

MySQL 5.1 リファレンスマニュアル

MySQL 5.1 リファレンスマニュアル

Copyright © 1997, 2011, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used without Oracle's express written authorization. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, for details on how the MySQL documentation is built and produced, or if you are interested in doing a translation, please visit [MySQL Contact & Questions](#).

For additional licensing information, including licenses for libraries used by MySQL products, see [はじめに](#).

If you want help with using MySQL, please visit either the [MySQL Forums](#) or [MySQL Mailing Lists](#) where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the [MySQL Documentation Library](#).

概要

本書はMySQLのリファレンスマニュアルです。MySQLの 5.1 から 5.1.15-betaについて解説します。

このマニュアルの詳細および最新の翻訳には、最近かもしれないいくつかのセクション。については、[MySQL 5.1 リファレンスマニュアル \(オンラインヘルプ\)](#)。

本書の作成日: 2012-09-28 (revision: 580)

目次

はじめに	xxi
1. 一般情報	1
このマニュアルについて	2
このマニュアルの表記規則	2
MySQL AB の概要	3
MySQL データベース管理システムの概要	4
MySQL とは?	4
MySQL の歴史	5
MySQL の主な機能	6
MySQL の開発ロードマップ	8
MySQL 5.1 での新機能	9
MySQL 5.2で計画されている新機能	10
MySQL の情報源	11
MySQL メーリング リスト	11
MySQL フォーラムにおける MySQL コミュニティサポート	13
IRC (インターネット リレー チャット) の MySQL コミュニティ サポート	13
質問またはバグの報告	14
MySQL の標準への準拠	17
MySQLが準拠する標準	18
SQLモードの選択	18
ANSIモードでのMySQLの実行	18
SQL標準に対するMySQL拡張機能	19
MySQLと標準SQLとの違い	21
MySQL における制約の処理	26
2. MySQL のインストールと更新	31
一般的なインストールの問題	32
MySQL Community Server がサポートしているオペレーティング システム	33
インストールする MySQL 配布版の選択	34
MySQL の取得方法	43
MD5 チェックサムあるいは GnuPG を用いたパッケージの品質の検証	43
インストールのレイアウト	45
バイナリの配布を使用した標準 MySQL のインストール	47
Windows に MySQL をインストールする	47
インストール用パッケージの選択	48
自動インストーラで MySQL をインストールする	48
MySQL インストール ウィザードを使用する	49
設定ウィザードを使用する	51
非インストール Zip アーカイブからのインストール	55
インストール アーカイブを取り出す	55
オプション ファイルの作成	55
MySQL サーバ タイプの選択	56
サーバを最初に起動する	57
MySQL の Windows のコマンドラインからの起動	58
Windows のサービスとして MySQL を起動する	59
MySQL インストールのテスト	61
Windows への MySQL インストールにおけるトラブルシューティング	61
Windows を使用した MySQL をアップグレードする	62
Windows 上の MySQL と Unix 上の MySQL の比較	63
Linux に MySQL をインストールする	65
Mac OS X に MySQL をインストールする	67
Solaris に MySQL をインストールする	69
MySQL を NetWare にインストールする	70
他の Unix 系システムへの MySQL のインストール	71
ソースのディストリビューションを使用した MySQL のインストール	73
ソースのインストール概要	74
典型的な configure オプション	76
開発ソース ツリーからのインストール	80

MySQL のコンパイルに関する問題	82
MIT-pthreads ノート	84
Windows にソースから MySQL をインストールする	85
Windows で MySQL クライアントをコンパイルする	88
インストール後の設定とテスト	89
Windows のインストール後のプロシージャ	89
Unix のインストール後のプロシージャ	90
最初の MySQL アカountの確保	99
MySQL のアップグレード	101
MySQL 5.0 から 5.1 へのアップグレード	102
MySQL データベースの他のマシンへのコピー	105
MySQL のダウングレード	106
MySQL 5.0 へのダウングレード	107
オペレーティング システムに特化した注釈	107
Linux の注釈	107
Mac OS X に関する注釈	113
Solaris に関する注釈	114
BSD に関する注釈	117
他の Unix に関する注釈	120
OS/2 に関する注釈	133
環境変数	133
Perl のインストールに関する注釈	134
Unix に Perl をインストールする	134
Windows に ActiveState Perl をインストールする	135
Perl DBI/DBD インターフェースを使用した際の問題	136
3. MySQL プログラムの使用	139
MySQL プログラム の概要	139
MySQL プログラムの起動	140
プログラム・オプションの指定	141
コマンドラインにおけるオプションの使用	141
オプションファイルの使用	143
オプション指定のための環境変数の使用	147
プログラム変数セットのためのオプション使用	148
4. データベース管理	149
サーバ サイド プログラムの概略	150
<code>mysqld</code> — MySQL サーバ	151
オプションと変数のリファレンス	152
コマンド オプション	152
システム変数	162
システム変数の使用	187
ステータス変数	195
SQL モード	204
シャットダウン プロセス	209
サーバ サイド ヘルプ	210
MySQL サーバ スタートアップ プログラム	210
<code>mysqld_safe</code> — MySQL サーバ スタートアップ スクリプト	210
<code>mysql.server</code> — MySQL サーバ スタートアップ スクリプト	213
<code>mysqld_multi</code> — 複数のMySQL サーバ管理	213
<code>mysqlmanager</code> — MySQL Instance Manager	217
MySQL Instance Manager コマンド オプション	217
MySQL Instance Manager の設定ファイル	220
MySQL Instance Manager で MySQL Server の起動	221
Instance Manager のユーザとパスワード管理	222
MySQL サーバ インスタンス ステータスの監視	224
MySQL Instance Manager へ接続	225
MySQL Instance Manager のコマンド	225
インストール関連プログラム	228
<code>make_win_bin_dist</code> — Package MySQL 配布 (ZIP アーカイブ)	228
<code>mysql_fix_privilege_tables</code> — MySQL システム テーブルのアップグレード	229
<code>mysql_install_db</code> — MySQL データ ディレクトリ 初期化スクリプト	230

mysql_upgrade — MySQL アップグレードのテーブル チェック	231
mysql_tzinfo_to_sql — タイム ゾーン テーブルのロード	232
セキュリティ問題	232
セキュリティ ガイドライン	232
MySQL のクラッカー対策	234
セキュリティ関連の mysqld オプション	236
LOAD DATA LOCAL のセキュリティ関連事項	237
ユーザによる MySQL の実行	238
MySQL アクセス権限システム	238
権限システムの役割	239
権限システムの機能	239
MySQL 提供の権限	243
MySQL サーバへの接続	246
アクセス制御の段階 1: 接続確認	246
アクセス制御の段階 2: 接続確認	250
権限の変更が反映するタイミング	252
Access denied エラーの原因	252
MySQL 4.1 のパスワードハッシュ	256
MySQL ユーザ アカウント管理	260
MySQL ユーザ名とパスワード	260
MySQL への新規ユーザの追加	261
MySQL ユーザの削除	264
ユーザ リソースの制限	264
パスワードの設定	265
パスワードのセキュリティ	266
接続安全	267
バックアップとリカバリ	274
データベースのバックアップ	274
バックアップとリカバリ手法の例示	276
任意時点のリカバリ	279
テーブル保守とクラッシュ リカバリ	280
MySQL のローカライズと国際的使用	289
データおよびソート用キャラクタ セット	289
英語以外のエラーメッセージ	290
新しいキャラクタ セットの追加	291
キャラクタ定義配列	292
文字列照合サポート	292
マルチ バイト文字サポート	293
キャラクタ セットに関する問題	293
MySQL サーバのタイム ゾーン サポート	293
MySQL サーバのローケル サポート	295
MySQL サーバ ログ	297
一般クエリとスロー クエリのログ出力先の選択	298
エラー ログ	299
一般クエリ ログ	300
バイナリ ログ	301
スロー クエリ ログ	304
ログ ファイルの保守	305
同じマシン上での複数 MySQL サーバの実行	306
Windows で複数サーバの実行	307
Unix で複数サーバの実行	310
複数サーバ環境でのクライアントプログラムの使用	311
MySQL クエリ キャッシュ	311
クエリ キャッシュの動作	312
クエリ キャッシュでの SELECT オプション	313
クエリ キャッシュの設定	314
クエリ キャッシュのステータスと保守	315
5. レプリケーション	317
レプリケーション設定	318
レプリケーションのセットアップ方法	319

レプリケーション フォーマット	326
レプリケーションのオプションと変数	330
レプリケーションでの管理タスク	336
Replication Topologies	338
Replication with a Single Slave	338
Replication with Multiple Slaves	338
Replication with Two Masters	339
Replication with Circular Masters	340
Replication Chains	340
Replicating Multiple Masters to One Slave	341
レプリケーション ソリューション	341
バックアップのレプリケーション	341
ストレージ エンジンが異なるマスタとスレーブのレプリケーション	343
スケールアウトのレプリケーション	344
異なるデータベースから異なるスレーブへのレプリケーション	346
レプリケーション パフォーマンスの改善	346
フェイルオーバーでのマスタ切り替え	348
SSLを使用するレプリケーションの設定	351
レプリケーション ノートとヒント	352
レプリケーション機能と既知問題	352
MySQL バージョン間のレプリケーション互換性	357
レプリケーション セットアップのアップグレード	357
レプリケーション FAQ	358
レプリケーションのトラブルシューティング	361
レプリケーション バグまたは問題を報告する方法	362
レプリケーションの実装	363
レプリケーション実装の詳細	363
マスタレプリケーションのスレッド状態	365
スレーブ レプリケーションの I/O スレッド状態	365
スレーブ レプリケーションの SQL スレッド状態	366
レプリケーション リレーとステータス ファイル	366
サーバのレプリケーション ルール評価	368
6. 最適化	371
最適化の概要	372
MySQL の設計上の制約とトレードオフ	372
移植性のためのアプリケーション設計	372
MySQL 使用実績	373
MySQL ベンチマークスイート	374
独自のベンチマークの使用	374
SELECTステートメントおよびその他のクエリの最適化	375
EXPLAINを使用して、クエリを最適化する	375
クエリパフォーマンスの推定	384
SELECTクエリ速度	384
WHERE 節最適化	384
Range 最適化	386
インデックス結合最適化	389
IS NULL最適化	391
DISTINCT最適化	391
LEFT JOINとRIGHT JOIN最適化	392
入れ子結合最適化	393
外側Join 単純化	397
ORDER BY最適化	399
GROUP BY最適化	401
LIMITの最適化	402
テーブルスキャンを避ける方法	403
INSERTステートメントの速度	403
UPDATEステートメントの速度	405
DELETEステートメントの速度	405
その他の最適化のヒント	405
ロック関連の問題	408

MySQL のテーブルロック方法	408
テーブルロック関連の問題	410
同時挿入	411
データベース構造の最適化	411
設計上の選択	411
データの小型化	412
カラムインデックス	413
複合インデックス	413
MySQLにおけるインデックスの使用	414
MyISAMキーキャッシュ	416
MyISAMインデックス統計コレクション	420
MySQL でのテーブルのオープンとクローズの方法	422
1つのデータベースに大量のテーブルを作成した場合の欠点	423
MySQL サーバの最適化	423
システム、コンパイル時間およびスタートアップパラメータのチューニング	423
サーバパラメータのチューニング	424
クエリオプティマイザパフォーマンスの管理	428
MySQL の速度に対するコンパイルとリンクの影響	429
MySQL でのメモリの使用	430
MySQLの DNS の使用	431
ディスク関連の問題	431
シンボリックリンクの使用	432
7. クライアントプログラムとユーティリティ プログラム	435
クライアントプログラムとユーティリティ プログラムの概要	435
my_print_defaults — オプション ファイルから オプションを表示する	437
mysam_ftdump — フル テキスト インデックス情報を表示する	438
myisamchk — MyISAM テーブル メンテナンス ユーティリティ	439
myisamchk 一般的なオプション	440
myisamchk チェック オプション	442
myisamchk修復オプション	442
他の myisamchk オプション	444
myisamchkメモリ使用量	445
myisamlog — Display MyISAM Log File Contents	445
mysampack — 圧縮された、読み取り専用MyISAM テーブルを作成する。	446
mysql — MySQL コマンド ライン ツール	452
mysql オプション	452
mysql Commands	457
mysqlサーバサイドヘルプ	460
テキストファイルからSQLステートメントを実行する	461
mysqlヒント	461
mysqlaccess — アクセス権限をチェックするクライアント	462
mysqladmin — MySQL サーバの管理を行うクライアント	464
mysqlbinlog — バイナリログファイルを処理するためのユーティリティ	468
mysqlcheck — テーブル メンテナンスと修復プログラム	474
mysqldump — データベースバックアッププログラム	477
mysqlhotcopy — データベースバックアッププログラム	485
mysqlimport — データインポートプログラム	487
mysqlshow — データベース、テーブル、カラム情報を表示します。	490
mysqlslap — クライアント負荷エミュレーション	491
mysql_zap — パターンとマッチする処理を消去します。	494
perror — エラーコードの説明	494
replace — 文字列置き換えユーティリティ	495
8. 言語構造	497
リテラル値	497
文字列	497
数値	499
16 進値	499
ブール値	500
ビットフィールド値	500
NULL値	500

識別子	500
識別子の修飾語	502
識別子の大文字/小文字区別	502
ファイル名への識別子のマッピング	503
関数名の構文解析と名前解決	505
MySQLでの予約語の扱い	507
ユーザによって定義された変数	510
コメント構文	511
9. キャラクタセットサポート	513
一般のキャラクタセットおよび照合順序	514
MySQLにおけるキャラクタセットおよび照合順序	514
デフォルトのキャラクタセットおよび照合順序の指定	516
サーバのキャラクタセットおよび照合順序	516
データベースのキャラクタセットおよび照合順序	516
テーブルのキャラクタセットおよび照合順序	517
カラムのキャラクタセットおよび照合順序	517
文字列リテラルのキャラクタセットおよび照合順序	518
各国キャラクタセット	519
キャラクタセットと照合順序の割り当ての例	520
他のDBMSとの互換性	520
接続のキャラクタセットおよび照合順序	520
照合順序に関して	522
SQLステートメントCOLLATE節を使用する	522
COLLATE節の優先順位	523
BINARY オペレータ	523
照合順序を決定するのが難しい特殊なケース	524
照合順序は適切なキャラクタセットに対応していること。	525
照合順序がもたらす結果の例	525
キャラクタセットのサポートによる影響を受ける演算	525
結果文字列	525
CONVERT() と CAST()	526
SHOW ステートメントとINFORMATION_SCHEMA	527
Unicodeのサポート	528
メタデータ用の UTF8	528
カラムキャラクタセット変換	529
MySQL でサポートされるキャラクタセットと照合順序	530
Unicode キャラクタセット	531
西ヨーロッパのキャラクタセット	533
中央ヨーロッパのキャラクタセット	535
南ヨーロッパおよび中東のキャラクタセット	535
バルト語のキャラクタセット	536
キリル語のキャラクタセット	536
アジアのキャラクタセット	537
10. データタイプ	541
データタイプ概要	541
数値タイプの概要	541
データと時刻タイプの概要	544
文字列タイプの概要	545
データタイプデフォルト値	547
数値タイプ	548
日付と時刻タイプ	550
DATETIME、DATE、そして TIMESTAMP タイプ	551
TIME タイプ	555
YEAR タイプ	556
2000年問題とデータタイプ	556
文字列タイプ	557
CHAR と VARCHAR タイプ	557
BINARY と VARBINARY タイプ	558
BLOBとTEXT タイプ	559
ENUM タイプ	560

SET タイプ	562
データタイプが必要とする記憶容量	563
カラムに適したタイプの選択	566
その他のデータベースエンジンのデータタイプの利用	566
11. 関数と演算子	569
演算子	570
演算子の優先順位	570
式評価でのタイプ変換	570
比較関数と演算子	572
論理演算子	575
制御フロー関数	577
文字列関数	578
文字列比較関数	587
正規表現	589
数字関数	592
算術演算子	592
数学関数	593
日付時刻関数	599
MySQL が使用するカレンダーは ?	613
全文検索関数	613
ブール全文検索	617
クエリ拡張を伴う全文検索	619
全文ストップワード	619
全文制限	622
微調整 MySQL 全文検索	622
キャスト関数と演算子	624
XML 関数	626
その他の関数	629
ビット関数	629
暗号化関数と圧縮関数	630
情報関数	634
その他の関数	640
GROUP BY 句との関数および修飾子の使用	642
GROUP BY (集約) 関数	642
GROUP BY 修飾子	645
非常時フィールドとの GROUP BY および HAVING	647
12. SQL ステートメント構文	649
データ定義ステートメント	650
ALTER DATABASE 構文	650
ALTER TABLE 構文	650
ALTER LOGFILE GROUP 構文	657
ALTER TABLESPACE 構文	658
ALTER SERVER 構文	659
CREATE DATABASE 構文	659
CREATE INDEX 構文	660
CREATE TABLE 構文	661
CREATE LOGFILE GROUP 構文	675
CREATE TABLESPACE 構文	676
CREATE SERVER 構文	677
DROP DATABASE 構文	677
DROP INDEX 構文	678
DROP TABLE 構文	678
DROP LOGFILE GROUP 構文	679
DROP TABLESPACE 構文	679
DROP SERVER 構文	679
RENAME DATABASE 構文	680
RENAME TABLE 構文	680
データ取り扱いステートメント	681
DELETE 構文	681
DO 構文	683

HANDLER 構文	683
INSERT 構文	684
LOAD DATA INFILE 構文	690
REPLACE 構文	697
SELECT 構文	698
サブクエリ構文	710
TRUNCATE 構文	719
UPDATE 構文	719
MySQL ユーティリティ ステートメント	721
DESCRIBE 構文	721
HELP 構文	721
USE 構文	723
MySQL トランザクションとロッキング関連のステートメント	724
START TRANSACTION、COMMIT、そして ROLLBACK 構文	724
ロールバックできないステートメント	726
暗黙のコミットを引き起こすステートメント	726
SAVEPOINT と ROLLBACK TO SAVEPOINT 構文	726
LOCK TABLES と UNLOCK TABLES 構文	727
SET TRANSACTION 構文	729
XA トランザクション	730
データベース管理ステートメント	733
アカウント管理ステートメント	733
テーブル メンテナンス ステートメント	742
SET 構文	747
SHOW 構文	752
その他の管理ステートメント	776
複製ステートメント	780
マスタ サーバをコントロールする SQL ステートメント	780
スレーブ サーバをコントロールする SQL ステートメント	783
プリペアド ステートメントの為の SQL 構文	790
13. ストレージエンジンとテーブルタイプ	793
MySQLストレージエンジンアーキテクチャの概要	794
共通データベースサーバ層	795
プラグابلなストレージエンジンアーキテクチャ	795
サポートされたストレージエンジン	796
ストレージエンジンの選択	797
トランザクションエンジンと、非トランザクションエンジンの比較	797
その他のストレージエンジン	798
ストレージエンジンの設定	799
MyISAM ストレージエンジン	799
MyISAM スタートアップオプション	801
キーに必要な領域	802
MyISAM テーブルストレージフォーマット	802
MyISAM テーブルの問題点	804
InnoDB ストレージ エンジン	806
InnoDB 概要	806
InnoDB 連絡先情報	806
InnoDB 設定	807
InnoDB 起動オプションとシステム変数	812
InnoDB テーブルスペースを作成する	818
InnoDB テーブルの作成と利用	819
InnoDB データとログ ファイルの追加と削除	825
InnoDB データベースのバックアップと復旧	826
InnoDB データベースを別のマシンに移動する	829
InnoDB トランザクション モデルとロック	829
InnoDB パフォーマンス チューニング ヒント	838
マルチバージョンの実装	843
InnoDB テーブルとインデックス構造	844
InnoDB ファイル領域の管理とディスク I/O	846
InnoDB エラー処理	848

InnoDB テーブル上の制約	853
InnoDB トラブルシューティング	855
MERGE ストレージエンジン	856
MERGE テーブルの問題点	859
MEMORY (HEAP) ストレージエンジン	860
EXAMPLE ストレージエンジン	862
FEDERATED ストレージエンジン	862
FEDERATED ストレージエンジン概要	862
FEDERATED テーブルの作成方法	863
FEDERATED ストレージエンジン 注意とヒント	866
FEDERATED ストレージエンジンリソース	866
ARCHIVE ストレージエンジン	866
CSV ストレージエンジン	867
CSVテーブル修正と確認	868
CSV制限	869
BLACKHOLE ストレージエンジン	869
14. MySQL Cluster	871
MySQL Cluster の概要	873
基本的な MySQL Cluster のコンセプト	875
MySQL Cluster ノード、ノード グループ、レプリカ、およびパーティション	876
簡単なマルチ コンピュータの手引き	878
ハードウェア、ソフトウェア、およびネットワークの構築	880
マルチ コンピュータのインストール	881
マルチ コンピュータの設定	883
最初の起動	885
サンプル データのローディングとクエリの実行	886
安全なシャットダウンと再起動	889
MySQL Cluster の設定	889
ソースコードによる MySQL Cluster の構築	890
ソフトウェアのインストール	890
MySQL Cluster の簡単なテストの設定	890
設定ファイル	892
クラスタ設定パラメータの概要	917
ローカル チェックポイントのパラメータの設定	923
MySQL Cluster のアップグレードおよびダウンロード	924
クラスタのローリング再起動の実行	924
クラスタのアップグレードおよびダウングレードの互換性	926
MySQL Cluster のプロセス管理	929
MySQL Cluster のための MySQL Server プロセスの使用法	929
ndbd、ストレージ エンジン ノード プロセス	930
ndb_mgmd、マネジメント サーバード プロセス	931
ndb_mgm、マネジメント クライアント プロセス	932
MySQL Cluster プロセスのコマンド オプション	932
MySQL Cluster の管理	935
MySQL Cluster 起動フェーズ	935
マネジメント クライアントのコマンド	937
MySQL Cluster で生成されたイベント レポート	937
単一ユーザーモード	944
MySQL Cluster のオンライン バックアップ	944
クラスタ バックアップの概念	944
バックアップを作成するためのマネジメント クライアントの使用	945
クラスタのバックアップの復旧方法	946
クラスタ バックアップの設定	948
バックアップのトラブルシューティング	949
クラスタ ユーティリティ プログラム	949
ndb_config — NDB 設定情報の抽出	950
ndb_delete_all — NDB テーブルからのすべての行を削除する	952
ndb_desc — NDB テーブルの説明	953
ndb_drop_index	954
ndb_drop_table	954

ndb_error_reporter	955
ndb_print_backup_file	955
ndb_print_schema_file	955
ndb_print_sys_file	956
ndb_select_all	956
ndb_select_count	958
ndb_show_tables	958
ndb_size.pl — NDBCluster サイズ仕様エスティメーター	959
ndb_waiter	961
MySQL Cluster レプリケーション	962
略語と記号	963
仮定条件と一般要件	964
既知の問題	964
レプリケーション スキーマおよびテーブル	965
レプリケーションにクラスタを準備する	967
レプリケーションの開始 (シングル レプリケーション チャンネル)	968
2 つのレプリケーション チャンネルを使用する	969
MySQL Cluster にフェールオーバーを導入する	970
MySQL Cluster のレプリケーションによるバックアップ	970
MySQL Cluster ディスク データ ストレージ	975
MySQL Cluster での高速インターコネクトを使用する	978
SCI ソケットを使用するための MySQL Cluster の設定	978
Cluster インターコネクトの理解する	981
MySQL Cluster の既知の制限	983
MySQL Cluster 開発ロードマップ	988
MySQL 5.1 における MySQL Cluster の変更点	988
MySQL Cluster の用語	989
15. パーティショニング	995
MySQL パーティショニングの概要	996
パーティショニングのタイプ	998
RANGE パーティショニング	999
LIST パーティショニング	1001
HASH パーティショニング	1003
KEY パーティショニング	1006
サブパーティショニング	1007
MySQLパーティショニングの NULL 値の取り扱い	1010
パーティショニング管理	1013
RANGE と LIST パーティションの管理	1014
HASH や KEY パーティションの管理	1019
パーティションのメンテナンス	1019
パーティション情報の取得	1020
パーティションの刈り込み	1022
パーティショニングの制約と制限	1025
16. Spatial Extensions	1029
Introduction to MySQL Spatial Support	1030
The OpenGIS Geometry Model	1030
The Geometry Class Hierarchy	1030
Class Geometry	1031
Class Point	1032
Class Curve	1033
Class LineString	1033
Class Surface	1033
Class Polygon	1033
Class GeometryCollection	1034
Class MultiPoint	1034
Class MultiCurve	1034
Class MultiLineString	1035
Class MultiSurface	1035
Class MultiPolygon	1035
Supported Spatial Data Formats	1035

Well-Known Text (WKT) Format	1035
Well-Known Binary (WKB) Format	1036
Creating a Spatially Enabled MySQL Database	1037
MySQL Spatial Data Types	1037
Creating Spatial Values	1037
Creating Spatial Columns	1040
Populating Spatial Columns	1040
Fetching Spatial Data	1041
Analyzing Spatial Information	1041
Geometry Format Conversion Functions	1041
Geometry Functions	1042
Functions That Create New Geometries from Existing Ones	1047
Functions for Testing Spatial Relations Between Geometric Objects	1047
Relations on Geometry Minimal Bounding Rectangles (MBRs)	1048
Functions That Test Spatial Relationships Between Geometries	1048
Optimizing Spatial Analysis	1049
Creating Spatial Indexes	1050
Using a Spatial Index	1051
MySQL Conformance and Compatibility	1052
17. ストアドプロシージャとファンクション	1053
ストアドルーチンとグラントテーブル	1054
ストアドルーチン構文	1054
CREATE PROCEDURE および CREATE FUNCTION 構文	1054
ALTER PROCEDURE および ALTER FUNCTION 構文	1058
DROP PROCEDURE および DROP FUNCTION 構文	1058
CALL ステートメント構文	1058
BEGIN ... END 複合ステートメント構文	1059
DECLARE ステートメント用構文	1060
ストアドルーチン内の変数	1060
条件とハンドラ	1061
カーソル	1063
フローコントロール・コンストラクト	1064
ストアドプロシージャ、ファンクション、トリガ並びに LAST_INSERT_ID()	1066
ストアドルーチンとトリガのバイナリログ	1066
18. トリガ	1071
CREATE TRIGGER 構文	1071
DROP TRIGGER 構文	1074
トリガの使用	1074
19. Event Scheduler	1077
Event Scheduler Overview	1077
Event Scheduler Syntax	1080
CREATE EVENT Syntax	1080
ALTER EVENT Syntax	1083
DROP EVENT Syntax	1085
Event Metadata	1085
Event Scheduler Status	1085
The Event Scheduler and MySQL Privileges	1086
Event Scheduler Limitations and Restrictions	1088
20. ビュー	1091
ALTER VIEW 構文	1091
CREATE VIEW 構文	1091
DROP VIEW 構文	1097
21. INFORMATION_SCHEMA データベース	1099
INFORMATION_SCHEMA SCHEMATA テーブル	1101
INFORMATION_SCHEMA TABLES テーブル	1101
INFORMATION_SCHEMA COLUMNS テーブル	1102
INFORMATION_SCHEMA STATISTICS テーブル	1103
INFORMATION_SCHEMA USER_PRIVILEGES テーブル	1104
INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル	1104
INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル	1105

INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル	1105
INFORMATION_SCHEMA CHARACTER_SETS テーブル	1106
INFORMATION_SCHEMA COLLATIONS テーブル	1106
INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル	1106
INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル	1106
INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル	1107
INFORMATION_SCHEMA ROUTINES テーブル	1108
INFORMATION_SCHEMA VIEWS テーブル	1109
INFORMATION_SCHEMA TRIGGERS テーブル	1109
INFORMATION_SCHEMA PLUGINS テーブル	1111
INFORMATION_SCHEMA ENGINES テーブル	1111
INFORMATION_SCHEMA PARTITIONS テーブル	1111
INFORMATION_SCHEMA EVENTS テーブル	1114
INFORMATION_SCHEMA FILES テーブル	1117
INFORMATION_SCHEMA PROCESSLIST テーブル	1121
INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル	1122
INFORMATION_SCHEMA GLOBAL_STATUS および SESSION_STATUS テーブル	1123
INFORMATION_SCHEMA GLOBAL_VARIABLES および SESSION_VARIABLES テーブル	1123
その他の INFORMATION_SCHEMA テーブル	1123
SHOW ステートメントへの拡張	1123
22. 精密計算	1127
数値のタイプ	1127
DECIMAL データ タイプの変更	1128
式の取り扱い	1129
丸め挙動	1130
精密計算の例	1131
23. APIとライブラリー	1135
埋め込まれたMySQLサーバライブラリ、libmysqld	1135
埋め込まれたMySQLサーバライブラリーの概括	1135
libmysqld使って行うプログラムの翻訳	1136
埋め込まれたMySQLサーバの使用に対する規制	1136
埋め込まれたサーバに対するオプション	1137
埋め込まれたサーバの例	1137
埋め込まれたサーバに対するのライセンスの供与	1140
MySQL C API	1140
C APIデータタイプ	1141
C API機能の概要。	1144
C API機能の説明	1148
準備されたC APIステートメント。	1193
準備されたC APIステートメントデータタイプ	1193
準備されたC APIステートメント機能の概要	1198
準備されたC APIステートメント機能の詳細	1201
準備されたC API ステートメントの問題	1220
マルチプルステートメントを実行する C APIハンドリング	1221
日付とタイム値のC API式取り扱い	1223
C APISレッド機能の説明	1224
埋め込まれたC API機能の説明	1225
自動再接続挙動の管理	1225
C APIを使うときよく尋ねられる質問と問題	1226
クライアントプログラムの構築	1228
スレッド付きクライアントを作る方法	1228
MySQL PHP API	1229
MySQLとPHPに対する共通問題	1230
mysql と mysqli の両方を PHP内で可能にする	1230
MySQL Perl API	1230
MySQL C++ API	1231
MySQL Python API	1231
MySQL Tcl API	1231
MySQLエッフェルラッパー	1231
MySQLプログラム開発ユーティリティ	1231

mysql2mysql — MySQLと一緒に使うため、mSQLプログラムを変換してください。	1232
mysql_config — コンパイルオプションをコンパイルクライアントのために (用に) 取得してください。	1232
24. MySQL コネクタ	1235
MySQL Connector/ODBC	1236
Connector/ODBC の概要	1236
Connector/ODBC のインストール	1240
Connector/ODBC の構成	1259
Connector/ODBC 例	1275
Connector/ODBC 参考資料	1301
Connector/ODBC に関する注釈とヒント	1306
Connector/ODBC サポート	1315
MySQL Connector/NET	1316
Connector/NET のバージョン	1317
Connector/NET のインストール	1317
Connector/NET の例と使用ガイド	1323
Connector/NET に関する注記とヒント	1367
Connector/NET のサポート	1375
MySQL Visual Studio プラグイン	1376
MySQL Visual Studio プラグインのインストール	1376
MySQL サーバとの接続の作成	1378
MySQL Visual Studio プラグインの使用	1379
Visual Studio プラグインのサポート	1387
MySQL Connector/J	1387
Connector/J バージョン	1388
Connector/J のインストール	1388
Connector/J 例	1392
Connector/J (JDBC) の参考	1392
Connector/J に関する注記とヒント	1408
Connector/J のサポート	1425
Connector/PHP	1427
25. Extending MySQL	1429
MySQL Internals	1429
MySQL Threads	1429
MySQL Test Suite	1430
The MySQL Plugin Interface	1430
Characteristics of the Plugin Interface	1431
Full-Text Parser Plugins	1432
INSTALL PLUGIN Syntax	1433
UNINSTALL PLUGIN Syntax	1433
Writing Plugins	1434
Adding New Functions to MySQL	1446
Features of the User-Defined Function Interface	1447
CREATE FUNCTION Syntax	1447
DROP FUNCTION Syntax	1447
Adding a New User-Defined Function	1448
Adding a New Native Function	1455
Adding New Procedures to MySQL	1456
Procedure Analyse	1456
Writing a Procedure	1456
A. Frequently Asked Questions About MySQL 5.1	1459
MySQL 5.1 FAQ — General	1459
MySQL 5.1 FAQ — Storage Engines	1460
MySQL 5.1 FAQ — Server SQL Mode	1461
MySQL 5.1 FAQ — Stored Procedures	1462
MySQL 5.1 FAQ — Triggers	1464
MySQL 5.1 FAQ — Stored Routines, Triggers, and Replication	1466
MySQL 5.1 FAQ — Views	1467
MySQL 5.0 FAQ — INFORMATION_SCHEMA	1468
MySQL 5.1 FAQ — Migration	1469

MySQL 5.1 FAQ — Security	1469
MySQL 5.1 FAQ — MySQL Cluster	1470
MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets	1478
MySQL 5.1 FAQ — Connectors & APIs	1489
B. Errors, Error Codes, and Common Problems	1491
Problems and Common Errors	1491
How to Determine What Is Causing a Problem	1491
Common Errors When Using MySQL Programs	1492
Installation-Related Issues	1503
Administration-Related Issues	1505
Query-Related Issues	1510
Optimizer-Related Issues	1515
Table Definition-Related Issues	1515
Known Issues in MySQL	1517
Server Error Codes and Messages	1519
Client Error Codes and Messages	1559
C. MySQL Change History	1563
Changes in release 5.1.x (Development)	1564
Changes in release 5.1.16 (Not yet released)	1564
Changes in release 5.1.15 (25 January 2007)	1568
Changes in release 5.1.14 (05 December 2006)	1575
Changes in release 5.1.13 (Not released)	1580
Changes in release 5.1.12 (24 October 2006)	1584
Changes in release 5.1.11 (26 May 2006)	1611
Changes in release 5.1.10 (Not released)	1614
Changes in release 5.1.9 (12 April 2006)	1620
Changes in release 5.1.8 (Not released)	1622
Changes in release 5.1.7 (27 February 2006)	1630
Changes in release 5.1.6 (01 February 2006)	1633
Changes in release 5.1.5 (10 January 2006)	1637
Changes in release 5.1.4 (21 December 2005)	1638
Changes in release 5.1.3 (29 November 2005)	1639
Changes in release 5.1.2 (Not released)	1640
Changes in release 5.1.1 (Not released)	1641
MySQL Connector/ODBC (MyODBC) Change History	1641
Changes in Connector/ODBC 5.0.11 (31 January 2007)	1641
Changes in Connector/ODBC 5.0.10 (14 December 2006)	1641
Changes in Connector/ODBC 5.0.9 (22 November 2006)	1642
Changes in Connector/ODBC 5.0.8 (17 November 2006)	1642
Changes in Connector/ODBC 5.0.7 (08 November 2006)	1643
Changes in Connector/ODBC 5.0.6 (03 November 2006)	1643
Changes in Connector/ODBC 5.0.5 (17 October 2006)	1643
Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)	1644
Changes in Connector/ODBC 5.0.2 (Never released)	1644
Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)	1644
Changes in Connector/ODBC 3.51.13 (Not yet released)	1645
Changes in Connector/ODBC 3.51.12	1645
Changes in Connector/ODBC 3.51.11	1646
Connector/NET Change History	1646
Changes in MySQL Connector/NET Version 5.0.4 (Not yet released)	1646
Changes in MySQL Connector/NET Version 5.0.3 (05 January 2007)	1646
Changes in MySQL Connector/NET Version 5.0.2 (06 November 2006)	1647
Changes in MySQL Connector/NET Version 5.0.1 (01 October 2006)	1648
Changes in MySQL Connector/NET Version 5.0.0 (08 August 2006)	1648
Changes in MySQL Connector/NET Version 1.0.10 (Not yet released)	1649
Changes in MySQL Connector/NET Version 1.0.9 (02 February 2007)	1649
Changes in MySQL Connector/NET Version 1.0.8 (20 October 2006)	1650
Changes in MySQL Connector/NET Version 1.0.7 (21 November 2005)	1651
Changes in MySQL Connector/NET Version 1.0.6 (03 October 2005)	1652
Changes in MySQL Connector/NET Version 1.0.5 (29 August 2005)	1652

Changes in MySQL Connector/NET Version 1.0.4 (20 January 2005)	1652
Changes in MySQL Connector/NET Version 1.0.3-gamma (12 October 2004)	1653
Changes in MySQL Connector/NET Version 1.0.2-gamma (15 November 2004)	1653
Changes in MySQL Connector/NET Version 1.0.1-beta2 (27 October 2004)	1654
Changes in MySQL Connector/NET Version 1.0.0 (01 September 2004)	1655
Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)	1655
Changes in MySQL Connector/NET Version 0.76	1658
Changes in MySQL Connector/NET Version 0.75	1659
Changes in MySQL Connector/NET Version 0.74	1660
Changes in MySQL Connector/NET Version 0.71	1661
Changes in MySQL Connector/NET Version 0.70	1662
Changes in MySQL Connector/NET Version 0.68	1663
Changes in MySQL Connector/NET Version 0.65	1664
Changes in MySQL Connector/NET Version 0.60	1664
Changes in MySQL Connector/NET Version 0.50	1664
MySQL Visual Studio Plugin Change History	1664
Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)	1664
Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)	1664
Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)	1665
MySQL Connector/J Change History	1665
Changes in MySQL Connector/J 5.1.x	1665
Changes in MySQL Connector/J 5.0.x	1665
Changes in MySQL Connector/J 3.1.x	1669
Changes in MySQL Connector/J 3.0.x	1684
Changes in MySQL Connector/J 2.0.x	1695
Changes in MySQL Connector/J 1.2b (04 July 1999)	1699
Changes in MySQL Connector/J 1.2.x and lower	1699
D. 制限と規制	1703
ストアド ルーチンとトリガの規制	1703
サーバサイドカーソルの規制	1705
サブクエリの規制	1705
ビューの規制	1707
XA トランザクションの規制	1709
MySQL の制限	1709
結合の制限	1709
E. Credits	1711
Developers at MySQL AB	1711
Contributors to MySQL	1716
Documenters and translators	1719
Libraries used by and included with MySQL	1721
Packages that support MySQL	1721
Tools that were used to create MySQL	1722
Supporters of MySQL	1722
索引	1725

はじめに

これは、MySQLデータベースシステムのリファレンスマニュアルです。バージョン 5.1、リリース 5.1.15-betaのMySQLソフトウェアについて記述しています。機能など多くの違いがバージョン 5.1 とそれ以前のバージョンにはあるため、古いバージョンのMySQLソフトウェアの説明については意図していません。もしあなたが前リリースのMySQLソフトウェアを使用している場合には、これはMySQL 5.0 について説明した MySQL 5.0 Reference Manualを参照して下さい。または、MySQL 4.1以前のバージョンについて記述された MySQL 3.23, 4.0, 4.1 リファレンスマニュアルを参照して下さい。MySQL 5.1のマイナーバージョンとの変更点は、リリース番号(5.1.x)への参照とともに、このテキストに書かれています。

第1章 一般情報

目次

このマニュアルについて	2
このマニュアルの表記規則	2
MySQL AB の概要	3
MySQL データベース管理システムの概要	4
MySQL とは?	4
MySQL の歴史	5
MySQL の主な機能	6
MySQL の開発ロードマップ	8
MySQL 5.1 での新機能	9
MySQL 5.2で計画されている新機能	10
MySQL の情報源	11
MySQL メーリング リスト	11
MySQL フォーラムにおける MySQL コミュニティサポート	13
IRC (インターネット リレー チャット) の MySQL コミュニティ サポート	13
質問またはバグの報告	14
MySQLの標準への準拠	17
MySQLが準拠する標準	18
SQLモードの選択	18
ANSIモードでのMySQLの実行	18
SQL標準に対するMySQL拡張機能	19
MySQLと標準SQLとの違い	21
MySQL における制約の処理	26

MySQL ® ソフトウェアによって、非常に高速で堅牢なマルチスレッド形式のマルチユーザ SQL (Structured Query Language) データベース サーバが実現します。MySQL サーバは、ミッション クリティカルで負荷が高い運用システムにも、大規模に展開されるソフトウェアへの組み込みにも対応するように設計されています。MySQL は MySQL AB の商標です。

MySQL ソフトウェアはデュアル ライセンス製品です。ユーザは、GNU 一般公衆利用許諾契約書(<http://www.fsf.org/licenses/>) の条件に基づいてオープン ソース/フリー ソフトウェア製品として MySQL ソフトウェアを使用することも、MySQL AB から標準の商用ライセンスを購入することもできます。さらに詳しいライセンス ポリシーや情報については <http://www.mysql.com/company/legal/licensing/> を参照してください。

このマニュアルで特に興味深いセクションは以下のとおりです。

- MySQL データベース サーバの機能に関する説明については、[MySQL の主な機能](#) を参照してください。
- 今後のプランに関しては、次を参照してください [MySQL の開発ロードマップ](#)。
- インストールの手順については、次を参照してください [2章MySQL のインストールと更新](#)。アップグレードに関する情報については、[MySQL のアップグレード](#) を参照してください。
- MySQL データベース サーバのチュートリアルについては、[Tutorial](#) を参照してください。
- MySQL サーバの管理とコンフィギュレーションに関する情報は、[4章データベース管理](#) を参照してください。
- レプリケーション サーバを設定する情報については、[5章レプリケーション](#) を参照してください。
- MySQL データベース サーバとその機能についてよくある質問に対する答えは、[付録A Frequently Asked Questions About MySQL 5.1](#) を参照してください。
- 既知のバグや機能上の問題の一覧については、[Known Issues in MySQL](#) を参照してください。
- このプロジェクトへの全貢献者の一覧については、[付録E Credits](#) を参照してください。
- 新機能とバグ修正の履歴については、[付録C MySQL Change History](#) を参照してください。

- 新しいアーキテクチャまたはオペレーティング システムへの MySQL Database Software の移植に関するヒントについては、[Debugging and Porting MySQL](#) を参照してください。
- SQL のサンプルとベンチマーク情報については、ベンチマーク ディレクトリ (ディストリ ビューションの [sql-bench](#)) を参照。

重要:

エラー (多くの場合、 「bugs」 と呼ばれます) の報告は、[質問またはバグの報告](#) で記されている手順に従ってください。

MySQL サーバで重大なセキュリティバグを見つけた場合は、[<security@mysql.com>](mailto:security@mysql.com) に電子メールをお送りください。

このマニュアルについて

これは、MySQL データベース システムのリファレンス マニュアルです。バージョン 5.1、リリース 5.1.15-beta の MySQL ソフトウェアについて記述しています。機能など多くの違いがバージョン 5.1 と前のバージョンにあるため、古いバージョンの MySQL ソフトウェアの説明については意図していません。もしあなたが前リリースの MySQL ソフトウェアを使用している場合には、これは MySQL 5.0 について説明した [MySQL 5.0 Reference Manual](#) を参照して下さい。または、MySQL 3.23、4.0、そして 4.1 をカバーする [MySQL 3.23, 4.0, 4.1 リファレンス マニュアル](#) を参照して下さい。MySQL 5.1 のマイナー バージョンとの変更点は、リリース番号 (5.1.x) への参照とともに、このテキストに書かれています。

これはリファレンス マニュアルなので、SQL やリレーショナル データベースの概念に関する一般的な説明は記載していません。また、ユーザの OS やコマンドライン インタープレターの説明も記載していません。

MySQL データベース ソフトウェアは継続して開発が行われているので、リファレンス マニュアルも頻繁に更新されます。このマニュアルの最新版は、<http://dev.mysql.com/doc/> から入手することができます。HTML、PDF、Windows CHM 版などのさまざまな形式で入手することも可能です。

リファレンス マニュアルのソース ファイルは、DocBook XML フォーマット内に書かれています。HTML バージョンとその他のフォーマットは、主に DocBook XSL スタイル シートを使用して自動的に作成されます。DocBook についてさらに詳しい情報は、<http://docbook.org/> を参照してください。

MySQL を利用するにあたって質問がある場合は、メーリング リストやフォーラムを使用してお尋ねください。[MySQL メーリング リスト](#) と [MySQL フォーラムにおける MySQL コミュニティサポート](#) を参照してください。このマニュアルへの追加や修正に関する提案は、マニュアル チーム (<http://www.mysql.com/company/contact/>) にお送りください。

このマニュアルは当初、David Axmark と Michael 「Monty」 Widenius によって執筆されました。現在は、Paul DuBois、Stefan Hinz、Jon Stephens、Martin MC Brown および Peter Lavin で構成される MySQL マニュアル チームによって管理されています。その他多数の貢献者については [付録E Credits](#) を参照してください。

このマニュアルの著作権はスウェーデンの会社 MySQL AB が所有しています。MySQL ® と MySQL のロゴは、MySQL AB のトレードマークとして登録されています。本マニュアル内で用いられている上記以外のトレードマークおよび登録トレードマークにはそれぞれ所有者が存在し、識別目的でのみ使用されます。

このマニュアルの表記規則

このマニュアルは、以下の表記規則に従って記載されています。

- [このスタイルのフォント](#) は、SQL ステートメント、データベース名、テーブル名、カラム名、C コード、Perl コード、および環境変数に使用されます。例: 「供与テーブルをリロードするには [FLUSH PRIVILEGES](#) を使用してください。」
- [このスタイルのフォント](#) は例であなたが入力する文字を示します。
- [このスタイルのフォント](#) は実行プログラムとスクリプトの名前、たとえば、[mysql](#) (MySQL コマンド ラインのクライアントプログラム) や [mysqld](#) (MySQL サーバ実行プログラム) を示します。
- [このスタイルのフォント](#) はあなたが値を設定すべき変数の入力に使用されます。
- ファイル名と階層名は、このように書かれます: 「グローバルな [my.cnf](#) ファイルは階層 [etc](#) に置かれます。」

- 文字シーケンスはこのように書かれます: 「ワイルドカードを指定するには '`%`' 文字を使用します。」
- このスタイルのフォントは強調に使用されます。
- このスタイルのフォントは表の見出しや特に強い強調を表す場合に使用されます。

特定のプログラムからコマンドを実行する必要があることを示す場合、コマンドの前にプログラムとプロンプトを記述します。たとえば、`shell>` はログインシェルから実行するコマンドを示し、`mysql>` は `mysql` クライアント プログラムから実行するコマンドを示します。

```
shell> シェルコマンドをここに入力します
mysql> mysqlステートメントをここに入力します
```

「シェル」とはコマンド インタープリタのことです。Unixでは通常、`sh`、`csh`、`bash`などのプログラムです。Windows では、Windows コンソールで通常実行される `command.com` や `cmd.exe` です。

コマンドやステートメントを例で入力したときには、プロンプトまでを入力しないでください。

データベース名、テーブル名、およびカラム名は多くの場合、コマンドに代入する必要があります。このような代入が必要であることを示す場合、このマニュアルでは `db_name`、`tbl_name`、および `col_name` を使用します。たとえば、次のようなステートメントがあるとしします。

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

これは、同様のステートメントを入力する場合、データベース名、テーブル名、およびカラム名を自分で指定することを意味します。たとえば、次のようになります。

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL キーワードは大文字と小文字を区別しないので、大文字で記述されることも小文字で記述されることもあります。ただし、このマニュアルでは大文字を使用します。

構文の説明で省略可能な単語や節を表す場合、角かっこ ('`[` および `]`') を使用します。たとえば、次のステートメントでは、`IF EXISTS` は省略可能です。

```
DROP TABLE [IF EXISTS] tbl_name
```

構文要素が複数の選択候補で構成される場合、それらの候補を縦線 ('`|`') で区切ります。選択候補のいずれかを選択できる場合は、次のように、それらの候補を角かっこ ('`[` および `]`') 内に列挙します。

```
TRIM([BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

選択候補のいずれかを選択できる場合は、次のように、それらの候補を角かっこ ('`[` および `]`') 内に列挙します。

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

省略記号(...) はステートメントのセクションの省略、とりわけ複雑な構文の簡略化を示します。たとえば、`INSERT ...SELECT` は `INSERT` ステートメントの後に `SELECT` ステートメントが続く場合を簡略して表記します。

省略記号はステートメントの反復も示します。たとえば次の例では、最初のコンマの後に続き複数の `reset_option` 値を与えられています。

```
RESET reset_option [,reset_option] ...
```

シェル変数をセットするコマンドは Bourne シェルの構文として示されます。たとえば、`CC` 環境変数をセットして `configure` コマンドを実行するシーケンスは、Bourne シェルの構文では次のようになります。

```
shell> CC=gcc ./configure
```

`csh` または `tcsh` をご使用の際には、適宜変更して使用してください。

```
shell> setenv CC gcc shell> ./configure
```

MySQL AB の概要

MySQL ABはMySQLの創始者と主な開発者からなる会社です。最初はDavid Axmark, Allan Larsson, and Michael 「Monty」 Wideniusによってスウェーデンに創設されました。

当社は、MySQLソフトウェアの開発と新しいユーザへの当社のデータベースの普及に専念しております。MySQL AB は、MySQL のソースコード、MySQL のロゴと商標、およびこのマニュアルの著作権を所有しています。[MySQL データベース管理システムの概要](#)を参照してください。

MySQL の本質的価値が、MySQL およびオープンソースへの当社の献身を示しています。

当社は、MySQL データベースソフトウェア が次のようなものであることを願っています。

- 世界中で最も広く使用される、世界最高のデータベースである。
- だれもが入手でき、価格が手ごろである。
- 使いやすい。
- 速度と安全性を確保しながら、改良を続ける。
- 楽しく使用および改良することができる。
- バグがない。

MySQL AB と MySQL AB の従業員は、以下のことを心がけています。

- オープンソースの理念を推進し、オープンソースコミュニティをサポートする。
- 健全なメンバーである。
- 当社の価値観と考え方を共有するパートナーを優先する。
- 電子メールに回答し、サポートを提供する。
- 他者とネットワークで結ばれたバーチャルな企業である。
- ソフトウェアの特許権に反対する立場をとる。

MySQL の Web サイト (<http://www.mysql.com/>) には、MySQL と MySQL AB に関する最新情報が掲載されています。

会社名の「AB」の部分は、スウェーデン語の「aktiebolag,」つまり「株式会社」の頭字語です。したがって、「MySQL 株式会社」という意味です。実際、MySQL AB の子会社として、MySQL Inc. や MySQL GmbH などがあります。これらの子会社はそれぞれ、アメリカとドイツにあります。

MySQL データベース管理システムの概要

MySQL とは？

MySQL は最もよく知られているオープンソース SQL データベース管理システムで、MySQL AB によって開発、提供、サポートが行われています。MySQL AB は MySQL の開発者によって創設された営利会社で、MySQL データベース管理システムに関連するサービスを提供することでビジネスを構築しています。

MySQL の Web サイト (<http://www.mysql.com/>) には、MySQL ソフトウェアと MySQL AB に関する最新情報が掲載されています。

- MySQL はデータベース管理システムです。

データベースとは、構造化されたデータの集合です。データベースには、簡単なショッピングリストから絵画のギャラリー、または社内ネットワークの膨大な量の情報など、さまざまなものがあります。コンピュータデータベースに格納されているデータに対して追加、アクセス、および処理などを行うには、MySQL サーバのようなデータベース管理システムが必要となります。コンピュータは大量のデータの処理に非常に優れているので、データベース管理システムは、スタンドアロンのユーティリティまたは他のアプリケーションの一部として、データ処理の中心的な役割を果たします。

- MySQL はリレーショナルデータベース管理システムです。

リレーショナルデータベースでは、1つの大きな保管領域にすべてのデータが格納されるのではなく、個別のテーブルにデータが格納されます。これにより、速度と柔軟性が向上します。「MySQL」の SQL は「Structured Query Language」を表します。SQL はデータベースへのアクセスに使用される最も一般的な標準化言語で、ANSI/

ISO SQL 標準によって定義されています。SQL 標準は 1986 年以降開発が繰り返され、複数のバージョンがあります。本マニュアルでは、「SQL-92」は 1992 年にリリースされた標準に、「SQL:1999」は 1999 年にリリースされた標準に、そして「SQL:2003」は標準の現バージョンに対応しています。「the SQL standard」という用語は、任意の時点における現行バージョンの SQL 標準を表す場合に使用します。

- MySQL ソフトウェアはオープンソースです。

オープンソースとは、だれもがそのソフトウェアを使用し、変更できることを意味します。MySQL ソフトウェアは、だれもが無料でインターネットからダウンロードし、使用することができます。必要に応じて、ソースコードを調べ、ニーズに合わせて変更することができます。MySQL ソフトウェアでは、GPL (GNU 一般公衆利用許諾契約書<http://www.fsf.org/licenses/>) に基づいて、さまざまな状況でのソフトウェアの使用において許可されている事項と禁止されている事項が規定されています。GPL では不都合な場合や商用アプリケーションに MySQL コードを組み込む必要がある場合は、商用ライセンス版を購入することができます。詳しくは MySQL ライセンスを参照してください。(http://www.mysql.com/company/legal/licensing/).

- MySQL データベース サーバ は、非常に高速で信頼性があり、簡単に使用することができます。

それを求めているのであれば、ぜひ試してください。また、MySQL サーバには、ユーザと密接に協力して開発された実用的な機能が備わっています。ベンチマークページで、他のデータベース管理システムと MySQL サーバのパフォーマンスを比較することができます。[MySQL ベンチマークスイート](#) を参照してください。

MySQL サーバは当初、既存のソリューションよりもはるかに速く大規模なデータベースを処理することを目的として開発され、多くを要求される運用環境で数年間にわたって使用されています。引き続き開発が行われていますが、MySQL サーバは現在、便利で多彩な機能を備えています。その接続性、速度、安全性によって、MySQL サーバはインターネット上のデータベースへのアクセスに非常に適しています。

- MySQL サーバはクライアント/サーバもしくは組み込まれたシステム。

MySQL データベースソフトウェアは、さまざまなバックエンドをサポートするマルチスレッド形式の SQL サーバ、複数の異なるクライアントプログラムおよびライブラリ、管理ツール、幅広いアプリケーションプログラミングインタフェース (API) から成るクライアント/サーバシステムです。

また、MySQL サーバはアプリケーションへのリンクが可能なマルチスレッド ライブラリとして提供されており、さらに小規模で高速な、管理しやすい製品を作り出すことができます。

- MySQL をサポートする多数のソフトウェアが提供されています。

必要なアプリケーションや言語ですでに MySQL データベース サーバがサポートされていると思われます。

「MySQL」の正式な発音は「My Ess Que Ell」ですが(「my sequel」ではありません)、「my sequel」と発音したり、ローカライズされた他の方法で発音してもかまいません。

MySQL の歴史

当初は、高速で低レベルな独自の (ISAM) ルーチンを使用してテーブルに接続するために [mSQL](#) を使用するつもりでした。しかし、いくつかのテストを行った結果、[mSQL](#) はそれほど高速でも柔軟でもないため、ニーズに合わないという結論に至りました。これが要因となって、私たちのデータベースへの新しい SQL インタフェースを開発することになりました。ただし、[mSQL](#) とほとんど同じ API インタフェースを使用することにしました。この API を選択したのは、[mSQL](#) で使用するために記述されたサードパーティのコードを MySQL で使用するために簡単に移植できるようにするためです。

MySQL という名前の由来は明らかではありません。当社の基本ディレクトリおよび多数のライブラリやツールには、10 年以上にわたって「my」というプリフィックスが使われています。また、共同創設者 Monty Widenius の娘の名前も My です。そのため、このうちのどちらが MySQL の名前の由来となったのかは、私たちにとってもいまだに謎なのです。

MySQL のドルフィン (当社のロゴ) の名前は「Sakila」です。Sakila は、当社の「ドルフィンのネーミング」コンテストで、ユーザによって提案された膨大な数の名前の中から MySQL AB の創設者が選んだものです。この名前を提案したのは、アフリカのスワジランド出身の Ambrose Twebaze というオープンソースソフトウェアの開発者でした。Ambrose によると、Sakila という名前は、スワジランドの現地語であるシスワティ語にルーツがあるということです。また、Sakila は、Ambrose の出生国であるウガンダに近い、タンザニアのアルーシャにある町の名前でもあります。

MySQL の主な機能

このセクションでは MySQL データベース ソフトウェアの重要な特徴の一部を説明します。現行版と最新情報を [MySQL の開発ロードマップ](#) で参照してください。おおむね、MySQL 全てのバージョンに対応しています。シリーズごとに新しく紹介される MySQL の機能については、対応するマニュアルの「一言」セクションを参照してください。

- MySQL 4.0 と 4.1: [MySQL 4.0 in a Nutshell](#), and [MySQL 4.1 in a Nutshell](#).
- MySQL 5.0: [MySQL 5.0 in a Nutshell](#).
- MySQL 5.1: [MySQL 5.1 in a Nutshell](#).

内部および移植性：

- C および C++ で記述されています。
- さまざまなコンパイラでテストされています。
- さまざまなプラットフォームで動作します。 [MySQL Community Server](#) がサポートしているオペレーティング システムを参照してください。
- GNU 移植性のために GNU Automake、Autoconf、および Libtool を使用しています。
- MySQL サーバは多層で、独立モジュールでデザインされています。
- カーネルスレッドを使用した完全なマルチスレッド。そのため、使用可能な場合、複数の CPU を簡単に使用することができます。
- トランザクションストレージエンジンと非トランザクションストレージエンジンを備えています。
- インデックス圧縮を備えた非常に高速な B-tree ディスクテーブル([MyISAM](#)) を使用しています。
- 別のストレージエンジンの追加が比較的容易です。これは、社内データベースへの SQL インタフェースを追加する場合に便利です。
- スレッドベースの非常に高速なメモリ割り当てシステム。
- 最適化された one-sweep multi-join を使用した非常に迅速な結合。
- テンポラリ テーブルとして使用されるメモリ内ハッシュテーブル。
- 高度に最適化されたクラス ライブラリから SQL 関数が実装されるため、最大限の速度が確保されます。通常は、クエリの初期化後にメモリ割り当てが行われることはありません。
- MySQL コードは Purify (市販のメモリリーク検出システム) と Valgrind と呼ばれる GPL ツール(<http://developer.kde.org/~sewardj/>)を使用してテストされています。
- クライアント/サーバまたは組み込み (リンク) バージョンとして使用可能です。単独のアプリケーションに組み込み (リンク) できるライブラリとしても提供されています。このようなアプリケーションは単一で、あるいはネットワーク環境の整っていない場所でも使用することができます。

データ タイプ：

- 多数のデータ タイプ：1、2、3、4、および 8 バイト長の符号付き/符号なし整数、[FLOAT](#)、[DOUBLE](#)、[CHAR](#)、[VARCHAR](#)、[TEXT](#)、[BLOB](#)、[DATE](#)、[TIME](#)、[DATETIME](#)、[TIMESTAMP](#)、[YEAR](#)、[SET](#)、[ENUM](#) および OpenGIS 空間型。 [10章データタイプ](#) を参照してください。
- 固定長および可変長のレコード。

コマンドと関数：

- クエリの [SELECT](#) 節および [WHERE](#) 節での演算子と関数の完全なサポート。たとえば、次のように使用することができます。例：

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
```



```
-> FROM citizen
-> WHERE income/dependents > 10000 AND age > 30;
```

- SQL の **GROUP BY** 節および **ORDER BY** 節の完全なサポート。グループ関数 (**COUNT()**、**COUNT(DISTINCT ...)**、**AVG()**、**STD()**、**SUM()**、**MAX()**、**MIN()**、そして **GROUP_CONCAT()**) のサポート。
- 標準の SQL 構文および ODBC 構文での **LEFT OUTER JOIN** および **RIGHT OUTER JOIN** のサポート。
- 標準 SQL で必要な、テーブルおよびカラムにおけるエイリアスのサポート。
- **DELETE**、**INSERT**、**REPLACE**、および **UPDATE** 変更された (影響を受けた) レコードの数を返す。サーバに接続する際にフラグを設定することで、代わりに一致したレコードの数を返すことも可能です。
- MySQL 固有の **SHOW** コマンドを使用すると、データベース、テーブル、およびインデックスに関する情報を取得することができます。MySQL 5.0 では、**INFORMATION_SCHEMA** データベースのサポートも、標準SQLに基づき追加されています。
- **EXPLAIN** コマンドを使用すると、オプティマイザによるクエリの解決方法を決定することができます。
- 関数名は、テーブル名やカラム名と衝突しません。例、**ABS** は有効なカラム名である。関数呼び出しで、関数名とその後続く '(' との間にスペースを使用できない点が唯一の制限事項です。[MySQLでの予約語の扱い](#) を参照してください。
- 同ステートメント内のさまざまなデータベースのテーブルを参照することができます。

セキュリティ :

- 非常に柔軟で安全な権限およびパスワードシステム。ホストベースの検証が可能です。
- サーバに接続する際にすべてのパスワードトラフィックが暗号化されるので、パスワードは安全です。

拡張性と範囲 :

- 大規模なデータベースを処理します。当社は、MySQL サーバを使用して 50,000,000 レコードが格納されたデータベースを処理しています。また、MySQL サーバを使用して 60,000 テーブル、約 5,000,000,000 レコードを処理しているユーザもいます。
- 各テーブルで最高 64個のインデックスが使用可能です。(MySQL 4.1.2では32個)。各インデックスは、1 から 16 個のカラムまたはカラムの一部で構成されます。インデックスの最大幅は 1000 バイトである (これは、MySQL サーバのコンパイル時に変更可能である)。(InnoDB では 767) MySQL 4.1.2 では 500 が限度でした。インデックスでは、**CHAR**、**VARCHAR**、**BLOB**、あるいは **TEXT** 型のカラムのプリフィックスを使用することができます。

接続性 :

- クライアントは複数のプロトコルを使用して MySQL に接続できます。
 - クライアントは、あらゆるプラットフォームで TCP/IP ソケットを使用して MySQL サーバに接続することができます。
 - WindowsNT ファミリ (NT、2000、XP、2003、または Vista) の Windows システムでは、サーバが **--enable-named-pipe** オプションで起動された場合、クライアントは名前付きパイプを使用して接続することができます。MySQL 4.1以降では、**--shared-memory** オプションで起動されていれば Windows のサーバは共有メモリコネクションもサポートされます。クライアントは **--protocol=memory** オプションを使用して共有メモリで接続できます。
 - Unix システムでは、Unix ドメイン ソケット ファイルを使用して接続することができます。
- MySQL クライアント プログラムはさまざまな言語で記述できます。C 言語で記述されたクライアントライブラリは C、C++、あるいは C バインディングを提供するどの言語でも提供されています。Eiffel、Java、Perl、PHP、Python、Ruby、Tcl、そして他言語での API が提供されています。[23章APIとライブラリー](#) を参照してください。
- C、C++、Eiffel、Java、Perl、PHP、Python、Ruby、そして Tcl の API が提供されています。よって、MySQL のクライアントがさまざまな言語で記述できます。[23章APIとライブラリー](#) を参照してください。

- Connector/ODBC (MyODBC) インタフェースによって、ODBC (Open-DataBase-Connectivity) 接続を使用するクライアントプログラムに MySQL サポートが提供されます。たとえば、MS Access を使用して MySQL サーバに接続することができます。クライアントは、Windows と Unix のどちらで実行されていてもかまいません。MyODBC ソースが使用可能です。他の多くの機能と同様に、ODBC 2.5 のすべての機能がサポートされます。[24章MySQL コネクタ](#) を参照してください。
- Connector/J interface は JDBC 接続を使用する Java クライアント プログラムの MySQL サポートを提供しています。クライアントは、Windows と Unix のどちらで実行されていてもかまいません。Connector/J ソースが使用可能です。[24章MySQL コネクタ](#) を参照してください。
- MySQL Connector/NET は、開発者に MySQL 上で安全性の高い高性能データ接続性を要する NET アプリケーションの作成を容易に行えるようにします。要求される ADO.NET インターフェースを実行し、ADO.NET アウェア ツールに統合します。開発者は .NET 言語を選択してアプリケーションを作成できます。MySQL Connector/NET は 100% C# で記述され、完全にマネージされた ADO.NET ドライバです。[24章MySQL コネクタ](#) を参照してください。

ローカライズ：

- サーバからクライアントへ多数の言語でエラー メッセージを送信することができます。[英語以外のエラーメッセージ](#) を参照してください。
- [latin1](#) (cp1252)、[german](#)、[big5](#)、[ujis](#)、などのさまざまなキャラクタセットの完全なサポート。例えば、スカンジナビア語の文字 'ä'、'å' および 'ö' をテーブル名やカラム名で使用することができます。このオプションは MySQL MySQL 4.1.1以降で利用できます。
- すべてのデータが、選択したキャラクタ セットで保存されます。
- ソートは、選択したキャラクタ セットに基づいて行われます (デフォルトは [latin1](#) とスウェーデン語によるソート)。これは、MySQL サーバの起動時に変更することができます。非常に高度なソートの例については、チェコ語のソートコードを参照してください。MySQL サーバではさまざまなキャラクタ セットがサポートされており、コンパイル時および実行時に指定することができます。
- MySQL 4.1.1以降、サーバのタイム ゾーンは大幅に変更可能で、各クライアントは自身のタイム ゾーンを指定できます。[MySQL サーバのタイム ゾーン サポート](#)。

クライアントとツール：

- MySQL AB は複数のクライアント/ユーティリティ プログラムを提供しています。これらの中には [mysqldump](#) や [mysqladmin](#) といったコマンドライン プログラム、そして MySQL Administrator や MySQL Query Browser などのグラフィック プログラムも含まれます。
- MySQL サーバには、テーブルのチェック、最適化、および修復を行う SQL ステートメントのサポートが組み込まれています。これらのステートメントは、[mysqlcheck](#) クライアントを介してコマンドラインから使用可能です。また、MySQL には、[MyISAM](#) テーブルでこれらの操作を実行するための [myisamchk](#) という非常に高速なコマンドライン ユーティリティが組み込まれています。[7章クライアントプログラムとユーティリティ プログラム](#) を参照してください。
- [--help](#) または [-?](#) オプションを指定して呼び出すと、すべての MySQL プログラムでオンライン ヘルプを参照することができます。

MySQL の開発ロードマップ

このセクションでは、開発ロードマップの概略、現在のリリース シリーズ(5.1)にみられる新機能の概略、そして次期リリース シリーズ(5.2)にみられる追加事項や変更事項についての概略を述べます。

本マニュアル(5.1)でカバーされているリリース シリーズのグレードはbetaです。グレードについての詳しい情報は、[インストールする MySQL のバージョンの選択](#) を参照してください。

次期リリース シリーズにアップグレードする前に、[MySQL のアップグレード](#) にある注意事項をチェックしてください。

次の表は、最も要望があった機能の一部についての計画をまとめたものです。

機能	MySQL シリーズ

Unions	4.0
R-trees	4.1 (MyISAM ストレージ エンジン用)
サブクエリ	4.1
カーソル	5.0
ストアド プロシージャ	5.0
トリガ	5.0および5.1
ビュー	5.0
XA トランザクション	5.0
イベント スケジューラ	5.1
パーティション	5.1
プラグイン API	5.1
行ベースのレプリケーション	5.1
サーバ ログ テーブル	5.1
外部キー	5.2 (InnoDB については3.23で実装済み)

MySQL 5.1 での新機能

以下の機能が MySQL 5.1 に追加されました。

- ・パーティション:この機能では、テーブル作成時に設定されたルールに従い、ファイル システムに渡って各テーブルのポーシオンを分散することが可能になります。結果として、同一テーブルの異なるポーシオンが異なる場所に独立テーブルとして保存されますが、ユーザの観点からは、分割されたテーブルは依然として1つのテーブルとなります。構文上、このことは新しい拡張数値を [CREATE TABLE](#)、[ALTER TABLE](#) そして [EXPLAIN ... SELECT](#) ステートメントに対して実行します。MySQL 5.1.6 以降は、テーブルの分割クエリとして パーティション プルーニングが利用できます。この結果、テーブルを分割させない場合の同じクエリと比較して、はるかに高速でクエリの実行が可能となるケースもあります。この機能に関してさらに詳しい情報を知りたい場合は、[15章パーティショニング](#) を参照してください。(執筆: Mikael Ronström)
- ・行ベースのレプリケーション:MySQL におけるレプリケーション機能は本来、マスタからスレーブまでSQLステートメントの伝播に基づいて行われていました。これは、ステートメントベースのレプリケーションと呼ばれています。MySQL 5.1.5 以降、レプリケーションのためにもう1つのベースが利用できるようになりました。これは、行ベースのレプリケーションと呼ばれています。スレーブに SQL ステートメントを送る代わりに、マスタは各テーブルの行がどのように影響を受けるかを指定するバイナリ ログへのイベントを書き込みます。MySQL 5.1.8 以降では、3つ目のオプションが利用できます。組み合わせです。これは、デフォルトではステートメントベースのレプリケーションが利用され、ある特定の場合にのみ行ベースのレプリケーションに切り替わるものです。詳しくは [レプリケーション フォーマット](#) を参照してください。(執筆: Lars Thalmann、Guilhem Bichot、Mats Kindahl)
- ・プラグインAPI:MySQL 5.1 では、サーバの再起動をしなくても、ランタイムに様々なコンポーネントのロード/アンロードを可能にするとてもフレキシブルなプラグイン API も追加でサポートしています。実行が終了していても、プラグイン フルテキスト パーサ は実行内容の最初のステップとなります。この結果、ユーザは、PDF ファイルやその他の文書フォーマットのような任意データにおいてフルテキスト検索機能を可能にすることで、インデックス テキストの入力フィルタを実行できるようになります。プレ パーサ フル テキスト プラグインは、テキストの実効パーシングおよび抽出を行い、MySQL のビルトイン フル テキスト検索にそれを渡します。詳しくは [The MySQL Plugin Interface](#) を参照してください。(執筆: Sergey Vojtovich)
- ・イベント スケジューラ:MySQL イベントは、スケジュールに従って実行するタスクです。1つのイベント作成時は、1つかそれ以上の正規インタバルで実行される、同じく1つかそれ以上の SQL ステートメントを含む名前つきデータベース オブジェクトを作成していることになります。このオブジェクトは、指定されたデータや時間で開始および終了するものです。概念として、このことはUnixの [crontab](#) (「コロンジョブ」としても知られる) や、Windows のタスク スケジューラの考え方に似ています。詳しくは [19章Event Scheduler](#) を参照してください。(執筆: Andrey Hristov)
- ・サーバ ログ テーブル:MySQL 5.1 以前では、サーバはログ ファイルに一般クエリ ログや低速クエリ ログ エントリを書き込みます。MySQL 5.1 以降では、サーバのこういったログ書き込み機能はより柔軟性のあるものとなります。ログのエントリはログ ファイル (上記のように) あるいは [general_log](#) および [mysql](#) データベースの

[slow_log](#) に書き込まれます。ロギングが有効になると、ディスティネーションのいずれかまたは両方が選択されます。[--log-output](#) オプションはディスティネーションあるいはログ出力のディスティネーションを管理します。詳しくは [一般クエリとスロー クエリのログ出力先の選択](#) を参照してください。(執筆: Petr Chardin)

- インスタンス マネージャー(IM) に新しい機能が追加されました。[SHOW instance_name LOG FILES](#) はすべてのログ ファイルをリストアップし、[SHOW instance_name LOG {ERROR | SLOW | GENERAL} size](#) は指定ログファイルの一部を検索し、[SET instance_name.option_name=option_value](#) はオプションを指定値に設定して構成ファイルに書き込みます。詳しくは [mysqlmanager — MySQL Instance Manager](#) を参照してください。(執筆: Petr Chardin)
 - アップグレード プログラム:mysql_upgrade プログラム(MySQL 5.1.7 以降で利用可能)では、MySQL サーバの現バージョンとの不適合性を調べるためにすべての現存テーブルがチェックされます。また、必要であればそれらの修復も行われます。このプログラムは各MySQLのアップグレードに対して起動されなければなりません。詳しくは [mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照してください。(執筆: Alexey Botchkov, Mikael Widenius)
 - MySQL クラスタ間のレプリケーションがサポートされるようになりました。MySQL クラスタと非クラスタデータベース間のレプリケーションも可能になりました。[MySQL Cluster レプリケーション](#) を参照してください。
 - MySQL クラスタディスクデータ:5.1.6以前のMySQLバージョンでは、[NDBCluster](#)ストレージエンジンは必ずメモリ内に保存されていたが、MySQL 5.1.6からはディスク上のクラスタデータ (ただし、インデックスではない) に保存されることが可能になりました。このことで、MySQL クラスタは以前より少ないハードウェア(RAM)で拡大できるようになりました。[MySQL Cluster ディスク データ ストレージ](#) を参照してください。
- ディスクデータの実行では、大量データ (テラバイト値域) の保存時に、高速ノード再起動のための新「no-steal」修復アルゴリズム も実行されます。
- MySQL クラスタに対するオンライン[ADD INDEX](#)および[DROP INDEX](#):テーブルへのインデックス追加と削除は、MySQL クラスタの前バージョンよりも、[NDB](#)ストレージエンジンを使用したほうが遥かに高速で処理されます。これは、[NDB](#)がテーブルを再作成する必要がないかわりにこれらのスキーマを現存テーブルに直接変換できるため、実現可能となっています。
 - MySQL クラスタ上での改良バックアップ実行:クラスタバックアップ中に単一データノードに生じる障害が、MySQL クラスタの前バージョンのように、アバートするために完全バックアップを起こすことはありません。
 - テーブルスペースのバックアップ:mysql_dump機能では、テーブルスペースのダンプオプションがサポートされています。[-Y](#)または[--all-tablespaces](#)を使用して、この機能を有効にしてください。
 - [INFORMATION_SCHEMA](#)に対する改良:MySQL 5.1では、メタデータ データベースでさらに詳しい情報が提供されています。そのデータベース上の新しいテーブルには、[FILES](#)、[EVENTS](#)、[PARTITIONS](#)、[PROCESSLIST](#)、[ENGINES](#)そして[PLUGINS](#)が含まれています。
 - XML 機能:[ExtractValue\(\)](#)は、作成されたXPath表現に適合するXMLフラグメントの内容を返します。[UpdateXML\(\)](#)は、XMLフラグメントから選択された要素を、ユーザが第二XMLフラグメント (これもユーザが与えたもの) を与えたXPath表現に置換します。そして、修正XMLを返します。詳しくは [XML 関数](#) を参照してください。(執筆: Alexander Barkov)
 - ロードエミュレータ:mysqlslapプログラムは、MySQLサーバにクライアントのロードをエミュレートし、各ステージのタイミングを報告するよう設定されています。それは、複数のクライアントがサーバにアクセスしているかのように働きます。詳しくは [mysqlslap — クライアント負荷エミュレーション](#) を参照してください。(執筆: Patrick Galbraith, Brian Aker)

MySQL 5.2で計画されている新機能

以下の機能がMySQL 5.2に追加もしくは変更されることが計画されています。このセクションは、MySQL 5.2開発が初期段階にある場合に變更される可能性があります。

次の概念は、MySQL 5.2ではほとんど扱われないか、もしくはなくなる可能性があります。他のオプションがあれば、それを利用するためにアプリケーションをアップデートする必要があります。

- [table_type](#)変数 ([storage_engine](#)の使用)
- [log_bin_trust_routine_creators](#)変数([log_bin_trust_function_creators](#)の使用)
- [TIMESTAMP\(N\)](#):Nのディスプレイ幅を指定できます。([N](#)は使用しない)

- ストレージエンジンを指定するための **TYPE** テーブルオプション (**ENGINE** を使用する)
- **SHOW TABLE TYPES** SQL ステートメント (**SHOW ENGINES** を使用する)
- **SHOW INNODB STATUS** SQL ステートメント (**SHOW ENGINE INNODB STATUS** を使用する)
- **SHOW MUTEX STATUS** SQL ステートメント (**SHOW ENGINE INNODB MUTEX** を使用する)
- **SHOW LOGS** SQL ステートメント
- **LOAD TABLE FROM MASTER** SQL ステートメント
- **RESTORE TABLE** SQL ステートメント
- **BACKUP TABLE** SQL ステートメント

MySQL の情報源

このセクションでは、MySQL メーリング リスト、ユーザ フォーラム、IRC など、役に立つと思われる情報を提供します。

MySQL メーリング リスト

このセクションでは MySQL メーリング リストの概要を説明するとともに、メーリング リストの使用ガイドラインを提供します。メーリング リストを購読すると、メーリング リストに投稿されたすべてのメッセージを電子メールとして受け取ることができます。自分の質問や回答をメーリング リストに送信することもできます。

このセクションに記載されているメーリング リストの購読を開始または購読を中止する場合には、<http://lists.mysql.com/> にアクセスしてください。ほとんどのメーリング リストは、個別のメッセージを得られる通常版、または一日おきにひとつの大きなメッセージを受け取れるダイジェスト版から選択できます。

購読または購読中止に関するメッセージはいずれのメーリング リストにも送信しないでください。送信した場合、そのメッセージを購読する多くのユーザへ自動的に配信されてしまいます。

あなたのローカル サイトには MySQL メーリング リストに登録した多くの購読者がいるかもしれません。その場合、lists.mysql.com から送信されたメッセージは、あなたのローカルなメーリング リストから受信できるように用意されている場合があります。その際は、ローカルの MySQL リストへの追加または削除については、あなたのシステム管理者にお問い合わせください。

メール プログラム内の個別のメール ボックスにメーリング リストが送信されるようにするには、フィルタをメッセージヘッダから判別するように設定してください。**List-ID:** または **Delivered-To:** ヘッダを使用して、リストのメッセージを識別することができます。

MySQL メーリング リストは以下のとおりです。

- **announce**

このリストは、新しいバージョンの MySQL および関連するプログラムの通知に使用されます。これは、サイズの小さいリストであり、すべての MySQL ユーザは購読するべきです。

- **mysql**

これは MySQL の一般的な議論に使用される主要なリストです。議題によっては、より細分化されたメーリング リストで議論を行ったほうがよい場合があります。適切でないリストに投稿すると、回答が得られないことがあります。

- **bug**

これは、MySQL の最新リリース以降に報告された問題を常に把握しておきたいユーザや、バグの検出と修正の過程に積極的に参加したいユーザのためのリストです。詳細は [質問またはバグの報告](#) を参照してください。

- **internals**

このリストは MySQL のプログラミングに携わる人を対象とします。MySQL の開発およびパッチ投稿に関する議論を行うフォーラムでもあります。

- [mysql-doc](#)

このリストは MySQL の文書に携わる人、すなわち MySQL AB の社員、翻訳者、その他のコミュニティ会員を対象とします。

- [ベンチマーク](#)

このリストは、パフォーマンスに関する問題に関心がある人を対象としています。データベースのパフォーマンス (MySQL に限らない) に関する議論のみでなく、カーネル、ファイルシステム、ディスクシステムのパフォーマンスなど、より広範なカテゴリも含まれます。

- [packagers](#)

このリストは、MySQL のパッケージと配布に関する議論に使用されます。これは、MySQL のパッケージに関するアイデアを交換し、サポートするすべてのプラットフォームとオペレーティングシステム上でできる限り同じような MySQL の外観と操作性を確保する目的のため、配布管理者によって使用されるフォーラムです。

- [java](#)

このリストは、MySQL サーバと Java に関する議論に使用されます。ほとんどが、MySQL Connector/J などの JDBC ドライバに関する議論に使用されています。

- [win32](#)

このメーリングリストは、Microsoft のオペレーティングシステム (Windows 9x/Me/NT/2000/XP/Server 2003 など) で実行される MySQL ソフトウェアに関するすべてのトピックに使用されます。

- [myodbc](#)

このメーリングリストは、ODBC を使用した MySQL サーバへの接続に関するすべてのトピックに使用されます。

- [gui-tools](#)

このメーリングリストは MySQL のグラフィカル ユーザ インタフェース、たとえば [MySQL Administrator](#) や [MySQL Query Browser](#) などについて使用されます。

- [cluster](#)

このメーリングリストは MySQL クラスターの議論に使用されます。

- [dotnet](#)

このメーリングリストは MySQL サーバおよび .NET プラットフォームに関する議論に使用されます。ほとんどが MySQL Connector/Net に関連する議論です。

- [plusplus](#)

このリストは、MySQL への C++ API を使用したプログラミングに関するすべてのトピックに使用されます。

- [perl](#)

このリストは、MySQL の `DBD::mysql` を使用した Perl サポートに関するすべてのトピックに使用されます。

質問に対する回答が MySQL メーリングリストから得られなかった場合、MySQL AB から有料サポートを受ける方法があります。この方法は、MySQL の開発者と連絡を直接取ることができます。

以下は、英語以外の MySQL メーリング リストです。これらのリストは MySQL AB による運営ではありません。

- <mysql-france-subscribe@yahoogroups.com>

フランス語のメーリングリスト。

- <list@tinc.net>

韓国語のメーリングリスト。購読するには、このメーリングリストに [subscribe mysql your@email.address](#) と書いた電子メールを送信してください。

- [<mysql-de-request@lists.4t2.com>](mailto:mysql-de-request@lists.4t2.com)

ドイツ語のメーリングリスト。購読するには、このメーリングリストに subscribe:mysql-de:your@email.address と書いた電子メールを送信してください。このメーリングリストに関する詳細な情報は、<http://www.4t2.com/mysql/> を参照してください。

- [<mysql-br-request@listas.linkway.com.br>](mailto:mysql-br-request@listas.linkway.com.br)

ポルトガル語のメーリングリスト。購読するには、このメーリングリストに subscribe:mysql-br:your@email.address と書いた電子メールを送信してください。

- [<mysql-alta@elistas.net>](mailto:mysql-alta@elistas.net)

スペイン語のメーリングリスト。購読するには、このメーリングリストに subscribe:mysql:your@email.address と書いた電子メールを送信してください。

メーリングリストを使用する際のガイドライン

HTML モードがオンになっているブラウザからメールメッセージを投稿しないでください。多くのユーザはブラウザでメールを読みません。

メーリングリストの質問に回答するとき、多数のユーザにとっても興味深いと思われる回答は、質問した個人に直接返信する代わりにメーリングリストに投稿しても構いません。その場合、元の投稿者以外のユーザにも役立つように、包括的な回答をするようにしてください。メーリングリストに投稿する際には、回答が前の回答と重複していないことを確認してください。

質問の重要な部分を回答へ要約するように努めてください。元のメッセージ全体を引用する必要はありません。

もし回答があなた個人に送られた場合、そしてメーリングリストではない場合、回答を要約してメーリングリストに載せることは、よいエチケットです。そうすれば他の人も、あなたが得たトラブル解決方法を共有できます。

MySQL フォーラムにおける MySQL コミュニティサポート

フォーラムは重要なコミュニケーションの場です。場所は<http://forums.mysql.com>です。多くのフォーラムに参加可能です。以下のカテゴリがあります。

- Migration (移入)
- MySQL Usage (MySQL の使い方)
- MySQL Connectors (MySQL コネクタ)
- Programming Languages (プログラム言語)
- Tools (ツール)
- 3rd-Party Applications (サード パーティーによるアプリケーション)
- Storage Engines (ストレージ エンジン)
- MySQL Technology (MySQL テクノロジー)
- SQL Standards (SQLの標準化)
- Business (ビジネス)

IRC (インターネット リレー チャット) の MySQL コミュニティ サポート

さまざまな MySQL メーリング リストやフォーラムに加え、IRC (インターネット リレー チャット) にも経験豊富なコミュニティがあります。以下は、当社が現在把握している最もすぐれたネットワーク/チャネルです。

freenode (サーバは <http://www.freenode.net/> を参照してください。)

- [#mysql](#) MySQL に関する質問が中心ですが、他のデータベースや SQL に関する質問も受け付けています。MySQL と PHP、Perl、C 言語の組み合わせに関する質問も行われています。

IRC ネットワークに接続するための IRC クライアントソフトウェアを探している場合には、[xChat \(http://www.xchat.org/\)](http://www.xchat.org/)を参照してください。X-Chat (GPL ライセンス) は、Unix および Windows プラットフォームで使うことができます。(無料Windows版 X-Chat は次の場所から入手できます。<http://www.silverex.org/download/>)

質問またはバグの報告

問題に対する解決策が新しいバージョンにすでに組み込まれている可能性がありますので、バグがすでに報告/解決されているかどうかを確認してください。

- このサイト<http://dev.mysql.com/doc/>でマニュアルを検索することができます。マニュアルは、常に最新の状態にしておくために、新しく見つかった問題に対する解決策によって頻繁に更新されています。問題に対する解決策が新しいバージョンにすでに組み込まれている可能性が高いので、変更履歴に関する付録 (<http://dev.mysql.com/doc/mysql/en/news.html>) は特に便利です。
- SQL ステートメントに対するパースエラーが生じた場合は、構文を念入りにチェックしてください。その構文に問題が見当たらなければ、MySQLの最新バージョンサーバがその構文をサポートしていない可能性が高くなります。最新のバージョンでマニュアルが使用された構文をカバーしていない場合、MySQL サーバはそのステートメントをサポートしません。この場合、ご自身でその構文を実行されるか、もしくは[<licensing@mysql.com>](mailto:licensing@mysql.com)宛てに電子メールを送り、サポートについてお問い合わせください。

マニュアルがその構文をカバーしているにも関わらず、MySQLが旧バージョンサーバの場合、MySQLの変更履歴を調べいつその構文が実行されたのかを確認してください。この場合、MySQLサーバへ新しくアップグレードするという選択肢もあります。

- 一般的な問題の解決法については以下を参照してください。[Problems and Common Errors](#)。
- バグデータベース<http://bugs.mysql.com/>を検索して、バグがすでに報告/解決されているかどうかを確認します。
- 次の場所でMySQL メーリングリストのアーカイブを検索します。<http://lists.mysql.com/>。詳細は [MySQL メーリングリスト](#) を参照してください。
- また、<http://www.mysql.com/search/>を使用して、MySQL AB Web サイトにある Web ページすべて (マニュアルを含む) を検索することもできます。

マニュアルやアーカイブで回答を見つけることができなかった場合、ローカルの MySQL の専門家とともに調べてください。それでも質問に対する回答を見つけることができなかった場合は、次のセクションに記載されているガイドラインに従ってMySQL メーリングリストにメールをお送りください。

通常バグを報告する場合、バグデータベースのアドレス<http://bugs.mysql.com/>を参照してください。当社のバグデータベースは公開されているので、すべてのユーザが参照および検索することができます。システムにログインすると、新しいレポートを入力することもできます。Webがアクセスできる環境が整っていない場合、この章の最後に記載のmysqlbugスクリプトを使用してバグレポートを作成することができます。

<http://bugs.mysql.com/>にあるバグデータベースにバグが報告されると、変更履歴にてどのバージョンで修正されたか確認できます。

MySQL で重大なセキュリティ バグを見つけた場合は、[<security@mysql.com>](mailto:security@mysql.com) に電子メールをお送りください。

他のユーザと問題について話し合う場合、MySQL メーリング リストの一部を利用することもできます。 [MySQL メーリング リスト](#)。

正確なバグ レポートを書くのは時間がかかるものですが、レポートを一度で済ませることで、報告者と弊社、双方の時間を節約することができます。そのバグの完全なテスト ケースを含むバグ レポートを提供していただければ、次のリリースではその問題を修正できる可能性が高くなります。このセクションでは、ユーザの時間短縮と効果的な情報提供のため、レポートの正しい書き方を紹介します。このセクションを注意深く読み、ここに記載されている全ての情報がユーザレポートに含まれているか、確認してください。

できれば、MySQL サーバの最新の製品版または開発版を使用して問題をテストしてから投稿してください。だれもが記載されているテストケースで `mysql test <script file>` を使用するだけでバグを再現したり、バグレポートに含まれているシェルまたは Perl スクリプトを実行したりすることができるようにする必要があります。弊社で再現が可能なバグであれば、次の MySQL リリースで修正される可能性が高くなります。

問題に関する適切な説明がバグレポートに記載されていると、最も効果的です。そのため、問題につながったすべての操作の適切な例を挙げ、問題自体を詳細に記述してください。最も効果的なレポートは、バグや問題を再現する方

法を示す詳細な例が記載されたものです。詳細は [Making a Test Case If You Experience Table Corruption](#) を参照してください。

情報が多すぎるメッセージに対応することはできますが、少なすぎるメッセージに対応することはできません。多くの場合、問題の原因がわかっていると思い、細部を重要でないと考えるため、事実を省略してしまいます。記載するかどうかを迷ったときは、記載することをお勧めします。最初に十分な情報を記載していなかったために再度質問して、回答を待たなければならなくなるよりも、レポートに数行を追加する方が、はるかに時間が節約される上に、煩わしくありません。

バグレポートで最もよくある誤りは、(a) 使用している MySQL ディストリビューションのバージョン番号を記載していない、(b) MySQL サーバがインストールされているプラットフォームの説明（プラットフォームの種類およびバージョン番号を含む）が十分でないというものです。これは非常に重要な情報なので、ほとんどの場合、この情報が記載されていないバグレポートは役に立ちません。「なぜうまく作動しないのか？」という質問が頻繁に寄せられます。その場合、要求した機能がその MySQL バージョンに実装されていなかったり、レポートに記載されているバグが新しい MySQL バージョンですでに修正されていたりすることがあります。エラーがプラットフォーム依存である場合もあります。そのような場合、オペレーティングシステムやプラットフォームのバージョン番号を知らないで問題を修正することはほとんど不可能です。

また、コンパイラが問題に関連している場合は、コンパイラに関する情報も記載してください。ユーザがコンパイラのバグを見つけると、MySQL 関連の問題であると考えることがよくあります。ほとんどのコンパイラは常に開発中なので、バージョンごとに改良されています。題がコンパイラに関連するものであるかどうかを判断するには、使用しているコンパイラを知る必要があります。コンパイルに関するすべての問題はバグと見なし、適宜報告してください。

プログラムでエラーメッセージが生成された場合、そのメッセージをレポートに記載することが非常に重要です。プログラムを使用するアーカイブから情報を検索しようとする場合、報告されたエラーメッセージがプログラムで生成されたものと正確に一致している方が効果的です。（大文字小文字の違いにも注意してください。）エラーメッセージを覚えるのではなく、メッセージ全体をレポートにコピーして貼り付けてください。

MyODBC に関する問題がある場合、MyODBC トレースファイルを生成し、レポートとともに送信してください。 [24 章MySQL コネクタ](#)のMyODBCセクションをご参照ください。

レポートに `mysql` コマンドラインツールが使用され、テストケースから長い出力クエリが含まれる場合、そのようなディスプレイの有効な幅を超える出力については、`--vertical` オプション（または `\G` ステートメント終端記号）を使用する必要があります。このセクションに後述される [EXPLAIN SELECT](#) 例では `\G` の使用方法を詳しく述べます。

レポートには以下の情報を記載してください。

- 使用している MySQL ディストリビューションのバージョン番号（MySQL バージョン 5.0.19 など）。実行しているバージョンを確認するには、`mysqladmin version` を実行します。`mysqladmin` は、MySQL インストールディレクトリ下の `bin` ディレクトリにあります。
- 問題が発生したマシンの製造元とモデル。
- オペレーティングシステムの名前とバージョン。Windows を使用している場合、名前とバージョン番号を取得するには、通常「マイ コンピュータ」アイコンをダブルクリックし、「ヘルプ/バージョン情報」メニューをプルダウンします。ほとんどのオペレーティングシステムでは、この情報を取得するには、Unix コマンド `uname -a` を実行します。
- メモリ（実メモリと仮想メモリ）の量が関連することもあります。関連していると思われる場合は、これらの値を記載します。
- MySQL ソフトウェアのソースディストリビューションを使用している場合、使用しているコンパイラの名前とバージョン番号が必要です。バイナリディストリビューションを使用している場合は、ディストリビューション名が必要です。
- コンパイル時に問題が発生した場合、正確なエラーメッセージ、およびエラーが発生したファイル内の問題のあるコード付近の数行のコンテキストを記載します。
- `mysqld` が停止した場合、`mysqld` のクラッシュの原因となったクエリも報告する必要があります。通常、ログを有効にして `mysqld` を実行すると、`mysqld` がクラッシュした後でこれを検出することができます。詳細は [Using Server Logs to Find Causes of Errors in mysqld](#) を参照してください。

- データベーステーブルが問題に関連している場合、`SHOW CREATE TABLE db_name.tbl_name`からの出力を記載します。これを行うのは非常に簡単ですが、データベース内のテーブルに関する情報を取得する強力な方法です。この情報は、発生した状況と同じ状況を再現する際に役立ちます。
- 問題発生時のSQLモードも有効な情報になります。`sql_mode`システム変数値を報告してください。保存されたプロシージャ、ファンクション、そしてトリガオブジェクトの関連する`sql_mode`値は、そのオブジェクトが作成された際に有効だったものになります。保存されたプロシージャかファンクションに関して、`SHOW CREATE PROCEDURE`または`SHOW CREATE FUNCTION`ステートメントは有効なSQLモードを示しています。また、これに関する情報に対しては`INFORMATION_SCHEMA`クエリを利用することができます。

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.ROUTINES;
```

トリガに対しては以下のステートメントが利用できます。

```
SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.TRIGGERS;
```

- 性能に関連するバグや`SELECT`ステートメントに関する問題については、`EXPLAIN SELECT ...`の出力や、少なくとも`SELECT`ステートメントが生成する行の数も含んでください。関連する各テーブル毎に、`SHOW CREATE TABLE tbl_name`からの出力もまた含んでください。状況について情報を提供できればできるほど、有効な手助けが可能となります。

下記はバグレポートの良い例です。`mysql`コマンドラインツールを使ってステートメントが実行されています。読むのが困難なほど長い出力行に対して、`\G`ステートメント終端記号がどのように使用されているか確認してください。

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ... \G
      <output from SHOW COLUMNS>
mysql> EXPLAIN SELECT ... \G
      <output from EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
      <A short version of the output from SELECT,
        including the time taken to run the query>
mysql> SHOW STATUS;
      <output from SHOW STATUS>
```

- `mysqld`の実行中にバグまたは問題が発生した場合、その異常を再現する入力スクリプトを提供します。このスクリプトには、必要なソースファイルを含める必要があります。スクリプトによって再現される状況が実際に発生した状況に近いほど、効果的です。再現可能なテストケースを作成できる場合は、バグレポートにアタッチメントとしてアップロードしてください。

スクリプトを提供できない場合は、少なくとも`mysqldadmin variables extended-status processlist`からの出力をメールに記載して、システムの動作に関する情報を提供する必要があります。

- 数行ではテストケースを再現できない場合や、テストテーブルが大きすぎてメーリングリストに送信できない場合（10レコードを超えるテーブル）、`mysqldump`を使用してテーブルをダンプし、問題を説明する`README`ファイルを作成する必要があります。`tar`と`gzip`または`zip`を使用してファイルの圧縮アーカイブを作成し、FTPを使用してそのアーカイブを<ftp://ftp.mysql.com/pub/mysql/upload/>に転送します。その後、バグデータベース<http://bugs.mysql.com/>に問題を入力します。
- MySQL サーバでクエリによって正しくない結果が生成されたと思う場合、結果だけでなく、正しいと考える結果と、その考えの根拠を示す説明も記載します。
- 問題のサンプルを提供する際には、新しい名前を使用するよりも、実際の状況で使用している変数名やテーブル名などを使用するのが適切です。問題が、変数やテーブルの名前に関連している可能性があるためです。このようなケースはほとんどないと思われませんが、後悔するよりも安全策をとるべきです。結局、実際の状況に基づいたサンプルを提供する方がユーザにとって簡単であると同時に、当社にとっても効率的です。データを他のユーザに公開したくない場合は、ftp を使用して <ftp://ftp.mysql.com/pub/mysql/upload/> に転送することができます。データが実際に最高機密で、当社にも公開したくない場合は、他の名前を使用したサンプルを提供することもできますが、これは最後の選択肢です。
- 可能な場合、関連するプログラムのすべてのオプションを記載します。たとえば、`mysqld`サーバを開始する際に使用したオプションやMySQL クライアントプログラムの実行に使用したオプションを記載します。`mysqld`や`mysql`の

ようなプログラムのオプション、[configure](#)スクリプトのオプションは多くの場合、回答への手がかりとなり、非常に重要です。これらを記載することは、決して無駄ではありません。Perl や PHP などのモジュールを使用している場合は、それらの言語プロセッサバージョン番号だけでなく、プログラムが使用するモジュールバージョンも同様に記載します。例えば、[DBI](#)や[DBD::mysql](#)を使用するPerl スクリプトがある場合、Perl、[DBI](#)、そして[DBD::mysql](#)バージョン番号を記載します。

- 質問が権限システムに関連する場合、[mysqlaccess](#)の出力、[mysqladmin reload](#)の出力、および接続しようとした際に表示されたすべてのエラーメッセージを記載します。権限をテストする際には、まず[mysqlaccess](#)を実行します。その後、[mysqladmin reload version](#)を実行し、問題が発生したプログラムに接続します。[mysqlaccess](#)は、MySQL インストールディレクトリ下の `bin`ディレクトリにあります。
 - バグのパッチを作成した場合、それを含めます。ただし、当社はパッチだけを必要としているわけではありません。パッチによって修正されるバグを示すテストケースなどの必要な情報が記載されていない場合、当社はそのパッチを使用しません。当社がパッチに関連する問題を見つける場合もあれば、パッチをまったく理解できない場合もあります。そのような場合は、パッチを使用することはできません。
- パッチの目的を正確に確認することができない場合、当社はそのパッチを使用しません。この場合、テストケースが役立つことになります。発生する可能性があるすべての状況がパッチによって処理されることを示す必要があります。パッチが機能しないボーダーラインケースが見つかった場合（それがまれなケースでも）、そのパッチは役に立たない可能性があります。
- 何がバグであるか、そのバグがなぜ発生するのか、そのバグが何に関連するのかを推測することは、通常、間違いです。MySQL チームでさえも、最初にデバッグを使用せずにそのようなことを推測して、バグの実際の原因を特定することはできません。
 - ユーザ自身で問題を解決しようとしたことを他のユーザがわかるように、リファレンスマニュアルやメールアーカイブを確認したことをバグレポートに記載します。
 - データが壊れている点が問題である場合や、特定のテーブルにアクセスした際にエラーが発生した場合、[myisamchk](#)または [CHECK TABLE](#)と[REPAIR TABLE](#)を使用して、まずテーブルをチェックしてから、修復を試みる必要があります。詳細は [4章データベース管理](#) を参照してください。

Windows起動時、[SHOW VARIABLES LIKE 'lower_case_table_names'](#)コマンドを使用して[lower_case_table_names](#)値をバリエーションしてください。変数は、データベースおよびテーブル名の大文字/小文字をサーバがどのように扱うかに対して影響を与えます。ある値に対しての効果は[識別子の大文字/小文字区別](#)で説明されている通りです。

- テーブルが頻繁に壊れる場合、その問題が発生する状況と理由を特定する必要があります。この場合、`.err`ファイルに、発生した問題に関する情報が格納されていることがあります。詳しくは[エラー ログ](#)を参照してください。このファイル内の関連する情報をバグレポートに記載します。通常、更新の途中で [mysqld](#)が停止されない限り、[mysqld](#)によってテーブルが破壊されることはありません。[mysqld](#)が停止した原因が特定されれば、当社はその問題の解決策をはるかに簡単に提供することができます。詳細は [How to Determine What Is Causing a Problem](#) を参照してください。
- 可能な場合、最新バージョンの MySQL サーバをダウンロードしてインストールし、これによって問題が解決されるかどうかを確認します。MySQL ソフトウェアのすべてのバージョンが詳細にテストされているため、問題なく動作するはずです。すべてにおいて最大限の下位互換性が確保されているため、MySQL のバージョンを問題なく切り替えることができると当社は確信しています。詳細は [インストールする MySQL 配布版の選択](#) を参照してください。

Webがアクセスできる環境が整っておらず、<http://bugs.mysql.com/>を参照してもバグレポートができない場合、この章の最後に記載の[mysqlbug](#)スクリプトを使用してバグレポート（もしくは他の問題に対するレポート）を作成することができます。[mysqlbug](#)は自動的に以下の情報を確認することでユーザのレポート作成を助けますが、大事な情報が欠如している場合は、メッセージに記載してください。[mysqlbug](#)はMySQLインストールディレクトリ(バイナリディストリビューション)内の[scripts](#)ディレクトリ(ソースディストリビューション)と[bin](#)ディレクトリで見つけることができます。

MySQLの標準への準拠

このセクションでは、MySQL と ANSI/ISO SQL 標準との関連を説明します。MySQL サーバには SQL 標準に対する多数の拡張機能があります。ここでは、それらの拡張機能と使用方法を説明します。また、MySQL サーバに不足している機能と、差異を解決する方法に関する情報も提供します。

SQL標準は1986年から発展し続けており、現在いくつかのバージョンがあります。本マニュアルでは、「SQL-92」は1992年にリリースされた標準に、「SQL:1999」は1999年にリリースされた標準に、そして「SQL:2003」は標準の現バージョンに対応しています。「SQL標準」や「標準SQL」といった言い回しは、常にSQL標準の現バージョンを意味します。

製品に関する当社の主な目標の1つは、速度や信頼性を犠牲にすることなく、SQL-99標準への準拠に向けて開発を続けることです。大部分のユーザにとってMySQLサーバが大幅に使いやすくなるのであれば、当社はSQLに対する拡張機能やSQL以外の機能のサポートを積極的に追加します。[HANDLER](#)インターフェースはこの方針の一例です。[HANDLER 構文](#)を参照してください。

当社は、負荷が高いWeb/ログの使用とミッションクリティカルな年中無休の使用の両方に対応するために、トランザクションデータベースと非トランザクションデータベースのサポートを継続します。

MySQLサーバは、小規模なコンピュータシステムで中規模のデータベース（1,000万～1億行、または1テーブルあたり約100MB）を使用することを目的として最初から設計されました。テラバイト規模のデータベースでも適切に動作するとともに、ハンドヘルドデバイスや組み込み用途により適した小規模なMySQLをコンパイルできるように、MySQLサーバの拡張を続けます。MySQLサーバのコンパクトな設計によって、ソースツリーで競合することなく、これらのどちらの方向も実現することができます。

現時点では、リアルタイムのサポートは考えていません（ただし、レプリケーションサービスを使用して、すでに多くのことを実行することができます）。

MySQLでは、[NDBCluster](#)ストレージエンジンの実装によって、有効性の高いデータベースクラスタがサポートされています。[14章MySQL Cluster](#)を参照してください。

MySQL 5.1でW3C XPath標準のほとんどがサポートされることで、XML機能が提供されています。今後のMySQL開発の一部として、XMLへのサポートの充実を計画しています。[XML 関数](#)を参照してください。

MySQLが準拠する標準

コードの速度と品質を犠牲にすることなく、ANSI/ISO SQL標準を完全にサポートすることを目指しています。

ODBC レベルは0～3.51です。

SQLモードの選択

MySQLサーバは異なるSQLモードで機能し、これらのモードを各クライアントにそれぞれ適合させることができます。この機能では、各アプリケーションがサーバのオペレーティングモードをその要求に合わせることができます。

SQLモードでは、MySQLがどのSQL構文をサポートすべきかまたはどの種類のデータバリデーションを実行すべきか、といったサーバオペレーション分野が管理されています。つまり、モードを定義することにより、異なる環境でMySQLを使用でき、MySQLを別のデータベースサーバと併用できるようになります。

`--sql-mode="mode_value"`オプションで`mysqld`を起動させると、デフォルトのSQLモードが設定できます。MySQL 4.1からは、`SET [SESSION|GLOBAL] sql_mode='mode_value'`ステートメントを使用して`sql_mode`システム変数を設定することで、ランタイムのモード変更もできます。

SQLモードの設定に関するさらに詳しい情報は、[SQL モード](#)を参照してください。

ANSIモードでのMySQLの実行

`mysqld`を使用して、`--ansi`スタートアップオプションでANSIモードを実行させることができます。ANSIモードでサーバを実行するのは、以下のオプションを指定してサーバを起動するのと同じことです

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

MySQL 4.1以降では、次の2つのステートメントで同じ動作を実現することができます。

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'ANSI';
```


`sql_mode`システム変数値は'ANSI'に設定されることで、ANSIモードに関連するすべてのSQLモードオプションに設定されます。結果を確認するには、以下を実行します。

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
-> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

--ansiを指定してANSIモードでサーバを実行することと、SQLモードを'ANSI'に設定することは、同じではありません。--ansiオプションはSQLモードに影響を与え、トランザクション分離レベルも設定します。SQLモードを'ANSI'に設定しても、分離レベルに影響はありません。

コマンド オプションおよびSQLモードの選択を参照してください。

SQL標準に対するMySQL拡張機能

MySQLサーバには、他のSQLデータベースにはない拡張機能があります。それらの拡張機能を使用した場合、他のSQLサーバにコードを移植できなくなるので注意してください。場合によっては、MySQL拡張機能を含むコードを記述しても、次の形式のコメントを使用することで移植することができます。

```
/*! MySQL-specific code */
```

その場合、MySQLサーバでは、他のMySQLステートメントと同様にコメント内のコードが解析および実行されますが、他のSQLサーバでは拡張機能が無視されます。例えば、MySQLサーバは次のステートメント内の**STRAIGHT_JOIN**キーワードを認識しますが、他のサーバでは認識されません。

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

!'の後ろにバージョン番号を追加すると、MySQLバージョンが、使用されているバージョン番号以降の場合にのみ、構文が実行されます。バージョン3.23.02以降を使用している場合、MySQLサーバで次のコメント内の**TEMPORARY**キーワードが使用されます。

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

以下は、MySQL拡張機能の一覧です。

- ディスク上のデータ構成

MySQLサーバでは、各データベースはMySQLデータディレクトリ下のディレクトリに、データベース内のテーブルはデータベースディレクトリ内のファイル名にマップされます。これには、次のような意味があります。

- ほとんどのUnixシステムのようにファイル名で大文字と小文字が区別されるオペレーティングシステム上のMySQLサーバでは、データベース名およびテーブル名は大文字と小文字を区別されます。[識別子の小文字/大文字区別](#)を参照してください。
- **MyISAM**ストレージエンジンを使用して、テーブルのバックアップ、名前の変更、移動、削除、およびコピーを行うことができます。たとえば、**MyISAM**テーブルの名前を変更するには、テーブルに対応する**.MYD**、**.MYI**、および**.frm**ファイルの名前を変更します。(ただし、**RENAME TABLE**や**ALTER TABLE ... RENAME**を使用して、サーバにファイル名を変更させるほうが好ましいでしょう。)

データベースとテーブル名は、パスネーム分離文字(/や\)を含むことはできません。

- 一般言語構文

- デフォルトでは、文字列は"だけでなく、""や""のいずれかでも囲むことができます。(ANSI_QUOTES SQLモードが有効な場合、文字列は"のみでしか囲むことができず、サーバは識別子として""で囲まれた文字列を実行します。)
- \は文字列内のエスケープ文字です。
- SQLステートメントで、**db_name.tbl_name**構文を使用して、さまざまなデータベース内のテーブルにアクセスできます。同様の機能備えたSQLサーバもありますが、これは**User space**と呼ばれます。MySQLサーバでは、以下のステートメントで使われるようなテーブルスペースはサポートされていません。**CREATE TABLE ralph.my_table ... IN my_tablespace**

- SQLステートメント構文

- [ANALYZE TABLE](#)、[CHECK TABLE](#)、[OPTIMIZE TABLE](#)そして[REPAIR TABLE](#)ステートメント
- [CREATE DATABASE](#)、[DROP DATABASE](#)そして[ALTER DATABASE](#) ステートメント。[CREATE DATABASE 構文](#)、[DROP DATABASE 構文](#)、[ALTER DATABASE 構文](#)などを参照してください。
- [DO](#)ステートメント
- クエリオプティマイザによるテーブルの結合方法に関する説明を取得する[EXPLAIN SELECT](#)。
- [FLUSH](#)および[RESET](#)ステートメント
- [SET](#)ステートメント[SET 構文](#)を参照してください。
- [SHOW](#)ステートメント詳しくは [SHOW 構文](#)を参照してください。MySQL 5.0以降では、MySQL特有の[SHOW](#)ステートメントの多くから得られた情報は、[INFORMATION_SCHEMA](#)クエリに対して[SELECT](#)を使用することで、より標準化されます。[21章INFORMATION_SCHEMA データベース](#)を参照してください。
- [LOAD DATA INFILE](#)の使用。多くの場合、この構文はOracleの[LOAD DATA INFILE](#)と互換性があります。[LOAD DATA INFILE 構文](#)を参照してください。
- [RENAME TABLE](#)の使用。[RENAME TABLE 構文](#)を参照してください。
- [DELETE](#) + [INSERT](#)の代わりとしての[REPLACE](#)使用。[REPLACE 構文](#)を参照してください。
- [ALTER TABLE](#)ステートメントにおける[CHANGE col_name](#)、[DROP col_name](#)、または[DROP INDEX](#)、[IGNORE](#)または[RENAME](#)の使用。[ALTER TABLE](#)ステートメントにおける、複数[ADD](#)、[ALTER](#)、[DROP](#)、または[CHANGE](#)節の使用。[ALTER TABLE 構文](#)を参照してください。
- インデックス名、カラムのプリフィックス上のインデックス、および[CREATE TABLE](#)ステートメントでの[INDEX](#)または[KEY](#)の使用。[CREATE TABLE 構文](#)を参照してください。
- [CREATE TABLE](#)を用いた[TEMPORARY](#)または[IF NOT EXISTS](#)の使用。
- [DROP TABLE](#)および[DROP DATABASE](#)を用いた[IF EXISTS](#)の使用。
- 1つの[DROP TABLE](#)ステートメントで複数のテーブルを破棄することができます。
- [UPDATE](#)および[DELETE](#)ステートメントの[ORDER BY](#)および[LIMIT](#)節。
- [INSERT INTO tbl_name SET col_name = ...](#)構文。
- [INSERT](#)および[REPLACE](#)ステートメントの[DELAYED](#)節。
- [INSERT](#)、[REPLACE](#)、[DELETE](#)および[UPDATE](#)ステートメントの[LOW_PRIORITY](#)節。
- [SELECT](#)ステートメントにおける[INTO OUTFILE](#)または[INTO DUMPFILE](#)の使用。[SELECT 構文](#)を参照してください。
- [SELECT](#)ステートメントにおける、[STRAIGHT_JOIN](#)もしくは[SQL_SMALL_RESULT](#)のようなオプション。
- [GROUP BY](#)節で、選択したすべてのカラムの名前を列挙する必要はありません。これにより、ごく一部ではありますが、きわめて一般的なクエリのパフォーマンスが向上します。[GROUP BY 句との関数および修飾子の使用](#)を参照してください。
- [ORDER BY](#)を用いるだけでなく[GROUP BY](#)を用いても、[ASC](#)および[DESC](#)を指定できます。
- [:=](#)代入演算子を使用して、ステートメント内で変数を設定できます。

```
mysql> SELECT @a:=SUM(total),@b:=COUNT(*),@a/@b AS avg
-> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

- データ型

- **MEDIUMINT**、**SET**および**ENUM**データ型、そして様々な**BLOB**および**TEXT**データ型。
- **AUTO_INCREMENT**、**BINARY**、**NULL**、**UNSIGNED**そして**ZEROFILL**データ型の属性。
- 関数と演算子
 - 他の SQL 環境を使用していたユーザにわかりやすいように、MySQL サーバでは多数の関数のエイリアスがサポートされています。たとえば、すべての文字列関数で、標準の SQL 構文と ODBC 構文の両方がサポートされています。
 - MySQL サーバでは、C プログラミング言語のように、**||** および **&&** 演算子が論理 OR および AND を意味すると解釈されます。MySQL サーバでは、**||**と **OR**、および**&&**と **AND** はシノニムです。このすぐれた構文のために、MySQL サーバでは、文字列の連結に標準 SQL の **||** 演算子を使用することができません。その代わりに、**CONCAT()** を使用します。**CONCAT()**には引数をいくつでも使用できるので、**||** 演算子の使用を MySQL サーバに変換するのは簡単です。
 - **value_list**に 1 つ以上の要素がある場合の、**COUNT(DISTINCT value_list)**の使用。
 - すべての文字列比較は、デフォルトでは大文字と小文字を区別せず、現在のキャラクタセット照合順序 で決められたソート順で行われます。このキャラクタセット照合順序は、デフォルトでは**latin1**(cp1252 West European)です。これを変更するには、**BINARY**属性を指定してカラムを宣言するか、**BINARY** キャストを使用して、字句順序よりもキャラクタコード値を使用して比較が行われるようにする必要があります。
 - **%**演算子は**MOD()**のシノニムです。したがって、**N % M**は**MOD(N,M)**と同じです。**%**は、C プログラマを対象として、また PostgreSQL との互換性を確保するためにサポートされています。
 - **=**、**<>**、**<=**、**<**、**>=**、**>**、**<<**、**>>**、**<=>**、**AND**、**OR**、または**LIKE**演算子を、**SELECT**ステートメントの**FROM**の左側のカラム比較で 사용할 ことができます。例：


```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```
 - **LAST_INSERT_ID()**関数は、最新の**AUTO_INCREMENT**値を返します。**情報関数** を参照してください。
 - **LIKE**は数値上で許可されます。
 - **REGEXP**および**NOT REGEXP** 拡張正規表現演算子。
 - 1 つまたは複数の引数を使用する**CONCAT()**または**CHAR()**。(MySQL サーバでは、これらの関数は引数をいくつでも使用することができます。)
 - **BIT_COUNT()**、**CASE**、**ELT()**、**FROM_DAYS()**、**FORMAT()**、**IF()**、**PASSWORD()**、**ENCRYPT()**、**MD5()**、**ENCODE()**、**DECODE()**、**PERIOD_ADD()**、**PERIOD_DIFF()**、または**WEEKDAY()**関数。
 - 部分文字列を削除する**TRIM()**の使用。標準SQLでは、1つの文字しか削除できない。
 - **GROUP BY**関数**STD()**、**BIT_OR()**、**BIT_AND()**、**BIT_XOR()**、および**GROUP_CONCAT()**。**GROUP BY 句との関数および修飾子の使用** を参照してください。

優先順位に従って並べられた、新しい拡張機能が MySQL サーバに追加される時期を示す一覧については、<http://dev.mysql.com/doc/mysql/en/roadmap.html>にあるオンラインのMySQL開発ロードマップを参照してください。

MySQLと標準SQLとの違い

MySQL サーバを ANSI SQL 標準および ODBC SQL 標準に準拠したものにする 것을目標としていますが、以下のよう に、MySQL サーバが異なる動作を示すことがあります。

- **VARCHAR**型のカラムでは、値が格納される際に後続のスペースが削除されます。(これは、MySQL 5.0.3で修復されています。) **Known Issues in MySQL** を参照してください。
- テーブルを定義した場合やその構成を変更した場合、**CHAR**型のカラムが自動的に **VARCHAR**型のカラムに変更されることがあります。(これは、MySQL 5.0.3で修復されています。)

- MySQLと標準SQLの権限システムには、いくつかの違いがあります。たとえばMySQLでは、テーブルを削除しても、テーブルの権限が自動的に取り消されません。テーブルの権限を取り消すには、明示的に[REVOKE](#)を発行する必要があります。詳細は [REVOKE 構文](#) をご覧ください。
- [CAST\(\)](#)関数は、[REAL](#)または[BIGINT](#)に対するキャストをサポートしていません。[キャスト関数と演算子](#) を参照してください。
- 標準SQLは、[SELECT](#)ステートメント内の[HAVING](#)節が、[GROUP BY](#)節のカラムを参照できるように要求します。このことは、MySQL 5.0.2以前ではできません。

サブクエリのサポート

MySQL 4.1以降では、サブクエリと派生テーブルがサポートされています。「サブクエリ」は、他のステートメント内にネストされた[SELECT](#)ステートメントです。「派生テーブル」(名前のないビュー)は他のステートメントの[FROM](#)節内のサブクエリです。[サブクエリ構文](#) を参照してください。

4.1 より前のバージョンの MySQL については、結合などの方法によって、ほとんどのサブクエリを記述し直すことができます。詳しい方法例は、[MySQL 初期バージョンにおいて、サブクエリの接合としての書き換え](#)を参照してください。

SELECT INTO TABLE

MySQL サーバでは、Sybase SQL 拡張機能[SELECT ... INTO TABLE](#)はまだサポートされていません。代わりに、標準SQL構文[INSERT INTO ... SELECT](#)がサポートされています。これらは、基本的には同じです。詳しくは [INSERT ... SELECT 構文](#) を参照してください。例：

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

また、[SELECT ... INTO OUTFILE](#)または[CREATE TABLE ... SELECT](#)を使用することもできます。

MySQL 5.0以降では、ユーザによって定義された変数で[SELECT ... INTO](#)を使用することができます。同じ構文も、カーソルとローカル変数を用いてストアードルーチン内で使用できます。[SELECT ... INTO ステートメント](#) を参照してください。

トランザクションとアトミックオペレーション

MySQL サーバ (バージョン 3.23-max およびすべてのバージョン 4.0 以降) では、[InnoDB](#)トランザクションストレージエンジンでトランザクションがサポートされています。[InnoDB](#)は完全に [ACID](#)に準拠しています。詳しくは [13章 ストレージエンジンとテーブルタイプ](#) を参照してください。トランザクションエラーの取り扱いにおける、標準SQLと[InnoDB](#)との違いについては[InnoDB エラー処理](#)を参照してください。

MySQL サーバのその他の非トランザクションストレージエンジン ([MyISAM](#) など) は、「アトミックオペレーション」と呼ばれる別のデータ整合性のパラダイムに従います。トランザクションの観点では、[MyISAM](#) テーブルは事実上、常に [AUTOCOMMIT=1](#)モードで動作すると言えます。アトミックオペレーションでは多くの場合、パフォーマンスの高さに値する整合性が確保されます。

両方のパラダイムをサポートする MySQL サーバでは、ユーザはアトミックオペレーションの速度を必要とするか、アプリケーションでトランザクション機能を使用する必要があるかを選択することができます。この選択は、テーブルごとに行うことができます。

ほとんどの場合、トランザクションストレージエンジンと非トランザクションストレージエンジンのどちらを選ぶかの決め手となるのはパフォーマンスです。トランザクションテーブルの場合、はるかに大きなメモリとディスク領域が必要で、CPU のオーバーヘッドも大きくなります。しかし、[InnoDB](#)のようなトランザクションストレージエンジンには特有の機能も多数あります。MySQLサーバのモジュール設計によって、このようなさまざまなストレージエンジンすべてを同時に使用することができるので、さまざまな要件に合わせて、あらゆる条件で最適なパフォーマンスを確保することが可能です。

しかし、MySQL サーバの機能を使用して、非トランザクションの [MyISAM](#) テーブルでも厳密な整合性を確保するにはどのようにすればよいのでしょうか。また、非トランザクションテーブルの機能はトランザクションストレージエンジンにどのように対抗できるのでしょうか。

- ・ トランザクションパラダイムでは、重大な状況で **COMMIT** ではなく **ROLLBACK** の呼び出しに依存するようにアプリケーションが作成されている場合、トランザクションの方が便利です。また、トランザクションでは、完了していない更新や失敗した活動がデータベースにコミットされることはありません。サーバには自動ロールバックを行う機会が与えられ、データベースは保護されます。

非トランザクションテーブルを使用している場合、MySQL サーバでは、ほとんどの場合、更新前に簡単なチェックを組み込んだり、データベースの不整合をチェックして、不整合が発生した場合には自動的に修復または警告する簡単なスクリプトを実行したりすることで、発生する可能性がある問題を解決することができます。MySQL ログを使用したり、別のログを 1 つ追加したりするだけで、通常、データの整合性が失われることなく、完全にテーブルを修復することができます。

- ・ ほとんどの場合、重要なトランザクション更新はアトミックな更新に記述し直すことができます。通常、トランザクションによって解決される整合性の問題はすべて、**LOCK TABLES** またはアトミックな更新を使用して解決することができます。これによって、サーバが自動的に停止するという、トランザクションデータベースシステムと共通する問題が回避されます。
- ・ MySQL サーバを安全に使用するには、トランザクションテーブルを使用するかどうかに関係なく、バックアップを作成し、バイナリログをオンにするだけです。これにより、他のトランザクションデータベースシステムで可能なように、どのような状況からでもリカバリすることができます。使用するデータベースシステムに関係なく、どのような場合でもバックアップを作成することが望ましいのは当然です。

トランザクションパラダイムには長所と短所があります。多数のユーザおよびアプリケーション開発者は、停止が発生する、または停止が必要な問題に関するコード化の容易さに頼っています。しかし、アトミックオペレーションのパラダイムに慣れていなかったり、トランザクションの方が詳しいという場合でも、非トランザクションテーブルの速度が、最も高速で最適に調整されたトランザクションテーブルの速度の 3 倍から 5 倍も速いという長所を考えてみてください。

整合性が最も重要な状況では、MySQL サーバは、非トランザクションテーブルでもトランザクションレベルの信頼性と整合性を提供します。**LOCK TABLES** を使用してテーブルをロックすると、整合性チェックが行われるまで、すべての更新が延期されます。テーブルエンドで同時挿入が可能な **READ LOCAL** ロック（書き込みロックの逆）が設定されている場合、他のクライアントによる読み取りと挿入は引き続き行うことができます。新しく挿入したレコードは、読み取りがロックされているクライアントには、読み取りロックが解除されるまで表示されません。**INSERT DELAYED** を使用すると、ロックが解除されるまで挿入をローカルキューに入れることができ、クライアントは挿入が完了するまで待機する必要はありません。**同時挿入** および **INSERT DELAYED 構文** を参照してください。

ここでいう「アトミック」とは、魔法のようなものではありません。個々の更新の実行中は、他のユーザがそれを妨害できないようにするとともに、自動ロールバック（あまり注意を払わなかった場合に、トランザクションテーブルで行われることがあります）が行われないようにすることができますというだけです。また、MySQL サーバでは、ダーティリードが行われることもありません。

非トランザクションテーブルを使用する際のテクニックは、以下のとおりです。

- ・ **LOCK TABLES** を使用して、通常はトランザクションを必要とするループをコード化することができる。そのため、実行中にレコードを更新するカーソルが不要です。
- ・ **ROLLBACK** を使用しないように、以下の方法を使用することができます。
 1. **LOCK TABLES** を使用して、アクセスするすべてのテーブルをロックします。
 2. 更新する前に条件をテストします。
 3. すべてに問題がなければ、更新します。
 4. **UNLOCK TABLES** を使用して、ロックを解除します。

常にではありませんが、これは通常ロールバックが行われる可能性があるトランザクションを使用するよりも、はるかに速い方法です。ただし、更新の途中でスレッドが停止された場合は、この方法では対応することができません。その場合、すべてのロックが解除されるが、一部の更新が実行されていない可能性があります。

- ・ 関数を使用して、1 回の操作でレコードを更新することもできます。次のようなテクニックを使用すると、非常に効率的なアプリケーションを取得することができます。
 - ・ 現在の値に関連してカラムを変更します。

- 実際に変更されたカラムのみを更新します。

たとえば、顧客情報に対して更新を行う場合、変更された顧客データのみを更新し、変更されたデータ、または変更されたデータに依存するデータが元のレコードと比較して変更されていないことだけをテストします。変更されたデータのテストは、[UPDATE](#)ステートメントで [WHERE](#) 節を使用して行われます。レコードが更新されなかった場合、「Some of the data you have changed has been changed by another user.」というメッセージがクライアントに表示されます。その場合、ウィンドウに元のレコードと新しいレコードを表示して、使用する必要がある顧客レコードのバージョンをユーザが決定できるようにします。

これによって、カラムロックと類似しているが、現在の値に関連する値を使用して、カラムの一部のみが更新される点で、実際はカラムロックよりすぐれた機能が実現します。つまり、一般的な[UPDATE](#)ステートメントは次のようになります。

```
UPDATE tablename SET pay_back=pay_back+125;

UPDATE customer
SET
  customer_date='current_date',
  address='new address',
  phone='new phone',
  money_owed_to_us=money_owed_to_us-125
WHERE
  customer_id=id AND address='old address' AND phone='old phone';
```

これは非常に効率的で、別のクライアントが [pay_back](#) または [money_owed_to_us](#) カラムの値を変更していても動作します。

- 多くの場合、ユーザは、複数のテーブルの一意の識別子を管理するために [LOCK TABLES](#) や [ROLLBACK](#) を使用していました。これは、[AUTO_INCREMENT](#)カラムおよびSQL関数[LAST_INSERT_ID\(\)](#)またはC API関数[mysql_insert_id\(\)](#)を使用することで、ロックやロールバックをせずにはるかに効率的に処理することができます。[情報関数](#) および [mysql_insert_id\(\)](#) を参照してください。

通常、行レベルのロックをコード化することができます。状況によっては実際にこれが必要なので、[InnoDB](#) テーブルでは行レベルのロックがサポートされています。[MyISAM](#)テーブルでは、テーブルでフラグカラムを使用し、以下のようなことを実行することができます。

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID;
```

レコードが見つかり、元のレコードで [row_flag](#) がすでに1でなくなっていた場合、MySQL は、影響を受けた行の数として1を返します。MySQL サーバでは前述のステートメントが次のように変更されると考えることができます。

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
```

ストアドプロシージャとトリガ

ストアドプロシージャと関数は、MySQL 5.0から実装されます。[17章ストアドプロシージャとファンクション](#) を参照してください。

基本的なトリガの機能はMySQL 5.0.2から実装されており、MySQL 5.1に向けて更なる開発が計画されています。[18章トリガ](#) を参照してください。

外部キー

MySQL サーバ 3.23.44 以降では、[CASCADE](#)、[ON DELETE](#)、および [ON UPDATE](#) を含む外部キー制約のチェックが [InnoDB](#)ストレージエンジンでサポートされています。[FOREIGN KEY 制約](#) を参照してください。

[InnoDB](#)以外のストレージエンジンについては、MySQL サーバでは現在、[CREATE TABLE](#) ステートメントで [FOREIGN KEY](#)構文のみが解析されますが、この情報は使用/保存されません。近いうちに、この情報がテーブル仕様ファイルに保存され、[mysqldump](#) および ODBC によって取得できるように、この実装を拡張する予定です。さらにその後は、[MyISAM](#)テーブルについても外部キー制約を実装する予定です。

データベース開発者にとって、外部キーを使用するメリットは以下のとおりです。

- 関係が適切に設計されている場合、外部キー制約によって、プログラマがデータベースで不整合を引き起こすことが少なくなります。

- データベースサーバによって集中的に制約チェックが行われるため、アプリケーション側でのこれらのチェックが不要になります。同様にこのことで、各アプリケーションで制約がすべてチェックされないといった可能性がなくなります。
- 連鎖更新および削除を使用すると、アプリケーションコードを単純化することができます。
- 適切に設計された外部キールールは、テーブル間の関係の記述に役立ちます。

これらのメリットには、必要なチェックを行なうため、データベースサーバの追加オーバーヘッドという犠牲が付随することに留意してください。サーバでの余分なチェックによってパフォーマンスに影響が生じるため、一部のアプリケーションでは、可能であればこれを避けるよう設定されています。(このため、主要な商用アプリケーションの中には、アプリケーションレベルでこの外部キーロジックがコード化されているものもあります。)

MySQLでは、データベース開発者が使用するアプローチを選択できます。外部キーが不要で、かつ参照整合性をチェックすることで生じるオーバーヘッドを避ける必要がなければ、[MyISAM](#)のような他のストレージエンジンを代わりに選択できます。(たとえば、[MyISAM](#)ストレージエンジンでは、[INSERT](#)や[SELECT](#)オペレーションでのみ行なわれるアプリケーションのパフォーマンスが、非常に高速に行なわれるようになります。この場合、テーブルの中間に欠如はなく、挿入は検索と同時に行なわれます。[同時挿入](#) 参照。)

参照整合性チェックを利用しない場合は、次のことに留意してください。

- サーバ側の外部キー関係チェックを使用しない場合、アプリケーション自身で関係についての問題を処理しなければなりません。たとえば、適切な順序でテーブルに行を挿入し、分断段落レコードの作成を避けるように注意してください。複合レコード挿入オペレーションの途中で生じるエラーから回復することも可能です。
- [ON DELETE](#)が、アプリケーションが必要とする参照整合性能力のみの場合、1つのステートメントで多くのテーブルから行を削除するために、複数テーブルの[DELETE](#)ステートメントを使用してMySQL4.0サーバと同様の結果を得ることができます。[DELETE](#) 構文 を参照してください。
- [ON DELETE](#)が実装されていないという問題に対処するには、外部キーがあるテーブルからレコードを削除する際に適切な[DELETE](#)ステートメントをアプリケーションに追加します。実際、この方法は外部キーを使用する場合と同じくらい簡単で、移植性はそれよりもはるかに高くなります。

外部キーを使用するデメリットは以下のとおりです。

- 外部キーサポートは多くの参照性合成問題をアドレス指定しますが、キー関係を設計する上で犯しやすい間違いによって、循環ルール、連鎖削除の不適切な組み合わせなどの深刻な問題が生じることがあります。
- DBAにとって、個々のテーブルのバックアップやリストアが非常に困難になり、場合によっては不可能になるような複雑な関係のトポロジを作成することはめったにありません。(MySQLは、他のテーブルに依存するテーブルをリロードする際、一時的に外部キーチェックを使用できないようにすることで、この問題を軽減します。詳しくは[FOREIGN KEY 制約](#)を参照してください。MySQL 4.1.1以降では、[mysqldump](#)は、リロードの際に自動的にこの能力を利用するダンプファイルを生成します。)

SQLの外部キーは、テーブルを結合させずに参照整合性をチェックおよび実行します。[SELECT](#)ステートメントで複数テーブルからの結果を得たい場合は、以下のようにjoinを行なってください。

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

[JOIN](#) 構文および[Using Foreign Keys](#)を参照してください。

[ON DELETE ...](#)を使用しない[FOREIGN KEY](#)構文は、自動的に[WHERE](#)節を作成するために、ODBCアプリケーションで使用されることが多々あります。

ビュー

(更新可能なビューを含む)ビューは、MySQLサーバ5.0.1から実装されます。[20章ビュー](#)を参照してください。

ビューは、1つのテーブルのように一連の関係(テーブル)にユーザがアクセスできるようにし、ユーザのアクセスをそのみに制限する場合に便利です。また、ビューを使用して、行(特定のテーブルのサブセット)へのアクセスを制限することもできます。MySQLサーバには高度な権限システムがあるので、カラムへのアクセスを制限する場合にはビューを使用する必要はありません。[MySQL アクセス権限システム](#)を参照してください。

当社はビューの実装を設計する際に、リレーショナルデータベースシステムに関する「コードのルール #6」に、SQL の範囲内で可能な限り準拠することを目指しています。「これは、理論的に更新可能なすべてのビューは実際にも更新可能でなければならないというものです。」

コメントの開始記号としての '--'

標準SQLでは、コメントにC構文 `/* this is a comment */` が使用され、MySQLサーバでも同様にこの構文がサポートされています。MySQLでもまた、[コメント構文](#)で記述されているように、MySQL固有のSQLをコメントに組み込ませる構文に対する拡張子がサポートされています。

標準SQLでは、コメント開始シーケンスとして `--'` が使用されます。MySQLサーバでは、コメント開始文字として `#` が使用されます。MySQLサーババージョン 3.23.3 以降でも、`--'` コメントスタイルを使用することができます。ただし、`--'` コメント開始シーケンスは、後にスペース（または、改行などの制御文字）が続かなければなりません。これは、このコメントスタイルによって、`payment` の値を自動的に挿入する次のようなコードを使用した自動生成のSQLクエリで多数の問題が発生していたためです。

```
UPDATE account SET credit=credit-payment
```

`payment` の値が負数（`-1` など）の場合、どうなるかを考えてみてください。

```
UPDATE account SET credit=credit--1
```

SQLでは`credit--1`は有効表現ですが、`--'`はコメント開始の一部として解釈され、表現の一部が破棄されてしまいます。結果として、意図したものとまったく異なる意味を持つステートメントとなってしまいます。

```
UPDATE account SET credit=credit
```

ステートメントでは、値が変更されることは一切ありません。このことは、コメントが`--'`で開始する場合に、重大な影響があることを示しています。

MySQLサーバ3.23.3以降でこの方法の実装を使用する場合、コメント開始シーケンスとして認識されるように、`--'`の後にスペースが続かなければなりません。したがって、`credit--1`は安全に使用できます。

もう1つの安全な機能は、`mysql` コマンドラインクライアントによって `--'` で始まるすべての行が削除されるというものです。

以下の情報は、3.23.3 より前のバージョンのMySQLを実行している場合にのみ関連します。

テキストファイルのSQLスクリプトに`--'`コメントが含まれている場合、スクリプトを実行する前に、`#`文字を使用するためのコメントを以下のように変換する[replace](#)機能を使用する必要があります。

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
| mysql db_name
```

これは通常の方法でスクリプトを実行するよりも安全です。

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

また、スクリプトファイルを「その場で」編集して、`--'`コメントを `#` コメントに変更することもできます。

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

次のコマンドで元に戻してください。

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

[replace](#) — 文字列置き換えユーティリティ を参照してください。

MySQL における制約の処理

MySQL ではトランザクションテーブルとロールバックが有効でない非トランザクションテーブルの両方を使用することができます。このため、MySQL と他のデータベースとでは制約の処理が多少異なります。エラー時にロールバックすることができない非トランザクションテーブルで、多数の行を更新または挿入した場合の処理を実装しなければなりません。

基本的な概念は、MySQLサーバが検出可能なエラーをコンパイル時に生成し、受け取ったエラーから実行時にリカバリするというものですほとんどの場合にこれが実装されていますが、まだすべてについて実装されているわけではありません。

MySQL における基本的なオプションは、途中でステートメントを中止するか、問題からリカバリするためにできる限りのことを行って、処理を続行するかです。デフォルトでは、サーバは後者の方法に従います。たとえば、サーバは無効値を、近い値を持つ有効値に強制変換する可能性があります。

いくつかのSQLモードオプションでは、不良データ値の処理やエラー時にステートメント実行が継続されるか中止されるかといった処理がコントロールできます。これらのオプションを使用して、他のデータベースが不適切な入力を拒絶するのと同じように、MySQLサーバをより従来に近い方法で働くようにコンフィギュレーションを行なえます。SQLモードがサーバ起動時に広域で設定されることで、すべてのクライアントに影響を与えることができます。ランタイムに各クライアントがSQLモードを設定できることで、要求に最も適した起動状態が選択できるようになります。[SQL モード](#) を参照してください。

さまざまな種類の制約に関する、MySQLサーバの処理方法を以下で説明します。

PRIMARY KEYおよびUNIQUEインデックス制約

通常、主キー、ユニークキー、または外部キー違反を引き起こす行のINSERT/UPDATEを行おうとすると、エラーが発生します。[InnoDB](#) のようなトランザクションストレージエンジンを使用している場合、MySQL ではステートメントが自動的にロールバックされます。非トランザクションストレージエンジンを使用している場合、MySQL はエラーが発生した行で停止し、残りの行は未処理のままになります。

このキー違反を無視する場合、MySQLではINSERTやUPDATEに対するIGNOREキーワードが使用できます。この場合、キー違反は無視され、引き続き次の行が処理されます。[INSERT 構文](#) および [UPDATE 構文](#) を参照してください。

[mysql_info\(\)](#) C API関数で、実際に挿入/更新される行数についての情報が取得できます。MySQL 4.1以降では、[SHOW WARNINGS](#)ステートメントも利用可能です。[mysql_info\(\)](#) および [SHOW WARNINGS 構文](#) を参照してください。

現時点では、外部キーがサポートされているのは[InnoDB](#)テーブルのみです。詳しくは [FOREIGN KEY 制約](#) を参照してください。[MyISAM](#)テーブルでの外部キーサポートは、MySQL 5.2で実装される予定です。[MySQL の開発ロードマップ](#) を参照してください。

無効値の制約

MySQLでは、デフォルトで無効や不適切なデータ値が許可されており、データ入力の際にそれらを有効値に強制変換します。しかし、サーバSQLモードを変更することで、不良値が生じた際にサーバが拒絶やステートメントの実行を中止するような、従来により近い不良値の処理方法が選択できます。[SQL モード](#) を参照してください。

このセクションでは、MySQLのデフォルト（許可されている）処理方法と厳格SQLモードの処理方法の相違点について説明しています。

厳格モードを使用していない場合、[NOT NULL](#)カラムにおけるNULLや数値カラムにおける大きすぎる数値のように「正しくない」値をカラムに挿入すると、MySQL ではエラーを生成するのではなく、カラムを「最適可能値」に設定します。この動作規則について、以下に詳しく説明します。

- 数値カラムに範囲外の値を格納しようとする、代わりにMySQLサーバは0、使用可能な最小値、または使用可能な最大値（いずれも無効値に近い値）を格納します。
- 文字列については、MySQLは空白文字列、またはカラム内で格納可能な最長文字列を格納します。
- 数値カラムに数値で始まらない文字列を格納しようとする、MySQLサーバは0を格納します。
- [ENUM](#)や[SET](#)カラムに対する無効値は、[ENUMおよびSET制約](#)に記載のとおり処理されます。
- MySQLでは、[DATE](#)や[DATETIME](#)カラム('2000-02-31'や'2000-02-00'のようなもの)に、特定の不正確なデータ値を格納することが許可されています。これは、SQLサーバはデータ検証を行わないことを意味します。データ値が格納できるか、または正確に同値が検索できる場合、MySQLはそれを既存値として格納します。日付が完全に不正確(サーバの格納能力を超えている)の場合は、代わりに特定の「ゼロ」日付値'0000-00-00'がカラムに格納されず。

- **NULL**値を使用することができないカラムに**NULL**を格納しようとする、単一行**INSERT**ステートメントに対するエラーが生じます。複数行**INSERT**ステートメントや**INSERT INTO ... SELECT**ステートメントに対して、MySQLサーバは、カラムデータ型に含まれるデフォルト値を格納します。たいていこれは、数値型の**0**、文字列型の空白文字列(")、および日時型の「ゼロ」値です。暗黙のデフォルト値に関しては [データタイプデフォルト値](#) で説明されています。
- **INSERT**ステートメントがカラムに対して値を指定しない場合、MySQLはカラム定義が既存の**DEFAULT**節を含んでいれば、そのデフォルト値を挿入します。定義に**DEFAULT**節のような節が含まれていなければ、MySQLはカラムデータ型に含まれるデフォルト値を挿入します。

非厳格モードにおける前述のルールの理由は、ステートメントの実行が開始される前にこれらの条件をチェックできないためです。複数の行を更新した後で問題が発生した場合、ストレージエンジンでサポートされていない可能性があるため、単にロールバックすることはできません。停止するというオプションは、この場合、更新が「未完了」のため、最悪のシナリオになる可能性があるため、適切ではありません。この場合「できる限りのことを行って」、何も問題が発生していないものとして処理を続行する方が適切です。

MySQL 5.0.2以降では、**STRICT_TRANS_TABLES**や**STRICT_ALL_TABLES** SQLモードを使用して、より厳格な入力値処理を行なうことができます。

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

STRICT_TRANS_TABLESでは、トランザクショナルストレージエンジンに対して厳格モードが動作可能となり、同様に非トランザクショナルエンジンに対してもいくらかの拡張が可能となります。これは、以下のように動作します。

- トランザクショナルストレージエンジンにとって、ステートメント内で生じる不良データ値は、ステートメントの実行中止やロールバックを引き起こします。
- 非トランザクショナルストレージエンジンにとって、挿入や更新がなされる最初の行でエラーが生じると、ステートメントの実行は中止されます。(トランザクショナルテーブルについては、最初の行でエラーが生じた場合、テーブルを変換させないようにするためにステートメントの実行を中止することができます。)2行以降でエラーが生じた場合は、ステートメント実行は中止されません。これは、最初の行によってテーブルが既に変換されているためです。エラー発生の代わりに不良データ値が適応され、この結果警告が発せられます。言い換えると、テーブルが変換されない場合は、**STRICT_TRANS_TABLES**によって、不正な値はMySQLにそれまでに行なわれたすべての更新をロールバックさせます。しかし、いったんテーブルが変換されると、それ以降に生じるエラーは適応や警告になります。

STRICT_ALL_TABLESは、より厳格なチェックについても使用できます。非トランザクショナルストレージエンジンでは2行以降の不良値に対してもエラーがステートメント実行を中止する、という点を除くと、上記のことは**STRICT_TRANS_TABLES**と同じです。これは、非トランザクショナルテーブルへの複数行挿入もしくは更新の途中でエラーが生じた場合、部分的な更新のみが行なわれるということを意味します。より前の行が挿入または更新されますが、それらはエラーから生じるものではありません。非トランザクショナルテーブルでこのことを避けるには、エラーよりも変換警告のほうが適切な場合、単一行ステートメントまたは**STRICT_TRANS_TABLES**のどちらかを使用してください。初期段階でこのような問題を避けるには、MySQLでカラム内容をチェックさせないようにしてください。データベースに対して確実に有効値のみが通過するようにアプリケーションを設定することが、最も安全(であり、時には高速)な方法です。

厳格モードオプションのいずれかで、**IGNORE**を用いずに**INSERT**や**UPDATE**よりも、**INSERT IGNORE**または**UPDATE IGNORE**を用いることで、エラーを警告として処理することができます。

ENUMおよびSET制約

ENUMおよび**SET**カラムによって、特定の値セットのみを含むカラムを効率的に定義することができます。[ENUMタイプ](#)と[SETタイプ](#)を参照してください。ただし、MySQL 5.0.2以前では、**ENUM**および**SET**カラムは無効値の入力に対する実際の制約ではありません。

- **ENUM**カラムには常にデフォルト値があります。デフォルトでない値を指定した場合、それは**NULL**を持ちうるカラムに対する**NULL**となります。もしくは、カラム定義における最初の列挙値となります。
- **ENUM**カラムに正しくない値を挿入した場合、もしくは**IGNORE**を用いて**ENUM**カラムにある値を強制した場合は、予約された列挙値**0**に設定され、文字列コンテキストでは空白文字列として表示されます。

- **SET**カラムに正しくない値を挿入した場合、その値は無視されます。たとえば、カラムが'a'、'b'、そして'c'値を含む場合、'a,x,b,y'を割り当てようとすると'a,b'値になってしまいます。

MySQL 5.0.2以降では、厳格SQLモードを使用するためにサーバのコンフィギュレーションを行なうことができます。詳しくは [SQL モード](#) を参照してください。厳格モードを使用している場合、**ENUM**や**SET**カラムの定義は、カラム入力値の制約としては振舞いません。これらの条件を満たさない値に対しては、エラーが生じます。

- **ENUM**値はカラム定義に一覧表示されているものか、もしくはそれについて同等の内部数値でなければなりません。その値はエラー値（つまり、0または空白文字列）にはなりません。**ENUM**('a','b','c')として定義されるカラムに対して、'、'd'または'ax'といった値は無効であり、かつ拒絶されます。
- **SET**値は空白文字列、もしくはコンマで区切られたカラム定義に挙げられている値のみで構成された値でなければなりません。**SET**('a','b','c')として定義されるカラムにとって、'd'または'a,b,c,d'といった値は無効であり、かつ拒絶されます。

無効値に対するエラーは、**INSERT IGNORE**や**UPDATE IGNORE**を使用している場合、厳格モードで抑制されます。この場合、エラーではなく警告が発せられます。**ENUM**に対しては、その値はエラーメンバー(0)として挿入されます。**SET**に対しては、どの無効部分文字列も消去されるという点を除いて、その値は既存のものとして挿入されます。たとえば、'a,x,b,y'は'a,b'値となります。

第2章 MySQL のインストールと更新

目次

一般的なインストールの問題	32
MySQL Community Server がサポートしているオペレーティング システム	33
インストールする MySQL 配布版の選択	34
MySQL の取得方法	43
MD5 チェックサムあるいは GnuPG を用いたパッケージの品質の検証	43
インストールのレイアウト	45
バイナリの配布を使用した標準 MySQL のインストール	47
Windows に MySQL をインストールする	47
インストール用パッケージの選択	48
自動インストーラで MySQL をインストールする	48
MySQL インストール ウィザードを使用する	49
設定ウィザードを使用する	51
非インストール Zip アーカイブからのインストール	55
インストール アーカイブを取り出す	55
オプション ファイルの作成	55
MySQL サーバ タイプの選択	56
サーバを最初に起動する	57
MySQL の Windows のコマンドラインからの起動	58
Windows のサービスとして MySQL を起動する	59
MySQL インストールのテスト	61
Windows への MySQL インストールにおけるトラブルシューティング	61
Windows を使用した MySQL をアップグレードする	62
Windows 上の MySQL と Unix 上の MySQL の比較	63
Linux に MySQL をインストールする	65
Mac OS X に MySQL をインストールする	67
Solaris に MySQL をインストールする	69
MySQL を NetWare にインストールする	70
他の Unix 系システムへの MySQL のインストール	71
ソースのディストリビューションを使用した MySQL のインストール	73
ソースのインストール概要	74
典型的な configure オプション	76
開発ソース ツリーからのインストール	80
MySQL のコンパイルに関する問題	82
MIT-pthreads ノート	84
Windows にソースから MySQL をインストールする	85
Windows で MySQL クライアントをコンパイルする	88
インストール後の設定とテスト	89
Windows のインストール後のプロシージャ	89
Unix のインストール後のプロシージャ	90
最初の MySQL アカウントの確保	99
MySQL のアップグレード	101
MySQL 5.0 から 5.1 へのアップグレード	102
MySQL データベースの他のマシンへのコピー	105
MySQL のダウングレード	106
MySQL 5.0 へのダウングレード	107
オペレーティング システムに特化した注釈	107
Linux の注釈	107
Mac OS X に関する注釈	113
Solaris に関する注釈	114
BSD に関する注釈	117
他の Unix に関する注釈	120
OS/2 に関する注釈	133
環境変数	133

Perl のインストールに関する注釈	134
Unix に Perl をインストールする	134
Windows に ActiveState Perl をインストールする	135
Perl DBI/DBD インターフェースを使用した際の問題	136

この章では MySQL の取得とインストールについて説明します。最初の手順の概要を、後半の項で詳細を説明します。MySQL を最初にインストールするのではなく、現行の MySQL を新しいバージョンにアップグレードするには、[MySQL のアップグレード](#) を参照してください。アップグレード前に考慮すべき問題点およびアップグレードの手順に関する情報を網羅しています。

MySQL を別のデータベースシステムから移行する場合には [MySQL 5.1 FAQ — Migration](#) を参照してください。この資料の中には移行の問題に関する質問に対する解答が含まれています。

- MySQL の実行と現在使用中のプラットフォームのサポート。MySQL はどのプラットフォームでも同様に動作する訳ではなく、また MySQL が動作するといわれているプラットフォームのすべてを MySQL AB がサポートしている訳ではありません。
 - MySQL Enterprise Server を公式にサポートしているプラットフォームの一覧は <http://www.mysql.com/support/supportedplatforms.html> にあります。
 - MySQL Community Server は [MySQL Community Server がサポートしているオペレーティング システム](#) の一覧にあるプラットフォームで動作します。
- どの配布版をインストールするか選択します。MySQL にはいくつかのバージョンがあり、その殆どがいくつかのプラットフォームで動作します。バイナリ (コンパイル済み) プログラムあるいはソースコードを含むパッケージを選択できます。疑問がある場合にはバイナリの配布版が無難です。弊社ではまた弊社の最新の開発に興味があり、新しいコードのテストにご協力頂ける方にソースを公開しています。どのバージョンのどの配布版を選択したらよいかは [インストールする MySQL 配布版の選択](#) をご覧ください。
- インストールする配布版をダウンロードします。インストールの説明は、[MySQL の取得方法](#) をご覧ください。配布版の品質を確認するには、[MD5 チェックサムあるいは GnuPGを用いたパッケージの品質の検証](#) の説明を参照してください。
- 配布版をインストールします。MySQL をバイナリでインストールするには、[バイナリの配布を使用した標準 MySQL のインストール](#) の説明書を参照してください。MySQL をソースの配布版からインストールする、あるいは現在の開発版からインストールするかは、[ソースのディストリビューションを使用した MySQL のインストール](#) を参照してください。
インストールで問題がありましたら、[オペレーティング システムに特化した注釈](#) をご覧ください。特定のプラットフォームに関する問題の解決法を載せてあります。
- インストール後の必要な設定を実行します。MySQL をインストールしたら、[インストール後の設定とテスト](#) をお読みください。この項では MySQL の動作検証に関する重要な情報を提供します。また最初の MySQL ユーザーアカウントの設定方法、特にパスワードを割り当てられるまでのパスワードが無い場合 の設定についても説明します。この項はバイナリおよびソース配布双方の MySQL のインストールを網羅しています。
- MySQL のベンチマーク スクリプトの実行を希望される場合、Perl の MySQL サポートが必要になります。[Perl のインストールに関する注釈](#) 参照。

一般的なインストールの問題

MySQL のインストール手順は MySQL Enterprise Server あるいは MySQL Community Server によって異なります。使用できるプラットフォームはどの配布版をインストールするかによって異なります。

- MySQL Enterprise Server を公式にサポートしているプラットフォームの一覧は <http://www.mysql.com/support/supportedplatforms.html> にあります。
- MySQL Community Server は [MySQL Community Server がサポートしているオペレーティング システム](#) の一覧にあるプラットフォームで動作します。

MySQL Enterprise Server の場合、Enterprise のインストーラを使用してメインの配布版および必要なサービスパックあるいはホットフィックスをインストールします。Enterprise インストーラをまだ用意していないプラットフォームの場合、Community Server の説明書を使用します。

MySQL Community Server には、メインの配布版とホットフィックスとその更新をインストールします。

- バイナリのリリースあるいはソースのリリースをダウンロードしてソースコードで MySQL を構築します。
- MySQL を BitKeeper ツリーから取り出してソースから構築します。BitKeeper ツリーには最新の開発コードが含まれています。

次の数項で配布版の選択、ダウンロード、および検証に必要な情報を掲げています。本章の次項以降で選択された配布版のインストールについて説明します。バイナリの配布版については、[バイナリの配布を使用した標準 MySQL のインストール](#)の説明書を参照してください。MySQL をソースから構築するには、[ソースのディストリビューションを使用した MySQL のインストール](#)の説明書を使用します。

MySQL Community Server がサポートしているオペレーティング システム

この項では MySQL Community Server が動作するオペレーティング システムを採り上げます。

重要MySQL ABではこの項で採り上げるすべてのオペレーティング システムに公式のサポートは提供していません。MySQL AB が公式にサポートしているプラットフォームは MySQL のウェブサイトにある <http://www.mysql.com/support/supportedplatforms.html> を参照してください。

弊社では GNU Autoconf を使用していますので、C++ のコンパイラを実装した最新のシステムに MySQL をポートして POSIX スレッドを実装できます。(スレッドのサポートがサーバに必要です。クライアントコードのみをコンパイルする場合には、C++ コンパイラのみが必要です。)弊社では主に Linux (SuSE および Red Hat)、FreeBSD、および Sun の Solaris (バージョン 8 と 9) を使用して自社でソフトウェアの開発を行っています。

MySQL は以下のオペレーティング システムとスレッド パッケージの組み合わせでのコンパイルでよい結果が出ます。多くのオペレーティング システムでは、ネイティブのスレッドのサポートは最新バージョンのみで動作します。

- AIX 4.x, 5.x (ネイティブ スレッド)。 [IBM-AIX に関する注釈](#) 参照。
- Amiga。
- BSDI 2.x (MIT-pthreads パッケージ) 。 [BSD/OS Version 2.x に関する注釈](#) 参照。
- BSDI 3.0, 3.1 および 4.x (ネイティブ スレッド)。 [BSD/OS Version 2.x に関する注釈](#) 参照。
- Digital Unix 4.x (ネイティブ スレッド)。 [Alpha-DEC-UNIX に関する注釈 \(Tru64\)](#) 参照。
- FreeBSD 2.x (MIT-pthreads パッケージ)。 [FreeBSD に関する注釈](#) 参照。
- FreeBSD 3.x and 4.x (ネイティブ スレッド)。 [FreeBSD に関する注釈](#) 参照。
- FreeBSD 4.x (ネイティブ スレッド)。 [FreeBSD に関する注釈](#) 参照。
- HP-UX 10.20 (DCE スレッドあるいは MIT-pthreads パッケージ)。 [HP-UX バージョン 10.20 に関する注釈](#) 参照。
- HP-UX 11.x (ネイティブ スレッド)。 [HP-UX バージョン 11.x に関する注釈](#) 参照。
- Linux 2.0+ LinuxThreads 0.7.1+ あるいは [glibc 2.0.7+](#) (いくつかの CPU アーキテクチャ用)。 [Linux の注釈](#) 参照。
- Mac OS X. [Mac OS X に関する注釈](#) 参照。
- NetBSD 1.3/1.4 Intel および NetBSD 1.3 Alpha (GNU 仕様)。 [NetBSD に関する注釈](#) 参照。
- Novell NetWare 6.0 および 6.5。 [MySQL を NetWare にインストールする](#) 参照。
- OpenBSD 2.5 およびネイティブ スレッド付き。OpenBSD 2.5 以前 (MIT-pthreads パッケージ) [OpenBSD 2.5 に関する注釈](#) 参照。
- OS/2 Warp 3, FixPack 29 および OS/2 Warp 4, FixPack 4。 [OS/2 に関する注釈](#) 参照。
- SCO OpenServer 5.0.X の最新 FSU Pthreads パッケージ サポート版 [SCO UNIX および OpenServer 5.0.x に関する注釈](#) 参照。
- SCO Openserver 6.0.x。 [SCO OpenServer 6.0.x に関する注釈](#) 参照。

- SCO UnixWare 7.1.x。 [SCO UnixWare 7.1.x および OpenUNIX 8.0.0 に関する注釈](#) 参照。
- SGI Irix 6.x (ネイティブ スレッド)。 [SGI Irix に関する注釈](#) 参照。
- Solaris 2.5 およびそれ以降で SPARC および x86 のネイティブ スレッド付き。 [Solaris に関する注釈](#) 参照。
- SunOS 4.x (MIT-pthreads パッケージ) [Solaris に関する注釈](#) 参照。
- Tru64 Unix。 [Alpha-DEC-UNIX に関する注釈 \(Tru64\)](#) 参照。
- Windows 9x、Me、NT、2000、XP、および Windows Server 2003。 [Windows に MySQL をインストールする](#) 参照。

すべてのプラットフォームが MySQL の動作に適しているという訳ではありません。どのプラットフォームが高負荷のミッション クリティカルな MySQL サーバに適しているかは以下の要因によります。

- スレッド ライブラリの総合的な安定性。プラットフォームそのものは評判が良くすべてがパーフェクトであっても、MySQL の安定性はそのプラットフォームではなくスレッドのライブラリに因ります。
- kernel および スレッド ライブラリの特徴は SMP (対称型マルチプロセッサ) を利点を活かせることにあります。換言すれば、プロセッサがスレッドを作成すると、オリジナル プロセスの異なる CPU でスレッドを実行できることです。
- kernel およびスレッド ライブラリの特徴は過剰なコンテキスト スイッチなしで、狭いクリティカルな領域で頻繁に mutex を取得してリリースする多くのスレッドを実行できることです。 [pthread_mutex_lock\(\)](#) の実装によって CPU の時間節約の汲々とした場合、MySQL に大きな影響を与えます。この問題に適切に対処しないと、別の CPU を追加することにより実質的に MySQL が遅くなります。
- 総合的なファイルシステムの安定性とパフォーマンス。
- テーブルが大きい場合、パフォーマンスはファイルシステムの大きなファイルの処理およびその効果的な処理能力によって影響を受けます。
- MySQL AB のプラットフォームに対する専門知識プラットフォームに対する知識および経験が豊富な場合、プラットフォームに特化した最適化および強化をコンパイル時にできます。弊社はまた MySQL を実装したお客様の最適なシステム構築に助言することもできます。
- 実際の事例に基づいた類似の構築に対する豊富な経験
- MySQL を使用した類似の構築によるプラットフォームの多くの成功事例 (お客様) この成功事例の数が多いほどプラットフォーム独特の問題に対する対処能力が向上し、問題を迅速に処理できます。

以前説明した基準では、この段階での MySQL の特性を最適に活かす組み合わせは x86 の 2.4 あるいは 2.6 kernel、および ReiserFS (あるいは同様の Linux 配布版) および SPARC 搭載 Solaris (2.7-9) です。三番目に FreeBSD が来ますが、弊社はこのスレッド バイナリが改善されてトップ グループの仲間入りすることを期待しています。弊社ではまた近いうちに MySQL は現在コンパイルされ動作できるものの、トップ グループに比べるとその安定性とパフォーマンスで劣る他のすべてのプラットフォームをトップ グループに含めたいと希望していますこれを実現するには MySQL が依存している オペレーティング システムやライブラリの開発社と共同作業する必要があります。お客様がそれらのコンポーネントの改善にご興味があり、またその開発に影響を及ぼすに十分な立場の方で、MySQL をさらによくするための詳細な説明書が必要な場合には、MySQL [internals](#) メーリング リストにメールをお願いします。 [MySQL メーリング リスト](#) 参照。

様々なオペレーティング システムの比較表を以前作成しましたが、この目的はそれらのオペレーティング システムの良し悪しを論ずることが目的ではないことをご理解願います。弊社では MySQL を特定の目的に対して OS を選択しているのであって他意はありません。このことを念頭に頂くと、弊社の比較の結果は他の要素が加わった場合には異なる場合があることもご理解頂けるものと思います。あるケースに於いて、ある一つの理由である一つ OS が MySQL の試験でいい結果であるということは、弊社がその特定のプラットフォームに関して試験および最適化で少しばかり多くの時間を割いたということに他なりません。弊社では最近お客様のためにどのプラットフォームが MySQL の動作に一番適しているかの検討を始めました。

インストールする MySQL 配布版の選択

MySQL をインストールする際に、どのバージョンをインストールするか決める必要があります。MySQL の開発にはいくつかのリリース シリーズがありますので、お客様の仕様に一番適合したものを選択できます。インストールする

バージョンを決めたら、配布フォーマットを選択します。リリースはバイナリあるいはソース フォーマットでご利用頂けます。

インストールする MySQL のバージョンの選択

最初に決めることは生産 (安定) リリースあるいは開発リリースのどちらにつるかと言うことです。MySQL の開発に於いては、開発の完成度に応じた複数のリリース シリーズがあります。

- MySQL 5.1 は現在開発中のリリース シリーズです。
- MySQL 5.0 は現在の安定した (量産品) リリース シリーズです。新しいリリースはバグ出しのためのみのリリースです。安定性の効果をもたらす新しい機能は追加されていません。
- MySQL 4.1 は以前の安定した (量産品) リリース シリーズです。新しいリリースは重大なバグ出しやセキュリティ対策のためにリリースされます。このシリーズには重要な新しい機能は導入されていません。
- MySQL 4.0 および 3.23 は旧バージョンの安定した (量産用) リリース シリーズです。これらのバージョンは既に廃番になっていますので、新しいリリースは特に重大なバグ出し (主にセキュリティの問題) のためにのみリリースされます。

弊社としては完全にコードが使用できなくなる (凍結) とは想定していません。というのは、これによってバグ出しや修正など必要が対策が出来なくなるからです。「ある程度凍結」は現在の量産リリースの機能に影響を及ぼさずに小さな修正ができるものを意味しています。当然のことながら、旧バージョンのバグ出しで関連するものはその後のバージョンに反映されています。

お客様が MySQL を最初にお使いになる場合やバイナリの配布のないシステムへのポートをご検討されている場合には、量産シリーズを選択されるようお勧めします。現在の量産シリーズは MySQL 5.0 です。ご利用前に MySQL リリースのすべてを、開発段階のものも含めて、MySQL のベンチマークでチェックし、広範な一連のテストを行うようお願いします。

お客様が旧バージョンを使用していてアップグレードを希望される場合で、シームレスのアップグレードを好まれる場合には、最新バージョンへアップグレードする際には現在お使いのシリーズ (バージョン番号の最後の数字が現在お使いのものより新しいもの) と同じシリーズでアップグレードする必要があります。弊社では致命的なバグの修正および小さな、比較的「安全な」変更のみをそのバージョンに行っています。

量産リリースのシリーズで新しい機能の使用を希望される場合には、開発シリーズのバージョンを使用できます。開発シリーズは量産シリーズに比べて安定性に欠けることをご理解ください。

現在までのすべてのパッチおよびバグ修正を含む最も最新のソースを希望される場合、弊社の BitKeeper レポジトリの中から選択できます。これらは「リリース版」ではありませんが、機能ベースのリリースを基にしたコードを試作品を利用できます。

MySQL の命名規則は 3 つの番号に接尾辞を使用しています。例えば、mysql-5.0.12-beta のようになります。リリースの番号の内訳は以下ようになります。

- 最初の (5) は主なバージョンおよびファイル フォーマットを表します。MySQL 5 のすべてのリリースのファイルフォーマットは同じです。
- 2 番目の番号は (0) はリリースのレベルを表します。主なバージョン番号とリリース レベルを一緒にすると、リリースのシリーズ番号を表します。
- 3 番目の番号 (12) はリリース シリーズのバージョン番号を表します。これは新しいリリース毎に数が増えます。一般的には選択したシリーズの最新のバージョンを選択されます。

マイナーな更新が行われた場合、最後のバージョン文字列の数字が大きくなります。重要な機能の追加あるいは旧バージョンに互換性のないマイナーな変更が加えられた場合には、バージョン文字列の 2 番目の番号が大きくなります。ファイル フォーマットが変更になった場合、最初の番号の値が増えます。

リリース名にはまたリリースの安定性を示す接尾辞が含まれています。一連の接尾辞のシリーズの進展があった場合のリリースは安定性のレベルの改善の状態を表します。接尾辞には以下のようなものがあります。

- アルファ は検討用のみのリリースを意味します。既知のバグはニュース セクション ([付録C MySQL Change History](#) 参照) に纏められています。殆どのアルファ リリースには新しいコマンドや拡張を導入されています。重要なコー

ドの変更を含む能動的な開発がアルファ リリースで行われます。しかし、リリースを公開する前に弊社でテストを行っています。

- ベータ は新規の開発に使用できることを意味しています。ベータ リリースでは機能および互換性の変更ありません。しかし、ベータのリリースには多くの重大なバグが含まれている場合があります。

すべてのAPI、外部から可視可能な構成、および SQL ステートメントのカラムは将来のベータ、リリース候補、あるいは量産リリースでも変更はありません。

- rc はリリース候補を意味します。リリース候補は安定していると思われており、MySQL の社内のすべてのテストに合格したもので、すべての既知の致命的なランタイムのバグは修正しています。しかしながら、これらのリリースはすべてのバグが間違いなく取り除かれたと確信するに足る広範な使用事例を経たものではありません。マイナーは修正のみが追加されています。(リリース候補は以前公式にはガンマ リリースと呼ばれていたものです。)
- 接尾辞が無い場合、そのリリースは一般的に利用できる (GA) あるいは量産リリースを意味します。GA リリースは安定しており、旧バージョンのすべてのバグ出しを完了したもので、信頼でき、重大なバグのない、量産システムの使用に適したものと認識されています。重大なバグ出しのみがそのリリースに行われます。

MySQL は命名規則を使用していますが、それは殆どの他の製品と多少異なります。一般的には、同じリリースのシリーズに置き換えることなく数週間使用できた場合にはそのバージョンは使用しても通常は安全です。

すべての MySQL のリリースは標準のテストおよびベンチマーク実施して使用に関しては相対的な安全を確認されています。標準のテストは旧バージョンのすべてのバグ修正の経験と事例に基づいていますので、テスト項目とその精度はその都度向上しています。

リリースはすべて最低以下の項目でテストを実施しています。

- 社内のテスト項目

[mysql-test](#) ディレクトリには広範なテスト項目含まれます。弊社ではこれらのテストをすべてのサーババイナリに実施しています。これらのテスト項目の詳細に関しては、[MySQL Test Suite](#) を参照してください。

- MySQL ベンチマーク項目

このテストでは一連の共通のクエリを実施しています。最新の最適化パッチが実際にコードの処理速度の向上させるかについてもテストを実施して決めています。[MySQL ベンチマークスイート](#) 参照。

- [クラッシュ](#) テスト

テストではデータベースがサポートしている機能およびその可能性と限界に以下テストします。[MySQL ベンチマークスイート](#) 参照。

弊社ではまた最新の MySQL バージョンを社内の生産環境で、最低でも 1 台のマシンで実施しています。100GB 以上のデータで検証しています。

配布フォーマットの選択

インストールする MySQL を決定したら、次にバイナリの配布にするかソースの配布にするか決める必要があります。殆どの場合、お客様のプラットフォームに適したものがあ場合、バイナリの配布を使用することになります。バイナリの配布は多くの Linux の RPM あるいは Mac OS X または Solaris の PKG パッケージ インストーラなど多くのプラットフォームがネイティブのフォーマットで利用できます。配布は Zip アーカイブあるいは圧縮 [tar](#) ファイルで利用できます。

バイナリの配布を選択する理由は以下のようになります。

- バイナリの配布は一般的にはソースの配布よりインストールが簡単です。
- ユーザー毎の仕様に満足するために、弊社ではいくつかのサーバをバイナリの配布で用意しています。[mysqld](#) は最適化したサーバで小規模ながら高速のバイナリです。[mysqld-debug](#) はデバッグのサポートでコンパイルしています。

これらのサーバはすべて同じソースの配布でしかも異なる設定オプションでコンパイルされています。すべてのネイティブ MySQL クライアントはどの MySQL バージョンにも接続できます。

環境によっては、MySQL をソースの配布でインストールしたほうが良い場合もあります。

- 明示のロケーションでの MySQL をインストールが望まれる場合もあります。標準のバイナリの配布はインストールのロケーションにこだわりますが、MySQL コンポーネントを希望する場所に配置することで柔軟性を更に向上させる必要に駆られる場合があります。
- `mysqld` を標準のバイナリの配布には含まれていない機能を使用した設定を希望される場合があります。機能の可用性を確認するための最も一般的な予備オプションの一覧を以下に示します。
 - `--with-libwrap`
 - `--with-named-z-libs` (これはいくつかのバイナリに実行されます)
 - `--with-debug[=full]`
- `mysqld` を標準のバイナリの配布に含まれるいくつかの機能を使用しないで設定する場合に使用します。例えば、バイナリは通常すべての文字セットをサポートするようにコンパイルされています。小規模な MySQL サーバを希望される場合、使用する文字セットのみのサポートでコンパイルできます。
- 特別なコンパイラ (`pgcc` など) あるいはお客様のプロセッサに適したコンパイラを使用できます。バイナリの配布は同じプロセッサ ファミリーの様々なプロセッサで動作するオプションにコンパイルされています。
- BitKeeper レポジトリの最新のソースを使用して現行のすべての bugfix にアクセスできます。例えば、バグが見つかりそれを MySQL の開発チームに送ると、バグはソースのレポジトリで修正され、そのレポジトリにアクセスできます。バグの修正はリリースが実際に公開されるまでリリースには表示されません。
- MySQL の開発に使用された C および C++ コードを読む (修正し) ことができます。これを行うには、ソースの配布を取得する必要があります。ソースコードは常に最適なマニュアルです。
- ソースの配布にはバイナリの配布より多くのテスト項目および例があります。

リリースの更新方法と時期

MySQL の展開は極めて速いため、他の MySQL ユーザーと新しい開発を共有したいと考えています。弊社では他でも必要と思われる新規の有用な機能あればいつでも新しいリリースを公開していきます。

導入が容易なユーザー要求の新機能についてもユーザーを支援していきます。ライセンス供与を受けたユーザーのご希望、特に弊社がサポートしているお客様のご要望に耳を傾け、それを支援して参ります。

新しいリリースをダウンロードする必要はありません。ニュース欄をご覧になれば新しいリリースがお客様の仕様に満たしているか確認できます。[付録C MySQL Change History](#) 参照。

MySQL の更新は以下の方針に基づいています。

- Enterprise Server は 18 カ月毎にリリースし、その都度 4 カ月毎のサービス バックおよび毎月の更新を補充していきます。Community Server は年に 2-3 回リリースを出します。
- リリースは各シリーズ毎に発行します。Enterprise Server のリリースの番号は偶数 (例えば、5.1.20) を使用しています。Community Server のリリースには奇数 (例えば、5.1.21) を使用します。
- いくつかのプラットフォームのバイナリの配布は主要なリリースの際に配布します。他のシステムのバイナリの配布は他で行われる場合がありますが、その頻度は多少少ないものと思われます。
- バグが見つかり次第できるだけ早急に、難解なバグを除く小さいものや余り重大でないものから最初に対処します。バグ修正版は一般に公開している弊社の BitKeeper のレポジトリのソースから即座に入手できるようになっており、そのバグの修正は次のリリースに更新されます。
- リリースでセキュリティの脆弱性および深刻なバグが見つかった場合には、弊社の方針に則ってそれを新しいリリースで早急に対応します。(弊社では他の会社もこのように早急な対策するよう希望しています！)

リリースの原理—リリースの未知のバグ

弊社ではバグのないリリースを目指してかなりの時間と労力を費やしています。弊社の方針で MySQL バージョンに既知の致命的な再現性のバグがある場合には量産品をリリースしません。

弊社では設計に関わるすべての一般的な問題、バグ、および懸案事項をすべて文書にまとめています。[Known Issues in MySQL](#) 参照。

弊社では MySQL バージョンの安定性を損なわずに修正可能なバグをすべて修正することを目標としています。場合によっては、これは安定したバージョン（量産品）ではなく開発バージョンの問題の修正を意味します。当然のことですが、そのような問題をユーザーに認識させるために文書にまとめています。

以下に弊社のビルド プロセスを説明します。

- <http://bugs.mysql.com/> にあるバグ データベースのカスタマーサポート リスト、および MySQL 外部からのメーリング リストでバグを監視します。
- 現在使用されているバージョンでレポートされたすべてのバグはバグの データベースに入力されます。
- バグを修正した場合には必ずテストケースを作成し、バグが検知されずに再現しないようにテスト システムに加えます。(修正したバグのおよそ 90% にテストケースがあります。)
- MySQL に追加する各新規機能にテストケースを作成します。
- 新しい MySQL をリリースする前に、報告されたすべての再現性バグが MySQL バージョン (3.23.x、4.0.x、4.1.x、5.0.x、5.1.x など) で修正されているか確認します。MySQL の内部の設計上の問題で修正が不可能な場合には、これをマニュアルに詳細に記録します。[Known Issues in MySQL](#) 参照。
- サポートしているバイナリのすべてのプラットフォームにビルドを行い、それらのすべてに一連のテストおよびベンチマークを実行します。
- 一連のテストあるいはベンチマークに失敗したプラットフォームにはバイナリを公開しません。その問題がソースの一般的な問題の場合には、その問題を解決し、ビルドしてすべてのシステムに再度ゼロからテストを実施します。
- ビルドとテスト プロセスに一週間かかります。このプロセスで致命的なバグに関する報告があった場合 (例えば、コアのダンプにつながるバグ)、その問題を修正し再度ビルド プロセスを実行します。
- バイナリを <http://dev.mysql.com/> で公開した後、その通知メッセージを [mysql](#) に送り [announce](#) メーリング リストに発表します。[MySQL メーリング リスト](#) 参照。通知メッセージにはリリースに対するすべての変更およびそのリリースの既知の問題が含まれています。このリリース ノートの既知の問題セクションはこれまではほんの一握りのリリースに必要でした。
- ユーザーに MySQL の機能に素早くアクセスして頂くために、新しい MySQL リリースを 4-8 週間でリリースする予定です。ソースコードのスナップショットは毎日ビルドしており、<http://downloads.mysql.com/snapshots.php> から入手できます。
- 弊社で最大限の努力をした場合でも、リリース公開後に特定のプラットフォームで構築中に重大な問題があるとの報告を受ける場合があります。弊社ではそれを直ぐに修正してそのプラットフォームに新しい 'a' リリースをビルドします。多数にユーザーにご利用頂いているお陰で、問題の発見および解決が非常に迅速にできます。
- 安定したリリースを可能にする弊社の追跡記録は非常に優れています。過去 150 のリリースで、これまで新しいビルドはそのうち 10 件以下です。これらの内 3 件は、バグが弊社のビルド マシンの一つで間違った [glibc](#) ライブラリにあったために、バグの追跡に長い時間がかかりました。

MySQL AB でコンパイルした MySQL バイナリ

MySQL AB のサービスの一環として、弊社では弊社のサイトでコンパイルしたシステムあるいは MySQL のサポーターのご好意によりサポーターのマシンにアクセスしてコンパイルした MySQL のバイナリを配布しております。

プラットフォームに特化したパッケージ フォーマットのバイナリに加え、弊社では圧縮した [tar](#) ファイル ([tar.gz files](#)) のフォームで多数のバイナリ配布を提供しています。[バイナリの配布を使用した標準 MySQL のインストール](#) 参照。

弊社のウェブ サイトから入手可能な MySQL のリリースの RPM 配布は 5.1 MySQL AB が作成したものです。

Windows の配布は、[Windows に MySQL をインストールする](#) を参照してください。

これらの配布はスクリプト [Build-tools/Do-compile](#) を使用して生成したもので、そのスクリプトでソースコードをコンパイルして、[scripts/make_binary_distribution](#) を使用して [tar.gz](#) アーカイブを作成します。

これらのバイナリは以下のコンパイラとオプションを使用して設定およびビルドします。これらの情報は各バイナリの [tar](#) ファイル配布のスクリプト [bin/mysqlbug](#) にある変数 [COMP_ENV_INFO](#) および [CONFIGURE_LINE](#) をご覧頂くことで取得できます。

以下のいずれかの `configure` コマンドに最適なオプションをお持ちの方は MySQL [internals](#) メーリング リストにメールして頂ければ幸いです。MySQL [メーリング リスト](#) 参照。

MySQL のデバッグ バージョンのコンパイルを希望される場合、`--with-debug` あるいは `--with-debug=full` を以下の `configure` コマンドに追加して `-fomit-frame-pointer` オプションを削除します。

以下のバイナリは MySQL AB の開発システムでビルドされます。

- Linux 2.4.xx x86 / [gcc](#) 2.95.3:

```
CFLAGS="-O2 -mcpu=pentiumpro" CXX=gcc CXXFLAGS="-O2 -mcpu=pentiumpro
-felide-constructors" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --disable-shared
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.x x86 / [icc](#) (Intel C++ コンパイラ 8.1 あるいはそれ以降のリリース):

```
CC=icc CXX=icpc CFLAGS="-O3 -unroll2 -ip -mp -no-gcc-restrict"
CXXFLAGS="-O3 -unroll2 -ip -mp -no-gcc-restrict" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --enable-assembler
--disable-shared --with-client-ldflags=-all-static
--with-mysqld-ldflags=-all-static --with-embedded-server --with-innodb
```

Intel のコンパイラのバージョン 8.1 以降は 'pure' C ([icc](#)) および C++ ([icpc](#)) 個別のドライバを使用しています。[icc](#) バージョン 8.0 あるいはそれ以前のバージョンを MySQL の構築に使用する場合は、`CXX=icc` を設定する必要があります。

- Linux 2.4.xx Intel Itanium 2 / [ecc](#) (Intel C++ Itanium コンパイラ 7.0):

```
CC=ecc CFLAGS="-O2 -tpp2 -ip -nolib_inline" CXX=ecc CXXFLAGS="-O2
-tpp2 -ip -nolib_inline" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile
```

- Linux 2.4.xx Intel Itanium / [ecc](#) (Intel C++ Itanium コンパイラ 7.0):

```
CC=ecc CFLAGS=-tpp1 CXX=ecc CXXFLAGS=-tpp1 ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile
```

- Linux 2.4.xx alpha / [ccc](#) (Compaq C V6.2-505 / Compaq C++ V6.3-006):

```
CC=ccc CFLAGS="-fast -arch generic" CXX=ccc CXXFLAGS="-fast -arch
generic -noexceptions -nortti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-mysqld-ldflags=-non_shared
--with-client-ldflags=-non_shared --disable-shared
```

- Linux 2.x.xx ppc / [gcc](#) 2.95.4:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-embedded-server
--with-innodb
```

- Linux 2.4.xx s390 / [gcc](#) 2.95.3:

```
CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.xx x86_64 (AMD64) / [gcc](#) 3.2.1:

```
CXX=gcc ./configure --prefix=/usr/local/mysql
```



```
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- Sun Solaris 8 x86 / [gcc 3.2.3](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-innodb
```

- Sun Solaris 8 SPARC / [gcc 3.2](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-z-libs=no
--with-named-curses-libs=lcurses --disable-shared
```

- Sun Solaris 8 SPARC 64-bit / [gcc 3.2](#):

```
CC=gcc CFLAGS="-O3 -m64 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-m64 -fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no
--with-named-curses-libs=lcurses --disable-shared
```

- Sun Solaris 9 SPARC / [gcc 2.95.3](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-curses-libs=lcurses
--disable-shared
```

- Sun Solaris 9 SPARC / [cc-5.0](#) (Sun Forte 5.0):

```
CC=cc-5.0 CXX=CC ASFLAGS="-xarch=v9" CFLAGS="-Xa -xstrconst -mt
-D_FORTEC_ -xarch=v9" CXXFLAGS="-noex -mt -D_FORTEC_ -xarch=v9"
./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --enable-assembler
--with-named-z-libs=no --enable-thread-safe-client --disable-shared
```

- IBM AIX 4.3.2 ppc / [gcc 3.2.3](#):

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2
-mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 4.3.3 ppc / [xlc_r](#) (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=xlc_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z-libs=no
--disable-shared --with-innodb
```

- IBM AIX 5.1.0 ppc / [gcc 3.3](#):

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2 -mcpu=powerpc
-Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z-libs=no
--disable-shared
```

- IBM AIX 5.2.0 ppc / [xlc_r](#) (IBM Visual Age C/C++ 6.0):


```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=xlc_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z-libs=no
--disable-shared --with-embedded-server --with-innodb
```

- HP-UX 10.20 pa-risc1.1 / [gcc 3.1](#):

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc CXXFLAGS="-DHPUX
-I/opt/dce/include -felide-constructors -fno-exceptions -fno-rtti
-O3 -fPIC" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-pthread --with-named-thread-libs=ldce
--with-lib-cflags=-fPIC --disable-shared
```

- HP-UX 11.00 pa-risc / [aCC](#) (HP ANSI C++ B3910B A.03.50):

```
CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-embedded-server --with-innodb
```

- HP-UX 11.11 pa-risc2.0 64bit / [aCC](#) (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS="+DD64" CXXFLAGS="+DD64" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
```

- HP-UX 11.11 pa-risc2.0 32bit / [aCC](#) (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-innodb
```

- HP-UX 11.22 ia64 64bit / [aCC](#) (HP aC++/ANSI C B3910B A.05.50):

```
CC=cc CXX=aCC CFLAGS="+DD64 +DSitanium2" CXXFLAGS="+DD64 +DSitanium2"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shared
--with-embedded-server --with-innodb
```

- Apple Mac OS X 10.2 powerpc / [gcc 3.1](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- FreeBSD 4.7 i386 / [gcc 2.95.4](#):

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --with-named-z-libs=not-used
--disable-shared
```

- FreeBSD 4.7 i386 /LinuxThreads [gcc 2.95.4](#):

```
CFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT
-D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads"
CXXFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT
-D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --enable-thread-safe-client
--enable-local-infile --enable-assembler
--with-named-thread-libs="-DHAVE_GLIBC2_STYLE_GETHOSTBYNAME_R
-D_THREAD_SAFE -I /usr/local/include/pthread/linuxthreads
```

```
-L/usr/local/lib -lthread -lgcc_r" --disable-shared
--with-embedded-server --with-innodb
```

- QNX Neutrino 6.2.1 i386 / [gcc 2.95.3](#)qnx-nto 20010315:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

以下のバイナリは他のユーザーのご好意により MySQL AB に提供されたサードパーティのシステムにビルドされたものです。これらは単なる興味から提出します。MySQL AB はこのシステムを完全に管理していないため、それらにビルドされたバイナリのサポートには限界があります。

- SCO Unix 3.2v5.0.7 i386 / [gcc 2.95.3](#):

```
CFLAGS="-O3 -mpentium" LDFLAGS=-static CXX=gcc CXXFLAGS="-O3 -mpentium
-felide-constructors" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-safe-client
--disable-shared
```

- SCO UnixWare 7.1.4 i386 / [CC 3.2](#):

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-safe-client
--disable-shared --with-readline
```

- SCO OpenServer 6.0.0 i386 / [CC 3.2](#):

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-safe-client
--disable-shared --with-readline
```

- Compaq Tru64 OSF/1 V5.1 732 アルファ / [cc/cxx](#) (Compaq C V6.3-029i / DIGITAL C++ V6.1-027):

```
CC="cc -pthread" CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline
speed -speculate all" CXX="cxx -pthread" CXXFLAGS="-O4 -ansi_alias
-fast -inline speed -speculate all -noexceptions -nortti" ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile
--with-named-thread-libs="-lpthread -lmach -lexc -lc" --disable-shared
--with-mysqld-ldflags=-all-static
```

- SGI Irix 6.5 IP32 / [gcc 3.0.1](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- FreeBSD/sparc64 5.0 / [gcc 3.2.1](#):

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-innodb
```

以下のコンパイル オプションは MySQL AB が以前提供したバイナリのパッケージに使用されていたものです。これらのバイナリはもはや現在使用されていませんが、このコンパイル オプションは参考のためにのみこのリストに載せています。

- Linux 2.2.xx SPARC / [egcs 1.1.2](#):

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
```

```
--enable-local-infile --enable-assembler --disable-shared
```

- Linux 2.2.x with x86 / [gcc 2.95.2](#):

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro  
-felide-constructors -fno-exceptions -fno-rtti" ./configure  
--prefix=/usr/local/mysql --enable-assembler  
--with-mysqld-ldflags=-all-static --disable-shared  
--with-extra-charsets=complex
```

- SunOS 4.1.4 2 sun4c / [gcc 2.7.2.1](#):

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors" ./configure  
--prefix=/usr/local/mysql --disable-shared --with-extra-charsets=complex  
--enable-assembler
```

- SunOS 5.5.1 (またはそれ以降) sun4u / [egcs 1.0.3a](#) or 2.90.27 or [gcc 2.95.2](#) およびそれ以降:

```
CC=gcc CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors  
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql  
--with-low-memory --with-extra-charsets=complex --enable-assembler
```

- SunOS 5.6 i86pc / [gcc 2.8.1](#):

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql  
--with-low-memory --with-extra-charsets=complex
```

- BSDI BSD/OS 3.1 i386 / [gcc 2.7.2.1](#):

```
CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex
```

- BSDI BSD/OS 2.1 i386 / [gcc 2.7.2](#):

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex
```

- AIX 4.2 / [gcc 2.7.2.2](#):

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql  
--with-extra-charsets=complex
```

MySQL の取得方法

MySQL の現在のバージョン情報およびダウンロードの説明書に関してはダウンロード ページ <http://dev.mysql.com/downloads/> を参照してください。MySQL のダウンロードのミラーサイトの最新の完全な更新リストは、<http://dev.mysql.com/downloads/mirrors.html> を参照してください。そこにはまた MySQL ミラーサイトのなり方および不良ミラーおよび旧式ミラーのレポート方法に関する情報が載せてあります。

弊社の主要なミラーは <http://mirrors.sunsite.dk/mysql/> にあります。

MD5 チェックサムあるいは [GnuPG](#) を用いたパッケージの品質の検証

お客様の仕様に適した MySQL のパッケージをダウンロードしそれをインストールする前に、そのパッケージが元のままで改ざんされていないか確認する必要があります。MySQL AB では品質の確認に 3 つの方法を提供しています。

- MD5 チェックサム
- [GnuPG](#)、GNU Privacy Guard を使用した暗号署名
- RPM パッケージの場合、内蔵 RPM 品質検証メカニズム

以下の項ではこれらのメソッドの使用方法について説明します。

MD5 チェックサムおよび GPG 署名が一致しない場合には、別のミラーサイトからそれぞれのパッケージを再度ダウンロードします。何回繰返してもパッケージの完全性を確認できない場合には、その状況と完全なパッケージ名および使用したダウンロード サイトを明記の上、[<webmaster@mysql.com>](mailto:webmaster@mysql.com) あるいは [<build@mysql.com>](mailto:build@mysql.com) までご連絡ください。バグのレポート システムではダウンロードの問題をレポートしないでください。

MD5 チェックサムの検証

MySQL パッケージをダウンロードしたら、MD5 チェックサムが MySQL のダウンロード ページのチェックサムと同じであることを確認します。各パッケージには以下おコマンドで検証できるそれぞれのチェックサムがあり、その `package_name` はお客様がダウンロードするパッケージの名前です。

```
shell> md5sum package_name
```

例:

```
shell> md5sum mysql-standard-5.1.15-beta-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945 mysql-standard-5.1.15-beta-linux-i686.tar.gz
```

結果おチェックサム (16 桁の数字) が各パッケージの真下のダウンロード ページのチェックサムと一致していることを確認する必要があります。

注:アーカイブ (例えば、`.zip` あるいは `.tar.gz` ファイル) のチェックサムでそのアーカイブの中に含まれるファイルのチェックサムでないことを確認します。

すべてのオペレーティングシステム `md5sum` コマンドをサポートしている訳ではないのでその点ご注意ください。中には、単に `md5` と呼ばれるものもあり、その他は含まれていません。Linux の場合、それは GNU Text Utilities パッケージの一部で、様々なプラットフォームに利用できます。ソースコードは <http://www.gnu.org/software/textutils/> からダウンロードできます。OpenSSL の場合には、代わりにコマンド `openssl md5 package_name` を使用できます。Windows の `md5` コマンドライン ユーティリティは <http://www.fourmilab.ch/md5/> から入手できます。`winMd5Sum` はグラフィカルな MD5 チェック ツールで <http://www.nullriver.com/index/products/winmd5sum> にあります。

GnuPG を使用した署名確認

パッケージの完全性およびその典拠を確認する別のメソッドとして暗号署名を使用します。このメソッドは MD5 チェックサムを使用するより信頼できますが、手間がかかります。

MySQL AB では、MySQL ダウンロード パッケージを GnuPG (GNU プライバシーガード) で署名しています。GnuPG はオープン ソースで周知の Phil Mimmerman 氏の周知の Pretty Good Privacy (PGP) の代替品です。GnuPG の詳細およびその取得並びにシステムへのインストール方法については <http://www.gnupg.org/> を参照してください。殆どの Linux の配布にはデフォルトで GnuPG がインストールされています。GnuPG の詳細は、<http://www.openpgp.org/> を参照してください。

特定のパッケージの署名を証明するには、最初に MySQL AB's public GPG ビルド キーを取得する必要があります。それは <http://www.keyserver.net/> で入手できます。取得するキーは `build@mysql.com` です。また以下のテキストから直接そのキーを切り取り/貼り付けできます。

```
Key ID:
pub 1024D/5072E1F5 2003-02-03
  MySQL Package signing key (www.mysql.com) <build@mysql.com>
Fingerprint: A4A9 4068 76FC BD3C 4567 70C8 8C71 8D3B 5072 E1F5

Public Key (ASCII-armored):

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGIBD4+owwRBAC14GifUfCyEDSlePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPIUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPKbDck96+OmSLN9brZ
fw2vOUgCmYv2Hw0yDHuvYIQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqQxRznNCRcRxAuAuVztHRCeAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxrR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QUEXM6FSbi0LLiZciNIYsafwAPEOMDKpMqAK6lyisNtPvaLd8IH0bPAnWqcyefep
rv0sxxqUEMcM3o7wggfN83POkDasDbs3pjwPhxvhz6//62zQJ7QT7XITUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkpPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQUcPj6jDAUJCWYBgaULBwoDBAMVAwIDFglBAheAAAoJElxjTtQ
cuH1cY4AnilUwTXn8MatQOiG0a/bPxrK/gCAJ4oinSNZRYTnblChwFaazt7PF3q
zlhMBBMRAGAMBQI+PqPRBYMJZgC7AAoJEEIQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/peCUmippiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRAGAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQlpEaolhpb6BozKI8p6eaabzF5MIJH58pAKCu/ROofk8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfbMdY4wslPnjH9rF4N2qfWsEN/
```

```
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1Ql4YFteR47gCvtgb6O1JHffOo2HfLmRDRi
Rjd1DTCHqeyX7CHhcgj/dNRIW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkbf4fm
Le11EzlaCa4PqARXQZc4dhSinMt6K3X4BrRsKtfozBu74F47D8llbf5vSYHbuE5p
/1oIDznkg/p8kVW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUib/F5GwADBQf+Lwqq
a8CGrRfsOAJxim63CHf5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFwvFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOW
l39EcyzgsIzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTPtI0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+iWJ0mghkKHt92
6s/ymfdf5HkdQ1cyvsz5tryVl3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZ
Whe70YGNPw1yjWJT1lhMBBgRAgAMBQI+PqMdBQkJZgGAAAOJElxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFApmr1+0uUq/SlSgQ==
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

ビルド キーを個人の公開 GPG キーリングにインポートするには、`gpg --import` を使用します。例えば、キーを `mysql_pubkey.asc` ファイルに保存した場合、インポート コマンドは以下のようになります。

```
shell> gpg --import mysql_pubkey.asc
```

ダウンロードして公開ビルド キーをインポートした後、所望の MySQL パッケージと同じくダウンロード ページで入手できるそれに相当する署名をダウンロードします。署名ファイルは配布ファイルの `.asc` 拡張と同じ名前です。例えば、

配布ファイル	<code>mysql-standard-5.1.15-beta-linux-i686.tar.gz</code>
署名ファイル	<code>mysql-standard-5.1.15-beta-linux-i686.tar.gz.asc</code>

両方のファイルが同じディレクトリにあることを確認し、次に以下のコマンドを実行して配布ファイルの署名を証明します。

```
shell> gpg --verify package_name.asc
```

例:

```
shell> gpg --verify mysql-standard-5.1.15-beta-linux-i686.tar.gz.asc
gpg: Signature made Tue 12 Jul 2005 23:35:41 EST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

Good signature メッセージは署名が証明されたことを意味します。`insecure memory` が表示された場合無視して構いません。

公開キーの取扱いに関する情報は GPG の資料を参照してください。

RPMを使用した署名の確認

RPM には、個別の署名はありません。RPM パッケージには内蔵の GPG 署名および MD5 チェックサムがあります。以下のコマンドを実行してパッケージを検証できます。

```
shell> rpm --checksig package_name.rpm
```

例:

```
shell> rpm --checksig MySQL-server-5.1.15-beta-0.i386.rpm
MySQL-server-5.1.15-beta-0.i386.rpm: md5 gpg OK
```

注: RPM 4.1 を使用していてそれが次を表示した場合 (**GPG) NOT OK (MISSING KEYS:GPG#5072e1f5)**、MySQL 公開キー作成のキーをお客様の GPG キーリングにインポートしている場合でも、このキーを最初に RPM キーリングにインポートする必要があります。RPM 4.1 はこの段階でお客様個人の GPG キーリング (あるいは GPG そのもの) を使用しません。むしろ、それはシステム全体のアプリケーションでユーザーの GPG 公開キーリングはユーザー特定のものですのでそれは自身のキーリングとして残ります。MySQL 公開キーを RPM キーリングにインポートするには、以下の説明に従って最初にキーリングを取得します。**GnuPG を使用した署名確認**次に `rpm --import` を使用してそのキーをインポートします。例えば、公開キーをファイル名 `mysql_pubkey.asc` に保存してある場合、以下おコマンドを使用して公開キーをインポートします。

```
shell> rpm --import mysql_pubkey.asc
```

MySQL 公開キーを取得する必要がある場合には、**GnuPG を使用した署名確認** を参照してください。

インストールのレイアウト

この項では MySQL AB が供給したバイナリあるいはソースの配布のインストールにより作成されたディレクトリのデフォルトのレイアウトについて説明します。他のベンダーから供給された配布は以下に示すレイアウトと異なるレイアウトを使用する場合があります。

Windows 5.1 で動作している MySQL のデフォルトのインストール ディレクトリは `C:\Program Files\MySQL\MySQL Server 5.1` になります。(Windows のユーザーの中には公式にデフォルトとして使用されていた `C:\mysql` にインストールを希望される場合があります。しかし、サブディレクトリのレイアウトはそのまま変わりません。) インストール ディレクトリには以下のサブディレクトリがあります。

ディレクトリ	ディレクトリの中身
<code>bin</code>	クライアント プログラムと <code>mysqld</code> サーバ
<code>data</code>	ログ ファイル、データベース
<code>Docs</code>	CHM フォーマットのマニュアル
<code>examples</code>	プログラムおよびスクリプト例
<code>include</code>	(ヘッダー) ファイルを含む
<code>lib</code>	ライブラリ
<code>scripts</code>	ユーティリティ スクリプト
<code>share</code>	エラーメッセージ ファイル

MySQL AB の Linux RPM 配布で作成されたインストールは以下のシステム ディレクトリのファイルに格納されます。

ディレクトリ	ディレクトリの中身
<code>/usr/bin</code>	クライアント プログラムおよびスクリプト
<code>/usr/sbin</code>	<code>mysqld</code> サーバ
<code>/var/lib/mysql</code>	ログ ファイル、データベース
<code>/usr/share/info</code>	情報フォーマットのマニュアル
<code>/usr/share/man</code>	Unix マニュアル ページ
<code>/usr/include/mysql</code>	(ヘッダー) ファイルを含む
<code>/usr/lib/mysql</code>	ライブラリ
<code>/usr/share/mysql</code>	エラー メッセージと文字列セット ファイル
<code>/usr/share/sql-bench</code>	ベンチマーク

Unix では、`tar` ファイルのバイナリ配布が選択したインストールのロケーション (一般的には `/usr/local/mysql`) で解凍するとインストールされそのロケーションで以下のディレクトリが作成されます。

ディレクトリ	ディレクトリの中身
<code>bin</code>	クライアント プログラムおよび <code>mysqld</code> サーバ
<code>data</code>	ログ ファイル、データベース
<code>docs</code>	情報フォーマットのマニュアル
<code>man</code>	Unix マニュアル ページ
<code>include</code>	(ヘッダー) ファイルを含む
<code>lib</code>	ライブラリ
<code>scripts</code>	<code>mysql_install_db</code>
<code>share/mysql</code>	エラーメッセージ ファイル
<code>sql-bench</code>	ベンチマーク

ソースの配布がそれを設定してコンパイルした後にインストールされます。デフォルトでは、インストール手順によってファイルを `/usr/local` の以下のサブディレクトリにインストールします。

ディレクトリ	ディレクトリの中身
bin	クライアントプログラムおよびスクリプト
include/mysql	(ヘッダー) ファイルを含む
Docs	Info のマニュアル、CHM フォーマット
man	Unix マニュアル ページ
lib/mysql	ライブラリ
libexec	mysqld サーバ
share/mysql	エラーメッセージ ファイル
sql-bench	ベンチマークおよび crash-me テスト
var	データベースおよびログ ファイル

インストールしたディレクトリの中では、ソースのインストールのレイアウトが以下の点でバイナリのインストールと異なります。

- [mysqld](#) サーバは [bin](#) ディレクトリではなく [libexec](#) ディレクトリにインストールされます。
- データ ディレクトリは [data](#) ではなく [var](#) です。
- [mysql_install_db](#) は [scripts](#) ディレクトリではなく [bin](#) ディレクトリにインストールされます。
- ヘッダーファイルとライブラリのディレクトリは [include](#) および [lib](#) ではなく [include/mysql](#) および [lib/mysql](#)。

ソース配布の一番上の ディレクトリにある [scripts/make_binary_distribution](#) スクリプトを実行してコンパイルされたソースの配布でお客様ご自身のバイナリのインストールを作成できます。

バイナリの配布を使用した標準 MySQL のインストール

次の数項ではそれぞれのプラットフォームのネイティブなパッケージ オーマットを使用したパッケージで提供しているプラットフォーム上の MySQL のインストールについて説明します。(これはまた「[binary install.](#)」の実行として知られています。)しかし、MySQL のバイナリの配布はその他多くのプラットフォームでも利用できます。すべてのプラットフォームに適用できるこれらのパッケージの包括的なインストールの説明は [他の Unix 系システムへの MySQL のインストール](#) を参照してください。

利用できるその他のバイナリの配布およびその取得方法に関する詳細は [一般的なインストールの問題](#) を参照してください。

Windows に MySQL をインストールする

バージョン 3.21 以来 MySQL AB は、MySQL の Windows 版 を提供しており、現在 MySQL の日々のダウンロードでかなりの部分のパーセントを占めています。この項では MySQL の Windows へのインストールのプロセスについて説明します。

注:MySQL を MySQL 4.15 以前のバージョンからアップグレードする場合、最初に以下の手順を踏む必要があります。[Windows を使用した MySQL をアップグレードする](#)。

- 9x、Me、NT、2000、XP、あるいは Windows Server 2003 などの 32 ビットの Windows オペレーティングシステムの場合。

Windows NT-ベースのオペレーティングシステム (NT、2000、XP、2003) の場合サービスで MySQL サーバをご利用頂けます。Windows NT-ベースのオペレーティングシステムの使用を強くお勧めします。[Windows のサービスとして MySQL を起動する](#) 参照。

一般的には Windows に MySQL をインストールするには管理者権限のアカウントを使用してインストールする必要があります。管理者権限が無い場合、[PATH](#) 環境変数の編集あるいは [サービス管理マネージャにアクセスする](#)などの操作の場合に問題に遭遇する場合があります。。

- TCP/IP のプロトコルのサポート。

- お客様の仕様に基いた解凍、インストール、およびデータベースの作成に十分なスペース（一般的には最低でも 200 メガバイトを推奨します。）

お客様の MySQL の利用の仕方によって他の条件が必要になる場合もあります。

- ODBC を用いて MySQL サーバに接続する場合には、コネクタ/ODBC ドライバが必要です。[24章MySQL コネクタ 参照](#)。
- 4GB 以上のテーブルが必要な場合、NTFS あるいは最新のファイルシステムに MySQL をインストールします。テーブルを作成する時には `MAX_ROWS` あるいは `AVG_ROW_LENGTH` の使用を忘れないでください。[CREATE TABLE 構文 参照](#)。

Windows 用の MySQL にはいくつかの配布フォーマットがあります。

- バイナリの配布が利用できるのでサーバを直ぐに起動できるよう必要なすべてをインストールできる設定プログラムを利用できます。別のバイナリの配布フォーマットではインストールの場所に単に解凍してインストールし、お客様ご自身で設定できるアーカイブがあります。詳細に関しては [インストール用パッケージの選択](#) を参照して下さい。
- ソースの配布には Visual Studio のコンパイラ システムを使用した実行ファイル作成用のすべてのコードおよびサポートファイルが含まれています。

一般的には、インストーラを含むバイナリの配布が必要です。他のソリューションを使用するよりは簡単で、MySQL を設定して起動するための特別なツールは必要ありません。MySQL の Windows インストーラを GUI 設定ウィザードと一緒に用いると MySQL を自動的にインストールし、オプション ファイルを作成してサーバを起動し、デフォルトのユーザーアカウントをセキュアにします。

次項ではバイナリの配布を用いた Windows への MySQL のインストールについて説明します。インストーラが実装されていないインストール パッケージを使用する際は、以下の説明のプロシージャに従います。[非インストール Zip アーカイブからのインストール](#)。ソースの配布を使用したインストールは、[Windows にソースから MySQL をインストールする](#) を参照してください。

Windows 用 MySQL の配布は、<http://dev.mysql.com/downloads/> からダウンロードできます。[MySQL の取得方法 参照](#)。

インストール用パッケージの選択

Windows 用 MySQL 5.1 のインストールには、3 種類のインストール パッケージがあります。

- 基本パッケージ:このパッケージは [mysql-essential-5.1.15-beta-win32.msi](#) に類似したファイル名を持ち、設定ウィザードを含む Windows への MySQL インストールの最低限のファイルを含んでいます。このパッケージには埋め込みサーバおよびベンチマーク スイートなどのオプションのコンポーネントは含まれていません。
- 完全パッケージ:このパッケージは [mysql-5.1.15-beta-win32.zip](#) の類似のファイル名を持ち、設定ウィザードを含む MySQL の Windows への完全なインストールを実現します。このパッケージは埋め込みサーバおよびベンチマーク スイートなどのオプションのコンポーネントを含んでいます。
- インストールなしのアーカイブ:このパッケージは [mysql-noinstall-5.1.15-beta-win32.zip](#) 類似のファイル名を持ち、設定ウィザードを除く完全なインストールパッケージと同様のすべてのファイルを含んでいます。このパッケージには自動インストーラが含まれていないので、手動でインストールして設定する必要があります。

殆どのユーザーには基本パッケージをお勧めします。それは `.msi` ファイルにあり、Windows Installer と一緒に使用します。完全およびインストールなしの配布は Zip アーカイブに格納されています。それらを使用するには、`.zip` ファイルを解凍するツールが必要です。

インストールのパッケージの選択によってインストールのプロセスが変わります。基本パッケージあるいは完全パッケージのいずれかを選択する場合、[自動インストーラで MySQL をインストールする](#) を参照します。インストールしないアーカイブを選択する際は、[非インストール Zip アーカイブからのインストール](#) を参照します。

自動インストーラで MySQL をインストールする

初めて MySQL を使用するユーザーは MySQL インストール レイション ウィザードと MySQL 設定ウィザードを使用して MySQL を Windows にインストールします。これらは MySQL を初めて使用するユーザーが MySQL を使用して直ぐ起動できるように MySQL をインストールして設定できるようになっています。

MySQL インストール ウィザード および MySQL 設定ウィザードは基本および完全インストール パッケージで利用できます。それらを最も標準的な MySQL のインストールにお勧めしています。その例外としては、複数の MySQL インスタンスを 1 台のサーバホストにインストールするあるいはサーバ設定の完全な管理を希望する熟練したユーザーです。

MySQL インストール ウィザードを使用する

インストール ウィザードへようこそ

MySQL インストール ウィザードは Microsoft 社の Windows 用に最新のインストール テクノロジーを使用した MySQL サーバのインストーラです。MySQL インストール ウィザードと MySQL 設定ウィザードと一緒に用いることによって、ユーザーが MySQL サーバをインストール・設定したら直ぐに使用できます。

MySQL インストール ウィザードは MySQL サーバディストリビューション用の標準にインストーラで、バージョン 4.1.5 およびそれ以降があります。MySQL の以前のバージョンを使用しているユーザーはコンピュータをシャットダウンし、既存の MySQL を手動でアンインストールしてから MySQL を MySQL インストール ウィザードでインストールする必要があります。以前のバージョンからのアップグレードに関する詳細は、[インストール ウィザードによる MySQL のアップグレード](#) を参照してください。

Microsoft 社は Microsoft Windows Installer (MSI) の改善版を Windows の最新機種に使用しています。MSI は Windows 2000、Windows XP、および Windows Server 2003 のアプリケーション インストールの事実上の標準です。MySQL インストール ウィザードはこのテクノロジーを活用することでスムーズでさらに柔軟なインストール プロセスを提供しています。

Microsoft Windows インストール エンジン は Windows XP のリリース時に更新されています。旧バージョンの Windows を使用ユーザーは [this Microsoft Knowledge Base article](#) をご覧頂ければ最新の Windows インストーラ エンジンへのアップグレード情報を得ることが出来ます。

さらに、Microsoft 社は最近 WiX (Windows Installer XML) ツールキットをリリースしました。これが Microsoft 社の最初のオープン ソース プロジェクトです。弊社では WiX がオープン ソース で完全な Windows インストール プロセスをスクリプトを使用して柔軟にできるため Wix に切り換えました。

MySQL インストール ウィザードをお客様のようなユーザーのサポートとフィードバックによる改善します。。お客様にとって重要な機能が MySQL インストール ウィザードに欠けている場合、あるいはバグを見つけた場合には、[質問またはバグの報告](#) の説明に従ってそれを弊社のバグ データベースまでご連絡をお願いします。

MySQL インストール ウィザードのダウンロードおよび起動

MySQL のインストール パッケージは <http://dev.mysql.com/downloads/> からダウンロードできます。ダウンロードするパッケージが Zip のアーカイブにある場合には、最初にアーカイブを取り出す必要があります。

ウィザードと起動するプロセスはダウンロードしたインストール パッケージの中身に因ります。[setup.exe](#) ファイルがある場合、それをダブル クリックしてインストール プロセスを開始します。[.msi](#) ファイルがある場合、それをダブル クリックしてインストール プロセスを開始します。

インストールの種類を選択

インストールの方法は 3 種類あります。標準、完全、およびカスタム の 3 種類です。

標準 のインストールでは MySQL サーバ、[mysql](#) コマンドライン クライアント、およびコマンドライン ユーティリティをインストールします。コマンドライン クライアントおよびユーティリティには [mysqldump](#)、[myisamchk](#)、および MySQL サーバを管理するいくつかのツールが含まれています。

完全 なインストールはインストール パッケージに含まれるすべてのコンポーネントをインストールします。完全なインストールパッケージには埋め込みサーバ ライブラリ、ベンチマーク スイート、サポート スクリプト、および説明書が含まれています。

カスタム のインストールではインストールするパッケージおよび使用するインストール パスなどを選択できます。詳細およびカスタムのインストールを実施するには、[カスタムのインストール ダイアログ](#) を参照してください。

標準および完全のインストールを選択して [Next](#) ボタンをクリックすると、選択の確認画面が表示されインストールを開始します。カスタムのインストールを選択して [Next](#) ボタンをクリックすると、[カスタムのインストール ダイアログ](#) で説明したカスタムのインストール ダイアログが表示されます。

カスタムのインストール ダイアログ

MySQL インストール ウィザードに含まれるインストールのパスあるいは特定のコンポーネントを変更する場合には、カスタムインストールを選択します。

カスタム インストール ダイアログの左側のツリー表示に利用できるすべてのコンポーネントが表示されます。インストールしなかったコンポーネントには赤の X アイコン、インストールしたコンポーネントには灰色のアイコンが付きまします。インストールするコンポーネントを変更するには、そのコンポーネントのアイコンをクリックし、表示されているドロップダウンリストからインストールするアイコンを選択します。

デフォルトのインストール パスを **Change...** ボタンをクリックして希望するインストールするパスに変更します。

インストールするコンポーネントとインストール パスを選択したら、**Next** ボタンをクリックすると選択の確認のダイアログが表示されます。

選択確認のダイアログ

インストールの種類とオプションのコンポーネントを選択すると、選択確認のダイアログが表示されます。選択を確認するための選択したインストールの種類とインストールのパスがダイアログに表示されます。

選択した設定に間違いがない場合、**Install** ボタンをクリックすると MySQL のインストールが開始されます。設定を変更するには、**Back** ボタンをクリックします。MySQL をインストールしないで MySQL インストール ウィザードを終了するには、**Cancel** ボタンをクリックします。

インストールが完了すると、MySQL のウェブサイトに登録するかどうか問われます。登録することによって forums.mysql.com の MySQL フォーラムに投稿したり、バグを bugs.mysql.com にレポートしたり、ニュースレターを購読できます。インストーラの最後の画面ではインストールの概要を表示し、MySQL 設定ウィザードの起動オプションを提供します。そこで設定ファイルを作成し、MySQL のサービスをインストールし、セキュリティを設定します。

MySQL インストール ウィザードに変更

Install ボタンをクリックすると、MySQL インストール ウィザードがインストールを開始し、次項以降で説明する変更をお客様のシステムに加えます。

レジストリの変更

MySQL インストール ウィザードは標準的なインストールの状況で、`HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB` にある Windows レジストリ キーを作成します。

MySQL インストール ウィザードは [MySQL Server 5.1](#) のようなインストールされる主なサーバのバージョン名でキーを作成します。それには 2 つの文字列の値、**Location** および **Version** が含まれます。**Location** 文字列にはインストールディレクトリへのパスが含まれます。デフォルトのインストールでは `C:\Program Files\MySQL\MySQL Server 5.1\` が含まれます。**Version** 文字列にはリリース番号が含まれます。例えば、MySQL Server 5.1.15-beta のインストールでは、そのキーには `5.1.15-beta` の値が含まれます。

これらのレジストリ キーは外部のツールに MySQL サーバのインストール場所を認識させるので、MySQL server のインストール パスを決定するためのハードディスクの完全スキャンをしなくて済みます。レジストリ キーはサーバの起動には必要ありません。MySQL を `noinstall` Zip アーカイブを使用してインストールすると、レジストリ キーは作成されません。

スタート メニューに戻る

MySQL インストール ウィザードが一般的な MySQL メニューの下にインストールした MySQL の主なバージョンの名前で Windows の **スタート** メニューに新たにエントリを作成します。例えば、MySQL 5.1 をインストールすると、MySQL インストール ウィザードが MySQL サーバ 5.1 の欄を **スタート** メニューに作成します。

以下のエントリが **スタート** メニュー欄に作成されます。

- MySQL コマンドライン クライアント:これは `mysql` コマンドライン クライアントへのショートカットで `root` ユーザーとして接続が設定されます。このショートカットは接続すると `root` ユーザーパスワード聞いてきます。
- MySQL サーバインスタンス設定ウィザード:これは MySQL 設定ウィザードへのショートカットです。このショートカットで新たにインストールしたサーバ、あるいは既存のサーバを再設定します。

- MySQL の説明資料:これは MySQL サーバのインストール ディレクトリにローカルで保存された MySQL サーバの説明資料へのリンクです。このオプションは MySQL サーバを基本インストール パッケージでインストールした場合には利用できません。

ファイルシステムへの変更

MySQL インストール ウィザードはデフォルトで MySQL 5.1 サーバを [C:\Program Files\MySQL\MySQL Server 5.1](#) にインストールします。そこでは [Program Files](#) はお客様のシステムのアプリケーションのデフォルトのロケーションで、[5.1](#) は MySQL サーバの中心となるバージョンです。これが MySQL サーバに推奨しているロケーションで、以前のデフォルトのロケーション [C:\mysql](#) を置き換えます。

デフォルトではすべての MySQL アプリケーションは共通のディレクトリ [C:\Program Files\MySQL](#) に保持され、[Program Files](#) は 客様の Windows にインストールした際のアプリケーション用デフォルトのロケーションです。開発マシンへの標準的なインストールは以下のようになります。

```
C:\Program Files\MySQL\MySQL Server 5.1
C:\Program Files\MySQL\MySQL Administrator 1.0
C:\Program Files\MySQL\MySQL Query Browser 1.0
```

これにより MySQL アプリケーションの特定にシステムでの管理と維持が用意になります。

インストール ウィザードによる MySQL のアップグレード

MySQL インストール ウィザードは MSI の機能を使用してサーバのアップグレードを自動的に行います。これにより以前のバージョンを手動でアンインストールすることなく新しいリリースをインストールできます。インストーラーは新しいバージョンをインストールする前に自動的にシャットダウンして以前のバージョンを削除します。

自動アップグレードは同じ大きな枝番と小さな枝番を持つバージョン番号でのインストールの際にのみ利用できます。例えば、MySQL 4.1.5 から MySQL 4.1.6 に自動的にアップグレードできますが、MySQL 5.0 から MySQL 5.1 へのアップグレードはできません。

[Windows を使用した MySQL をアップグレードする](#) 参照。

設定ウィザードを使用する

設定ウィザードへようこそ

MySQL 設定ウィザードは Windows 環境でサーバの設定プロセスを自動化します。MySQL 設定ウィザード一連の質問をお客様に問いかけた上ではカスタムの [my.ini](#) ファイルを作成し、次にお客様の回答をテンプレートに反映してインストールに使用する [my.ini](#) ファイルを作成します。

MySQL 設定ウィザードは MySQL 5.1 サーバに含まれており、現在は Windows のユーザーのみ利用できます。

MySQL 設定ウィザードの大部分は MySQL AB に多くのユーザーがここ数年もたらしてくれたフィードバックに結果です。それでも尚、お客様にとって重要な機能が欠けている場合には、[質問またはバグの報告](#) で説明した手順に従ってそれをバグ データベースにレポートお願いします。

MySQL 設定ウィザード起動

MySQL 設定ウィザードは一般的には MySQL インストール ウィザードがインストールされている場合 MySQL インストール ウィザードから起動します。また、MySQL 設定ウィザードを Windows の [スタート](#) メニューの MySQL 欄にある MySQL Server Instance Config Wizard のエントリをクリックすると起動できます。

あるいは、MySQL インストールの [bin](#) ディレクトリをナビゲートして [MySQLInstanceConfig.exe](#) ファイルのディレクトリから直接起動できます。

メンテナンス オプションの選択

MySQL 設定ウィザードが既存の [my.ini](#) ファイルを検知した場合、お客様の既存のサーバを再設定するか、あるいは [my.ini](#) ファイルを削除してサーバインスタンスを削除し MySQL のサービスを停止して削除します。

既存のサーバを再設定するには、再設定インスタンスオプションを選択して [Next](#) ボタンをクリックします。既存の [my.ini](#) ファイル名が [mytimestamp.ini.bak](#) に変わり、その [timestamp](#) が既存の [my.ini](#) ファイルが作成された日時で

す。既存のサーバ インスタンスを削除するには、インスタンスの削除オプションを選択し **Next** ボタンをクリックします。

インスタンスの削除オプションを選択すると確認ウィンドウが表示されます。**Execute** ボタンをクリックします。MySQL 設定ウィザードが停止して MySQL サービスを削除し、次に **my.ini** ファイルを削除します。サーバのインストールおよびその **data** フォルダは削除されません。

再設定インスタンスオプションを選択すると、**設定タイプ** のダイアログが表示されるので、そこで設定するインストールのタイプを選択します。

設定タイプの選択

MySQL 設定ウィザードを新規の MySQL インストールに起動する、あるいは 再設定インスタンスオプションを既存のインストールに選択すると、**設定タイプ** のダイアログが表示されます。

選定タイプは 2 種類あります。詳細設定 および 標準設定 の 2 種類です。標準設定オプションは新規のユーザーがサーバの設定を変更しないで MySQL を直ぐに起動する場合に適しています。詳細設定オプションは熟練したユーザー用でサーバの設定を細かく設定する場合に適しています。

MySQL を初めて使用されるユーザーで単一ユーザーの開発マシンとしての設定が必要な場合には、標準設定で十分です。標準設定オプションを選択するとサービス オプションおよびセキュリティ オプションを除くすべての設定オプションを MySQL 設定ウィザードが自動的に設定します。

標準設定は既存の MySQL がインストールされている場合にシステムに互換がないオプションを設定します。システムに設定するインストール以外に既存の MySQL をインストールしている場合、詳細設定オプションをお勧めします。

標準設定 を完了するには、それぞれ **サービス オプション ダイアログ** および **セキュリティ オプション ダイアログ** のサービス オプション並びにセキュリティ オプション の項を参照してください。

サーバタイプのダイアログ

選択可能な異なる 3 種類のサーバタイプがあります。選択するサーバの種類によって MySQL 設定ウィザードのメモリ、ディスク、および使用しているプロセッサに関する決定に影響を与えます。

- 開発用マシン:MySQL を個人の使用目的のみに使用する場合には、一般的なデスクトップのワークステーション用としてこのオプションを選択します。他の多くのデスクトップのアプリケーションが実行されているものと想定します。MySQL サーバは最低限のシステム リソース向けに設定されます。
- サーバ マシン:MySQL サーバが FTP、Eメール、およびウェブ サービスなど他のサーバアプリケーションと一緒に動作しているサーバ マシンにはこのオプションを選択します。MySQL サーバはシステム リソースの平均的な部分を使用するものとして設定されます。
- 専用 MySQL サーバ マシン:MySQL サーバのみを運用するサーバ マシンにはこのオプションを選択します。他のアプリケーションを実行していないことを想定した設定です。MySQL サーバは利用できるすべてのシステム リソースを使用する設定になります。

注記

前もって設定された設定を選択すると、お客様の **my.cnf** あるいは **my.ini** の様々なオプションの値や設定がそれぞれ変更されます。ですから、参照マニュアルで説明したデフォルトの値やオプションは設定ウィザードの実行中に作成されたオプションや値と異なる場合があります。

データベースの使用ダイアログ

データベース利用 ダイアログは MySQL のテーブルを作成する際に使用するストレージ エンジンを表示します。選択したオプションによって **InnoDB** ストレージ エンジンを利用できるか、何パーセントのサーバ リソースが **InnoDB** に利用できるかを決定します。

- 多機能データベース:このオプションは **InnoDB** および **MyISAM** ストレージ エンジン両方を有効にしリソースを両方に等分します。このオプションは両方のストレージ エンジンを定期的に使用するユーザーにお勧めします。

- ・ **トランザクション データベースのみ**:このオプションは **InnoDB** および **MyISAM** ストレージ エンジンの両方を有効にしますが、殆どのサーバリソースは **InnoDB** ストレージ エンジンが使用します。このオプションは **InnoDB** を中心に使用し **MyISAM** は最低限使用するユーザーに適しています。
- ・ **非-トランザクション データベースのみ**:このオプションは **InnoDB** ストレージ エンジン完全に無効にしすべてのサーバリソースは **MyISAM** ストレージ エンジンが使用します。このオプションは **InnoDB** を使用しないユーザーにお勧めです。

InnoDB テーブルスペース ダイアログ

ユーザーの中には **InnoDB** テーブルスペース ファイルを MySQL サーバのデータディレクトリではなく異なるロケーションに配置することを希望するユーザーもいます。テーブルスペース ファイルの個別のロケーションへの配置はお客様のシステムが高容量あるいは RAID ストレージ エンジンのような高パフォーマンスのストレージ デバイスを利用できる場合に適しています。

InnoDB テーブルスペース ファイルのロケーションを変更するには、ドライブ レターのドロップダウン リストから新しいドライブを選択し、パスのドロップダウン リストから新しいパスを選択します。カスタムのパスを作成するには、 ボタンをクリックします。

既存のサーバの設定を変更するには、パスを変更する前に **Modify** ボタンをクリックします。この状況ではサーバを起動する前に既存のテーブルスペース ファイルを新しいロケーションに手動で移動する必要があります。

同時接続ダイアログ

サーバのリソースが不足しないようにするために、確立できる MySQL サーバへの同時接続数を制限することが重要です。**同時接続**ダイアログでサーバの設定を任意に選択し、同時接続数の制限を設定できます。同接続の制限を手動で設定することも出来ます。

- ・ **デシジョン サポート (DSS)/OLAP**:サーバが多数の同時接続を必要としない場合にこのオプションを選択します。接続の最大数を 100 に設定し、平均の 20 の同時接続を想定します。
- ・ **オンライン トランザクション プロセッシング (OLTP)**:サーバが多数の同時接続を必要とする場合にこのオプションを選択します。同時接続の最大数を 500 に設定します。
- ・ **手動の設定**:サーバへの最大の同時接続数を手動で設定する場合にこのオプションを設定します。表示されたドロップダウン ボックスから任意の接続数を選択するか、任意の接続数がない場合は最大の接続数をドロップダウン ボックスに入力します。

ネットワークおよび厳格なモード オプション ダイアログ

ネットワーク オプション ダイアログを使用して TCP/IP ネットワークを有効/無効にし、MySQL サーバの接続に使用するポート番号を設定します。

TCP/IP ネットワークはデフォルトで有効になっています。TCP/IP ネットワークを無効にするには、TCP/IP ネットワーク オプションの隣にあるボックスのチェックを外します。

デフォルトでは Port 3306 を使用しています。MySQL の接続に使用するポートを変更するには、ドロップダウン ボックスから新しいポート番号を選択するか、ドロップダウン ボックスに直接ポート番号を入力します。ポート番号を選択すると、そのポート番号の選択を確認するプロンプトが表示されます。

サーバを SQL モードに設定して厳格モードを有効あるいは無効にします。厳格モード（デフォルト）に設定する MySQL が他のデータベース管理システムと同様の振る舞いをします。アプリケーションを MySQL の旧式の「許容」な振る舞いで動作させるには、それらのアプリケーションを使用するかあるいは厳格モードを無効にします。厳格モードの詳細については、**SQL モード** を参照してください。

文字セット ダイアログ

MySQL サーバは複数の文字セットをサポートしています。オーバーライドを除いてすべてのテーブル、カラム、およびデータベースに適用されるデフォルトのサーバ文字セットに設定できます。MySQL サーバのデフォルトの文字セットを変更するには**文字セット** のダイアログを使用します。

- ・ **標準の文字セット:latin1** をデフォルトのサーバ文字セットとして使用するにはこのオプションを選択します。**latin1** は英語および多くの西部ヨーロッパ言語で使用されています。

- 多言語のベスト サポート: `utf8` をデフォルトのサーバ文字セットとして使用する場合このオプションを選択します。これはユニコードの文字セットで多くの異なる言語の文字を保持できます。
- 手動選択のデフォルトの文字セット/照合: サーバのデフォルトの文字セットを手動で選択するにはこのオプションを選択します。任意の文字セットを表示されたドロップダウン リストから選択します。

サービス オプション ダイアログ

Windows の NT-ベースのプラットフォームでは、MySQL サーバは Windows のサービスの一貫としてインストールできます。サービスでインストールした場合、MySQL サーバはシステムの起動時に自動的に起動されサービスの不具合時でも Windows を起動すると自動的に再起動します。

MySQL 設定ウィザードがサービス名 `MySQL` で MySQL サーバをデフォルトでインストールします。このサービスのインストールを望まない場合、Windows のサービスとしてインストールする のオプションの隣にあるボックスのチェックを外します。表示されたドロップダウン ボックスから新しいサービス名を選択するか、ドロップダウン ボックスに新しいサービス名を入力するとサービス名を変更できます。

MySQL サーバをサービスとしてインストールしても起動時の自動的な起動を望まない場合、MySQL サーバを自動的に起動する のオプションの隣にあるボックスのチェックを外します。

セキュリティ オプション ダイアログ

`root` パスワードを MySQL サーバに設定することを強くお勧めします。MySQL 設定ウィザードはデフォルトでその設定を要求します。`root` パスワードの設定を望まない場合、セキュリティの設定変更 オプションの隣にあるボックスのチェックを外します。

`root` パスワードを設定するには、任意のパスワードを 新しいルートパスワードおよび確認 のボックスに入力します。既存のサーバを再設定するには、既存の `root` パスワードを現在のルート パスワード ボックスに入力する必要があります。

ネットワーク上で他の人が `root` にログインできないようにするには、ルートはローカル ホストのみからアクセス可能 のオプションの隣にあるボックスにチェックを入れます。これにより `root` アカウントのセキュリティが強化されます。

匿名のユーザーアカウントを作成するには、匿名のアカウントを作成する のオプションの隣にあるボックスにチェックを入れます。匿名のアカウントを作成するとサーバのセキュリティが脆弱になり、ログインおよび許可において困難が伴います。このため、匿名のアカウントはお勧めできません。

選択確認のダイアログ

MySQL 設定ウィザードの最後のダイアログは 確認ダイアログ です。設定プロセスを実行するには、`Execute` ボタンをクリックします。前のダイアログに戻るには、`Back` ボタンをクリックします。MySQL 設定ウィザードをサーバを設定しないで終了するには、`Cancel` ボタンをクリックします。

`Execute` ボタンをクリックすると、MySQL 設定ウィザードが一連のタスクを実行しタスクの実行に応じて進捗状況を画面に表示します。

MySQL 設定ウィザードは MySQL の開発者やエンジニアが用意したテンプレートを使用してお客様の選択に基づいて最初に設定ファイル のオプションを決定します。このテンプレート名は `my-template.ini` でサーバのインストールディレクトリにあります。

MySQL 設定ウィザードは次にこれらのオプションを `my.ini` ファイルに書き込みます。`my.ini` ファイルの最後のロケーションは `Write configuration file` タスクの隣に表示されます。

MySQL サーバのサービスの作成を選択すると、MySQL 設定ウィザードがサービスを作成して開始します。既存のサービスを再設定すると、MySQL 設定ウィザードがサービスを再起動して設定変更に応用します。

`root` パスワードの設定を選択すると、MySQL 設定ウィザードがサーバに接続し、新しい `root` パスワードを設定して選択した他のセキュリティの設定に応用します。

MySQL 設定ウィザードがタスクを完了すると、その概要が表示されます。MySQL 設定ウィザードを終了するには `Finish` ボタンをクリックします。

my.ini ファイルのロケーション

MySQL 設定ウィザードが **my.ini** ファイルを MySQL サーバのインストール ディレクトリに配置します。これにより設定ファイルを特定のサーバインスタンスに関連付けます。

MySQL サーバの **my.ini** ファイルのロケーションの確認に、このファイル名に類似した引数をサービス インストールの一貫として MySQL サーバに渡します。

```
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.1\my.ini"
```

ここでは、**C:\Program Files\MySQL\MySQL Server 5.1** が MySQL サーバへのインストール パスに置き換えられます。**--defaults-file** オプションが起動時に MySQL サーバに設定オプションに指定したファイルを読み込ませます。

my.ini ファイルの編集

my.ini ファイルを変更するには、そのファイルをテキスト エディタで開いて必要な変更を加えます。サーバの設定も <http://www.mysql.com/products/administrator/> ユーティティで変更できます。

mysql および **mysqldump** コマンドラインクライアントなどの MySQL クライアントおよびユーティリティはサーバのインストール ディレクトリにある **my.ini** ファイルを見つけることはできません。クライアントおよびユーティリティのアプリケーションを設定するには、新しい **my.ini** ファイルを **C:\WINDOWS** あるいは **C:\WINNT** ディレクトリ (お客様の Windows のバージョンに適したいずれか) に作成します。

非インストール Zip アーカイブからのインストール

非インストール パッケージからインストールするユーザーは手動で MySQL をインストールする際この項の説明を使用します。MySQL を Zip アーカイブからインストールするには以下の手順に従います。

1. アーカイブを任意のインストール ディレクトリに取り出す
2. 空のファイルを作成する
3. MySQL のサーバタイプを選択する
4. MySQL サーバを起動する
5. デフォルトのユーザーアカウントを確認する

このプロセスはこの後の項で説明します。

インストール アーカイブを取り出す

MySQL を手動でインストールするには、以下の手順に従います。

1. 以前のバージョンからアップグレードする場合にはアップグレードを始める前に **Windows を使用した MySQL をアップグレードする** を参照してください。
2. Windows NT、Windows 2000、Windows XP、あるいは Windows Server 2003 などの Windows NT-ベースのオペレーティング システムを使用している場合には、管理者権限のユーザーとしてログインする必要があります。
3. インストールのロケーションを選択します。従来、MySQL サーバは **C:\mysql** にインストールされます。MySQL インストール ウィザードが MySQL を **C:\Program Files\MySQL** にインストールします。MySQL を **C:\mysql** にインストールしない場合、起動時あるいはオープン ファイルでインストール ディレクトリへのパスを指定する必要があります。**オプション ファイルの作成** 参照。
4. 任意の Zip アーカイブ ツールを使用してインストールするアーカイブを選択したインストールのロケーションに取り出します。ツールによっては選択したインストール ロケーションのフォルダにアーカイブを取り出します。取り出したら、そのサブフォルダの中身を選択したインストール ロケーションに移動できます。

オプション ファイルの作成

サーバの起動時に起動オプションを指定する必要がある場合には、それらをコマンドラインで指示するかあるいはそれらをオープン ファイルに配置します。サーバの起動時に常に使用するオプションは、オープン ファイルを使用して MySQL の設定を指定すると一番便利です。これは以下の環境で特に便利です。

- インストールのロケーションあるいはデータ ディレクトリのロケーションはデフォルトのロケーション (C:\Program Files\MySQL\MySQL Server 5.1 あるいは C:\Program Files\MySQL\MySQL Server 5.1\data) とは異なります。
- サーバの設定を調整する必要があります。

MySQL サーバが Windows で起動すると 2 つのファイルでオプションを探します。Windows のディレクトリの `my.ini` ファイル、および `C:\my.cnf` ファイルです。Windows のディレクトリは一般的には `C:\WINDOWS` あるいは `C:\WINNT` のような名前で命名されます。以下のコマンドを使用して `WINDIR` 環境変数の値から正確なロケーションを割り出すことができます。

```
C:\> echo %WINDIR%
```

MySQL は最初に `my.ini` ファイルでオプションを探し、次に `my.cnf` ファイルで探します。しかし、混乱を避けるためにはファイルを 1 つだけ使用するほうが良いでしょう。お客様の PC がブート ロードーを使用している場合 `C:` はブートドライブではないので、`my.ini` ファイルしか使用できません。どちらのファイルにしる、それらは一般的なテキストファイルになります。

MySQL ディストリビューションに含まれている参考例のオプション ファイルを使用することもできます。詳細は [あらかじめ形成されたオプション・ファイル](#) を参照してください。

オプション ファイルはノートパッドなどのテキスト エディタで作成したり変更できます。例えば、MySQL を `E:\mysql` にインストールしデータ ディレクトリが `E:\mydata\data` にある場合、`[mysqld]` セクションを含むオプション ファイルを作成して `basedir` および `datadir` パラメータの値を指定できます。

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Windows のパラメータはオプション ファイルでバックスラッシュではなくスラッシュ (前向き) を使用して指定されていますのでその点ご注意ください。バックスラッシュを使用する際は 2 重のバックスラッシュを使用する必要があります。

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

MySQL Enterprise

```
#####MySQL #####
##http://www.mysql.com/products/enterprise/advisors.html #####
```

Windows では、MySQL インストーラによってデータ ディレクトリはお客様が MySQL をインストールしたディレクトリに配置されます。データ ディレクトリを異なるロケーションで使いたい場合、`data` ディレクトリのコンテンツ全体を新しいロケーションにコピーする必要があります。例えば、MySQL が `C:\Program Files\MySQL\MySQL Server 5.1` にインストールされている場合、デフォルトのデータ ディレクトリは `C:\Program Files\MySQL\MySQL Server 5.1\data` にあります。代わりに `E:\mydata` をデータ ディレクトリとして使用する場合、以下の 2 つを行う必要があります。

1. `data` ディレクトリ全体およびそのすべてのコンテンツを `C:\Program Files\MySQL\MySQL Server 5.1\data` から `E:\mydata` へ移動します。
2. サーバの起動時に常に新しいデータ ディレクトリのロケーションを指定するには `--datadir` オプションを使用します。

MySQL サーバ タイプの選択

以下の表は Windows 上の MySQL 5.1 で使用できるサーバを示したものです。

バイナリ	説明
<code>mysqld-debug</code>	フルのデバッグおよび自動メモリ割り当てチェックによるコンパイル、および <code>InnoDB</code> サポート。

<code>mysqld</code>	バイナリの InnoDB サポートによる最適化。
<code>mysqld-nt</code>	指定パイプのサポートによる Windows NT、2000 および XP へのバイナリの最適化

以前のバイナリはすべて最新の Intel プロセッサに最適化されていますが、Intel i386-クラスあるいはそれ以上で動作する必要があります。

Windows 用のすべての MySQL 5.1 サーバはデータベース ディレクトリのシンボリック リンキングをサポートしています。

MySQL はすべての Windows プラットフォームで TCP/IP をサポートしています。platforms.`mysqld-nt` サーバは Windows NT、2000、XP、および 2003 上の名前付きパイプをサポートしています。しかし、デフォルトではプラットフォームに関係なく TCP/IP を使用します。(名前付きパイプは多くの Windows 設定では TCP/IP より低速です。)

名前付きパイプの使用は以下の条件に因ります。

- 名前付きパイプはサーバを `--enable-named-pipe` オプションで起動したときのみ有効です。いくつかのユーザーが名前付きパイプを使用した際に MySQL サーバがシャットダウンする問題がありましたので、このオプションは明示的に使用する必要があります。
- 名前付きパイプの接続は `mysqld-nt` サーバにのみ可能で、サーバが名前付きパイプをサポートしている Windows のバージョン (NT、2000、XP、2003) で動作している時のみ使用できます。
- これらのサーバは Windows 98 あるいは Me でも動作しますが、TCP/IP がインストールされている時のみで、名前付き接続は使用できません。
- これらのサーバは Windows 95 では動作しません。

注:このマニュアルの殆どの例では `mysqld` をサーバ名として使用しています。`mysqld-nt` などの別のサーバを使用する場合には、例示のようにコマンドで適切な変更を加えます。

サーバを最初に起動する

この項では MySQL サーバの起動に関する一般的な概要を説明します。以下の数項では MySQL サーバのコマンドラインあるいは Windows のサービスとしての起動に特化した情報を提供します。

ここでは MySQL を `Noinstall` バージョンを使用してインストールした場合、あるいは MySQL を GUI ツールを使用しないで手動で設定してテストする場合に関する情報を提供します。

以下の例では MySQL をデフォルトのロケーション `C:\Program Files\MySQL\MySQL Server 5.1` にインストールしたものと説明します。異なるロケーションに MySQL をインストールしている場合には例示のパス名を変更します。

Windows NT、2000、XP、あるいは 2003 などの NT-ベースのシステムでは、クライアントには 2 つのオプションがあります。それらは TCP/IP を使用し、サーバが名前付きパイプの接続をサポートしている場合には名前付きパイプも使用できます。MySQL で TCP/IP を Windows NT 4 で使用する場合、サービス パック 3 (あるいはそれ以降) をインストールする必要があります。

Windows 95、98、あるいは Me 上では、MySQL クライアントは常に TCP/IP を使用したサーバに接続します。(これによりネットワークのどのマシンでもお客様の MySQL サーバに接続します。.)この理由により、MySQL を起動する前にお客様のマシンに TCP/IP のサポートがインストールされているか確認する必要があります。TCP/IP は Windows CD-ROM にあります。

旧 Windows 95 リリース (例えば、OSR2) を使用している場合、旧 Winsock を使用している可能性があります。MySQL には Winsock 2 が必要です。最新版の Winsock は <http://www.microsoft.com/> で入手できます。Windows 98 は新しい Winsock 2 ライブラリを使用していますので、ライブラリを更新する必要はありません。

サーバを `--shared-memory` オプションで起動した場合、Windows 上の MySQL は共有メモリの接続もサポートします。クライアントは `--protocol=memory` オプションを使用して共有メモリで接続できます。

どのサーババイナリを実行するかについては、[MySQL サーバタイプの選択](#) を参照してください。

テストはコンソール ウィンドウ (あるいは「DOS ウィンドウ」) のコマンド プロンプトで行います。このようにウィンドウにサーバの状況に関するメッセージが表示されますので状況を用意に確認できます。設定に何か問題があった場合には、これらのメッセージで問題を特定して修正できます。

サーバを起動するには、以下のコマンドを入力します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --console
```

InnoDB サポートを含むサーバでは、以下のメッセージに類似したメッセージがサーバの起動時に表示されます (パラメータをサイズは異なる場合があります)。

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25 InnoDB: Started
```

サーバが起動シーケンスを終了すると、以下のようなメッセージが表示されます。このメッセージが表示されるとサーバがクライアント接続の用意が整ったことを意味します。

```
mysqld: ready for connections
Version: '5.1.15-beta' socket: '' port: 3306
```

サーバは生成する分析に関する出力をコンソールに書き続けます。クライアント プログラムを実行する新しいコンソール ウィンドウを開くことができます。

`--console` オプションを削除すると、サーバは分析の出力をデータ ディレクトリ (C:\Program Files\MySQL\MySQL Server 5.1\data デフォルト) のエラーログに書き込みます。エラーログは `.err` 拡張付きのファイルです。

注:MySQL の Grant テーブルのアカウントには最初はパスワードがありません。サーバの起動後に [インストール後の設定とテスト](#) の説明に従ってパスワードをアカウントに設定する必要があります。

MySQL の Windows のコマンドラインからの起動

MySQL サーバはコマンドラインから手動で起動できます。この起動は Windows のどのバージョンでもできます。

コマンドラインから `mysqld` サーバを起動するには、コンソール ウィンドウ (あるいは「DOS ウィンドウ」) を開け、以下のコマンドを入力します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld"
```

`mysqld` へのパスはお客様のシステムの MySQL のインストール ケーションによって異なる場合があります。

NT バージョン以外の Windows 上では、このコマンドはバックグラウンドで `mysqld` を起動します。つまり、サーバの起動後に別のコマンド プロンプトが表示されます。サーバをこのように Windows NT、2000、XP、あるいは 2003 上で起動すると、サーバはフォアグラウンド (前景) で起動しそのサーバが動作している間はコマンド プロンプトは表示されません。このため、そのサーバが動作している間は別のコンソール ウィンドウを開けてクライアント プログラムを実行する必要があります。

MySQL サーバを停止するには以下のコマンドを実行します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

注:MySQL の `root` ユーザーアカウントにパスワードが設定されている場合、`mysqladmin` を `-p` オプションで実行し、プロンプトが表示されたらパスワードを入力します。

このコマンドは MySQL 管理ユーティリティ `mysqladmin` を実行してサーバに接続し、サーバをシャットダウンします。そのコマンドは MySQL に `root` ユーザーとして接続します。それは MySQL の Grant システムのデフォルトの管理アカウントです。MySQL の Grant システムのユーザーは Windows のログイン ユーザーとは全く別のものです。

`mysqld` が起動しない場合、エラーログを確認してその問題に関するメッセージが無いか確認します。そのエラーは `C:\Program Files\MySQL\MySQL Server 5.1\data` ディレクトリにあります。それは接尾辞 `.err` のファイルです。サーバを `mysqld --console` として起動することもできます。この場合、表示された画面で問題を解決するために役に立つ情報が得られる場合もあります。

最後のオプションは `mysqld` を `--standalone` および `--debug` オプションで起動することです。この場合、`mysqld` がログファイル `C:\mysqld.trace` を書き、その中に `mysqld` が起動しない理由が含まれています。[Creating Trace Files](#) 参照。

`mysqld --verbose --help` を使用して `mysqld` に関連したすべてのオプションを表示します。

Windows のサービスとして MySQL を起動する

NT ファミリー (Windows NT、2000、XP、2003) を使用する場合、MySQL を Windows のサービスとしてインストールすることを勧めます。サービスでインストールすると Windows の起動および停止と共に MySQL を自動的に起動、停止できます。MySQL サーバをサービスとしてインストールすると `NET` コマンドを使用してコマンドラインから、あるいはグラフィカル `Services` ユーティリティで管理することもできます。

`Services` ユーティリティ (Windows `Service Control Manager`) は Windows のコントロール パネル (Windows 2000、XP、および Server 2003 の 管理ツール に) にあります。衝突を避けるために、サーバのインストールの際に `Service` ユーティリティを閉じるか、またはコマンドラインからオペレーションを削除するようお願いします。

MySQL を Windows のサービスとしてインストールする前に、現在のサーバが以下のコマンドを使用している場合には最初に現在のサーバを停止する必要があります。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

注:MySQL の `root` ユーザーアカウントにパスワードが設定されている場合、`mysqladmin` を `-p` オプションで実行し、プロンプトが表示されたらパスワードを入力します。

このコマンドは MySQL 管理ユーティリティ `mysqladmin` を実行してサーバに接続し、サーバをシャットダウンします。そのコマンドは MySQL に `root` ユーザーとして接続します。それは MySQL のグラント システムのデフォルトの管理アカウントです。MySQL のグラント システムのユーザーは Windows のログイン ユーザーとは全く別のものです。

サーバをこのコマンドを使用してサービスとしてインストールします。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --install
```

サービスのインストール コマンドはサーバは起動しません。その件は後ほどこの項で説明します。

MySQL プログラムを容易に実行するには、MySQL `bin` ディレクトリのパス名を Windows システムの `PATH` 環境変数に追加します。

- Windows のデスクトップで マイ コンピュータ アイコンを右クリックし、プロパティを選択します。
- 次に 高度 タブを表示された システムのプロパティ メニューから選択し、`Environment Variables` ボタンをクリックします。
- `System Variables` で、パス を選択し、次に `Edit` ボタンをクリックします。システム変数の編集のダイアログが表示されます。
- カーソルを `Variable Value` の印のついたスペースに表示されたテキストの最後に持って行きます。(カーソルがそのスペースのテキストの一番後ろにあることを確認するには 終了 キーを使用します。)次にお客様の MySQL `bin` ディレクトリの完全なパス名 (例えば、`C:\Program Files\MySQL\MySQL Server 5.1\bin`) を入力します。このパス名の行にはパス名を他の値から分けるためのセミコロンを使用していることにご留意ください。OK をクリックして表示されているすべてのダイアログを順番に消します。この段階でシステムのどのディレクトリからでも DOS プロンプトに MySQL の実行プログラム名を入力してすべての MySQL 実行プログラムを呼び出すことができます。この実行プログラムにはサーバ、`mysql` クライアント、および `mysqladmin` 並びに `mysqldump` などのすべての MySQL コマンドライン ユーティリティが含まれています。

同じマシンで複数の MySQL サーバを動作させている場合には MySQL `bin` ディレクトリを Windows `PATH` に追加することはできません。

警告:システム `PATH` を手動で編集する際には最大限の注意が必要です。既存の `PATH` の値の一部でも間違えて削除したりあるいは変更したりすると誤動作を引き起こしたりあるいはシステムが使用できなくなったりする場合があります。

サービスをインストールする際に以下の引数を MySQL 5.1 で使用できます。

- `--install` オプションの後に直ぐにサービス名を指定できます。デフォルトのサービス名は `MySQL` です。
- サービス名を入力すると、一つのオプションで操作できます。操作手順では、`--defaults-file=file_name` でオプションのファイル名を指定し、そこでサーバが起動時にオプションを読み込みます。
`--defaults-file` 以外の一つのオプションを使用できますが、あまりお勧めできません。`--defaults-file` は名前付けオプション ファイルにサーバの複数の起動オプションを指定できるのでさらに柔軟です。
- サービス名の後に `--local-service` オプションも指定できます。これによりサーバをシステム権限の制限付き `LocalService` Windows アカウントを使用して起動できます。このアカウントは Windows XP あるいはそれ以降のみ利用できます。`--defaults-file` および `--local-service` は両方ともサービス名の後に続きます。順序は帰られます。

MySQL サーバを Windows のサービスでインストールした場合、サーバが使用するサービス名およびオプション ファイルは以下の規則で決められます。

- サービス インストールのコマンドがサービス名あるいはデフォルトのサービス名 (`MySQL`) を `--install` オプションの後に指定しない場合、サーバは `MySQL` のサービス名を使用し、標準のオプション ファイルの `[mysqld]` グループのオプションを読み込みます。
- サービス インストールのコマンドが `MySQL` 以外のサービス名を `--install` オプションの後に指定した場合、サーバはそのサービス名を使用します。サービスとして同じ名前を持つグループからオプションを読み込み、標準のオプション ファイルからオプションを読み込みます。

サーバはまた標準のオプション ファイルの `[mysqld]` グループからオプションを読み込みます。これによりすべての MySQL サービスで使用されるオプション、およびそのサービス名でインストールしたサーバがサービスとして同じ名前を持つオプション ション グループに `[mysqld]` グループを使用できます。

- サービス インストールのコマンドがサービス名の後に `--defaults-file` オプションを指定すると、サーバは名前付けファイルの `[mysqld]` グループのオプションのみを読み込み、標準のオプション ファイルは無視します。

さらに複雑な例として、以下のコマンドがあります。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld"
--install MySQL --defaults-file=C:\my-opts.cnf
```

ここでは、デフォルトのサービス名 (`MySQL`) は `--install` オプションの後に表示されます。`--defaults-file` オプションが指定されない場合、このコマンドよりサーバが標準のオプション ファイルから `[mysqld]` グループを読み込みます。しかし、`--defaults-file` オプションが指定されているので、サーバは `[mysqld]` オプション グループのオプションを、名前付けファイルのみから読み込みます。

MySQL サービスを実行する前に Windows の `Services` ユーティリティでスタート パラメータとしてオプションを指定することもできます。

MySQL サーバをサービスとしてインストールすると、Windows が起動されるたびにサービスが自動的に実行されます。サービスを `Services` ユーティリティで、あるいは `NET START MySQL` コマンドを使用して直ぐに実行することもできます。`NET` コマンドはケースセンシティブではありません。

サービスとして起動する場合、`mysqld` はコンソール ウィンドウにアクセスしないのでメッセージは表示されません。`mysqld` が起動しない場合、エラーログにサーバが記載したその問題の原因を知らせるメッセージがないか確認します。エラーログは MySQL データ ディレクトリ (例えば、`C:\Program Files\MySQL\MySQL Server 5.1\data`) にあります。そのファイルは `.err` の接尾辞が付いたファイルです。

MySQL サーバをサービスとしてインストールして、サービスを実行しているときに、Windows がシャットダウンすると Windows が自動的にサービスを停止します。そのサーバを `Services` ユーティリティ、`NET STOP MySQL` コマンド、あるいは `mysqladmin shutdown` コマンドを使用して手動で停止することもできます。

サービスがブート プロセスで自動的に実行されないようにサーバをマニュアル サービスとしてインストールすることもできます。これには `--install` オプションではなく `--install-manual` オプションを使用します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --install-manual
```

サービスとしてインストールしたサーバを削除するには、起動中の場合にはそれを最初に **NET STOP MySQL** を実行して停止します。次に **--remove** オプションを使用してそれを削除します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --remove
```

mysqld をサービスで稼働させていない場合は、コマンドラインでそれを起動できます。その手順は、[MySQL の Windows のコマンドラインからの起動](#) を参照してください。

インスとレーションで問題があった場合には、[Windows への MySQL インストールにおけるトラブルシューティング](#) を参照してください。

MySQL インストールのテスト

以下のコマンドのいずれかを実行して MySQL サーバが動作しているかテストできます。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqlshow"
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqlshow" -u root mysql
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" version status proc
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql" test
```

mysqld のクライアント プログラムから TCP/IP 接続への反応が鈍い場合、それは多分お客様の DNS の問題です。この場合 **mysqld** を **--skip-name-resolve** オプションで起動し **localhost** および MySQL 許諾テーブルの **Host** カラムにある IP 番号のみを使用します。

MySQL クライアントに TCP/IP ではなく名前付けパイプ接続を使用するには **--pipe** あるいは **--protocol=PIPE** オプションを指定するか、あるいは **.**(ピリオド) をホスト名として指定すると使用できます。デフォルトのパイプ名を使用しない場合には **--socket** オプションを使用してパイプ名を指定します。

root アカウントにパスワードを設定したら、匿名のアカウントを削除し、あるいは新しいユーザーアカウントを作成し、次に適切な **-u** および **-p** オプションを上述のコマンドで使用して MySQL サーバに接続する必要があります。[MySQL サーバへの接続](#) 参照。

mysqlshow に関する詳細は、[mysqlshow — データベース、テーブル、カラム情報を表示します](#)。を参照してください。

Windows への MySQL インストールにおけるトラブルシューティング

MySQL を最初にインストールして起動する際に、エラーが発生して MySQL サーバが起動できない場合があります。この項ではエラーが発生した場合の問題の分析方法およびエラーの修正について説明します。

サーバのトラブルシューティングに最初に使用するツールはエラーログです。MySQL サーバはサーバが起動しない原因となるエラーに関する情報を記録するエラーログを使用しています。エラーログはお客様が **my.ini** ファイルで指定したデータ ディレクトリにあります。デフォルトのデータ ディレクトリは **C:\Program Files\MySQL\MySQL Server 5.1\data** にあります。[エラー ログ](#) 参照。

エラーに関する別の情報源は MySQL サービスが実行された際に表示されるコンソール メッセージです。コマンドラインの **NET START MySQL** コマンドを **mysqld** をサービスとしてインストールした後に使用してサービスとしての MySQL サーバの起動に関するエラーメッセージを表示します。[Windows のサービスとして MySQL を起動する](#) 参照。

以下の例は MySQL のインストールおよびサーバを最初に起動する際に発生する共通の問題のエラーメッセージを示したものです。

- MySQL サーバが **mysql** 権限データベースあるいは重要なファイルを見つけられなかった場合、以下のメッセージが表示されます。

```
System error 1067 has occurred.
Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
```

これらのメッセージは MySQL のベース あるいはデータディレクトリがデフォルトのロケーション (それぞれ **C:\Program Files\MySQL\MySQL Server 5.1** および **C:\Program Files\MySQL\MySQL Server 5.1\data**) のロケーションにインストールされた場合によく表示されます。

この状況は MySQL がアップグレードされて新しいロケーションにインストールされたが、設定ファイルの新しいロケーション反映のための更新が行われていない場合に発生する場合があります。さらに、新旧の設定ファイルの

衝突がエラーの原因になる場合もあります。MySQL をアップグレードする際は必ず旧設定ファイルを削除するか名前を変えてください。

MySQL を `C:\Program Files\MySQL\MySQL Server 5.1` 以外のディレクトリにインストールした場合には、設定 (`my.ini`) ファイルを使用して MySQL サーバがこれを認識しているかどうか確認する必要があります。`my.ini` ファイルは Windows ディレクトリ、一般的には `C:\WINDOWS` あるいは `C:\WINNT` に配置される必要があります。コマンドプロンプトから以下のコマンドを発行して `WINDIR` 環境変数の値から正確なロケーションを割り出すことができます。

```
C:\> echo %WINDIR%
```

オプション ファイルはノートパッドなどのテキスト エディタで作成したり変更できます。例えば、MySQL を `E:\mysql` にインストールしデータ ディレクトリが `D:\MySQLdata` にある場合、オプション ファイルを作成して `[mysqld]` の欄を設けて値を `basedir` および `datadir` パラメータに指定できます。

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Windows のパラメータはオプション ファイルでバックスラッシュではなくスラッシュ (前向き) を使用して指定されていますのでその点ご注意ください。バックスラッシュを使用する際は 2 重のバックスラッシュを使用する必要があります。

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.1
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

`datadir` の値を MySQL 設定ファイルで変更する際は、MySQL サーバを起動する前に既存の MySQL データ ディレクトリのコンテンツを移動する必要があります。

オプション ファイルの作成 を参照してください。

- MySQL を再インストールあるいはアップグレードする際に、最初に既存の MySQL のサービスを停止して削除してから MySQL を MySQL の 設定ウィザードを使用してインストールしなかった場合、このエラーが発生する場合があります。

```
Error: Cannot create Windows service for MySql. Error: 0
```

これは設定ウィザードがサービスをインストールしようとした時に既存のサービスが同じ名前で存在する場合に発生します。

この問題の解決法の一つは設定ウィザードを使用する際 `mysql` 以外のサービス名を選択することです。これにより新しいサービスが正しくインストールされ、古いサービスはそのまま残ります。新旧のサービスが存在することは特に問題はありませんが、使用しない古いサービスは削除したほうが良いでしょう。

古い `mysql` サービスを永久に削除するには、コマンドラインで以下のコマンドを管理権限のあるユーザーとして実行します。

```
C:\> sc delete mysql
[SC] DeleteService SUCCESS
```

`sc` ユーティリティがお客様の Windows のバージョンで利用できない場合、`delsrv` ユーティリティを <http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp> からダウンロードし、`delsrv mysql` 構文を使用します。

Windows を使用した MySQL をアップグレードする

この項では Windows を使用した MySQL のアップグレードに必要なステップについて説明します。

1. Windows に特化しない MySQL のアップグレードに関する詳細は、[MySQL のアップグレード](#) を参照してください。

2. アップグレードを実行する前に常に現在の MySQL のインストールのバックアップを取る必要があります。[データベースのバックアップ](#) 参照。
3. <http://dev.mysql.com/downloads/> から MySQL の Windows ディストリビューションをダウンロードします。
4. MySQL をアップグレードする前に、サーバを停止する必要があります。サーバをサービスとしてインストールしている場合は、コマンドプロンプトの以下のコマンドでサービスを停止します。

```
C:\> NET STOP MySQL
```

MySQL サーバをサービスを使用せず使用している場合は、以下のコマンドを使用してサーバを停止します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin" -u root shutdown
```

注:MySQL の `root` ユーザーアカウントにパスワードが設定されている場合、`mysqladmin` を `-p` オプションで実行し、プロンプトが表示されたらパスワードを入力します。

5. MySQL 5.1 を 4.1.5 以前のバージョンからアップグレードする場合、あるいは Zip アーカイブからインストールした MySQL バージョンを MySQL インストール ウィザードでインストールしたバージョンにアップグレードする場合は、以前のインストールと MySQL サービス (サーバがサービスとしてインストールされている場合) を手動で削除する必要があります。

MySQL サービスを削除する場合、以下のコマンドを使用します。

```
C:\> C:\mysql\bin\mysqld --remove
```

既存のサービスを削除しなかった場合、MySQL インストール ウィザードは新しい MySQL サービスを適切にインストールできない場合があります。

6. MySQL インストール ウィザードを使用する場合は、[MySQL インストール ウィザードを使用する](#) の説明に従ってウィザードを起動します。
7. Zip アーカイブから MySQL をインストールする場合、アーカイブを取りだします。既存の MySQL のインストールを上書きする (通常 `C:\mysql` にある) か、あるいはそれを `C:\mysql5` などの異なるディレクトリにインストールします。既存のインストールの上書きをお勧めしています。
8. MySQL を Windows のサービスとして使用していて、この手順で以前のサービスを削除し、サービスを再インストールした場合。 ([Windows のサービスとして MySQL を起動する](#) 参照。)
9. サーバを再起動します。MySQL をサービスとして使用している場合、例えば `NET START MySQL` を使用するか、あるいは `mysqld` を直接実行します。
10. エラーが発生したら、[Windows への MySQL インストールにおけるトラブルシューティング](#) を参照します。

Windows 上の MySQL と Unix 上の MySQL の比較

Windows を使用した MySQL は実証済みで安定しています。Windows バージョンの MySQL は以下の例外を除いて Unix バージョンの MySQL と同じです。

- Windows 95 およびスレッド

Windows 95 は各スレッドの作成毎におよそ 200 バイトのメモリを消費します。MySQL の各接続毎にスレッドが作成されます。ですからサーバが多くの接続を扱う場合には Windows 95 では `mysqld` をあまり長時間使用しないようにします。Windows の新しいバージョンではこのバグによる影響はありません。

- ポート数の制限

Windows システムにはクライアント接続のポートがおよそ 4,000 あり、一つのポート接続が閉じるとそのポートを再度利用できるまで 2~4 分かかります。クライアントのサーバへに接続/切断頻度が高い環境では、すべての利用できるポートは閉じたポートが再度利用できるようになる前に使用してしまうことができます。このようになると、MySQL サーバはそれが動作中であっても反応していないようにみえます。ポートはマシンで実行されている他のアプリケーションでも同様に使用できます。その際、MySQL に利用できるポート数は少なくなります。

この問題に関する詳細は、<http://support.microsoft.com/default.aspx?scid=kb;en-us;196271> を参照してください。

- 同時読み込み

MySQL は `INSERT` と `SELECT` を混合する際 `pread()` および `pwrite()` のシステム コールに依存しています。現在、弊社では `pread()` および `pwrite()` をエミュレートするために `mutex` を使用しています。弊社では将来的に高速化を図るため `readfile()/writefile()` インターフェースを NT、2000、および XP に使用できるようにしてファイルレベルのインターフェースを仮想インターフェースに置き換える予定です。現在は MySQL 5.1 が使用できるオープンファイルは 2,048 に制限されており、Unix に相当する同時スレッドを Windows NT、2000、XP、および 2003 では使用できないということを意味しています。

- ブロック読み込み

MySQL は各接続にブロック読み込みを使用しています。それは名前付きパイプ接続が有効になった場合以下が想定されます。

- 接続は、Unix バージョンの MySQL とは異なり、8 時間後でも自動的に切断されない。
- 接続がハングした場合、MySQL を停止せずにハングを解消できない。
- `mysqladmin kill` は休眠中の接続には機能しない。
- `mysqladmin shutdown` は休眠接続がある限るシャットダウンできない。

弊社でこの問題を将来的に解決する予定です。

- `ALTER TABLE`

`ALTER TABLE` ステートメントを実行中は、テーブルは他のスレッドが使用できないようにロックされます。これは Windows の場合、使用中のファイルは別のスレッドで削除できないという事実に関連しています。将来的にはこの問題を解決する方法を見つける予定です。

- `DROP TABLE`

`MERGE` ハンドラーがテーブルのマッピングを上層の MySQL に非表示にしているために、`MERGE` テーブルで使用されている `DROP TABLE` は Windows では機能しません。Windows では開いているファイルは削除できないようになっているため、まずすべての `MERGE` テーブル (`FLUSH TABLES` で) をフラッシュするか、あるいはテーブルを削除する前に `MERGE` テーブルを削除する必要があります。

- `DATA DIRECTORY` および `INDEX DIRECTORY`

`CREATE TABLE` の `DATA DIRECTORY` オプションと `INDEX DIRECTORY` オプションは Windows がシンボリックリンクをサポートしていないので Windows では無視されます。これらのオプションはまた非機能 `realpath()` コールのシステムでは無視されます。

- `DROP DATABASE`

スレッドで使用中のデータベースは削除できません。

- タスク マネージャで MySQL をシャットダウンする

Windows 95 では、タスク マネージャあるいはシャットダウン ユーティリティで MySQL をシャットダウンできません。それを停止するには `mysqladmin shutdown` を使用します。

- ケース インセンシティブ ネーム

ファイル名は Windows ではケース センシティブではありません。ですから MySQL データベースおよびテーブル名も Windows ではケース センシティブではありません。唯一の制限はデータベースおよびテーブル名は所定のステートメントでは同じケースを使用して指定される必要があるということだけです。[識別子の犬文字/小文字区別](#) 参照。

- `\` パス名区切り文字

Windows ではパス名のコンポーネントは `\` 文字で区切られます。それは MySQL でもエスケープ文字です。`LOAD DATA INFILE` あるいは `SELECT...INTO OUTFILE` を使用している場合、Unix スタイルのファイル名を `'` 文字と一緒に使用します。


```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

また、`\` を 2 本にする必要があります。

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- パイプに関する問題

Pipes は Windows のコマンドライン プロンプトでは信頼性に欠けます。パイプに `^Z / CHAR(24)` が含まれている場合、Windows はファイルの最後だと勘違いしてプログラムを中止します。

これは以下のようにバイナリのログを適用する際の主な問題です。

```
C:\> mysqlbinlog binary_log_file | mysql --user=root
```

ログに関する問題が発生しその問題が `^Z / CHAR(24)` 文字によるものだと考えられる場合は、以下の回避法が使用できます。

```
C:\> mysqlbinlog binary_log_file --result-file=tmp/bin.sql
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

後者のコマンドはバイナリのデータを含む SQL ファイルを確実に読み込むためにも使用できます。

- Access denied for user エラー

MySQL がホスト名の問題を適切に解決できない場合、同じマシンで動作しているサーバに接続するために MySQL クライアントのプログラムを実行しようとした時に以下のエラーが発生する場合があります。

```
Access denied for user 'some_user'@'unknown'
to database 'mysql'
```

この問題を解決するためには、以下の情報を含むファイル名 `\\windows\hosts` を作成する必要があります。

```
127.0.0.1 localhost
```

ここに皆さんから改善に向けたご協力を頂けるであろう Windows を使用した MySQL の周知の問題を示します。

- Microsoft から提供された高速のスレッド セーフな増分/減分メソッドを使用するためにマクロを追加します。

Linux に MySQL をインストールする

Linux に MySQL をインストールする方法として RPM パッケージをお勧めしています。MySQL RPM版としては現在 SuSE Linux 7.3 のシステムにビルドしていますが、`rpm` をサポートし `glibc` を使用した殆どの Linux に対応する必要があります。RPM パッケージの取得については、[MySQL の取得方法](#) を参照してください。

MySQL AB ではプラットフォームに特化した RPM を提供しています。プラットフォーム特化の RPM と一般の RPM の違いはプラットフォーム特化版 RPM は対象のプラットフォームにビルドされて動的にリンクされているのに対し、一般の RPM は Linux のスレッドに静的にリンクされています。

注:MySQL の RPM ディストリビューションは他のベンダーもよく提供しています。他のベンダーが提供している RPM ディストリビューションは MySQL AB が提供しているものとその機能および特徴において異なっており、本マニュアルの説明は他のベンダーのインストールには必ずしも適用しません。ベンダーの説明書はそれでも検討する必要があります。

RPM ファイル (例えば、次のようなエラーメッセージが表示された場合 `Sorry, the host 'xxxx' could not be looked up`) に問題があった場合には、[Linux バイナリ ディストリビューションの注釈](#) を参照してください。

殆どの場合、[MySQL-server](#) パッケージおよび [MySQL-client](#) パッケージのインストールだけで MySQL インストールの機能面は十分です。標準のインストールには他のパッケージは必要ありません。

MySQL パッケージにのインストールの際に依存型不具合 (例えば、`error:removing these packages would break dependencies:libmysqlclient.so.10 is needed by ...`) が表示された場合、共有ライブラリを含む [MySQL-shared-](#)

`compat` パッケージをインストールして下位互換 (MySQL 4.0 用 `libmysqlclient.so.12` および MySQL 3.23 用 `libmysqlclient.so.10`) を取る必要があります。

Linux ディストリビューションの中にはまだ MySQL 3.23 で出荷しているものもあり、その組み合わせは通常ディスクスペースを節約するためにアプリケーションに動的にリンクします。これらの共有ライブラリが個別のパッケージに入っている場合 (例えば、`MySQL-shared`)、このパッケージをインストールしたままで MySQL サーバとクライアント パッケージ (スタティックにリンクし共有ライブラリに依存していないもの) を単にアップグレードするだけで十分です。MySQL サーバ (例えば、Red Hat Linux) のように同じパッケージに共有ライブラリを含むディストリビューションには、弊社の 3.23 `MySQL-shared` RPM をインストールするか、あるいは `MySQL-shared-compat` パッケージを使用します。(両方をインストールしないでください。)

以下の RPM パッケージが利用できます。

- `MySQL-server-VERSION.i386.rpm`

MySQL サーバ。別のマシンで動作している MySQL サーバに接続する際にのみこれが必要です。

注:MySQL 4.0.10 以前はサーバ RPM ファイルは `MySQL-VERSION.i386.rpm` と呼ばれていました。つまり、`-server` がその名前にありませんでした。

- `MySQL-client-VERSION.i386.rpm`

標準の MySQL クライアント プログラムこのパッケージは常にインストールの希望があるパッケージです。

- `MySQL-bench-VERSION.i386.rpm`

テストおよびベンチマーク Perl と `DBI` および `DBD::mysql` モジュールが必要です。

- `MySQL-devel-VERSION.i386.rpm`

Perl モジュールなど他の MySQL クライアントをコンパイルする際に必要なライブラリとファイルを含みます。

- `MySQL-shared-VERSION.i386.rpm`

このパッケージには言語およびアプリケーションが動的にロードされ MySQL を使用する必要のある共有ライブラリ (`libmysqlclient.so*`) を含みます。それにはシングル スレッドとスレッドセーフのバイナリが含まれます。このパッケージをインストールする場合は、`MySQL-shared-compat` パッケージはインストールしないでください。

- `MySQL-shared-compat-VERSION.i386.rpm`

このパッケージには MySQL 3.23、4.0、4.1、および 5.1 の共有ライブラリが含まれています。それにはシングル スレッドとスレッドセーフのバイナリが含まれます。旧バージョンの MySQL に動的にリンクしたアプリケーションをインストールしていて現在のバージョンをライブラリの依存性を壊すことなくアップグレードを希望する際には `MySQL-shared` の代わりにこのパッケージをインストールします。

- `MySQL-embedded-VERSION.i386.rpm`

埋め込み型 MySQL サーバライブラリ (MySQL 4.0 で利用可) です。

- `MySQL-VERSION.src.rpm`

これにはすべての旧パッケージのソースコードが含まれています。他のアーキテクチャ (例えば、Alpha あるいは SPARC) で RPM をビルドする際にも使用できます。

RPM パッケージ (例えば `MySQL-server` RPM) のすべてのファイルを表示するには、以下のコマンドを実行します。

```
shell> rpm -qpl MySQL-server-VERSION.i386.rpm
```

標準の最低限のインストールには、サーバとクライアント RPM をインストールします。

```
shell> rpm -i MySQL-server-VERSION.i386.rpm
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

クライアント プログラムのみをインストールする場合は、クライアント RPM のみインストールします。

```
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

RPM にはインストールするパッケージが完全であるかまたはその出処を確認する機能があります。この機能に関する詳細は、[MD5 チェックサムあるいは GnuPG を用いたパッケージの品質の検証](#) を参照してください。

サーバ RPM はデータを `/var/lib/mysql` でディレクトリに格納します。RPM はまた MySQL サーバを運用するためのユーザー `mysql` (存在しない場合) のログイン アカウントを作成し、サーバがブート時に自動的に起動するように適切なエントリを `/etc/init.d/` に作成します。(このことは以前インストールを実行しその起動スクリプトに変更を加えた場合、新バージョンの RPM をインストールする際に忘れないようにそのスクリプトのコピーを取っておくことを意味します。システム起動時の MySQL の自動的な起動に関する詳細は、[MySQL を自動的に起動・停止する](#) を参照してください。

MySQL RPM を `/etc/init.d` (直接あるいは symlink を介して) で初期化スクリプトをサポートしていない旧 Linux ディストリビューションにインストールするには、初期化スクリプトが実際にインストールされたロケーションを指すシンボリック リンクを作成する必要があります。例えば、ロケーションが `/etc/rc.d/init.d` の場合、RPM をインストールする前に以下のコマンドを使用して `/etc/init.d` をそこを指すシンボリック リンクとして作成します。

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

しかし、現在のすべての主な Linux ディストリビューションは `/etc/init.d` を使用している新しいディレクトリのレイアウトをサポートする必要があります。というのは、それが LSB (Linux Standard Base) 準拠に必要だからです。

インストールした RPM ファイルが `MySQL-server` を含んでいる場合、`mysqld` サーバをインストール後に設定して起動する必要があります。MySQL を使用して起動する必要があります。

上手くいかない場合には、バイナリのインストールの項で詳細を参照してください。[他の Unix 系システムへの MySQL のインストール](#) 参照。

注:MySQL のグラント テーブルのアカウントには最初はパスワードがありません。サーバの起動後に [インストール後の設定とテスト](#) の説明に従ってパスワードをアカウントに設定する必要があります。

Mac OS X に MySQL をインストールする

MySQL を Mac OS X 10.3.x (「Panther」) あるいは新バージョンに、バイナリの tarball ディストリビューションではなく Mac OS X バイナリ パッケージを PKG フォーマットでインストールします。Mac OS X の旧バージョン (例えば 10.1.x or 10.2.x) はこのパッケージではサポートされていません。

パッケージはディスク画像 (`.dmg`) ファイルにあります。そのファイルはファインダーにあるアイコンをダブルクリックして最初にインストールする必要があります。次に画像をインストールしてそのコンテンツを表示します。

MySQL を取得するには、[MySQL の取得方法](#) を参照してください。

注:インストールを始める前に、動作中のすべての MySQL サーバインスタンスを MySQL 管理アプリケーション (Mac OS X サーバ用) あるいはコマンドラインの `mysqladmin shutdown` でシャットダウンする必要があります。

MySQL PKG ファイルを実際にインストールするには、パッケージのアイコンをダブルクリックします。これで Mac OS X パッケージ インストーラーが起動し、MySQL のインストールを案内します。

Mac OS X パッケージ インストーラーにバグがある場合、宛先ディスク選択ダイアログにエラーメッセージが表示されます。

```
You cannot install this software on this disk. (null)
```

エラーが発生したら、[Go Back](#) ボタンを 1 回クリックして前の画面に戻ります。次に [Continue](#) をクリックして宛先ディスク選択にもう一度進み、宛先ディスクを選択します。弊社ではこのバグを Apple 社に連絡してこの問題の調査を依頼しています。

MySQL 用 Mac OS X PKG が `/usr/local/mysql-VERSION` にインストールされると symbolic link、`/usr/local/mysql` もインストールされ、新しいロケーションをポイントします。`/usr/local/mysql` の名前のディレクトリがあると、最初に `/usr/local/mysql.bak` に名前が変わります。さらに、インストーラは `mysql` データベースで `mysql_install_db` を実行して許諾テーブルを作成します。

インストールのレイアウトは `tar` ファイルのバイナリ ディストリビューションに類似しています。すべての MySQL バイナリは `/usr/local/mysql/bin` にあります。MySQL のソケット ファイルはデフォルトで `/tmp/mysql.sock` として作成されます。[インストールのレイアウト](#) 参照。

MySQL インストールには `mysql` 名の Mac OS X ユーザーアカウントが必要です。この名前のユーザーアカウントは Mac OS X 10.2 およびそれ以降ではデフォルトで終了します。

Mac OS X サーバを使用している場合は、MySQL のバージョンのどれかをインストールする必要があります。以下の表には Mac OS X サーバのバージョンごとの MySQL を示しています。

Mac OS X サーバのバージョン	MySQL バージョン
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a

本マニュアルのこの項では公式な MySQL Mac OS X PKG のインストールについてのみ説明します。MySQL をインストールするには Apple 社のヘルプ情報を必ず読んでください。「Help View」のアプリケーションを実行し、「Mac OS X サーバ」のヘルプを選択し、「MySQL」を探して「MySQL のインストール」の表題の項目を読みます。

<http://www.entropy.ch> から既に Mac OS X 用の Marc Liyanage の MySQL パッケージを使用している場合、そのページで提供しているバイナリのインストール レイアウトを使用したパッケージ用の更新情報に従ってください。

Marc の 3.23.x バージョンあるいは MySQL の Mac OS X サーバ バージョンから公式の MySQL PKG にアップグレードする際には、既存の MySQL 権限テーブルも現在のフォーマットに変換する必要があります。現在のフォーマットにはセキュリティの権限が追加されています。[mysql_upgrade](#) — [MySQL アップグレードのテーブル チェック](#) 参照。

MySQL をシステムの起動時に起動したい場合には、MySQL の起動アイテムもインストールする必要があります。それは個別のインストール パッケージで Mac OS X のインストール ディスク画像の一部をなすのもです。MySQLStartupItem.pkg アイコンをクリックするだけで、それをインストールするには説明の手順に従います。起動アイテムは一度だけインストールします。MySQL パッケージを後でインストールするたびにインストールする必要はありません。

MySQL の起動アイテムは `/Library/StartupItems/MySQLCOM` にあります。(MySQL 4.1.2 以前では、そのロケーションは `/Library/StartupItems/MySQL` でしたが、Mac OS X サーバでインストールされた MySQL 起動アイテムと衝突していました。)起動アイテムをインストールすると変数 `MYSQLCOM=-YES-` がシステムの設定ファイル `/etc/hostconfig` に追加されます。MySQL の自動的な起動を無効にするには、この変数を単純に `MYSQLCOM=-NO-` に変更するだけです。

Mac OS X サーバでは、デフォルトの MySQL インストールが `/etc/hostconfig` ファイルにある変数 `MYSQ` を使します。MySQL AB 起動アイテムのインストーラでこの変数を `MYSQ=-NO-` に設定すると無効にします。これによって MySQL AB の起動アイテムによる `MYSQLCOM` 変数との衝突を避けられます。しかし、それによって動作中の MySQL サーバをシャットダウンすることはありません。それはご自身で行ってください。

インストールしたら、端末ウィンドウで以下のコマンドを実行することで MySQL を起動できます。このタスクを実行するには管理者権限が必要です。

起動アイテムをインストールしたら、このコマンドを使用します。

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

起動アイテムを使用しない場合には、以下のコマンド シーケンスを入力します。

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
(Enter your password, if necessary)
(Press Control-Z)
shell> bg
(Press Control-D or enter "exit" to exit the shell)
```

MySQL サーバに、例えば `/usr/local/mysql/bin/mysql` を実行すると接続できます。

注:MySQL の Grant テーブルのアカウントには最初はパスワードがありません。サーバの起動後に [インストール後の設定とテスト](#) の説明に従ってパスワードをアカウントに設定する必要があります。

shell のリソース ファイルに別名を追加するとコマンドラインの `mysql` および `mysqladmin` などのよく使用するプログラムに容易にアクセスできます。`bash` の構文は以下のようになります。

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

`tcsh` には、以下を使用します。

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

`/usr/local/mysql/bin` to your `PATH` 環境変数を追加すると更によりくなります。例えば、shell が `bash` の場合には以下の行を `$HOME/.bashrc` ファイルに追加します。

```
PATH=${PATH}:/usr/local/mysql/bin
```

shell が `tcsh` の場合には以下の行を `$HOME/.tcshrc` ファイルに追加します。

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

`.bashrc` あるいは `.tcshrc` ファイルがホーム ディレクトリにない場合には、それをテキスト エディタで作成します。

既存のインストールのアップグレードの際、新しい MySQL PKG をインストールしても古いインストールのディレクトリは削除されません。残念ながら、Mac OS X のインストーラは以前インストールしたパッケージの適切なアップグレードに必要な機能をまだ提供していません。

既存のデータベースを新しいインストールで使用するには、古いデータ ディレクトリのコンテンツを新しいデータ ディレクトリにコピーする必要があります。このコピーを行う時には新旧のサーバが動作していないことを確認します。古いインストールからの MySQL データベース ファイルのコピーが終了して新しいサーバの起動が完了したら、ディスク スペースを空けるために古いインストールを削除します。さらに、[/Library/Receipts/mysql-VERSION.pkg](#) にある古いバージョンのパッケージを入れたディレクトリの削除する必要があります。

Solaris に MySQL をインストールする

MySQL をバイナリの tarball ディストリビューションで Solaris にインストールする際、MySQL のディストリビューションを開ける前に既に問題に出くわします。これは Solaris の `tar` が長いファイル名を扱えないからです。これは MySQL を解凍する時にエラーが表示されることを意味します。

この問題が発生したら、GNU `tar` (`gtar`) を使用してディストリビューションを解凍します。Solaris 用にコンパイルしたコピーは <http://dev.mysql.com/downloads/os-solaris.html> にあります。

MySQL の Solaris へのインストールをバイナリの tarball のディストリビューションではなく PKG フォーマットのバイナリのパッケージを使用して行うことができます。バイナリの PKG フォーマットを使用してインストールする前に、`mysql` のユーザーおよびグループなどを作成する必要があります。

```
groupadd mysql
useradd -g mysql mysql
```

基本的な PKG 取扱いのフロー例

- パッケージを追加する

```
pkgadd -d package_name.pkg
```

- パッケージを削除する

```
pkgrm package_name
```

- インストールしたパッケージの全リストを取得する

```
pkginfo
```

- パッケージに関する詳細を取得する

```
pkginfo -l package_name
```


- パッケージに属すファイルのリストを表示する

```
pkgchk -v package_name
```

- アービトラリ ファイルのパッケージ情報を取得する

```
pkgchk -l -p file_name
```

MySQL の Solaris へのインストールに関する詳細は、[Solaris に関する注釈](#) を参照してください。

MySQL を NetWare にインストールする

MySQL の NetWare へのポートは Novell 社先導の多大な貢献によるものです。NetWare 6.5 はバンドルした MySQL バイナリと同梱で、そのバージョンの NetWare 上で動作するすべてのサーバは自動商用ライセンス完備ですので Novell 社のお客様はきっと喜んでおられる事でしょう

NetWare 用 MySQL は NetWare 用 Metrowerks 社の CodeWarrior および GNU オートツールのクロス コンパイル バージョンの組み合わせでコンパイルされたものです。

NetWare 用最新のバイナリ パッケージは <http://dev.mysql.com/downloads/> で入手できます。[MySQL の取得方法](#) 参照。

MySQL をホストするには、NetWare サーバは以下の要件を満たす必要があります。

- [NetWare 6.5](#) の最新のサポート バックがインストールされていること。
- システムは NetWare のそれぞれのバージョンの動作に要する Novell 社の最低限の要件を満たしていること。
- MySQL データおよびプログラム バイナリが NSS ボリュームにインストールされていること。従来のボリュームはサポートされていません。

MySQL を NetWare にインストールするには、以下の手順を踏みます。

1. 以前のインストールからアップグレードするには、MySQL サーバを停止してください。これはサーバのコンソールから、以下のコマンドを使用して実行できます。

```
SERVER: mysqladmin -u root shutdown
```

注:MySQL の [root](#) ユーザーアカウントにパスワードが設定されている場合、[mysqladmin](#) を [-p](#) オプションで実行し、プロンプトが表示されたらパスワードを入力します。

2. MySQL をインストールするロケーションにアクセスしてクライアント マシンからターゲットのサーバにログオンします。
3. サーバにバイナリ パッケージの Zip ファイルを取り出します。Zip ファイルのパスが使用できるか確認します。単純にファイルを [SYS:](#) に取り出すほうが安全です。

以前のインストールをアップグレードするには、データ ディレクトリ (例えば、[SYS:MYSQL\DATA](#))、および [my.cnf](#) を、それをカスタマイズしている場合、コピーする必要があります。次に MySQL の古いコピーを削除します。

4. ディレクトリを分かり安く使いやすい名前に変更できます。本マニュアルの例ではインストール ディレクトリに [SYS:MYSQL](#) を使用しています。

MySQL を NetWare へのインストールする際、MySQL のあるバージョンが既に NetWare のリリース以外にインストールされている場合は検知しません。ですから、[SYS:MYSQL](#) のウェブ (例えば、MySQL 4.1 あるいはそれ以降) から最新の MySQL をインストールした場合、NetWare のサーバをアップグレードする前にフォルダの名前を変える必要があります。名前を変えないと [SYS:MySQL](#) のファイルは NetWare サポート バックにある MySQL で上書きされます。

5. サーバのコンソールで、MySQL NLM を含むディレクトリに検索パスを追加します。例えば、

```
SERVER: SEARCH ADD SYS:MYSQL\BIN
```

6. データ ディレクトリと許諾テーブルを必要に応じて [mysql_install_db](#) をサーバのコンソールで実行して初期化します。

7. サーバのコンソールで `mysqld_safe` を使用して MySQL サーバを起動します。
8. インストールを終了するには、以下のコマンドも `autoexec.ncf` に追加します。例えば、MySQL のインストールが `SYS:MYSQL` で MySQL を自動的に起動したい場合、以下のコマンドを追加します。

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE
```

MySQL を NetWare 6.0 で稼働している場合、コマンドラインの `--skip-external-locking` オプションを使用することを強くお勧めします。

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --skip-external-locking
```

`CHECK TABLE` および `REPAIR TABLE` を `myisamchk` の代わりに使用します。それは `myisamchk` が外部のロッキングを使用しているからです。外部のロッキングは NetWare 6.0 上では問題があることが知られています。その問題は NetWare 6.5 では無視されてきました。MySQL の Netware 6.0 への使用は公式にはサポートしていないことをご留意ください。

NetWare の `mysqld_safe` は画面に表示されます。`mysqld_safe` NLM をアンロード (シャットダウン) する際、画面はデフォルトで消えないようになっています。代わりに、それはユーザーの入力でプロンプトします。

```
*<NLM has terminated; Press any key to close the screen>*
```

NetWare の画面を自動的に閉じるようにするには、`--autoclose` オプションを使用して `mysqld_safe` にします。例えば、

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

NetWare の `mysqld_safe` の振る舞いに関する詳細は、[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) を参照してください。

9. MySQL を最初にインストール、あるいは以前のバージョンからアップグレードするには、NetWare 用の最新で適切な Perl および PHP 拡張をダウンロードしてインストールします。

- Perl: <http://forge.novell.com/modules/xfcontent/downloads.php/perl/Modules/>
- PHP: <http://forge.novell.com/modules/xfcontent/downloads.php/php/Modules/>

NetWare サーバに既存の MySQL のインストーラがある場合、`autoexec.ncf` で既存の MySQL の起動コマンドを確認し、必要に応じてそれらを編集あるいは削除します。

注:MySQL のグラント テーブルのアカウントには最初はパスワードがありません。サーバの起動後に [インストール後の設定とテスト](#) の説明に従ってパスワードをアカウントに設定する必要があります。

他の Unix 系システムへの MySQL のインストール

この項では圧縮 `tar` ファイル (`.tar.gz` 拡張のファイル) のフォームで様々なプラットフォームに提供されている MySQL のバイナリのディストリビューションのインストールについて説明します。詳細は、[MySQL AB でコンパイルした MySQL バイナリ](#) を参照してください。

MySQL を取得するには、[MySQL の取得方法](#) を参照してください。

MySQL `tar` ファイルのバイナリ ディストリビューションには `mysql-VERSION-OS.tar.gz` 形式の名前があり、`VERSION` は番号 (例えば、`5.1.15-beta`) で、`OS` はディストリビューションが意図するオペレーティング システムの種類を意味します (例えば、`pc-linux-i686`)。

これらの一般的なパッケージに加え、弊社ではまた選択したプラットフォームにはバイナリをプラットフォームに特化したパッケージ フォーマットで提供しています。これらのインストールの詳細に関しては、[バイナリの配布を使用した標準 MySQL のインストール](#) を参照してください。

MySQL [tar](#) ファイルのバイナリ ディストリビューションをインストールするには以下のツールが必要です。

- ディストリビューションの解凍用 GNU [gunzip](#)
- ディストリビューションのアンパック用 [tar](#)。GNU [tar](#) が機能することが知られています。オペレーティング システムの中には問題があるとされる [tar](#) のプリインストール バージョンがあります。例えば、Mac OS X [tar](#) および Sun [tar](#) は長いファイル名の場合問題があるが知られています。Mac OS X 上では、プリインストールの [gnutar](#) プログラムを使用できます。欠陥のある [tar](#) の他のシステムに、GNU [tar](#) を最初にインストールする必要があります。

問題が発生してバグをレポートする必要がある場合には、[質問またはバグの報告](#) の手順に従ってください。

MySQL のバイナリのディストリビューションのインストールおよびその使用の際に必要な基本的なコマンド

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

注:このプロシージャでは MySQL のアカウントのパスワードは設定しません。そのプロシージャの後には、[インストール後の設定とテスト](#) に進みます。

上述のバイナリ ディストリビューションのインストールに関する詳細な説明は以下のようになります。

1. [mysqld](#) にログイン ユーザーとグループを追加するには以下を実行します。

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

これらのコマンドにより [mysql](#) グループと [mysql](#) ユーザーを追加します。[useradd](#) および [groupadd](#) の構文は Unix のバージョンにより多少異なったり、あるいは [adduser](#) および [addgroup](#) など異なる名前が付く場合があります。

ユーザーやグループを [mysql](#) のではなく別の名前に変更することもできます。その場合、以下の手順で別の名前をつけます。

2. ディストリビューションを解凍するディレクトリを選択してその中にロケーションを変更します。以下の例では、ディストリビューションを [/usr/local](#) に解凍します。(この説明では、[/usr/local](#) にファイルとディレクトリを作成する権限を有するものとして説明を続けます。ディレクトリが保護されている場合、インストールを [root](#) として行う必要があります。)

```
shell> cd /usr/local
```

3. ディストリビューションを [MySQL の取得方法](#) の説明に従って取得します。所定のリリースでは、すべてのプラットフォームのバイナリのディストリビューションは同じ MySQL のソース ディストリビューションでビルドされています。
4. ディストリビューションを解凍すると、インストールのディレクトリが作成されます。次にそのディレクトリにシンボリック リンクを作成します。

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

[tar](#) コマンドが [mysql-VERSION-OS](#) 名のディレクトリを作成します。[ln](#) コマンドがディレクトリへのシンボリック リンクを作成します。これによって [/usr/local/mysql](#) のインストール ディレクトリに容易にアクセスできます。

GNU [tar](#) では、[gunzip](#) の個別の呼び出しは必要ありません。最初の行を以下のコマンドに置き換えてディストリビューションを解凍し取り出すことができます。

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

5. ロケーションをインストールのディレクトリに変更します。

```
shell> cd mysql
```

mysql ディレクトリにはいくつかのファイルとサブディレクトリがあります。インストールに最も重要なものは **bin** および **scripts** のサブディレクトリです。

- **bin** ディレクトリにはクライアント プログラムとサーバが含まれています。shell が MySQL プログラムを間違いないで見つけるにはこのディレクトリの完全なパス名を **PATH** 環境変数に追加する必要があります。[環境変数参照](#)。
- **scripts** ディレクトリは **mysql_install_db** スクリプトを含み、それによってサーバのアクセス権限を保持する許諾テーブルを含む **mysql** データベースを初期化します。

6. 今まで MySQL をインストールしたことがない場合には、MySQL 許諾テーブルを作成する必要があります。

```
shell> scripts/mysql_install_db --user=mysql
```

コマンドを **root** として実行する場合、以下の **--user** オプションを使用する必要があります。オプションの値はサーバに移動に使用する最初のステップで作成したログイン アカウント名になります。そのユーザーでログインしてそのコマンドを実行する場合、**--user** オプションは無視できます。

許諾テーブルを作成して更新した後にサーバを手動で再起動する必要があります。

7. プログラム バイナリの所有者を **root** に、データ ディレクトリの所有者を **mysqld** を運用するユーザーに変更します。インストール ディレクトリ (**/usr/local/mysql**) にいると想定した場合、そのコマンドは以下のようになります。

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

最初のコマンドはファイルの所有者属性を **root** ユーザーに変更します。2 番目のコマンドはデータ ディレクトリの所有者属性 **mysql** ユーザーに変更します。3 番目のコマンドはグループ属性を **mysql** グループに変更します。

8. マシンをブートしたときに MySQL を自動的に起動する場合は、**support-files/mysql.server** をシステムの起動ファイルがあるロケーションにコピーします。詳細 [support-files/mysql.server](#) のスクリプトおよび [MySQL を自動的に起動・停止する](#) にあります。
9. **DBI** および **DBD::mysql** Perl モジュールをインストールする場合、**bin/mysql_setpermission** スクリプトを使用して新しいアカウントを設定できます。その手順は、[Perl のインストールに関する注釈](#) を参照してください。
10. **mysqlaccess** を使用して MySQL ディストリビューションを標準とは異なる別のロケーションに入れるには、**mysqlaccess** が **mysql** クライアントを探すロケーションを変更する必要があります。**bin/mysqlaccess** スクリプトをおよそ行 18 で編集します。以下のような行を探します。

```
$MYSQL = '/usr/local/bin/mysql'; # path to mysql executable
```

パスを **mysql** が実際にシステムの保存されたロケーションに変更します。パスを変更しないと **Broken pipe** エラーが **mysqlaccess** を実行したときに発生します。

すべてを解凍してインストールしたら、ディストリビューションをテストします。MySQL サーバを起動するには、以下のコマンドを使用します。

```
shell> bin/mysqld_safe --user=mysql &
```

そのコマンドが直ぐに失敗し、**mysqld ended** をプリントした場合、そのデータ ディレクトリの **host_name.err** ファイルから情報を入手できます。

mysqld_safe に関する詳細は、[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) にあります。

注:MySQL の Grant テーブルのアカウントには最初はパスワードがありません。サーバの起動後に [インストール後の設定とテスト](#) の説明に従ってパスワードをアカウントに設定する必要があります。

ソースのディストリビューションを使用した MySQL のインストール

ソースからのインストールを始める前に、弊社のバイナリがお客様のプラットフォームで利用できるかまたは機能するかチェックしてください。弊社では弊社のバイナリが最良のオプションでビルドされたものであることを確認するために最大限の努力を払っています。

MySQL のソース ディストリビューションを取得するには、[MySQL の取得方法](#) を参照してください。

MySQL のソース ディストリビューションは圧縮 `tar` アーカイブで提供しており、フォームの名前は `mysql-VERSION.tar.gz` です。`VERSION` は `5.1.15-beta` のような番号です。

MySQL をソースからビルドしてインストールするには以下のツールが必要です。

- ディストリビューションの解凍用 GNU `gunzip`
- ディストリビューションのアンパック用 `tar`。GNU `tar` が機能することが知られています。オペレーティング システムの中には問題があるとされる `tar` のプリインストール バージョンがあります。例えば、初期バージョンの Mac OS X `tar` で提供されている `tar`、SunOS 4.x および Solaris 8 またはそれ以前のバージョンは長いファイル名の場合に問題があることが知られています。Mac OS X 上では、プリインストールの `gnutar` プログラムを使用できます。欠陥のある `tar` の他のシステムに、GNU `tar` を最初にインストールする必要があります。
- ANSI C++ コンパイラ。gcc 2.95.2 あるいはそれ以降、egcs 1.0.2 またはそれ以降あるいは egcs 2.91.66、SGI C++、および SunPro C++ は問題なく機能するものとして知られています。libg++ は gcc を使用する際は必要ありません。gcc 2.7.x は `sql/sql_base.cc` などの完全な legal C++ のファイルのいくつかをコンパイルできなくするバグを含んでいます。gcc 2.7.x だけの場合、gcc を MySQL をコンパイルできるようにアップグレードする必要があります。gcc 2.8.1 もいくつかのプラットフォームでは問題があるとされており、そのプラットフォームに新しいコンパイラがある場合には使用しないでください。

gcc 2.95.2 あるいはそれ以降を MySQL 3.23.x のコンパイルにお勧めします。

- 優良な `make` プログラム。GNU `make` を常に推奨しており、また必要な場合もあります。問題に遭遇した場合には GNU `make` 3.75 あるいはそれ以降をお勧めします。

gcc のバージョンを使用していて、それが `-fno-exceptions` オプションを理解できるほど新しい場合は、このオプションを使用することが非常に重要です。それを使用しない場合、無作為にクラッシュするバイナリをコンパイルすることになります。また、`-felide-constructors` および `-fno-rtti` を `-fno-exceptions` と一緒に使用することをお勧めします。疑念がある場合、以下を実行します。

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
-fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-asm \
--with-mysqld-ldflags=-all-static
```

ほとんどのシステムでは、これによって速くて安定したバイナリが得られます。

問題が発生してバグをレポートする必要がある場合には、[質問またはバグの報告](#) の手順に従ってください。

ソースのインストール概要

MySQL のソース ディストリビューションをインストールする際に実行する基本的なコマンドは以下のようになります。

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

ソースの RPM から起動する場合、以下を実行します。

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

これによりインストールできるバイナリの RPM が作成されます。旧バージョンの RPM は、コマンド `rpmbuild` を `rpm` で置き換える必要があります。

注:このプロシージャでは MySQL のアカウントのパスワードは設定しません。以下のプロシージャの後、[インストール後の設定とテスト](#) に進みインストール後の設定およびテストを行います。

上述のソース ディストリビューションを使用した MySQL のインストールに関する詳細な説明は以下のようになります。

1. `mysqld` にログイン ユーザーとグループを追加するには以下を実行します。

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

これらのコマンドにより `mysql` グループと `mysql` ユーザーを追加します。`useradd` および `groupadd` の構文は Unix のバージョンにより多少異なったり、あるいは `adduser` および `addgroup` など異なる名前が付く場合があります。

ユーザーやグループを `mysql` のではなく別の名前に変更することもできます。その場合、以下の手順で別の名前をつけます。

2. ディストリビューションを解凍するディレクトリを選択してロケーションをそれに変更します。
3. ディストリビューションを [MySQL の取得方法](#) の説明に従って取得します。
4. ディストリビューションを現在のディレクトリに解凍します。

```
shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
```

このコマンドで `mysql-VERSION` 名のディレクトリが作成します。

GNU `tar` では、`gunzip` の個別の呼び出しは必要ありません。ディストリビューションを解凍し取り出す際に以下のコマンドも使用できます。

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

5. ロケーションを解凍したディストリビューションの上段のディレクトリに変更します。

```
shell> cd mysql-VERSION
```

現在は MySQL をこの上段のディレクトリから設定してビルドする必要があります。別のディレクトリにビルドすることはできません。

6. リリースを設定してすべてをコンパイルします。

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

`configure` を実行する時、他のオプションを指定みてはいかがでしょう。オプション リストを表示するには `./configure --help` を実行します。[典型的な configure オプション](#)、には更に有用なオプションの説明がいくつかあります。

`configure` が失敗して MySQL メーリング リストに協力を求める際は、その問題の解決に役立つと思われる `config.log` の行を含めてください。また `configure` の出力の少なくとも最後の数行も含めてください。レポートを提出する際は、[質問またはバグの報告](#) の説明を使用してください。

コンパイルに失敗した場合には、[MySQL のコンパイルに関する問題](#) を参照してください。

7. ディストリビューションのインストール

```
shell> make install
```

オプション ファイルを設定する際は、`support-files` ディレクトリにあるどれか一つをテンプレートとして使用します。例えば、


```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

これらのコマンドを `root` として実行する必要があるかも知れません。

InnoDB テーブルのサポートを設定するには、`/etc/my.cnf` ファイルを編集し、`innodb...` で始まるオプション行の前にある `#` 記号を削除し、任意のオプション値に変更します。[オプションファイルの使用](#)、および [InnoDB 設定参照](#)。

8. ロケーションをインストールのディレクトリに変更します。

```
shell> cd /usr/local/mysql
```

9. 今まで MySQL をインストールしたことがない場合には、MySQL 許諾テーブルを作成する必要があります。

```
shell> bin/mysql_install_db --user=mysql
```

コマンドを `root` として実行する場合、以下の `--user` オプションを使用する必要があります。オプションの値はサーバに移動に使用する最初のステップで作成したログイン アカウント名になります。そのユーザーでログインしてそのコマンドを実行する場合、`--user` オプションは無視できます。

`mysql_install_db` を MySQL の許諾テーブルの作成に使用した後、手動でサーバを再起動する必要があります。これを行うための `mysqld_safe` コマンドは後のステップで説明します。

10. プログラム バイナリの所有者を `root` に、データ ディレクトリの所有者を `mysqld` を運用するユーザーに変更します。インストール ディレクトリ (`/usr/local/mysql`) にいると想定した場合、そのコマンドは以下のようになります。

```
shell> chown -R root .
shell> chown -R mysql var
shell> chgrp -R mysql .
```

最初のコマンドはファイルの所有者属性を `root` ユーザーに変更します。2 番目のコマンドはデータ ディレクトリの所有者属性 `mysql` ユーザーに変更します。3 番目のコマンドはグループ属性を `mysql` グループに変更します。

11. マシンをブートしたときに MySQL を自動的に起動する場合は、`support-files/mysql.server` をシステムの起動ファイルがあるロケーションにコピーします。詳細は `support-files/mysql.server` のスクリプトおよび [MySQL を自動的に起動・停止する](#) にもあります。
12. `DBI` および `DBD::mysql` Perl モジュールをインストールする場合、`bin/mysql_setpermission` スクリプトを使用して新しいアカウントを設定できます。その手順は、[Perl のインストールに関する注釈](#) を参照してください。

すべてをインストールしたら、ディストリビューションをテストする必要があります。MySQL サーバを起動するには、以下のコマンドを使用します。

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

そのコマンドが直ぐに失敗し、`mysqld ended` をプリントした場合、そのデータ ディレクトリの `host_name.err` ファイルから情報を入手できます。

`mysqld_safe` に関する詳細は、[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) にあります。

注:MySQL の Grant テーブルのアカウントには最初はパスワードがありません。サーバの起動後に [インストール後の設定とテスト](#) の説明に従ってパスワードをアカウントに設定する必要があります。

典型的な `configure` オプション

`configure` スクリプトを使用することによって MySQL のソース ディストリビューションの設定を柔軟にできます。通常は、`configure` コマンドラインのオプションを使用してこれを行います。また、特定の環境変数を使って `configure` を変更することもできます。[環境変数](#) 参照。 `configure` がサポートしているオプションの完全なリストを表示するには、このコマンドを実行します。

```
shell> ./configure --help
```

利用できる `configure` オプションを以下説明します。

- MySQL クライアント ライブラリおよびクライアント プログラムのみをコンパイルしてサーバを含まない場合、`--without-server` オプションを選択します。

```
shell> ./configure --without-server
```

C++ のコンパイラを持っていない場合、`mysql` などのクライアント プログラムは C++ が必要なためコンパイルできません。この場合 C++ コンパイラをテストするコードを `configure` から削除し、次に `./configure` を `--without-server` オプションで実行します。コンパイルのステップはそれでもすべてのクライアントをビルドしようとしていますが、`mysql.cc` のようなファイルに関する警告は無視して構いません。(もし `make` が停止しても、`make -k` を実行してエラーが発生してもビルドの最後まで継続させます。)

- 埋め込み MySQL ライブラリ (`libmysqld.a`) をビルドするには、`--with-embedded-server` オプションを使用します。
- ログ ファイルやデータベース ディレクトリの配置を `/usr/local/var` に望まない場合、これらの一つに似た `configure` コマンドを使用します。

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
--localstatedir=/usr/local/mysql/data
```

最初のコマンドはすべてがデフォルトの `/usr/local` ではなく `/usr/local/mysql` にインストールされるようインストールの接尾辞を変更します。2 番目のコマンドはデフォルトのインストール接頭辞を保持しますが、デフォルトのロケーションをデータベースのディレクトリ (通常は `/usr/local/var`) にオーバーライドしてそれを `/usr/local/mysql/data` に変更します。

インストールのディレクトリのロケーションとデータ ディレクトリのロケーションも `--basedir` および `--datadir` オプションを使用してサーバの起動時に変更できます。これらは、通常はオプション ファイルを使用するのが一般的ではあるが、コマンドラインあるいは MySQL オプション ファイルで変更可能です。[オプションファイルの使用](#) 参照。

- Unix を使用していて MySQL ソケット ファイルのロケーションをデフォルトのロケーション (通常は `/tmp` あるいは `/var/run` のディレクトリにある) とは別の場所に変更するには、以下のような `configure` コマンドを使用します。

```
shell> ./configure \
--with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

ソケットのファイル名は絶対パス名にする必要があります。MySQL のオプション ファイルを使用してサーバの起動時に `mysql.sock` のロケーションを変更することもできます。[How to Protect or Change the MySQL Unix Socket File](#) 参照。

- 静的にリンクしたプログラム (例えば、バイナリ ディストリビューションの作成、パフォーマンスの向上、あるいは Red Hat Linux ディストリビューションの問題回避) をコンパイルするには、以下のように `configure` を実行します。

```
shell> ./configure --with-client-ldflags=-all-static \
--with-mysqld-ldflags=-all-static
```

- `gcc` を使用していて `libg++` あるいは `libstdc++` インストールしていない場合、`configure` に `gcc` を C++ コンパイラとして使用するよう指定できます。

```
shell> CC=gcc CXX=gcc ./configure
```

`gcc` を C++ コンパイラとして使用すると、`libg++` あるいは `libstdc++` ではリンクしません。それらのライブラリをインストールしていたとしてもこれをしてほうがよいでしょう。それらのバージョンのいくつかはこれまで MySQL ユーザーに予想外な問題を起こしてきました。

以下のリストはそれぞれ一般に使用されてきたいくつかのコンパイラおよび環境変数の設定を示したものです。

- `gcc 2.7.2:`

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

- `egcs 1.0.3a:`

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors \
```

```
-fno-exceptions -fno-rtti"
```

- `gcc` 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

- `pgcc` 2.90.29 あるいはそれ以降

```
CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \
CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \
-felide-constructors -fno-exceptions -fno-rtti"
```

ほとんどの場合、上記のリストにあるオプションを使用して適度に最適化された MySQL のバイナリを得て、以下のオプションを `configure` の行に追加します。

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

フルの `configure` 行は、言い換えると、最近のすべての `gcc` バージョンに対しては多少なりとも以下ようになります。

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

MySQL Web サイト <http://dev.mysql.com/downloads/> で提供しているバイナリはすべて完全な最適化の下でコンパイルしているため、ほとんどのユーザーにパーフェクトに適合します。MySQL AB でコンパイルした MySQL バイナリ 参照。Configuration の設定の中には少し微調整して高速のバイナリをビルドできるものもありますが、しかしこれらの設定は熟練したユーザー向けです。MySQL の速度に対するコンパイルとリンクの影響 参照。

コンパイラでのビルドに失敗してエラーが発生した場合、あるいはリンクが共有ライブラリ `libmysqlclient.so.N` (`N` はバージョン番号) を作成できない場合、`--disable-shared` オプションを `configure` することでこの問題を回避できます。この場合、`configure` は共有 `libmysqlclient.so.N` ライブラリをビルドしません。

- デフォルトでは、MySQL は **latin1** (cp1252 西部ヨーロッパ) 文字セットを使用します。デフォルトの設定を変更するには、**--with-charset** オプションを使用します。

```
shell> ./configure --with-charset=CHARSET
```

CHARSET は次のいずれかで

す。binary、armSCII8、ascii、big5、cp1250、cp1251、cp1256、cp1257、cp850、cp852、cp866、cp932、dec8、eucjpms、euc
 タおよびソート用キャラクタセット参照。(その他の文字セットも利用できます。現在のリストは ./configure --help
 の出力をチェックしてください。)

デフォルトで照合を指定することもできます。MySQL はデフォルトで `latin1_swedish_ci` 照合を使用しています。この設定を変更するには `--with-collation` オプションを使用します。

```
shell> ./configure --with-collation=COLLATION
```

文字セットと照合の両方を変更するには、`--with-charset` および `--with-collation` オプションを使用します。文字セットの照合は規定の照合になります。(各文字列セットにどの照合を使用するかを決めるには `SHOW COLLATION` ステートメントを使用します。

警告: テーブル作成後に文字セットを変更するには `mysamchk -r -q --set-collation=collation_name` をすべての **MyISAM** テーブルに実行する必要があります。さもなければ、インデックスの分類が正しく行われない場合があります。この問題は MySQL のインストール、テーブルの作成、および異なる文字セットを MySQL に設定してそれを再インストールする際に起こります。

`configure` オプション `--with-extra-charsets=LIST` で、他にどの文字セットをサーバにコンパイルするか定義できます。`LIST` は以下のいずれかになります。

- スペースで区切られた文字列セット名のリスト
- `complex` は動的にロードできないすべての文字セットを含む

- `all` はバイナリのすべての文字セットを含む

サーバとクライアントで文字変換するクライアントは `SET NAMES` ステートメントを使用する必要があります。 [SET 構文](#)、および [接続のキャラクタセットおよび照合順序](#) 参照。

- MySQL をデバッグ コードで設定するには `--with-debug` オプションを使用します。

```
shell> ./configure --with-debug
```

これによりエラーを検出し状況に関する出力を提供する安全メモリ アロケータを含むことができます。 [Debugging a MySQL Server](#) 参照。

MySQL 5.1.12 では、`--with-debug` を使用して MySQL のデバッグのサポートを可能にしサーバの起動時に `--debug="d.parser_debug"` オプションを使用できるようにします。これにより SQL ステートメントの処理に使用される Bison parser に parser トレースをサーバの標準エラー出力にダンプさせます。一般的には、この出力はエラーログに記録されます。

- クライアント プログラムがスレッドを使用しているので、MySQL クライアント ライブラリのスレッド セーフなバージョンを `--enable-thread-safe-client` 設定オプションでコンパイルする必要があります。これにより `libmysqlclient_r` ライブラリが作成され、それによってスレッドしたアプリケーションをリンクします。 [スレッド付きクライアントを作る方法](#) 参照。
- `--with-big-tables` オプションを使用して大きなテーブルのサポートが付き MySQL をビルドできます。

このオプションではテーブルの行カウントを保存する変数を `unsigned long` ではなく `unsigned long long` で宣言します。これによりおよそ $1.844\text{E}+19$ ($(2^{32})^2$) 行を 2^{32} ($\sim 4.295\text{E}+09$) 行の代わりに保持できます。以前は `-DBIG_TABLES` コンパイラに手動で渡してこの機能を有効にする必要がありました。

- `configure` を `--disable-grant-options` オプションで実行し、`--bootstrap`、`--skip-grant-tables`、および `--init-file` オプションを `mysqld` に対して無効にします。Windows では、`configure.js` スクリプトが `DISABLE_GRANT_OPTIONS` フラグを認識し、同じ効果を持ちます。この機能は MySQL 5.1.15より利用できます。
- 特殊なオペレーティング システムに関するオプションは、[オペレーティング システムに特化した注釈](#) を参照してください。
- MySQL のセキュア（暗号化した）な接続のサポート設定に関するオプションは、[SSL接続](#) を参照してください。
- `configure` オプションのいくつかはプラグインの選択およびビルドに適用されます。プラグインを静的（サーバにコンパイル）あるいは動的（使用前に `INSTALL PLUGIN` ステートメントでインストールが必要な動的ライブラリとしてビルドされる）なプラグインとしてビルドできます。プラグインの中には静的あるいは動的ビルドをサポートしていないものもあります。

`configure --help` はプラグインの関する以下の情報を網羅しています。

- プラグイン関連オプション
- 利用可能なすべてのプラグイン名
- 各プラグインの、その目的の説明、サポートしているビルドの種類（静的あるいは動的）、およびそれが属するプラグインのグループ

以下の `configure` オプションでプラグインの選択および無効にします。

```
--with-plugins=PLUGIN[,PLUGIN]...
--with-plugins=GROUP
--with-plugin-PLUGIN
--without-plugin-PLUGIN
```

`PLUGIN` は `csv` あるいは `archive` など個別のプラグイン名です。

略語では、`GROUP` は `none`（プラグインの選択なし）あるいは `all`（すべてのプラグインの選択）など設定グループ名を表します。

`--with-plugins` にはコンマ区切られた、あるいはプラグイン グループ名の一つ以上のプラグイン名のリストがあります。名前付きプラグインは静的プラグインとしてのビルド用に構成されています。

`--with-plugin-PLUGIN` は所定のプラグインを静的なプラグインとしてビルドできるように設定します。

`--without-plugin-PLUGIN` は所定のプラグインのビルドを無効にします。

プラグインが `--with` あるいは `--without` オプションの両方で 名前がある場合、その結果は定義できません。

明示的な選択あるいは無効になっていないプラグインは、動的ビルドをサポートしている場合に動的にビルドに選択され、動的ビルドをサポートしていない場合は選択されません。(このように、プラグインのオプションが与えられない場合、動的ビルドをサポートするすべてのプラグインは動的プラグインとしてのビルドに選択されます。動的ビルドをサポートしていないプラグインはビルドされません。)

開発ソース ツリーからのインストール

注意 :このセクションは、当社の新しいコードのテストに協力していただける場合にのみお読みください。単に MySQL を立ち上げてお客様のシステムを運用する場合には、標準のディストリビューション (バイナリあるいはソースのディストリビューション) を使用されるようお願いいたします。

最新の開発ソース ツリーを取得するには、BitKeeper が手元がない場合は最初にダウンロードし、BitKeeper のフリークライアントをインストールします。クライアントは <http://www.bitmover.com/bk-client2.0.shar> から入手できます。BitKeeper フリークライアントをビルドするには `gcc` と `make`、および BitKeeper フリークライアントを使用するには `patch` と `tar` が必要です。BitKeeper フリークライアントの旧 1.1 バージョンは動作しませんのでご注意ください!

Unix に BitKeeper クライアントをインストールするには、以下のコマンドを使用します。

```
shell> /bin/sh bk-client2.0.shar
shell> cd bk-client2.0
shell> make
```

`cc` を取得すると `command not found` エラーが表示されたら、`make` を実行する前に以下のコマンドを実行します。

```
shell> make CC=gcc
```

上記のステップでユーティリティ `bkf` が作成されます。それが BitKeeper フリークライアントです。BitKeeper フリークライアントをメインのクライアントと同じように使用できます。`bkf` に関する詳細は、以下を使用します。

```
shell> bkf --help
```

BitKeeper クライアントを Windows にインストールするには、以下の説明書を使用します。

1. Cygwin を <http://cygwin.com> からダウンロードしてインストールします。
2. `patch`、`tar`、`gcc` および `make` が Cygwin でインストールされていることを確認します。各コマンドに `which gcc` を発行してこれをテストします。必要なツールがインストールされていない場合、Cygwin のパッケージ マネージャを実行して、必要なツールを選択してインストールします。
3. BitKeeper フリークライアントのインストールには、上記の Unix と同様なインストールを行います。

BitKeeper フリークライアントの出荷にはソースコードが同梱されています。そのフリークライアントで利用できる説明資料はそのソースコードのみです。

BitKeeper クライアントをインストールすると、MySQL 開発ソースのツリーにアクセスできます。

1. 使用するディレクトリにロケーションを変更し、以下のコマンドを使用して MySQL 5.1 バージョンのローカルのコピーを取ります。

```
shell> bkf clone bk://mysql.bkbits.net/mysql-5.1 mysql-5.1
```

上記の例で、ソースツリーが現在のディレクトリの `mysql-5.1/` のサブディレクトリに設定されます。

ソースツリーの最初のダウンロードは、接続の速度によって多少時間がかかります。気長にお待ちください。

2. 次のステップのコマンドを実行するには GNU [make](http://www.gnu.org/software/make/)、[autoconf](http://www.gnu.org/software/autoconf/) 2.58 (あるいはそれ以降)、[automake](http://www.gnu.org/software/automake/) 1.8、[libtool](http://www.gnu.org/software/libtool/) 1.5、および [m4](http://www.gnu.org/software/m4/) が必要です。オペレーティングシステムの多くはそれぞれの [make](http://www.gnu.org/software/make/) の実装を備えているとはいえ、予想外のエラーメッセージでコンパイルに失敗する確率が高いといえます。ですから、GNU [make](http://www.gnu.org/software/make/) ([gmake](http://www.gnu.org/software/gmake/) の名前の時もある) を使用することを強くお勧めします。

幸運にも、多くのオペレーティングシステムは GNU toolchain を実装しているか、あるいはこれらをインストールできるパッケージを提供しています。ほとんどの場合、それらは以下のロケーションからダウンロードすることもできます。

- <http://www.gnu.org/software/autoconf/>
- <http://www.gnu.org/software/automake/>
- <http://www.gnu.org/software/libtool/>
- <http://www.gnu.org/software/m4/>
- <http://www.gnu.org/software/make/>

MySQL 5.1 を設定するには、GNU [bison](http://www.gnu.org/software/bison/) も必要です。できる限り最初バージョンの [bison](http://www.gnu.org/software/bison/) を使用してください。バージョン 1.75 とバージョン 2.1 は動作確認が取れています。[bison](http://www.gnu.org/software/bison/) 1.875 には問題があることが報告されています。問題に遭遇したら、以前のバージョンよりは最近のバージョンにアップグレードするほうがよいでしょう。[bison](http://www.gnu.org/software/bison/) 1.75 以前のバージョンではこのエラーあるかも知れません。

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

注: テーブルの最大サイズを実際には超えていなくても、[bison](http://www.gnu.org/software/bison/) の旧バージョンのバグでエラーが発生します。

以下の例はソースツリーの設定に必要な一般的なコマンドを示したものです。最初の `cd` コマンドはロケーションをツリーが一番上のレベルに変更し、`mysql-5.1` を適切なディレクトリ名に置き換えます。2 番目の行は (`storage/innobase` 用) は MySQL 5.1.12 以前のバージョンにのみ必要です。.

```
shell> cd mysql-5.1
shell> (cd storage/innobase; autoreconf --force --install)
shell> autoreconf --force --install
shell> ./configure # Add your favorite options here
shell> make
```

あるいは `BUILD/autorun.sh` を以下のコマンドのシーケンスとして使用できます。

```
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd storage/innobase; aclocal; autoheader; autoconf; automake)
```

ディレクトリを `storage/innobase` ディレクトリに変更するコマンドラインは `InnoDB` ストレージ エンジンの設定に使用されます。`InnoDB` のサポートが必要ない場合にはこの行を削除できます。

注: MySQL 5.1 以降では、ストレージ エンジン専用のコードは `storage` ディレクトリに移動されています。例えば、`InnoDB` コードは現在 `storage/innobase` にあり `NDBCluster` コードは `storage/ndb` にあります。

このステージで予想外の問題が発生した場合には、[libtool](http://www.gnu.org/software/libtool/) が間違いなくインストールされているか確認します。

弊社の標準設定のスクリプト一覧は `BUILD/` サブディレクトリにあります。`BUILD/compile-pentium-debug` スクリプトを使用するほうが上述の `shell` コマンドよりも使い勝手がいいことがお分かりかと思います。異なるアーキテクチャでのコンパイルでは、Pentium 専用のフラグを削除してスクリプトを変更します。

3. ビルドが終了したら、`make install` を実行します。この実行は量産マシンでは注意してください。そのコマンドによって実際のリリースのインストールがオーバーライドされる場合があります。MySQL を別にインストールしている場合には、`./configure` を `--prefix`、`--with-tcp-port`、および `--unix-socket-path` オプションに対して量産サーバに使用した値と異なる値で実行することをお勧めします。
4. インストールしたら色々試し新機能をクラッシュさせてみてください。`make test` の実行から始めてみてください。[MySQL Test Suite](#) 参照。

5. `make` 段階になっても、ディストリビューションのコンパイルができない場合は、[質問またはバグの報告](#) の説明に従ってその問題を弊社のバグ データベースに入力お願いします。必要な GNU のツールの最新バージョンをインストールして、弊社の設定ファイルの処理中にクラッシュする場合には、その件も併せてご連絡をお願いします。しかし、`aclocal` を実行して `command not found` エラーが発生した場合あるいは同様の問題が発生した場合には、レポートしないでください。その代わりに、必要なすべてのツールがインストールされ shell がパスを見つけれられるように `PATH` 変数が正しく設定されているか確認してください。
6. ソースツリーを取得するためにレポジトリを最初に `bkf` でコピーしたら、`pull` オプションを使用して定期的にローカルのコピーを更新する必要があります。レポジトリ設定後にこれを行う時には、以下のコマンドを使用します。

```
shell> bkf pull
```

7. ツリーの変更セットのコマンドを `changes` オプション使用して `bkf` を実行すると確認できます。

```
shell> bkf changes
```

次の `bkf pull` と一緒に使用する変更リストを取得するには

```
shell> bkf changes -R
```

特定の変更セット (`CSETID`) のパッチ ファイルを取得するには、以下を使用

```
shell> bkf changes -vvrCSETID
```

疑問に思う diffs あるいはコードが表示された場合には、MySQL [internals](#) メーリング リストにいつでもメールしてください。[MySQL メーリング リスト](#) 参照。また、何でもいいですから何かよいアイデアが浮かんだ場合には、パッチと一緒にメールを上記のメーリング リストに送ってください。

オンラインでも変更セット、コメント、およびソースコードをブラウズできます。MySQL 5.1 のこの情報をブラウズするには、<http://mysql.bkbits.net:8080/mysql-5.1> に進みます。

MySQL のコンパイルに関する問題

すべての MySQL プログラムの `gcc` による Solaris あるいは Linux のコンパイルは警告なしのクリーンな形で行われます。他のシステムは、システムに含まれるファイルの違いによって警告が出る場合もあります。MIT-pthreads を使用した場合の警告については [MIT-pthreads ノート](#) を参照してください。他の問題に関しては以下のリストをチェックしてください。

多くの問題の解決には再設定が含まれます。再設定が必要な場合には、以下の備考を参照します。

- `configure` を前の実行後に実行する場合、その前に実行中に集められた情報を使用する場合があります。この情報は `config.cache` に保存されます。`configure` が実行されると、そのファイルを探し、コンテンツがある場合にはその情報がまだ正しいとの仮定の下にそのコンテンツを読みます。この仮定は再設定した場合には無効です。
- `configure` を実行するたびに、`make` を再度実行してコンパイルする必要があります。しかし、前のビルドの古いオブジェクト ファイルが異なる設定オプションでコンパイルされている場合、前のビルドの古いオブジェクト ファイルを最初に削除する場合があります。

古い設定情報あるいはオブジェクト ファイルが使用されないようにするには、`configure` を実行する前に以下のコマンドを実行します。

```
shell> rm config.cache
shell> make clean
```

代わりに `make distclean` を実行できます。

以下リストでは MySQL のコンパイル時にこれまでによく発生した問題のいくつかを説明したものです。

- `sql_yacc.cc` をコンパイル中に以下に示すようなエラーが発生した場合には、多分メモリあるいはスワップ スペースの不足です。

```
Internal compiler error: program cc1plus got fatal signal 11
Out of virtual memory
Virtual memory exhausted
```


問題は `gcc` が `sql_yacc.cc` をインライン機能でコンパイルするには非常に大きなメモリが必要だということです。`configure` を `--with-low-memory` オプションで実行してみます。

```
shell> ./configure --with-low-memory
```

このオプションは `gcc` を使用しているときには `-fno-inline` をコンパイル行に追加し、何か他のものを使用している場合には `-O0` を追加します。多分大丈夫だろうと思われる十分なメモリとスワップ スペースがある場合は、`--with-low-memory` オプションを試してみる必要があります。この問題は十分なハードウェア設定のシステムでも発生するところを観察されており、`--with-low-memory` オプションで通常その問題を修正できます。

- デフォルトでは、`configure` が `c++` をコンパイラ名として使用し、GNU `c++` は `-lg++` にリンクします。`gcc` を使用している時に、その振る舞いにより設定中に以下のような問題が発生します。

```
configure: error: installation or configuration problem:
C++ compiler cannot create executables.
```

また `g++`、`libg++`、あるいは `libstdc++` に関連したコンパイル中にも問題が起こる場合もあります。

これらの問題の一つに `g++` がないか、あるいは `g++` があっても `libg++`、あるいは `libstdc++` がないということです。`config.log` ファイルをご覧ください。その中に C++ コンパイラが機能しなかったまさにその理由が書いてあるはずです。これらの問題を回避するには、`gcc` を C++ コンパイラとして使用します。環境変数を `CXX` から `"gcc -O3"` に設定してみてください。例えば、

```
shell> CXX="gcc -O3" ./configure
```

`gcc` が C++ ソース ファイルを `g++` 同様にコンパイルするためこれは機能します。しかしデフォルトでは `libg++` あるいは `libstdc++` ではリンクしません。

これらの問題の別の解決方法は `g++`、`libg++`、および `libstdc++` をインストールすることです。しかし、`libg++` あるいは `libstdc++` を MySQL に使用しないようお勧めします。これらを使用してもなんの効果もないばかりでなく単に `mysqld` のバイナリのサイズを大きくするだけです。これらのバージョンのいくつかでこれまで MySQL ユーザーで予想外な問題が発生しています。

- コンパイルが以下のいずれかで失敗した場合、`make` のバージョンを GNU `make` にアップグレードする必要があります。

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

または

```
make: file 'Makefile' line 18: Must be a separator (:
```

または

```
pthread.h: No such file or directory
```

Solaris および FreeBSD は問題の多い `make` プログラムとして知られています。

GNU `make` 3.75 は動作が確認されています。

- フラグを C あるいは C++ コンパイラで使用するよう定義するにはフラグを `CFLAGS` および `CXXFLAGS` の環境変数に追加します。environment variables.コンパイラ名もこのように `CC` および `CXX` を使用して指定できます。例えば、

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

様々なシステムで有効性が確認されているフラグのリストは、MySQL AB でコンパイルした MySQL バイナリ を参照してください。

- `mysql` をコンパイル中に以下のようなエラーが発生した場合、`configure` が `accept()`、`getsockname()`、あるいは `getpeername()` の最後の引数の種類を正しく検知していなかったことになります。

```
cxx: Error: mysql.cc, line 645: In this statement, the referenced
  type of the pointer value "length" is "unsigned long",
  which is not compatible with "int".
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

この問題を解決するには、`config.h` ファイル (`configure` により生成される) を編集します。以下の行を探します。

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

XXX を `size_t` あるいは `int` にオペレーティングシステムに従って変更します。(`configure` が `config.h` を生成しますので、`configure` を実行するたびにこれを行います。)

- `sql_yacc.cc` ファイルは `sql_yacc.yy` から生成されます。通常は、ビルドプロセスは `sql_yacc.cc` を作成する必要はありません。というのは、MySQL は事前に生成されたコピーがあるからです。しかし、それを再度作成する必要がある場合、この問題に遭遇する場合があります。

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

これは `yacc` のバージョンに問題があることを意味しています。 `bison` (`yacc` の GNU バージョン) をインストールし、代わりにそれを使用する場合もあります。

- Linux 3.0 の Debian では、デフォルトの `mawk` の代わりに `gawk` をインストールする必要があります。
- `mysql` あるいは MySQL クライアントのデバッグの必要がある場合、`configure` を `--with-debug` オプションで実行し、次にコンパイルして新しいクライアントライブラリにリンクします。 [Debugging a MySQL Client](#) 参照。
- Linux (例えば、SuSE Linux 8.1 あるいは Red Hat Linux 7.3) のコンパイルで以下のようなエラーが発生した場合、多分 `g++` がインストールされていません。

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer
without a cast
make[2]: *** [libmysql.lo] Error 1
```

デフォルトでは、`configure` スクリプトが `g++` (GNU C++ コンパイラ) を使用して引数の正しい数字を決めようとします。 `g++` がインストールされていないとこのテストでは正しい結果は出ません。この問題を回避する 2 つの方法があります。

- GNU C++ `g++` がインストールされていることを確認してください。Linux のディストリビューションでは、必要なパッケージは `gpp` を呼ばれており、他では `gcc-c++` となります。
- `CXX` 環境変数を `gcc` に設定して `gcc` を C++ コンパイラとして使用します。

```
export CXX="gcc"
```

それらのいずれかの変更を加えた後に `configure` 再度実行します。

MIT-pthreads ノート

この項では MIT-pthreads を使用した幾つかの問題について説明します。 .

Linux には、MIT-pthreads は使用できません。その代わりに実装した LinuxThreads を使用します。 [Linux の注釈](#) 参照。

システムがネイティブのスレッドをサポートしていない場合、MIT-pthreads パッケージを使用して MySQL を構築する必要があります。これには旧 FreeBSD システム、SunOS 4.x、Solaris 2.4 およびそれ以前、並びにその他幾つかが含まれます。 [MySQL Community Server がサポートしているオペレーティングシステム](#) 参照。

MIT-pthreads は MySQL 5.1 のソースディストリビューションには含まれていません。このパッケージが必要な場合、http://dev.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz から別途ダウンロードします。

ダウンロードしたら、このソース アーカイブを MySQL ソース ディレクトリの最上段に抽出します。抽出すると `mit-pthreads` のサブディレクトリが作成されます。

- ほとんどのシステムで、MIT-pthreads を `configure` を `--with-mit-threads` オプションで実行すると使用できます。

```
shell> ./configure --with-mit-threads
```

このコードへの変更を最小限に抑えるために、MIT-pthreads を使用したソース以外のディレクトリでのビルドはサポートしていません。

- MIT-pthreads の使用を決定するチェックはサーバのコードを処理する設定プロセスの実行中にのみ行われます。 `--without-server` を使用してクライアント コードのみのビルドにディストリビューションを設定した場合、クライアントは MIT-pthreads を使用するかどうかを判断できないため、デフォルトで Unix ソケット接続を使用します。なぜなら、Unix ソケットはプラットフォームによっては MIT-pthreads では動作しないため、クライアントのプログラムを実行する際、 `-h` あるいは `--host` を `localhost` 以外の値で使用する必要があります。
- MySQL を MIT-pthreads でコンパイルした場合、システム ロックはパフォーマンス上の理由でデフォルトで無効になります。システムロックを `--external-locking` オプションで使用するようサーバに指定できます。2 台の MySQL サーバを同じデータ ファイルに稼動する場合にのみこれは必要ですが、いずれにしろこのやり方は推奨していません。
- スレッド `bind()` コマンドがエラーメッセージ (少なくとも Solaris では) なしでソケットとのバインドに失敗する場合があります。その結果サーバへのすべての接続が失敗に終わります。例えば、

```
shell> mysqladmin version
mysqladmin: connect to server at " failed;
error: 'Can't connect to mysql server on localhost (146)'
```

この問題を解決するには `mysqld` サーバを停止してそれを再起動します。これはサーバを強制的に停止して直ぐに再起動した場合にのみ起こります。

- MIT-pthreads の使用により、 `sleep()` のシステム コールの `SIGINT` (ブレーク) での割り込みはありません。これは `mysqladmin --sleep` を実行すると分かります。割り込みが行われプロセスが停止するまで `sleep()` コールが終了するのを待つ必要があります。
- リンク時に以下の警告メッセージ (少なくとも Solaris で) が表示される場合がありますが、無視して構いません。

```
ld: warning: symbol `__job' has differing sizes:
(file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
/my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
ld: warning: symbol `__job' has differing sizes:
(file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
/my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- その他の警告も無視して構いません。

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- MIT-pthread と連動する `readline` はまだありません。(これは必要ありませんが、興味あるユーザーがいるかも知れません。)

Windows にソースから MySQL をインストールする

これらの説明書は Windows 上でのソースからの MySQL 5.1 用バイナリのビルド方法を網羅しています。説明書では標準のソース ディストリビューションあるいは最新の開発ソースを含む BitKeeper ツリーからのバイナリのビルドを説明します。

注:これらの説明書は Microsoft 社の Windows 上で最新のソース ディストリビューションあるいは BitKeeper ツリーを MySQL でテストするユーザーに限定したものです。MySQL AB ではソースからユーザー自身がビルドした MySQL サーバを量産環境で使用することはお勧めしていません。通常は、MySQL AB で Windows 専用に最適なパフォーマンスでビルドした MySQL のプリコンパイル版のバイナリのディストリビューションを使用されるのがベストです。バイナリ ディストリビューションのインストール手順は [Windows に MySQL をインストールする](#) を参照してください。

MySQL を Windows 上でソースからビルドするには、以下のシステム、コンパイラ、およびソース要件を満たす必要があります。

- Windows 2000、Windows XP、あるいは以降のバージョン Windows Vista は Microsoft 社が Visual Studio 2005 を Vista で検証するまではサポートしていません。
- CMake は <http://www.cmake.org> からダウンロードできます。インストール後に、パスを変更して `cmake` バイナリを含めます。
- Microsoft Visual C++ 2005 Express Edition、Visual Studio .Net 2003 (7.1)、あるいは Visual Studio 2005 (8.0) コンパイラ システム。
- Visual C++ 2005 Express Edition を使用している場合、適切な Platform SDK もインストールする必要があります。詳細および利用できる様々な Windows プラットフォームのダウンロードのリンクは <http://msdn.microsoft.com/platformsdk/> で利用できます。
- BitKeeper ツリーからのコンパイルあるいは parser に変更を加える場合には、Windows 用 `bison` が必要です。それは <http://gnuwin32.sourceforge.net/packages/bison.htm> からダウンロードできます。「Complete package, excluding sources」のラベルのパッケージをダウンロードします。パッケージをインストールしたら、パスを変更して `bison` バイナリを含め、このバイナリに Visual Studio からアクセスできることを確認します。
- テスト スクリプトを実行したり、コンパイルしたバイナリおよびサポート ファイルを Zip アーカイブにパッケージするには Cygwin が必要になる場合があります。(Cygwin はディストリビューションをテストあるいはパッケージする際のみ必要で、ビルドには必要ありません。)Cygwin は <http://cygwin.com> から入手できます。
- 3GB から 5GB のディスク容量

正確なシステム要件は以下にあります。 <http://msdn.microsoft.com/vstudio/Previous/2003/sysreqs/default.aspx> および <http://msdn.microsoft.com/vstudio/products/sysreqs/default.aspx>

Windows 用の MySQL ソース ディストリビューションも必要です。それは 2 つの方法で取得できます。

- MySQL AB からソースのディストリビューションを入手する。これらは <http://dev.mysql.com/downloads/> にあります。
- 最新の BitKeeper 開発ソースツリーからご自身でソース ディストリビューションをパッケージする。最新のソース ファイルに関する説明書は、[開発ソース ツリーからのインストール](#) を参照してください。

何か期待通りに動作しない、あるいは Windows を使用した現在のビルド プロセスの改善のための助言が必要な場合には、[win32](#) メーリング リストにメッセージを送ってください。[MySQL メーリング リスト](#) 参照。

CMake および Visual Studio を使用したソースからの MySQL のビルド

MySQL をビルドするには以下の手順に従います。

1. パッケージのディストリビューションからインストールする場合、作業ディレクトリ (例えば、`C:\workdir`) を作成し、`WinZip` あるいは `.zip` ファイルを読める別の Windows のツールを使用してソース ディストリビューションを作業ディレクトリに解凍します。このディレクトリは作業ディレクトリでその手順は以下のようになります。
2. BitKeeper ツリーからインストールする場合、そのツリーのルート ディレクトリは作業ディレクトリで以下の手順に従います。
3. コマンド シェルを使用するには、作業ディレクトリをナビゲートして以下のコマンドを実行します。

```
C:\workdir>win\configure options
```

以下のオプションが利用できます。

- `WITH_INNOBASE_STORAGE_ENGINE:InnoDB` ストレージ エンジン を有効にします。
- `WITH_PARTITION_STORAGE_ENGINE`: ユーザー定義のパーティション化を有効にします。
- `WITH_ARCHIVE_STORAGE_ENGINE:ARCHIVE` ストレージ エンジン を有効にします。
- `WITH_BLACKHOLE_STORAGE_ENGINE:BLACKHOLE` ストレージ エンジン を有効にします。

- `WITH_EXAMPLE_STORAGE_ENGINE:EXAMPLE` ストレージ エンジンを有効にします。
- `WITH_FEDERATED_STORAGE_ENGINE:FEDERATED` ストレージ エンジンを有効にします。
- `__NT__`: 名前付きパイプのサポートを有効にします。
- `MYSQL_SERVER_SUFFIX=suffix`: サーバ接尾辞、デフォルトはなし
- `COMPILATION_COMMENT=comment`: Server コメント、デフォルトは「ソース ディストリビューション」
- `MYSQL_TCP_PORT=port`: サーバ ポート、デフォルトは 3306。
- `DISABLE_GRANT_OPTIONS:--bootstrap`、`--skip-grant-tables`、および `--init-file` オプションを `mysqld` に対して無効にします。このオプションは MySQL 5.1.15 から利用できます。

例えば (一つの行にコマンドを入力します):

```
C:\workdir>win\configure WITH_INNOBASE_STORAGE_ENGINE
WITH_PARTITION_STORAGE_ENGINE MYSQL_SERVER_SUFFIX=-pro
```

4. 作業でディレクトリから、インストールした Visual Studio のバージョンに基づいて `win\build-vs8.bat` あるいは `win\build-vs71.bat` ファイルを実行します。スクリプトが CMake を実行し、`mysql.sln` のソリューション ファイルを生成します。

また `win\build-vs8_x64.bat` を使用して 64-ビット バージョンの MySQL をビルドできます。しかし、64-ビット バージョンを Visual Studio Express Edition ではビルドできません。Visual Studio 2005 (8.0) あるいはそれ以上を使用する必要があります。

5. 作業ディレクトリで、生成された `mysql.sln` ファイルを Visual Studio で開いて、Configuration メニューで適切な設定を選択します。メニューには デバッグ、リリース、RelwithDebInfo、MinRelInfo オプションがあります。次に ソリューション > ビルド を選択してソリューションをビルドします。

このステップで使用した設定を覚えておきます。これは後でスクリプトを実行するときに重要になります。というのは、スクリプトはどの設定になっているか知る必要があるからです。

6. サーバのテスト。上記の手順でビルドしたサーバの MySQL ベース ディレクトリおよびデータ ディレクトリはデフォルトで `C:\mysql` および `C:\mysql\data` になります。サーバをソースのツリー ルートおよびそのデータディレクトリをベースのディレクトリおよびデータ ディレクトリとして使用してテストするには、サーバにそれらのパス名を指定する必要があります。これは `--basedir` オプションおよび `--datadir` オプションのコマンドライン、あるいはオプション ファイルで適切なオプションを設定することで指定できます。(オプションファイルの使用 参照。)使用したい既存のデータ ディレクトリがどこかにある場合、代わりにそのパス名を指定します。

サーバをスタンドアロンで稼働している場合あるいは設定上サービス ベースで運用している場合、`mysql` のインタラクティブ コマンドライン ユーティリティからそれに接続します。

標準のテスト スクリプト、`mysql-test-run.pl` を実行することもできます。このスクリプトは Perl で書かれているので、それを実行するには Cygwin あるいは ActiveState Perl のいずれかが必要です。またこのスクリプトに必要なモジュールをインストールしなければならない場合もあります。テスト スクリプトを実行するには、ロケーションを作業ディレクトリの `mysql-test` ディレクトリに変更し、`MTR_VS_CONFIG` 環境変数を以前選択した (あるいは `--vs-config` オプションを使用し) 設定を選択して、`mysql-test-run.pl` を実行します。例えば (Cygwin および `bash` シェルを使用して):

```
shell> cd mysql-test
shell> export MTS_VS_CONFIG=debug
shell> ./mysqltest-run.pl --force --timer
shell> ./mysqltest-run.pl --force --timer --ps-protocol
```

ビルドしたプログラムが正常に機能していることに満足したら、サーバを停止します。この段階でディストリビューションをインストールできます。インストールを行う一つの方法として MySQL のソース ディストリビューションの `scripts` ディレクトリにある `make_win_bin_dist` スクリプトを使用する方法があります (`make_win_bin_dist` — Package MySQL 配布 (ZIP アーカイブ) 参照)これはシェル スクリプトですので、これを使用する際は Cygwin をインストールする必要があります。それによりビルドした実行ファイルの Zip アーカイブを作成し、MySQL をインストールするロケーションに解凍するファイルをサポートします。

ディレクトリやファイルを直接コピーして MySQL をインストールすることもできます。

1. MySQL をインストールするディレクトリを作成します。例えば、`C:\mysql` にインストールするには、このコマンドを使用します。

```
C:\> mkdir C:\mysql
C:\> mkdir C:\mysql\bin
C:\> mkdir C:\mysql\data
C:\> mkdir C:\mysql\share
C:\> mkdir C:\mysql\scripts
```

他のクライアントをコンパイルしてそれらを MySQL にリンクさせるには、さらに幾つかのディレクトリを作成する必要があります。

```
C:\> mkdir C:\mysql\include
C:\> mkdir C:\mysql\lib
C:\> mkdir C:\mysql\lib\debug
C:\> mkdir C:\mysql\lib\opt
```

MySQL をベンチマークするには、このディレクトリを作成します。

```
C:\> mkdir C:\mysql\sql-bench
```

ベンチマークするには Perl のサポートが必要です。[Perl のインストールに関する注釈](#) 参照。

2. 作業ディレクトリから、以下のディレクトリを `C:\mysql` ディレクトリにコピーします。

```
C:\> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
C:\workdir> xcopy scripts\*. * C:\mysql\scripts /E
C:\workdir> xcopy share\*. * C:\mysql\share /E
```

他のクライアントをコンパイルしてそれらを MySQL にリンクさせるには、幾つかのライブラリとヘッダーファイルもコピー必要があります。

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

MySQL をベンチマークするには、以下が必要です。

```
C:\workdir> xcopy sql-bench\*. * C:\mysql\bench /E
```

インストールしたら、バイナリの Windows ディストリビューションと同じようにサーバを設定して起動します。[Windows に MySQL をインストールする](#) 参照。

Borland C++ を使用したソースからの MySQL のビルド

MySQL Windows のソースを Borland C++ 5.02 でコンパイルできます。(Windows のソースは Microsoft VC++ 用のプロジェクトしか含んでいないため、Borland C++ の場合にはご自身でプロジェクトファイルを作成する必要があります。

Borland C++ の既知の問題の一つは、VC++ とは構成の配列が異なっていることです。このとは、デフォルトの `libmysql.dll` ライブラリ (VCC++ でコンパイルした) を Borland C++ に使用すると問題が発生するということを意味します。この問題を避けるには、`mysql_init()` を `NULL` で引数としてコールするだけで、事前に割り当てられた `MYSQL` 構成としてコールしないことです。

Windows で MySQL クライアントをコンパイルする

ソース ファイルには、`my_global.h` を `mysql.h` の前に含める必要があります。

```
#include <my_global.h>
```



```
#include <mysql.h>
```

[my_global.h](#) には Windows でプログラムをコンパイルするために必要な Windows 互換 ([windows.h](#) など) の他のファイルが含まれています。

コードをオンデマンドで [libmysql.dll](#) にロードする単なるラッパーである動的な [libmysql.lib](#) ライブラリにリンクさせるか、あるいは静的な [mysqlclient.lib](#) ライブラリにリンクできます。

MySQL クライアント ライブラリはスレッド ライブラリとしてコンパイルされますので、コードもマルチ スレッドとしてコンパイルする必要があります。

インストール後の設定とテスト

MySQL のインストール後に、処理しなければならない問題がいくつかあります。例えば、Unix では、データ ディレクトリを初期化して MySQL グラント テーブルを作成する必要があります。すべてのプラットフォームで、重要なセキュリティの問題はグラント テーブルの最初のアカウントにはパスワードがないということです。MySQL サーバへの権限のないアクセスを妨げるにはパスワードを割り当てる必要があります。オプションで、タイム ゾーン テーブルを作成して名前付きタイム ゾーンの認識を有効にできます。

次項以降では、Windows システムおよび Unix システムに特定したインストール後の手順について説明します。別のセクション、[MySQL サーバの起動とトラブルシューティング](#) ではすべてのプラットフォームでのサーバの起動時の問題について説明します。[最初の MySQL アカウントの確保](#) もすべてのプラットフォームに関する説明です。MySQL アカウントをアカウントにパスワードを割り当てて適切に保護していることを確認するには以下の手順に従います。

新たにユーザー アカウントを作成する用意ができたら、[MySQL アクセス権限システム](#) および [MySQL ユーザ アカウント管理](#) にある MySQL アクセス管理システムに関する情報およびアカウント管理を参照してユーザーアカウントを設定します。

Windows のインストール後のプロシージャ

Windows では、データ ディレクトリおよびグラント テーブルは作成する必要はありません。MySQL の Windows のディストリビューションにはデータ ディレクトリの [mysql](#) データベースに初期化されたアカウント セットのグラント テーブルが含まれています。Unix で実行した [mysql_install_db](#) スクリプトを実行する必要はありません。パスワードの場合、MySQL を Windows のインストール ウィザードでインストールした場合、そのインストールが済むとそのアカウントに対するパスワードの割り当ては済んでいます。([MySQL インストール ウィザードを使用する](#) 参照。) そうでない場合には、[最初の MySQL アカウントの確保](#) にあるパスワード割り当て手順を使用します。

パスワードを設定する前に、クライアント プログラムを実行してサーバに接続できるかまたは適切に動作しているか確認する必要があります。サーバが稼働していること ([サーバを最初に起動する](#) 参照) を確認し、次に以下のコマンドを発行してサーバから情報を取り出せるか確認します。その出力は以下に示すものと類似しているはずです。

```
C:\> C:\mysql\bin\mysqlshow

+-----+
| Databases |
+-----+
| mysql |
| test  |
+-----+

C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| db            |
| func          |
| help_category |
| help_keyword  |
| help_relation |
| help_topic    |
| host          |
| proc          |
| procs_priv    |
```

```

| tables_priv |
| time_zone |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user |
+-----+
C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
+-----+-----+-----+
| host | db | user |
+-----+-----+-----+
| % | test% | |
+-----+-----+-----+

```

サービスをサポートしている Windows バージョンを使用していて、MySQL サーバを Windows の自動時に起動する場合には、[Windows のサービスとして MySQL を起動する](#) を参照してください。.

Unix のインストール後のプロシージャ

MySQL を Unix にインストールしたら、グラントテーブルを初期化し、サーバを起動し、サーバが期待通りに動作しているか確認します。サーバをシステムの起動、停止と共に自動的に起動あるいは停止させることもできます。グラントテーブルのアカウントにパスワードを割り当てることもできます。

Unix では、グラントテーブルは [mysql_install_db](#) プログラムで設定します。インストールの仕方によっては、このプログラムを自動的に実行することができます。

- MySQL を Linux に RPM ディストリビューションを使用してインストールする場合、サーバの RPM が [mysql_install_db](#) を実行します。
- MySQL を Mac OS X に PKG ディストリビューションを使用してインストールする場合、インストーラが [mysql_install_db](#) を実行します。

そうでない場合は、ご自身で [mysql_install_db](#) を実行します。

以下のプロシージャではグラントテーブルの初期化(初期化されていない場合)、次にサーバの起動手順について説明します。同時にサーバへのアクセスおよびサーバの正常動作をテストする幾つかのコマンドも示してあります。サーバの自動による起動および停止に関する情報は、[MySQL を自動的に起動・停止する](#) を参照してください。

プロシージャを完了してサーバを動作させたら、[mysql_install_db](#) で作成したアカウントにパスワードを割り当てます。その手順は、[最初の MySQL アカウントの確保](#) にあります。

以下の例では、サーバは[mysql](#) のログイン アカウントのユーザーID で動作します。これはそのようなアカウントが存在することを前提にしています。アカウントがない場合アカウントを作成するか、あるいはサーバの稼働に使用する別の既存のログイン アカウントで置き換えます。

1. ロケーションを MySQL の最上段のディレクトリに変更すると、ここでは [BASEDIR](#) になります。

```
shell> cd BASEDIR
```

[BASEDIR](#) は [/usr/local/mysql](#) あるいは [/usr/local](#) のようなものです。以下のステップはユーザーがこのディレクトリにあるものとの前提に基づいたものです。

2. 必要に応じて [mysql_install_db](#) プログラムを実行して、ユーザーのサーバ接続へのアクセス方法を決定する権限を含む最初の MySQL グラントテーブルを設定します。インストールのプロシージャではそのプログラムが起動しないディストリビューションを使用している場合にこの手順が必要です。

一般的には、[mysql_install_db](#) はMySQL を最初にインストールする際にのみ実行する必要があるため、既存のインストールをアップグレードした場合にはこのステップをスキップできます。しかし、[mysql_install_db](#) は既存の権限テーブルを上書きしないので、どの環境でもこれを実行するのが安全かも知れません。

グラントテーブルを初期化するには、[mysql_install_db](#) が [bin](#) にあるかあるいは [scripts](#) ディレクトリにあるかによって以下のコマンドのいずれかを使用します。

```
shell> bin/mysql_install_db --user=mysql
shell> scripts/mysql_install_db --user=mysql
```

`mysql_install_db` のスクリプトはサーバのデータ ディレクトリを作成します。サーバのディレクトリはすべてのデータベース権限および MySQL のテストに使用する `test` データベースを保持する `mysql` データベースのディレクトリを作成します。そのスクリプトはまた `root` および匿名ユーザーアカウントの権限テーブルのエントリを作成します。それらのアカウントには最初はパスワードはありません。最初の権限に関する説明は [最初の MySQL アカウントの確保](#) にあります。これらの権限では一時的に、MySQL `root` ユーザーが何かをしたり、誰でも `test` 名の下にデータベースを作成して使用したり、あるいは `test_` 名で動作できるようになっています。

あとでサーバを起動した時にサーバがそれらのアクセスを読んだり書いたりできるようにデータベースのディレクトリおよびファイルが `mysql` のログイン アカウントの所有になっていることを確認しておくことが重要です。これを確認するには、`--user` オプションは `mysql_install_db` を `root` として実行する際に以下のように使用される必要があります。または、`mysql` としてログインする際にそのスクリプトを実行する必要があります。その場合 `--user` オプションをコマンドから除外できます。

`mysql_install_db` は幾つかのテーブルを `mysql` データベースに作成します。その中には `user`、`db`、`host`、`tables_priv`、`columns_priv`、`func`、およびその他が含まれます。これらのテーブルの完全なリストおよびその説明は、[MySQL アクセス権限システム](#) を参照してください。

`test` データベースが必要ない場合には、サーバの起動後に `mysqladmin -u root drop test` でそれを削除できます。

この段階で `mysql_install_db` で問題に遭遇した場合、[mysql_install_db 実行中の問題](#) を参照してください。

3. MySQL サーバを起動する

```
shell> bin/mysqld_safe --user=mysql &
```

MySQL サーバを権限のない (非`root`) ログイン アカウントで起動することが重要です。これを確認するには、`--user` オプションは `mysql_safe` をシステム `root` として実行する際に以下のように使用される必要があります。または、`mysql` としてログインする際にそのスクリプトを実行する必要があります。その場合 `--user` オプションをコマンドから除外できます。

MySQL を権限のないユーザーとして動作させるための詳細は [ユーザによる MySQL の実行](#) にあります。

このステップに進む前にグラント テーブルを作成しなかった場合には、サーバを起動すると以下のメッセージがエラーのログ ライルに表示されます。

```
mysqld: Can't find file: 'host.frm'
```

サーバの起動時に他の問題が発生した場合には、[MySQL サーバの起動とトラブルシューティング](#) を参照してください。

4. `mysqladmin` を使用してサーバが動作していることを確認します。以下のコマンドにはサーバの起動および接続をチェックする簡単なテストが含まれています。

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

`mysqladmin version` の出力はプラットフォームおよび MySQL のバージョンによって多少異なりますが以下に類似します。

```
shell> bin/mysqladmin version
mysqladmin Ver 14.12 Distrib 5.1.15-beta, for pc-linux-gnu on i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version      5.1.15-beta-Max
Protocol version    10
Connection          Localhost via UNIX socket
UNIX socket         /var/lib/mysql/mysql.sock
Uptime:             14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
```

Queries per second avg: 0.000

`mysqladmin` の他に機能については、それを `--help` オプションで起動します。

5. サーバをシャットダウンできることを確認します。

```
shell> bin/mysqladmin -u root shutdown
```

6. サーバを再度起動できるか確認します。 `mysqld_safe` を使用するかあるいは `mysqld` 直接実行して起動を確認します。例えば、

```
shell> bin/mysqld_safe --user=mysql --log &
```

`mysqld_safe` が失敗したら、[MySQL サーバの起動とトラブルシューティング](#) を参照します。

7. 簡単なテストをしてサーバから情報を取り出せるか確認します。その出力は以下に示すものと類似しているはずで

```
shell> bin/mysqlshow
+-----+
| Databases |
+-----+
| mysql    |
| test     |
+-----+

shell> bin/mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| db           |
| func         |
| help_category |
| help_keyword |
| help_relation |
| help_topic   |
| host         |
| proc         |
| procs_priv   |
| tables_priv  |
| time_zone    |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user         |
+-----+

shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+-----+
| host | db | user |
+-----+
| %    | test | |
| %    | test_% | |
+-----+
```

8. [sql-bench](#) ディレクトリ (MySQL インストール ディレクトリ) に MySQL の異なるプラットフォームでの違いを比較したベンチマーク スイートがあります。ベンチマーク スイートは Perl で書かれています。それには様々なデータベース、および他の幾つかの Perl モジュールへのデータベース非依存型インターフェースを提供する Perl DBI が必要です。

```
DBI
DBD::mysql
Data::Dumper
Data::ShowTable
```

これらのモジュールは CPAN (<http://www.cpan.org/>) で入手できます。[Unix に Perl をインストールする](#) も参照してください。

[sql-bench/Results](#) ディレクトリには異なるデータベースおよびプラットフォームでの多くの動作結果が含まれています。すべてのテストを行うには、以下のコマンドを実行します。

```
shell> cd sql-bench
shell> perl run-all-tests
```

[sql-bench](#) ディレクトリがない場合、多分 MySQL をソースの RPM ではない RPM ファイルからインストールしています。(ソースの RPM には [sql-bench](#) ベンチマーク ディレクトリが含まれています。)この場合、それを使用する前に最初にベンチマーク スイートをインストールする必要があります。[mysql-bench-VERSION-i386.rpm](#) の名前の個別のベンチマーク RPM ファイルがあり、その中にはベンチマークのコードおよびデータが含まれています。

ソースのディストリビューションの場合には、実行できるテストも [tests](#) サブディレクトリにあります。例えば、[auto_increment.tst](#) を実行するには、このコマンドをソース ディストリビューションの最上段のディレクトリから実行します。

```
shell> mysql -vvf test < ./tests/auto_increment.tst
```

テストの予想結果は [./tests/auto_increment.res](#) ファイルにあります。

- この段階では、サーバが稼動していなければなりません。しかし、最初の MySQL アカウントにはどれもパスワードがないため、[最初の MySQL アカウントの確保](#) の手順に従ってパスワードを割り当てる必要があります。

MySQL 5.1 のインストール プロシージャが [mysql](#) データベースでタイム ゾーン テーブルを作成します。しかし、テーブルは [MySQL サーバのタイム ゾーン サポート](#) の手順に従って手動で構成する必要があります。

mysql_install_db 実行中の問題

[mysql_install_db](#) スクリプトの目的は新しい MySQL 権限テーブルを生成することです。それは既存の MySQL 権限テーブルを上書きしないので、他のデータに影響を及ぼすことはありません。

権限のテーブルを再度作成するには、稼動している [mysqld](#) サーバを停止します。次にデータ ディレクトリの [mysql](#) ディレクトリの名前を変更して保存し、次に [mysql_install_db](#) を実行します。お客様の現在のディレクトリが MySQL のインストール ディレクトリで [mysql_install_db](#) が [bin](#) ディレクトリにあり、データ ディレクトリ名が [data](#) のであるとします。[mysql](#) データベースの名前を変更し [mysql_install_db](#) を実行するには、以下のコマンドを使用します。

```
shell> mv data/mysql data/mysql.old
shell> bin/mysql_install_db --user=mysql
```

[mysql_install_db](#) を実行すると、以下の問題が発生する場合があります。

- [mysql_install_db](#) によるグラント テーブルのインストールの失敗

[mysql_install_db](#) がグラント テーブルのインストールに失敗し以下のメッセージの表示後に終了する場合があります。

```
Starting mysqld daemon with databases from XXXXXX
mysqld ended
```

この場合、エラーのログ ファイルを非常に慎重に調べる必要があります。ログ ファイルはエラーメッセージの名前が付いてディレクトリ [XXXXXX](#) に格納され、[mysqld](#) が起動しなかった理由が示されます。どんな問題が発生したか分からない場合、バグ レポートをポストする時にログを一緒に送ります。[質問またはバグの報告](#) 参照。

- [mysqld](#) プロセスが実行されている

これはサーバが稼動していることを示しています。この場合グラント テーブルは多分既に作成されています。その場合、[mysql_install_db](#) を起動する必要はまったくありません。なぜなら、それは一度だけ起動する必要があるからです (最初に MySQL をインストールした場合)。

- 1 台のサーバの起動中に 2 番目の [mysqld](#) サーバのインストールはできません。

これは既存の MySQL のインストールがあり、異なるロケーションに新たにインストールを行う場合に起こります。例えば、量産 (実稼働中) インストールが既にあり、テスト目的に 2 番目のインストールを希望される場合で

す。一般的に 2 番目のサーバを起動しようとした時に起こる問題は 2 台目のサーバが 1 台目のサーバが使用しているネットワーク インターフェースを使用しようとした場合に発生します。この場合、以下のエラーメッセージのいずれかが表示されます。

```
Can't start server: Bind on TCP/IP port:
Address already in use
Can't start server: Bind on unix socket...
```

複数のサーバの設定に関する説明は、[同じマシン上での複数 MySQL サーバの実行](#) 参照してください。

- あなたには `/tmp` ディレクトリへの書き込みアクセスがありません

一時ファイル作成の書き込みアクセスがない場合あるいは Unix ソケット ファイルがデフォルトのロケーション (`/tmp` ディレクトリ) にない場合に、`mysql_install_db` を実行したり `mysqld` サーバを起動するとエラーが発生します。

一時ディレクトリまたは Unix のソケット ファイルに `mysql_install_db` あるいは `mysqld` を起動する前に以下のコマンドを実行して別のロケーションを指定できます。そこでは `some_tmp_dir` は許可を書いたディレクトリへのフルのパス名です。

```
shell> TMPDIR=/some_tmp_dir/
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

次に `mysql_install_db` を起動し以下のコマンドでサーバを起動します。

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

If `mysql_install_db` が `scripts` ディレクトリにある場合には、最初のコマンドを `scripts/mysql_install_db` に変更します。

[How to Protect or Change the MySQL Unix Socket File](#)、および [環境変数](#) 参照。

MySQL で提供された `mysql_install_db` スクリプトを実行する幾つかの代案があります。

- 最初の権限を標準のデフォルトと別にしたい場合、それを実行する前に `mysql_install_db` を変更します。しかし、[GRANT](#) および [REVOKE](#) を GRANT テーブルを設定した後に 使用して権限を変更することが好ましい。換言すると、`mysql_install_db` を起動し、次に `mysql -u root mysql` を使用してサーバに MySQL `root` ユーザーとして接続します。これで必要な [GRANT](#) および [REVOKE](#) ステートメントを発行できます。

同じ権限で複数のマシンに MySQL をインストールする場合、[GRANT](#) および [REVOKE](#) ステートメントを一つのファイルに入れ、そのファイルを `mysql` を使用してスクリプトとして `mysql_install_db` を起動後に実行します。例えば、

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysql -u root < your_script_file
```

このようにすることで、各マシンでステートメントを手動で発行する必要がなくなります。

- GRANT テーブルを作成した後に再度完全に作成できます。これらは `MySQL_install_db` を起動後に [GRANT](#) および [REVOKE](#) の使用の仕方を学ぶために色々変更を加えその後にテーブルからそれらを変更を削除して起動する再に必要なになります。

GRANT テーブルを再度作成するには、`mysql` データベース ディレクトリにあるすべての `.frm`、`.MYI`、および `.MYD` ファイルのすべて削除します。次に `mysql_install_db` スクリプトを再度実行します。

- `mysqld` を `--skip-grant-tables` オプションを使用して手動で起動して `mysql` を使用してご自身で権限情報を追加することができます。

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

`mysql` から、`mysql_install_db` の SQL コマンドを手動で実行できます。`mysqladmin flush-privileges` あるいは `mysqladmin reload` を後で実行して GRANT テーブルをリロードするようサーバに指定します。

`mysql_install_db` を使用しなかった場合は、`grant` テーブルを手動で作成しなければならないのみならず、それらを最初に作成しなければなりません。

MySQL を自動的に起動・停止する

一般的には `mysqld` サーバを以下のいずれかで起動します。

- `mysqld` を直接実行します。これはプラットフォームで行われます。
- MySQL サーバを Windows のサービスで稼働します。これはサービスをサポートしている Windows (NT、2000、XP、および 2003 など) で可能です。このサービスでは Windows を起動したときにサーバを自動的に起動する、あるいは手動サービスとして要求に基づいて起動するように設定できます。その手順は、[Windows のサービスとして MySQL を起動する](#) を参照してください。
- `mysqld_safe` を実行すると、`mysqld` の適切なオプションを判断してそれらのオプションでそれを実行します。このスクリプトは Unix および Unix 様のシステムで使用できます。[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) 参照。
- `mysql.server` の起動。このスクリプトはシステム V-style 実行ディレクトリを使用しているシステムのシステムの起動時およびシャットダウン時に主に使用されます。それは通常 `mysql` の名前でインストールされます。`mysql.server` スクリプトは `mysqld_safe` を実行してサーバを起動します。[mysql.server — MySQL サーバ スタートアップ スクリプト](#) 参照。
- Mac OS X では、システムの起動で MySQL を自動的に起動する個別の MySQL 起動アイテムパッケージをインストールできます。起動アイテムは `mysql.server` を実行するとサーバを起動します。詳細は、[Mac OS X に MySQL をインストールする](#) を参照してください。

`mysqld_safe` および `mysql.server` スクリプト並びに Mac OS X 起動アイテムでシステムの起動時にサーバを手動、あるいは自動的に起動できます。`mysql.server` および起動アイテムはサーバを停止することもできます。

`mysql.server` スクリプトを使用してサーバを手動で起動・停止するには、サーバを `start` あるいは `stop` 引数で呼び出します。

```
shell> mysql.server start
shell> mysql.server stop
```

`mysql.server` がサーバを起動する前に、ロケーションを MySQL インストール ディレクトリに変更し、次に `mysqld_safe` を実行します。サーバを特定のユーザーとして起動するには、適切な `user` オプションを `/etc/my.cnf` オプション ファイルの `[mysqld]` グループにこの項の後で示すように追加します。(バイナリ ディストリビューションの MySQL を標準と異なるロケーションにインストールした場合には `mysql.server` の編集が必要になる場合があります。`mysqld_safe` を実行する前にそれを `cd` に変更して適切なディレクトリに入れます。これを行うと、変更したバージョンの `mysql.server` は将来 MySQL をアップグレードすると上書きされる場合がありますので、編集したバージョンをインストールできるようにコピーを取っておく必要があります。

`mysql.server stop` は停止する信号を送ってサーバを停止します。`mysqladmin shutdown` を実行してサーバを手動で停止することもできます。

サーバ上の MySQL を自動的に起動・停止するには、起動・停止コマンドを `/etc/rc*` ファイルの適切な場所に加える必要があります。

Linux サーバ RPM パッケージ (`MySQL-server-VERSION.rpm`) を使用する場合には、`mysql.server` スクリプトを `/etc/init.d` ディレクトリに `mysql` の名前でインストールします。それを手動でインストールする必要はありません。Linux RPM パッケージに関する詳細は、[Linux に MySQL をインストールする](#) を参照してください。

ベンダーによっては起動スクリプトをインストールする RPM パッケージを `mysqld` のような別名で提供している場合もあります。

MySQL をソース ディストリビューションからあるいは `mysql.server` を自動的にインストールしないバイナリのディストリビューション フォーマットを使用してインストールする場合、それを手動でインストールできます。そのスクリプトは MySQL インストール ディレクトリの `support-files` ディレクトリあるいは MySQL のソース ツリーにあります。

`mysql.server` を手動でインストールするには、それを `/etc/init.d` ディレクトリに `mysql` の名前でコピーし、次にそれを実行ファイルにします。実行ファイルにするには `mysql.server` を格納を格納しこれらのコマンドを実行する適切なディレクトリにロケーションを変更します。

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

古い Red Hat システムは `/etc/init.d` ではなく `/etc/rc.d/init.d` ディレクトリを使用しています。上記のコマンドを状況に応じて変更します。または、最初に `/etc/init.d` をシンボリックリンクとして `/etc/rc.d/init.d` にポイントして作成します。

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

スクリプトをインストールした後、使用しているオペレーティングシステムに基づいてシステムの起動時にコマンドを実行できるように有効化します。Linux では、`chkconfig` を使用します。

```
shell> chkconfig --add mysql
```

Linux システムの中には、`mysql` スクリプトをフルに有効にするには以下のコマンドが必要になる場合があります。

```
shell> chkconfig --level 345 mysql on
```

FreeBSD では、起動スクリプトは通常 `/usr/local/etc/rc.d/` にあります。`.rc(8)` のマニュアルではこのディレクトリのスクリプトはそれらのベースの名前が `*.sh` シェルのファイル名のパターンに一致したときのみ実行できると書いてあります。そのディレクトリの他のファイルあるいはディレクトリは無視されます。換言すれば、FreeBSD では、`mysql.server` スクリプトを `/usr/local/etc/rc.d/mysql.server.sh` としてインストールして自動による起動を有効にします。

上記の設定の代案として、オペレーティングシステムの中には `/etc/rc.local` あるいは `/etc/init.d/boot.local` を使用して起動時に追加のサービスを起動しています。この方法で MySQL を起動するには、以下のようなコマンドを適切な起動ファイルに追加します。

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

他のシステムの起動スクリプトのインストール方法についてはそのオペレーティングシステムの説明書をお読みください。

`mysql.server` のグローバル `/etc/my.cnf` ファイルにオプションを追加できます。一般的な `/etc/my.cnf` ファイルは以下のようになります。

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

`mysql.server` スクリプトは以下のオプションを実行します。`basedir`、`datadir`、および `pid-file`。指定する場合には、それらはコマンドラインではなくオプション ファイルに入れる 必要があります。`mysql.server` は `start` および `stop` をコマンドラインの引数として理解します。

以下のテーブルはどのオプション グループからサーバおよび各起動スクリプトがオプション ファイルを読むか示しています。

スクリプト	オプション グループ
<code>mysqld</code>	<code>[mysqld]</code> , <code>[server]</code> , <code>[mysqld-major_version]</code>
<code>mysqld_safe</code>	<code>[mysqld]</code> , <code>[server]</code> , <code>[mysqld_safe]</code>
<code>mysql.server</code>	<code>[mysqld]</code> , <code>[mysql.server]</code> , <code>[server]</code>

`[mysqld-major_version]` は `[mysqld-5.0]` および `[mysqld-5.1]` の名前のグループが 5.0.x、5.1.x、などのバージョンのサーバによって読まれることを意味しています。この機能は所定のリリース シリーズのサーバによってのみ読まれるオプションを指定するために使用されています。

下位互換の場合、`mysql.server` は `[mysql_server]` グループも読み込み `mysqld_safe` は `[safe_mysqld]` グループも読み込みます。しかし、MySQL 5.1 を使用する際は、`[mysql.server]` および `[mysqld_safe]` グループを読み込めるようにオプション ファイルをアップグレードする必要があります。

[オプションファイルの使用](#) を参照してください。

MySQL サーバの起動とトラブルシューティング

この項では Unix 上のサーバ起動時の問題に関するトラブルシューティングについて説明します。Windows を使用している場合には、[Windows への MySQL インストールにおけるトラブルシューティング](#) を参照してください。

サーバの起動時に問題がある場合、以下を試してみます。

- エラーログをチェックしてサーバが起動しない理由を調べます。
- 使用しているストレージ エンジンに必要なオプションを指定します。
- サーバがデータ ディレクトリの場所を検索できるか確認します。
- サーバがデータ ディレクトリにアクセスできるか確認します。データ ディレクトリおよびそのコンテンツの所有者の権利および権限をサーバがそれらを読み込んで変更できるように設定します。
- サーバに必要なネットワーク インターフェースが利用できるか確認します。

ストレージ エンジンにその振る舞いを管理するオプションが付いているものもあります。`my.cnf` ファイルを作成して使用するエンジンの起動オプションを指定します。トランザクション テーブル (InnoDB、NDB) をサポートするストレージ エンジンを使用する場合には、サーバを起動する前にそれらが所定の設定になっているか確認します。

MySQL Enterprise

```
#####MySQL #####
##http://www.mysql.com/products/enterprise/advisors.html #####
```

- InnoDB テーブルを使用している場合は、[InnoDB 設定](#) を参照してください。
- MySQL Cluster を使用している場合、[MySQL Cluster の設定](#) を参照してください。

ストレージ エンジンには特に指定しない場合はデフォルトの値を使用します。しかし、デフォルトの値が必ずしも適切であるとは限らないので利用できるオプションを確認して明示的に値を指定することをお勧めします。

`mysqld` サーバが起動すると、ロケーションをデータ ディレクトリに変更します。そこでデータベースを探し、ログ ファイルを書き込みます。サーバはデータ ディレクトリで pid (プロセス ID) ファイルも書き込みます。

サーバをコンパイルするとデータ ディレクトリのロケーションが組み込まれます。ここがデフォルトでサーバがデータ ディレクトリを探す場所です。データ ディレクトリがシステムのどこか別の場所にある場合には、サーバは正常に動作しません。`mysqld` を `--verbose` および `--help` オプションを実行してデフォルトのパスの設定を決定できます。

システム上のデフォルトのロケーションが MySQL のインストール レイアウトと一致しない場合、オプションをコマンドラインあるいはオプション ファイルの `mysqld` あるいは `mysqld_safe` に指定してそれらをオーバーライドできます。

データ ディレクトリのロケーションを明示的に指定するには、`--datadir` オプションを使用します。しかし、通常は MySQL をインストールしそこでデータ ディレクトリを探すベース ディレクトリのロケーションを `mysqld` に指定できます。`--basedir` オプションで指定できます。

パス オプションの指定の結果を知るには、`mysqld` をそれらのオプションで実行し次に `--verbose` および `--help` オプションを実行します。例えば、ロケーションを `mysqld` をインストールしたディレクトリに変更して次に以下のコマンドを実行すると、その結果 `/usr/local` のベース ディレクトリでサーバが起動します。

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

`--datadir` のようなオプションも同様に指定できますが、`--verbose` および `--help` は最後のオプションになります。

任意のパスを設定した後、サーバを `--verbose` および `--help` を使用しないで起動します。

`mysqld` が動作している時に、以下のコマンドを実行してどのパス設定が使用されているか確認できます。

```
shell> mysqladmin variables
```

または

```
shell> mysqladmin -h host_name variables
```

`host_name` は MySQL サーバのホスト名です。

`mysql` を実行したときに `Errcode 13 (Permission denied)` を意味する) が表示された場合は、データ ディレクトリの権限あるいはそのコンテンツがサーバのアクセスを許可していないことを意味します。この場合、関連するファイルおよびディレクトリの権限を変更してサーバがそれらのを使用できるようにします。サーバを `root` から起動できますが、この場合セキュリティが脆弱になるためこの立上げは避けるべきです。

Unix では、ロケーションをデータ ディレクトリに変更してデータ ディレクトリおよびそのコンテンツの所有者権限をチェックし、サーバにアクセス権があるか確認します。例えば、データ ディレクトリが `/usr/local/mysql/var` の場合は、以下のコマンドを使用します。

```
shell> ls -la /usr/local/mysql/var
```

データ ディレクトリあるいはそのファイルまたはサブディレクトリがサーバを運用するためのログイン アカウントの所有でない場合、それらの所有者権限をそのアカウントに変更します。そのアカウントが `mysql` の場合、以下のコマンドを使用します。

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

サーバが正常に起動できない場合、エラーログをチェックします。ログ ファイルはデータ ディレクトリ (一般的には Windows では `C:\Program Files\MySQL\MySQL Server 5.1\data`、Unix のバイナリ ディストリビューションでは `/usr/local/mysql/data`、Unix ソース ディストリビューションでは `/usr/local/var`) にあります。データ ディレクトリの `host_name.err` および `host_name.log` のフォーマットのファイルは、`host_name` はサーバのホスト名です。次にこれらのファイルの最後の数行を調べます。Unix では、それらの表示に `tail` を使用します。

```
shell> tail host_name.err
shell> tail host_name.log
```

エラーログにはサーバが起動しなかった情報が含まれています。

以下のいずれかのエラーが発生した場合、他のプログラム (多分別の `mysqld` サーバ) が `mysqld` が使用する TCP/IP ポートあるいは Unix のソケット ファイルを使用していることを意味します。

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

別の `mysqld` サーバを動作している場合には、`ps` を使用します。その場合、`mysqld` を再度起動する前にサーバをシャットダウンします。(別のサーバの動作中に、複数のサーバを稼働させる運用の仕方に関する情報は、[同じマシン上での複数 MySQL サーバの実行](#) にあります。)

サーバがどれも動作していない場合、コマンド `telnet your_host_name tcp_ip_port_number` を実行してみます。(デフォルトの MySQL ポート番号は 3306 です。)次に Enter を数回押します。次のようなエラーメッセージが表示されなかった場合 `telnet:Unable to connect to remote host:Connection refused`、`mysqld` が使用しようとしている TCP/IP ポートを他のプログラムが使用しています。その際どのプログラムが使用しているかを調べてそれを無効にするか、または `mysqld` に `--port` オプションで別のポートを指定します。この場合、サーバに TCP/IP 経由で接続する際にクライアント プログラムにポート番号を指定する必要があります。

ポートが接続できない別の理由にファイアウォールがその接続をブロックしている場合があります。その場合、ファイアウォールの設定をポートにアクセスできるように変更します。

サーバが起動しても接続できない場合、以下のようなエントリが `/etc/hosts` ないか確認します。

```
127.0.0.1 localhost
```

この問題は実行中のスレッド ライブラリがないシステムで MIT-pthreads を使用する設定が必要な MySQL でのみ発生します。

`mysqld` を起動できない場合、トレース ファイルを作成し `--debug` オプションを使用して問題を探します。[Creating Trace Files](#) 参照。

最初の MySQL アカウントの確保

MySQL インストール プロセスの一環でグラント テーブルを含む `mysql` データベースを設定します。

- Windows のディストリビューションには自動的にインストールされる初期化されたグラント テーブルが含まれています。
- Unix では、グラント テーブルは `mysql_install_db` プログラムで設定されます。インストール メソッドがお客様に代わってこのプログラムを実行します。その他は手動でそれを実行します。詳細に関しては [Unix のインストール後のプロシージャ](#) を参照して下さい。

グラント テーブルは最初の MySQL ユーザーアカウントおよびそれらのアクセス権限を定義します。これらのアカウントは以下のように設定されます。

- ユーザー名 `root` を持つアカウントが作成されます。通常とな異なる機能を持つスーパーユーザーのアカウントがあります。最初の `root` アカウントパスワードは空ですので、誰でも MySQL サーバに `root` として — パスワードなし — で接続できすべての権限を使用できます。
- Windows では、一つの `root` アカウントが作成されます。このアカウントではローカル ホストのみから接続できます。Windows のインストーラではオプションでアカウントを作成してインストール中にユーザーが `Enable root access from remote machines` オプションを選択したときのみどのホストからでも接続できます。
- Unix では、両方の `root` アカウントはローカル ホストの接続用です。接続はローカルホストから `localhost` のホスト名をアカウントの一つに指定するか、あるいは実際のホスト名または他への IP 番号を指定する必要があります。
- 2 つの匿名アカウントをそれぞれユーザー名なしで作成できます。匿名のアカウントにはパスワードがないため、誰でもそのアカウントを使用して MySQL サーバに接続できます。
- Windows では、一つの匿名アカウントはローカル ホストの接続用です。そのアカウントにはすべての権限が `root` アカウントと同様にあります。もう一つのアカウントはすべてのホストの接続用で `test` データベースおよび `test` で始まる名前の他のデータベースに対してすべての権限を持っています。
- Unix では、両方の匿名アカウントはローカル ホストの接続用です。接続はローカル ホストから `localhost` のホスト名をアカウントの一つあるいは実際のホスト名または他への IP 番号を指定する必要があります。これらのアカウントは `test` データベースおよび `test_` で始まる名前のすべてのデータベースに対してすべての権限を持っています。

上述のごとく、最初のアカウントにはどれもパスワードがありません。このことは MySQL のインストールはそれらを設定するまでは保護されていないということを意味します。

- クライアントが匿名ユーザーでパスワードなしで接続しないようにするには、それぞれの匿名のアカウントにパスワードを割り当てるか、あるいはそれらのアカウントを削除する必要があります。
- それぞれの MySQL `root` アカウントにパスワードを割り当てる必要があります。

以下の手順では最初の MySQL アカウントのパスワードの設定方法、最初に匿名のアカウント、次に `root` アカウントへの設定方法を説明します。例の「`newpwd`」を実際に使用するパスワードに置き換えます。以下の説明では匿名のアクセスを希望しない場合の匿名のアカウントの削除の仕方についても説明します。

パスワードを新たな設定やテストをしている間に指定する必要がないようにそれらの作業が終わるまでパスワードの設定を望まない場合もあります。しかし、量産用（実稼働）のインストールを使用する前にはそれらを忘れずに設定してください。

匿名アカウントのパスワードの割り当て

匿名アカウントにパスワードを割り当てるには、サーバに `root` として接続して `SET PASSWORD` あるいは `UPDATE` のいずれかを使用します。どちらの場合も、`PASSWORD()` 機能を使用してパスワードを暗号化します。

Windows の `SET PASSWORD` を使用するには、以下のようにします。

```
shell> mysql -u root
mysql> SET PASSWORD FOR '@localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR '@%' = PASSWORD('newpwd');
```


Unix で `SET PASSWORD` を使用するには、以下のようになります。

```
shell> mysql -u root
mysql> SET PASSWORD FOR '@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR '@'host_name' = PASSWORD('newpwd');
```

2 番目の `SET PASSWORD` ステートメントでは、`host_name` をサーバのホスト名で置き換えます。これは `user` テーブルの `root` に非-`localhost` レコードの `Host` カラムで指定された名前です。ホスト名が分からない場合には、`SET PASSWORD` を使用する前に以下のステートメントを発行します。

```
mysql> SELECT Host, User FROM mysql.user;
```

`User` カラムで `root` を持つレコードおよび `Host` カラムで `localhost` 以外の何かを探します。次にその `Host` 値を 2 番目の `SET PASSWORD` ステートメントで使用します。

匿名のアカウントにパスワードを割り当てる別の方法は `UPDATE` を使用して `user` テーブルを直接変更することです。サーバに `root` として接続して値を適切な `user` のテーブル レコードの `Password` カラムに割り当てる `UPDATE` ステートメントを発行します。その手順は Windows も Unix も同じです。以下の `UPDATE` ステートメントは両方の匿名アカウントに同時にパスワードを割り当てます。

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

`UPDATE` を使用して `user` テーブルのパスワードを直接更新したら、サーバに `FLUSH PRIVILEGES` でグラント テーブルを再度読み込むようにサーバに指定する必要があります。さもなければ、その変更はサーバを再起動するまで認識されません。

匿名アカウントの削除

匿名のアカウントを削除するには、以下のようになります。

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

`DELETE` ステートメントは Windows および Unix 双方に適用されます。Windows で `root` と同じ権限を持つ匿名のアカウントのみを削除するには、以下のようになります。

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

そのアカウントは匿名のアクセスが可能ですべての権限を持っているので、それを削除するとセキュリティが改善されます。

root アカウントのパスワードの割り当て

`root` アカウントへのパスワードの割り当てには幾つかの方法があります。以下の説明では 3 種類の方法を示します。

- `SET PASSWORD` ステートメントを使用する
- `mysqladmin` コマンドラインのクライアント プログラムを使用する
- `UPDATE` ステートメントを使用する

`SET PASSWORD` を使用してパスワードを割り当てるには、サーバに `root` として接続して 2 つの `SET PASSWORD` ステートメントを発行します。`PASSWORD()` 機能を使用してパスワードを忘れずに暗号化します。

Windows では、以下のようになります。

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

Unix では、以下のようになります。

```
shell> mysql -u root
```



```
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

2 番目の `SET PASSWORD` ステートメントでは、`host_name` をサーバのホスト名で置き換えます。これは匿名アカウトのパスワードを割り当てた時に使用したホスト名と同じものです。

パスワードを `root` アカウトに `mysqladmin` を使用して割り当てるには、以下のコマンドを実行します。

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

これらのコマンドは Windows および Unix の両方に適用されます。.2 番目のコマンドで、`host_name` をサーバのホスト名で置き換えます。パスワードの二重引用符は常に必要とは限りません、パスワードにスペースあるいは使用しているコマンド インタープリタに特殊な他の文字を使用している場合には使用する必要があります。

`UPDATE` を使用して `user` テーブルを直接変更することもできます。以下の `UPDATE` ステートメントは両方の `root` アカウトに同時にパスワードを割り当てます。

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

`UPDATE` ステートメントは Windows および Unix の両方に適用されます。

パスワードを設定すると、サーバに接続するたびに適切なパスワードを入力する必要があります。例えば、`mysqladmin` を使用してサーバをシャットダウンするには以下のコマンドを使用します。

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

注:パスワード設定後に `root` パスワードを忘れた場合には、[How to Reset the Root Password](#)、がその再設定を行います。

新たにアカウントを追加するには、`GRANT` ステートメントを使用します。その手順は、[MySQL への新規ユーザの追加](#) を参照してください。

MySQL のアップグレード

一般的な規則として一つのリリースのシリーズから別のリリースのシリーズにアップグレードするには、シリーズをスキップしないで次のシリーズにアップグレードすることをお勧めします。例えば、現在使用している MySQL 4.0 から新しいシリーズにアップグレードするには、5.0 あるいは 5.1 にアップグレードするのではなく MySQL 4.1 にアップグレードします。

以下の項目はアップグレードする際の必ず実施するチェックリストを示したものです。

- MySQL 5.0 から 5.1 にアップグレードする前に、[MySQL 5.0 から 5.1 へのアップグレード](#)、および [付録C MySQL Change History](#) をお読みください。これらは MySQL 5.1 で新しく追加された機能および MySQL 5.0 と異なる機能に関する情報を提供します。MySQL 5.0 以前のリリース シリーズからアップグレードするには、MySQL 5.0 にアップグレードするまでに各シリーズから順番にアップグレードし、その後で MySQL 5.1 にアップグレードします。MySQL 5.0 からのアップグレードに関する情報は、MySQL 5.0 Reference Manual を参照してください。以前のリリースに関しては、MySQL 3.23, 4.0, 4.1 の参照マニュアルを参照してください。
- アップグレードを実施する前に、グラント テーブルを含む `mysql` データベースを格納したデータベースのバックアップを取ります。
- MySQL のリリースの中にはテーブルに互換性のない変更をしている場合もあります。(弊社はこれらの変更を避けるように努めていますが、時にリリース間の互換性を取るよりも深刻な問題の修正に迫られる場合があります。)MySQL の リリースの中には新たに権限や機能を加えるためにグラント テーブルの構成に変更を加えているものもあります。

それらの変更による問題を避けるためには、MySQL の新しいバージョンにアップグレードした後に、テーブルをチェックし (必要に応じて修正する)、グラント テーブルをアップグレードして、新しい機能を利用できるようにそれらが現在の構成になっていることを確認する必要があります。[mysql_upgrade — MySQL アップグレードのテーブルチェック](#) 参照。

- Windows 上で MySQL サーバを稼働している場合には、[Windows を使用した MySQL をアップグレードする](#) を参照してください。
- レプリケーションを使用している場合、レプリケーション設定のアップグレードに関する情報、[レプリケーション セットアップのアップグレード](#)にあります。
- MySQL 5.1.9 の場合、[mysqld-max](#) サーバはバイナリのディストリビューションに含まれています。個別の MySQL-Max ディストリビューションはありません。MySQL 5.1.12 では、バイナリのディストリビューションは以前 [mysqld-max](#) に含まれていた機能を実装したサーバを搭載しています。
- ユーザー定義関数 (UDF) の所定の名前で作成して MySQL を同じ名前の内蔵機能を実装した新しいバージョンにアップグレードした場合、その UDF はアクセスできなくなります。これを修正するには、[DROP FUNCTION](#) を使用して UDF を削除し、次に [CREATE FUNCTION](#) を使用して衝突しない名前の UDF を新たに作成します。同じところが新しいバージョンの MySQL が既存の機能と同名の内蔵機能を実装した場合にも言えます。サーバの異なる種類の機能に対するリファレンスの解釈を説明した規則は、[関数名の構文解析と名前解決](#) を参照してください。

MySQL のフォーマット ファイルやデータ ファイルを MySQL の同じリリースのバージョン内に限り同じアーキテクチャの異なるバージョン間で常に移動できます。MySQL の動作中に文字セットを変更するには、[mysamchk -r -q --set-collation=collation_name](#) をすべての [MyISAM](#) テーブルで実行する必要があります。さもないと、文字セットを変更することによって分類順序も変更される場合があるので、インデックスは正しく順序付けされない場合があります。

新しいバージョンの使用に懸念がある場合には、常に古い [mysqld](#) の名前を新しいバージョンをインストールする前に変更します。例えば、MySQL 5.0.13 を使用していてそれを 5.1.10 にアップグレードする場合、現在のサーバ名を [mysqld](#) から [mysqld-5.0.13](#) に変更します。新しい [mysqld](#) に予想外の問題が発生した場合は、単にそれをシャットダウンして古い [mysqld](#) を起動するだけで済みます。

アップグレード後に、再コンパイルしたクライアント プログラムで [Commands out of sync](#) あるいは予想外のコア ダンプなどの問題が発生した場合、それは多分に古いヘッダーあるいはライブラリ ファイルをプログラムをコンパイルする際の使用したためです。この場合、[mysql.h](#) ファイルおよび [libmysqlclient.a](#) ライブラリの日付をチェックしてそれらが新しい MySQL のディストリビューションによるものであることを確認します。そうでない場合には、プログラムを新しいヘッダーおよびライブラリで再度コンパイルします。

新しい [mysqld](#) サーバが起動しないあるいはパスワードなしで接続できないなどの問題が発生した場合には、前のインストールの古い [my.cnf](#) ファイルが残っていないか確認します。これは [--print-defaults](#) オプション (例えば、[mysqld --print-defaults](#)) で確認できます。このコマンドがプログラム名以外の何かを表示した場合、サーバあるいはクライアント オペレーションに影響を及ぼすアクティブな [my.cnf](#) ファイルがあることになります。

新しいリリースの MySQL をインストールする際はいつでも Perl [DBD::mysql](#) モジュールを再構築して再インストールするのがいいでしょう。同じことが、PHP [mysql](#) 拡張および Python [MySQLdb](#) モジュールなどの他の MySQL インターフェイスにも言えます。

MySQL 5.0 から 5.1 へのアップグレード

5.0 のインストールから 5.0.10 あるいはそれ以上にアップグレードするには [グラント テーブルのアップグレード](#) が必要です。アップグレードしなかった場合、保持したプロシージャおよび機能を作成しても機能しません。この手順に関する説明は、[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照してください。

注:新しいバージョンのソフトウェアをインストールする前にデータのバックアップを必ず励行してください。MySQL が高品質の維持に努めたにしても、デバックアップを取ってデータを守るのはお客様ご自信です。MySQL AB ではテーブルをダンプして以前のバージョンからテーブルを再ロードすることをお勧めします。5.1.

一般的には、MySQL 5.0 から 5.1 にアップグレードするには以下を行う必要があります。

- [MySQL のアップグレード](#) の項目をチェックし、その項目でアプリケーションに影響を及ぼすものがないか確認します。
- この項で後述する変更リストをチェックしそれらの変更リストの一部にアプリケーションに影響を及ぼすものがないか確認します。非互換の変更 の印の付いた項目に特に注意します。これらの非互換の MySQL の以前のバージョンの結果、およびアップグレードする前に必要な注意事項。
- MySQL のリリースの中にはテーブルに互換性のない変更をしている場合もあります。(弊社はこれらの変更を避けるように努めていますが、時にリリース間の互換性を取るよりも深刻な問題の修正に迫られる場合があります)

す。)MySQL のリリースの中には新たに権限や機能を加えるために Grant テーブルの構成に変更を加えているものもあります。

それらの変更による問題を避けるためには、MySQL の新しいバージョンにアップグレードした後、`mysql_upgrade` を実行してテーブル (必要に応じて修正する) をチェックし、Grant テーブルをアップグレードして、新しい機能を利用できるようにそれらが現在の構成になっていることを確認します。[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) 参照。

- MySQL 5.1 の変更履歴を読んでどのような重要な新機能を 5.1 で使用できるか確認します。[Changes in release 5.1.x \(Development\)](#) 参照。
- Windows 上で MySQL サーバを稼働している場合には、[Windows を使用した MySQL をアップグレードする](#) を参照してください。
- レプリケーションを使用している場合、レプリケーション設定のアップグレードに関する情報、[レプリケーション セットアップのアップグレード](#)にあります。

以下のリストはアプリケーションに影響を及ぼす可能性があり MySQL 5.1 にアップグレードする際に注意を要する項目をリストに纏めたものです。

設定の変更:

- MySQL 5.1.11 以前で MySQL をソースから SSL サポート付きでビルドする場合、`configure` を `--with-openssl` あるいは `--with-yassl` オプションのいずれかで実行します。MySQL 5.1.11 では、それらのオプションは両方とも `--with-ssl` オプションで置き換えられています。デフォルトで、`--with-ssl` でバンドルした yaSSL ライブラリを使用できます。OpenSSL を選択するには、オプションを `--with-ssl=path` として与え、そこでは `path` はディレクトリで OpenSSL のヘッダーファイルおよびライブラリが格納されています。

サーバの変更:

- 非互換の変更:MySQL 5.1 の実装ではランタイム時のコンポーネントのローディングおよびアンローディングをサーバの起動なしで行うプラグイン API をサポートしています。[The MySQL Plugin Interface](#)。プラグイン API には `mysql.plugin` テーブルが必要です。MySQL の旧バージョンからアップグレードするには、`mysql_upgrade` コマンドを実行してこのテーブルを作成します。[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) 参照。

プラグインは `plugin_dir` システム変数で指定されたディレクトリにインストールされます。この変数はまたサーバがユーザー定義関数 (UDF) をロードするロケーションを管理し、この点が MySQL の旧バージョンから変更になっています。つまり、すべての UDF ライブラリ ファイルは現在はプラグインのディレクトリにインストールする必要があります。MySQL の旧バージョンからアップグレードする際、UDF ファイルをプラグインのディレクトリに移行する必要があります。

- 非互換の変更:`table_cache` システム変数は `table_open_cache` に名前が変わっています。`table_cache` を参照するすべてのスクリプトは新しい名前を使用できるように更新する必要があります。
- 非互換の変更:MySQL 5.1.15 では、以下の条件を `read_only` システム変数を有効にするために適用します。
 - `read_only` を明示のロック (`LOCK TABLES` で取得) 中に有効にしようとしたりあるいは未処理のトランザクションがある場合、エラーが発生します。
 - 他のクライアントが明示のテーブル ロックを掛けていたり未処理のトランザクションがある場合、`read_only` の有効化はロックが解除されトランザクションが終了するまでブロックされます。`read_only` の有効化がペンディング中には、他のクライアントによるテーブル ロックあるいはトランザクションの開始もまた `read_only` が設定されるまでブロックされます。
 - `read_only` はグローバル読み込みロック (`FLUSH TABLES WITH READ LOCK` で取得) 中でもそれがテーブル ロックを実行しないために有効にできます。

以前は、`read_only` の有効化は明示のロックあるいはトランザクションが未処理の場合でも直ぐに有効になり、ステートメントのデータの変更はサーバで同時に行われていました。

- 非互換の変更:`IGNORE_SPACE` によって影響を受ける機能名が MySQL 5.1.13 ではおよそ 200 から 30 に大幅に減少しました。(`IGNORE_SPACE` の詳細は、[関数名の構文解析と名前解決](#) を参照してください。)この変更によりペーパー操作の一貫性が改善されています。しかしながら、以下の条件に基づく旧 SQL コードの非互換性の可能性も加わっています。

- `IGNORE_SPACE` は無効になっています。
- 機能名に続く余白部分の存在の有無によって同名の (例えば、`PI()` versus `PI ()`) 内蔵機能および保持機能間の区別をしています。

MySQL 5.1.13 の `IGNORE_SPACE` によりもはや影響を受けない機能には、その戦略は通用しません。前述の非互換に基くコードがある場合以下の手法のいずれかを使用できます。

- 保持機能名と内蔵機能名が衝突する名前の場合、余白の有無に関係なくスキーマ名修飾語の名前のある保持機能を参照します。例えば、`schema_name.PI()` あるいは `schema_name.PI ()` と書きます。
- あるいは、保持機能の名前に衝突しない名前に変更して、機能の実行を新しい名前を使用できるように変更します。
- 非互換の変更: `utf8` カラムでは、フル テキスト parser が幾つかの非単語の句読点および余白文字を単語文字として不正確に解釈するため、検索した場合不正確な検索結果を返す場合があります。修正には MySQL 5.1.12 のフル テキスト parser の変更が含まれ、`FULLTEXT` インデックスを `utf8` カラムに持つテーブルは `REPAIR TABLE` で修正する必要があります。

```
REPAIR TABLE tbl_name QUICK;
```

- 非互換の変更: `FULLTEXT` インデックスの構成は MySQL 5.1.6 で変更になっています。MySQL 5.1.6 あるいはそれ以降にアップグレード後に、`REPAIR TABLE` ステートメントを `FULLTEXT` インデックスを含む各テーブルにコールします。
 - 非互換の変更: MySQL 5.1.6 以前では、サーバは一般的なクエリ ログを書いてログ ファイルへのクエリ ログのエントリを遅くします。MySQL 5.1.6 では、サーバのこれらのログのログ機能はさらに柔軟になっています。ログのエントリはログ ファイル (上記のように) あるいは `general_log` および `mysql` データベースの `slow_log` に書き込まれます。ロギングが有効になると、デイスティネーションのいずれかまたは両方が選択されます。`--log-output` オプションはデイスティネーションあるいはログ出力のデイスティネーションを管理します。[一般クエリとスロー クエリのログ出力先の選択](#) 参照。
- ロギングが有効になると、デフォルトのデイスティネーションはこの段階でテーブルにログされます。この点が以前のバージョンと異なる点です。ログをログ ファイルにする設定のサーバを所持している場合には、`--log-output=FILE` を使用してこの振る舞いを MySQL の 5.1.6 あるいはそれ以降にアップグレードした後に保持します。
- MySQL 5.1.12 の場合、`lc_time_names` システム変数は日付および省略文字の記載に使用される言語を管理するロケールを指定します。この変数は `DATE_FORMAT()`、`DAYNAME()` および `MONTHNAME()` 関数の出力に影響を与えます。[MySQL サーバのローケル サポート](#) 参照。
 - MySQL 5.1.6 では、データベースおよびテーブル識別子の特殊文字は相当するディレクトリ名およびファイル名を作成中にエンコードされます。これにより識別子として使用される文字制限が緩やかになります。[ファイル名への識別子のマッピング](#) 参照。`mysql_upgrade` を実施すると、データベース名およびテーブル名は、それらが特殊文字を含む場合、新しいフォーマットに更新されます。
 - MySQL 5.1.9 では、`mysqld_safe` はもはや默示的に `mysqld-max` をそれが存在した場合でも呼び出しません。代わりに、それは `--mysqld` あるいは `--mysqld-version` オプションが他のサーバを明示的に指定しない限り `mysql` を呼び出します。これまで默示的に `mysqld-max` の呼び出しを使用していた場合、これからは適切なオプションを使用する必要があります。

SQL の変更:

- 非互換の変更: MySQL 5.1.8 では、`TYPE = engine_name` は `ENGINE = engine_name` テーブル オプションに対してまだ類義語として受け入れられますが、警告が表示されます。このオプションは MySQL 5.1.7 では利用できず、MySQL 5.2 ではすべて削除されますので構文エラーが表示されます。.

`TYPE` は MySQL 4.0 以降は使用していません。

- 非互換の変更: トリガのネームスペースは MySQL 5.0.10 では変更になっています。トリガ名は以前はテーブルごとに一意でなければなりませんでした。現在はそれらはスキーマ (データベース) ごとに一意です。この変更によって `DROP TRIGGER` 構文は現在テーブル名の代わりにスキーマ名を使用しています (スキーマ名はオプションで、削除された場合は現在のスキーマが使用されます)。

旧バージョンの MySQL 5 から MySQL 5.0.10 あるいはそれ以降にアップグレードする場合、すべてのトリガを削除して再度作成しなければ **DROP TRIGGER** はアップグレード後は機能しなくなります。以下のその手順を示します。

1. MySQL 5.0.10 あるいはそれ以降にアップグレードすると **INFORMATION_SCHEMA.TRIGGERS** テーブルでトリガ情報にアクセスできます。(5.0.10 トリガでも機能する必要があります。)
2. 以下の **SELECT** ステートメントですべてのトリガ定義をダンプします。

```
SELECT CONCAT('CREATE TRIGGER ', t.TRIGGER_SCHEMA, ' ', t.TRIGGER_NAME,
              ' ', t.ACTION_TIMING, ' ', t.EVENT_MANIPULATION, ' ON ',
              t.EVENT_OBJECT_SCHEMA, ' ', t.EVENT_OBJECT_TABLE,
              ' FOR EACH ROW ', t.ACTION_STATEMENT, ' ')
INTO OUTFILE 'tmp/triggers.sql'
FROM INFORMATION_SCHEMA.TRIGGERS AS t;
```

そのステートメントは **INTO OUTFILE** を使用していますので、**FILE** 権限を持つ必要があります。ファイルはサーバのホストで、希望される場合任意のファイル名で、作成されます。100% 安全であるためには、トリガ定義を **triggers.sql** ファイルで確認し、そのファイルのバックアップを取るとよいでしょう。

3. サーバを停止してすべてのトリガをすべての **.TRG** ファイルをデータベースのディレクトリから削除して削除します。ロケーションをデータ ディレクトリに変更して以下のコマンドを発行します。

```
shell> rm */*.TRG
```

4. サーバを起動し **triggers.sql** ファイルを使用してすべてのトリガを再度作成します。例えば、私の場合には:

```
mysql> delimiter // ;
mysql> source /tmp/triggers.sql //
```

5. すべてのトリガの作成が **SHOW TRIGGERS** ステートメントを使用して完了したことを確認します。

- 非互換の変更:MySQL 5.1.6 では **TRIGGER** 権限を使用しています。以前は、**SUPER** 権限はトリガの作成あるいは削除に必要でした。今それらの操作には **TRIGGER** 権限が必要です。これやセキュリティの改善に行われたもので、トリガを作成するためにはやが Grant ユーザーには **SUPER** 権限は必要ありません。しかし、トリガの **DEFINER** 節で指定されたアカウントは **SUPER** 権限を持つという条件が **TRIGGER** 権限の条件に変わっています。MySQL 5.0 or 5.1 の旧バージョンから MySQL 5.1.6 あるいはそれ以上にアップグレードするには、[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) の説明に従って Grant テーブルを更新する必要があります。このプロセスにより **TRIGGER** 権限を **SUPER** 権限を持っていたすべてのアカウントに割り当てます。Grant テーブルの更新に失敗すると、トリガは実行に失敗する場合があります。(Grant テーブルの更新後、**SUPER** 権限をもはやそれを必要としないそれらのアカウントから取り消すことができます。)
- 4 桁の年を表す 200 以内の日付の場合、世紀を含める黙示の変換が **DATE_ADD()**、**DATE_SUB()**、**+ INTERVAL**、および **- INTERVAL** で実行される日付の計算式に適用されています。(例えば、**DATE_ADD('0050-01-01 00:00:00', INTERVAL 0 SECOND)** は **'2050-01-01 00:00:00'** になります。)MySQL 5.1.11 では、これらの操作は正確な非 **NULL** 値ではなく **NULL** を返します。(Bug #18997)
- キーワードの幾つかが MySQL 5.1 で予約されます。それらは MySQL 5.0 では予約されなかったものです。[MySQL での予約語の扱い](#) 参照。
- **LOAD DATA FROM MASTER** および **LOAD TABLE FROM MASTER** ステートメントは今後は使用しません。推奨している代案については、[LOAD DATA FROM MASTER 構文](#) を参照してください。
- プラグイン API に使用される **INSTALL PLUGIN** および **UNINSTALL PLUGIN** ステートメントは新規です。同様にフル テキスト インデックスの parser プラグインに関連した **FULLTEXT** インデックス作成用の **WITH PAPER** 節も新規です。[The MySQL Plugin Interface](#)。

C API の変更:

- 非互換の変更:MySQL 5.1.7 では、[mysql_stmt_attr_get\(\)](#) C API 関数は **STMT_ATTR_UPDATE_MAX_LENGTH** の無署名 int ではなく boolean を返します。(Bug #16144)

MySQL データベースの他のマシンへのコピー

.frm、.MYI、および .MYD ファイルを [MyISAM](#) テーブルに同じフローティング ポイント フォーマットをサポートしている異なるアーキテクチャ間でコピーでします。(MySQL はバイト スワッピング問題を処理します。)[MyISAM ストレージエンジン](#) 参照。

データベースを異なるアーキテクチャ間で移動する場合、[mysqldump](#) を使用して SQL ステートメントを含むファイルを作成します。次にそのファイルを別のマシンに転送して [mysql](#) クライアントの入力として扱います。

利用できるオプションを表示するには [mysqldump --help](#) を使用します。データを新しいバージョンの MySQL に移動するには [mysqldump --opt](#) を使用して最適化を活用することによってさらに小さくて迅速に処理が可能なダンプ ファイルを作成します。

データベースを 2 つのマシンで間で移動する最も容易な方法 (速くはないが) はデータベースを搭載したマシン上で以下のコマンドを実行することです。す。

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump --opt db_name | mysql -h 'other_hostname' db_name
```

データベースを遠隔のマシンから速度の遅いネットワークにコピーするには、以下のコマンドを使用します。

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --opt --compress db_name | mysql db_name
```

ダンプをファイルに保存して、そのファイルをターゲット マシンに転送し、そのファイルをそのデータベースにロードすることもできます。例えば、データベースをソース マシンの圧縮ファイルに以下のようにダンプします。

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

データベースのコンテンツを含んだファイルをターゲット マシンに転送しそこで以下のコマンドを実行します。

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

データベースの転送に [mysqldump](#) および [mysqlimport](#) を使用することもできます。大きなテーブルの場合、これは単に [mysqldump](#) を使用するよりも非常に速く転送できます。以下のコマンドで、[DUMPDIR](#) は [mysqldump](#) の出力の保存に使用されるディレクトリのフルのパス名です。

最初に、その出力ファイルのディレクトリを作成してデータベースをダンプします。

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

次に [DUMPDIR](#) ディレクトリのファイルをターゲット マシンの相当するディレクトリに転送して、そのファイルをその MySQL にロードします。

```
shell> mysqladmin create db_name # create database
shell> cat DUMPDIR/*.sql | mysql db_name # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt # load data into tables
```

[mysql](#) データベースのコピーを忘れないでください。そこにgrant テーブルが保存されています。コマンドを MySQL [root](#) ユーザーとして新しいマシンで [mysql](#) データベースの用意できるまで実行することもできます。

[mysql](#) データベースを新しいマシンにインポートしたら、[mysqladmin flush-privileges](#) を実行してサーバにgrant テーブルの情報をロードさせます。

MySQL のダウングレード

この項ではあまりあり得ないケースだが旧バージョンが新しいバージョンより優れている場合における旧 MySQL バージョンにへのダウングレードの手順について説明します。

同じリリースのシリーズ (例えば、5.0.13 から 5.0.12) 内でのダウングレード際の一般的な規則としては新しいバイナリを旧バージョンの上段にインストールすることです。この場合データベースにはなにもする必要はありません。いつものように、この場合もバックアップを取ります。

以下の項目はダウングレードする際に必ず実施するチェックリストを示したものです。

- ダウングレードするリリース シリーズのアップグレードの項を読み、実際に必要な機能がないことを確認します。[MySQL のアップグレード](#)。

- そのバージョンのダウングレードの項ある場合には、それも同様に読む必要があります。

MySQL のフォーマット ファイルやデータ ファイルを MySQL の同じリリースのバージョン内に限り同じアーキテクチャの異なるバージョン間で移動できます。

一つのリリース シリーズから別のリリース シリーズにダウングレードする場合、テーブル ストレージ フォーマットの互換性が取れなくなる場合があります。このような場合、[mysqldump](#) を使用してダウングレードする前にテーブルをダンプできます。ダウングレードしたら、[mysql](#) あるいは [mysqlimport](#) を使用してダンプ ファイルをロードしテーブルを作成します。参考例は、[MySQL データベースの他のマシンへのコピー](#) を参照してください。

ダウンロードした際のテーブル フォーマット変更による非互換問題の一般的な現象はテーブルを開くことができないことです。そのような場合には、以下の手順に従います。

1. 新バージョンからダウングレードしている旧 MySQL サーバを停止します。
2. 旧バージョンにダウングレードしている新しい MySQL サーバを再起動します。
3. [mysqldump](#) を使用して旧サーバにアクセスできなかったテーブルをダンプして新たにダンプファイルを作成します。
4. 新しい MySQL サーバを停止して旧サーバを再起動します。
5. 旧サーバにダンプ ファイルをロードします。これでテーブルにアクセスできるようになります。

MySQL 5.0 へのダウングレード

MySQL 5.0 はイベントをサポートしていません。データベースにイベントが含まれる場合には [mysqldump](#) を使用するとき `--skip-events` オプションを使用してイベントがダンプされないようにします。([mysqldump — データベースバックアッププログラム](#) 参照。)

オペレーティング システムに特化した注釈

Linux の注釈

この項では Linux を使用した場合に発生する問題について説明します。この項の前半ではバイナリおよびソースのディストリビューションを使用した際のオペレーティング システムに関する一般的な懸案事項および問題、およびインストール後の懸案事項について説明します。この項の後半では Linux の特定のプラットフォームでの問題について説明します。

これらの問題の多くは旧バージョンの Linux で発生します。最新バージョンを使用している場合には、何も問題が発生しない場合があります。

Linux オペレーティング システムの注釈

MySQL には最低でも Linux バージョン 2.0 が必要です。

警告: 弊社では予想外の問題が Linux 2.2.14 および SMP システムの MySQL で報告されています。弊社にはまた MySQL のユーザーから MySQL を kernel 2.2.14 で使用した場合深刻な安定性の問題が発生するとの報告も寄せられています。この kernel を使用している場合には、2.2.19 (あるいはそれ以降) あるいは 2.4 kernel にアップグレードする必要があります。複数の CPU ボックスを使用している場合には、2.4 のご使用を真剣にご検討ください。2.4 は速度を大幅に改善します。お客様のシステムがさらに安定します。

LinuxThread を使用する際は、少なくとも 3 種類の [mysqld](#) プロセスを実施する必要があります。これらは ファクトスレッドにあります。1 つ目は LinuxThread マネージャ用、2 つ目接続用、3 つ目はアラームおよび信号を扱います。

Linux バイナリ ディストリビューションの注釈

MySQL の Linux-Intel バイナリおよび RPM リリースは最高速度に設定されています。弊社では常に高速で安定したコンパイラの使用を心がけています。

バイナリのリリースは `-static` とリンクしていますので、通常はどのバージョンのシステム ライブラリを使用しているのか心配する必要はありません。Linux Thread をインストールする必要もありません。`-static` とリンクしたプログラ

ムは動的にリンクしたプログラムより少しだけ大きくなりますが、多少速く (3-5%) なっています。しかし、静的にリンクされたプログラムの一つの問題は、ユーザー定義関数 (UDF) を使用できないことです。UDF を書いたり使用する (C あるいは C++ プログラマーの場合のみ) 場合、動的リンクを使用してお自身で MySQL をコンパイルする必要があります。

バイナリ ディストリビューションの既知の問題は `libc` (Red Hat 4.x あるいは Slackware など) を使用した旧 Linux システムでは、ホスト名のリゾリューションに幾つかの (致命的ではない) 問題があることです。お客様のシステムが `glibc2` ではなく `libc` を使用している場合、多分ホスト名リゾリューションおよび `getpwnam()` で問題に遭遇します。これは `glibc` (不幸にも) が、`-static` でコンパイルする時でさえ、いくつかの外部のライブラリに依存してホスト名リゾリューションおよび `getpwnam()` を実装しているからです。これらの問題は 2 つの方法で明らかです。

- `mysql_install_db` を実行した時に以下のエラーメッセージが表示されます。

```
Sorry, the host 'xxxx' could not be looked up
```

この問題は `mysql_install_db --force` を実行して処理します。それは `mysql_install_db` の `resolveip` テストを実行しません。問題はホスト名をグラント テーブルで使用できないことです。`localhost` 以外に、IP 番号を使用する必要があります。`--force` をサポートしていない旧バージョンの MySQL を使用している場合、テキスト エディタを使用して `mysql_install` の `resolveip` テストを手動で削除する必要があります。

- `mysqld` を `--user` オプションで実行する際にも以下のエラーが表示される場合があります。

```
getpwnam: No such file or directory
```

この問題を回避するには、`--user` オプションを指定しないで `mysqld` を `su` コマンドで実行します。これによりシステムそのもので `mysqld` プロセスのユーザー ID を変更するため `mysqld` はそれをする必要がありません。

別の解決策、それは両方の問題を解決しますが、バイナリのディストリビューションを使用しません。MySQL のソース ディストリビューション (RPM あるいは `tar.gz` フォーマットで) を取得しそれをインストールします。

Linux 2.2 バージョンの中には、クライアントが TCP/IP で `mysqld` サーバへ新たに相当数の接続をしたときに `Resource temporarily unavailable` のエラーが表示される場合があります。この問題は Linux には TCP/IP ソケットを閉じる時間とシステムが実際のそれを開放する間に遅延があるためです。TCP/IP のスロット数には限りがあるため、クライアントが余りにも多くの TCP/IP 接続を短時間にしようとリソース不足のエラーに遭遇します。例えば、MySQL `test-connect` ベンチマークを TCP/IP で実行する時にエラーが発生する場合があります。

弊社ではこの問題に関して Linux の別々のメーリング リストを使用して数回連絡を取っていますが現在までのところまだ解決策は見つかっておりません。唯一これまで分かっている「fix」はクライアントの接続を執拗に続けること、あるいはデータベース サーバとクライアントが同じマシンで稼働している場合、TCP/IP 接続ではなく Unix のソケット ファイル接続を使用することです。

Linux バイナリ ディストリビューションの注釈

以下の `glibc` に関する注釈は MySQL をご自身でビルドする際にのみ適用します。Linux を x86 マシンで使用している場合、弊社のバイナリが多くのケースで最適です。弊社では弊社のバイナリを弊社が知り得る限りベスト パッチの `glibc` バージョンにリンクし最高のコンパイラ オプションを使用して、高負荷のサーバに最適なソリューションを目指しています。一般的なユーザーにとって、多数の同時接続あるいは 2 GB の制限を越えるテーブルでの設定においてさえ、弊社のバイナリはその殆どのケースで最適な選択とします。以下のテキストを読まれた後で、それでも何をすべきか分からない場合には、まず弊社のバイナリを試してみてもお客様のニーズに合うか決めてください。もし要求を十分満たすことができない場合には、ご自身でビルドしてみることもできます。そのような場合、お客様のご意見を寄せて頂ければ弊社はさらによいバイナリをビルドするための参考にさせていただきます。

MySQL は Linux に LinuxThread を使用しています。`glibc2` を実装していない旧 Linux バージョンを使用している場合には、MySQL をコンパイルする前に LinuxThread をインストールする必要があります。LinuxThread は <http://dev.mysql.com/downloads/os-linux.html> で取得できます。

`INSERT DELAYD` ステートメントが発行される時に使用されるバージョン 2.1.1 を含む以前の `glibc` のバージョンには `pthread_mutex_timedwait()` の処理で致命的なバグがあります。`glibc` をアップグレードするまでは `INSERT DELAYED` を使用しないようお勧めします。

Linux kernel および LinuxThread ライブラリはデフォルトで最大 1,024 スレッドを扱うことができます。1,000 以上の同時接続を計画している場合には、以下のように LinuxThread を変更する必要があります。

- `PTHREAD_THREADS_MAX` を `sysdeps/unix/sysv/linux/bits/local_lim.h` で 4096 に増やし `STACK_SIZE` を `linuxthreads/internals.h` で 256KB に減らします。パスは `glibc` のルートに関連しています。(MySQL は `STACK_SIZE` がデフォルトの 2MB の場合には 600-1000 接続では安定しません。)
- Recompile LinuxThreads to produce a new `libpthread.a` library, and relink MySQL against it.

LinuxThreads のスレッド制限の回避法の詳細は <http://www.volano.com/linuxnotes.html> にあります。

MySQL のパフォーマンスに大きな影響を与える別の問題が、特に SMP システム上であります。`glibc` 2.1 の LinuxThreads の mutex の実装が mutex をほんの短時間保持する多くのスレッドを使用するプログラムで非常に貧弱です。これにより逆説的な結果が生じます。MySQL を無修正の LinuxThreads にリンクさせる際、SMP からプロセスを削除すると MySQL パフォーマンスが実質的に多くのケースで改善されます。弊社ではこの振る舞いを修正するためのパッチを `glibc` 2.1.3 に用意しました<http://dev.mysql.com/Downloads/Linux/linuxthreads-2.1-patch>。

`glibc` 2.2.2 を使用する際、MySQL に adaptive の mutex を使用することによって `glibc` 2.1.3 にパッチ版を使用するよりはるかによくなります。しかし、使用条件によっては、`glibc` 2.2.2 の現在の mutex code はオーバースピンし、MySQL パフォーマンスを低下させるので、その点注意が必要です。この現象の再現を `mysqld` のプロセスを最高度まで再調整することで減らすことができます。弊社ではまたパッチを使用してオーバースピンを修正できるようにしました。そのパッチは <http://dev.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch> にあります。そのパッチでオーバースピン、スレッドの最大数、およびスタックのスペースをすべて一つにまとめるなどの修正を加えています。そのパッチは `linuxthreads` ディレクトリに `patch -p0 <tmp/linuxthreads-2.2.2.patch` を適用します。弊社ではそのパッチが `glibc` 2.2 の今後のリリースに反映されることを希望しています。いずれにしろ、`glibc` 2.2.2 にリンクするには、`STACK_SIZE` および `PTHREAD_THREADS_MAX` をまだ修正する必要があります。弊社ではデフォルトの値が、今後 MySQL の高負荷設定に対応できるように修正され、お客様ご自身でビルドする際に必要なコマンドが `./configure; make; make install` まで減少できるように期待しています。

これらのパッチは特定の `libpthread.a` の静的バージョンのビルドに使用し、MySQL に静的にリンクする際のみ使用することをお勧めします。これらのパッチは MySQL に使用する際は安全でそのパフォーマンスを大幅に改善しますが、他のアプリケーションに対する影響についてはなにも言う立場にありません。LinuxThreads をライブラリのパッチ版の静的バージョンが必要な他のアプリケーションにリンクする、あるいはパッチ共有バージョンをビルドしそれをお客様のシステムにインストールすることは、お客様ご自身のリスクで行うことになります。

MySQL のインストールの最中に予想外の問題に遭遇した場合、あるいはそれに一般的なユーティリティのハンギングが伴う場合、それらの問題はライブラリあるいはコンパイラにいずれかに起因するものである確立が高いといえます。このような場合には、弊社のバイナリがその問題を解決します。

お客様ご自身の MySQL クライアント プログラムをリンクさせる場合、ランタイムで以下のエラーが表示される場合があります。

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

この問題は以下のメソッドのいずれかで回避できます。

- クライアントを `-Lpath` ではなく `-Wl,r/full/path/to/libmysqlclient.so` フラグにリンクします。
- `libmysqlclient.so` を `/usr/lib` にコピーします。
- クライアントを起動する前に `libmysqlclient.so` のディレクトリのパス名を `LD_RUN_PATH` 環境変数に追加します。

富士通のコンパイラ (`fcc/FCC`) を使用する場合、Linux のヘッダーファイルは `gcc` に特化したものですので MySQL のコンパイルで問題が発生する場合があります。以下の `configure` 行が `fcc/FCC` の役に立つはずで

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
-DCONST=const -DNO_STRTOLL_PROTO" \
CXX=fcc CXXFLAGS="-O -K fast -K lib \
-K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE \
-DCONST=const -Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
-D_EXTERN_INLINE=static __inline" \
./configure \
--prefix=/usr/local/mysql --enable-assembly \
--with-mysqld-ldflags=-all-static --disable-shared \
--with-low-memory
```

Linux のインストール後の注釈

`mysql.server` は MySQL インストール ディレクトリの `support-files` ディレクトリあるいは MySQL のソース ツリーにあります。それを自動的起動・シャットダウンの `/etc/init.d/mysql` としてインストールします。[MySQL を自動的に起動・停止する](#) 参照。

MySQL が十分なファイルや接続ができない場合、十分なファイル処理する Linux を設定していない場合があります。

Linux 2.2 およびそれ以降では、割り当てられたファイル処理を以下のようにチェックできます。

```
shell> cat /proc/sys/fs/file-max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

16MB 以上のメモリがある場合、何か以下のようなものを init スクリプト (例えば、SuSE Linux の `/etc/init.d/boot.local`) を追加する必要があります。

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

コマンドラインの `echo` コマンドを `root` として実行することもできます。しかし、これらの設定は次回コンピュータを再起動した時には失われます。

代わりに、これらのパラメータを `sysctl` ツールを使用して起動時に設定できます。それは多くの Linux ディストリビューション (SuSE Linux 8.0 およびそれ以降を含む) で使用されています。以下の値を `/etc/sysctl.conf` の名前のファイルに入れます。

```
# Increase some values for MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

また以下を `/etc/my.cnf` に追加する必要があります。

```
[mysqld_safe]
open-files-limit=8192
```

これによりサーバの接続総数およびオープン ファイルの上限 8,192 まで可能になります。.

LinuxThreads の `STACK_SIZE` 定数によりアドレス スペースのスレッド スタックのスペースを管理します。その管理は各スレッド スタックに十分余裕をあたえられるように大きく、しかしスレッドのスタックが `mysqld` データに入り込まないように小さくなくてはなりません。幸いなことに、現在使用しているアドレスのマップを外す要求を出した場合 Linux の `mmap()` の実装によってマップした領域のマップを外し、エラーを返さずにページ全体のデータをゼロにすることが弊社の実験で分かりました。ですから `mysqld` あるいは他のスレッド アプリケーションの安全はスレッドを作成するコードの「紳士的な」振る舞いにかかっています。ユーザーは所定の時間での実行中のスレッド数がスレッド スタックがグローバル ヒープに近寄らないように十分に低くなっているようにするための対策を講じる必要があります。`mysqld` を使用して、適切な値を `max_connections` 変数に設定することでこの振る舞いを強化する必要があります。

MySQL をご自身でビルドする場合、スタックをよくするために LinuxThreads をパッチします。[Linux バイナリ ディストリビューションの注釈](#) 参照。LinuxThreads のパッチを望まない場合、`max_connections` を 500 以下の値に設定します。その値は大きなキーバッファ、大きなヒープ テーブル、あるいは `mysqld` に大きな容量のメモリを割り当てる何かがある、または 2.2 kernel を 2GB パッチで動作させる場合にはもっと小さい値にする必要があるかも知れません。弊社のバイナリあるいは RPM バージョンを使用する場合、大きなキーバッファあるいは多量のデータを伴う大きなヒープ テーブルがない場合、`max_connections` が 1500 に設定しても安全です。LinuxThreads の `STACK_SIZE` が小さければ小さいほど、多くのスレッドを安全に作成できます。弊社では 128KB から 256KB の間の値をお勧めしています。

多数の同時接続を使用する場合、フォーキングあるいは子のクローン化でプロセスを妨げることによってフォーク ボム アタックを妨げようとする 2.2 kernel の「feature」によって影響を受ける場合があります。これによって 同時クライアントが増えるに従って MySQL はうまくスケールできなくなります。シングルの CPU を使用したシステムで、この例を非常に遅いスレッドで経験しました。MySQL に接続するのに長い時間 (1 分近く) がかかる場合があり、まさにそれをシャットダウンするほどの時間がかかります。1 台の複数の CPU を使用したシステムでは、クライアント

数が増えるに従ってクエリ速度が徐々に遅くなっていきました。解決策を探す段階で、弊社のユーザーの 1 社からユーザーのサイトで効果があったとのことで kernel のパッチが届けられました。このパッチは <http://dev.mysql.com/Downloads/Patches/linux-fork.patch> で利用できます。弊社ではこのパッチに弊社の開発や実際の生産システムを含む広範なテストを実施しました。そのパッチは何ら問題なく MySQL のパフォーマンスを大幅に改善しましたので、弊社では高負荷のサーバを 2.2 kernel で運用している弊社のユーザーに推薦しました。

この問題は 2.4 kernel では修正されているため、現在稼働中のシステムのパフォーマンスに満足されていない場合には 2.2 のパッチ版よりはむしろ、2.4 へのアップグレードするほうが容易です。SMP システムでは、アップグレードすることによって既知のバグの修正に加えて SMP のブーストがよくなります。

弊社では MySQL を 2.4 kernel で 2 つの CPU を搭載したマシンでテストしましたが MySQL スケールが大幅に良くなっています。1,000 クライアントまでのテストを実施している間に実質的にクエリ速度が速くなり、MySQL スケーリング ファクターは (最大のスループットと 1 台のクライアント スループットの比率を計算したもの) は 180% でした。CPU を 4 つ搭載したシステムでも同様の結果が得られました。クライアント数を 1,000 まで増やしても実質的な減速がなく、スケーリング ファクターは 300% でした。これらの結果に基づいて、2.2 kernel を使用した高負荷の SMP サーバには、弊社でこの段階で 2.4 kernel にアップグレードするよう強くお勧めします。

最高のパフォーマンスを実現するには最優先課題として `mysqld` を 2.4 kernel で動作させることが必須であるということが分かりました。これを行うには `renice -20 $$` コマンドを `mysqld_safe` に追加します。4 CPU 搭載マシン上での弊社のテストでは、その優先度を上げることによって 400 クライアントまでの増加の間に 60% の結果が出ました。

弊社では現在 2.4 kernel を four-way および eight-way のシステムで使用した場合 MySQL の性能がどのように良くなるか情報を集めています。お客様がそのようなシステムにアクセスして何かベンチマークを行った場合、その結果を [<benchmarks@mysql.com>](mailto:benchmarks@mysql.com) まで Eメール頂ければ幸いです。弊社ではそれを検討してマニュアルに追加するかどうかを検討します。

`ps` で `mysqld` サーバプロセスがデッドした場合、これは通常 MySQL にバグがあるかあるいはテーブルが破損していることを意味します。[What to Do If MySQL Keeps Crashing](#) 参照。

`mysqld` が SIGSEGV 信号でデッドした場合に Linux でコア ダンプを取得するには、`mysqld` を `--core-file` オプションで起動します。多分コア ファイルのサイズを `ulimit -c 1000000` を `mysqld_safe` に追加あるいは `mysqld_safe` を `--core-file-size=1000000` で起動して増やす必要があります。[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) 参照。

Linux x86 注釈

MySQL には `libc` 5.4.12 あるいはそれ以降が必要です。`libc` 5.4.46 での動作は確認されています。`glibc` 2.0.6 およびそれ以降もまた動作します。Red Hat 社の `glibc` RPM に問題があります。問題が発生した場合には、何か更新があるかチェックしてください。`glibc` 2.0.7-19 および 2.0.7-29 RPM の動作は確認されています。

Red Hat 8.0 あるいは新しい `glibc` 2.2.x ライブラリを使用している場合、`mysqld` が `gethostbyaddr()` でデッドする場合があります。これは新しい `glibc` ライブラリのこの呼び出しにスタック サイズ 128KB 以上が必要だからです。この問題を修正するには、`mysqld` を `--thread-stack=192K` オプションで実行します。(MySQL 4 の前に `-O thread_stack=192K` を使用します。) このスタック サイズは MySQL 4.0.10 あるいはそれ以降ではデフォルトですので、問題が発生することはないはずです。

MySQL のコンパイルに `gcc` 3.0 あるいはそれ以降を使用している場合、MySQL をコンパイルする前に `libstdc++v3` ライブラリをインストールする必要があります。それをインストールしなかった場合、リンク中に `__cxa_pure_virtual` 記号が不明のメッセージが表示されます。

Linux ディストリビューションの旧バージョンの中には、`configure` が以下のようなエラーを表示するものもあります。

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

エラーメッセージに従ってその通りに作業します。アンダースコアが 1 つだけの `_P` マクロ名にもう 1 つのアンダースコアを加えて、再度試します。

コンパイルする時に警告が発せられる場合があります。以下の表示は無視して構いません。

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function `void init_signals()':
```

```
mysql.cc:315: warning: assignment of negative value '-1' to
'long unsigned int'
mysql.cc: In function 'void * signal_hand(void *)':
mysql.cc:346: warning: assignment of negative value '-1' to
'long unsigned int'
```

mysql が起動する時に常にコアをダンプする場合は、その問題は多分旧 `/lib/libc.a` に因ります。その名前を変更し、次に `sql/mysql` を削除し、新たに `make install` を行い再度試します。この問題は Slackware のインストールで既に幾つか報告を受けております。

mysql をリンク中に以下のエラーが表示された場合、`libg++.a` が正しくインストールされていないことを意味します。

```
/usr/lib/libc.a(putc.o): In function '_IO_putc':
putc.o(.text+0x0): multiple definition of '_IO_putc'
```

`configure` を以下のように実行すると `libg++.a` を使用しないで済みます。

```
shell> CXX=gcc ./configure
```

Linux SPARC に関する注釈

実装の際に、`readdir_r()` が破損する場合があります。その兆候としては `SHOW DATABASES` ステートメントが常に空のセットを返します。これは設定後のコンパイルする前に `HAVE_READDIR_R` を `config.h` から削除することで修正できます。

Linux Alpha に関する注釈

弊社の MySQL 5.1 の Alpha でのベンチマークおよびテストスイートのテストでは、動作に問題がないものと思われます。

弊社では現在 MySQL のバイナリパッケージを AXP 用 SuSE Linux 7.0、kernel 2.4.4-SMP、Compaq C compiler (V6.2-505) および Compaq C++ compiler (V6.3-006) に対して Alpha EV6 プロセッサを搭載した Compaq DS20 マシンでビルドしています。

供述のコンパイラは <http://www.support.compaq.com/alpha-tools/> にあります。これらのコンパイラを `gcc` の代わりに使用して、およそ 9-14% ほどよい MySQL パフォーマンスが出ています。

Alpha の MySQL には、`-arch generic` フラグをコンパイルに使用しており、このことはそのバイナリはすべての Alpha プロセッサで動作することを意味しています。また、バイナリの問題を避けるために静的にコンパイルしています。`configure` コマンドは以下のようになります。

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \
./configure --prefix=/usr/local/mysql --disable-shared \
--with-extra-charsets=complex --enable-thread-safe-client \
--with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared
```

`egcs` を使用する場合、以下の `configure` 行が弊社では機能しました。

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --disable-shared
```

MySQL を Linux-Alpha で動作させた場合の既知の問題:

- スレッドアプリケーションの MySQL のようなデバッグは `gdb 4.18` ではうまくいきません。代わりに `gdb 5.1` を使用する必要があります。
- `gcc` を使用して `mysql` を静的にリンクすると、結果のイメージが起動時にコアをダンプします。換言すると、`--with-mysqld-ldflags=-all-static` を `gcc` では使用しないということになります。

Linux PowerPC に関する注釈

MySQL は MkLinux 上では最新の `glibc` パッケージで動作します (`glibc 2.0.7` でのテスト)。

Linux MIPSに関する注釈

MySQL を Qube2 (Linux Mips) で動作させるには、最新の [glibc](#) ライブラリが必要です。[glibc-2.0.7-29C2](#) は動作するといわれています。また [egcs](#) C++ コンパイラ ([egcs](#) 1.0.2-9、[gcc](#) 2.95.2 あるいはそれ以降) を使用する必要があります。

Linux IA-64 に関する注釈

MySQL を Linux IA-64 でコンパイルするには、弊社では以下の [configure](#) コマンドを [gcc](#) 2.96 でビルドする際に使用しています。

```
CC=gcc \
CFLAGS="-O3 -fno-omit-frame-pointer" \
CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql \
"--with-comment=Official MySQL binary" \
--with-extra-charsets=complex
```

IA-64 には、MySQL クライアントは共有バイナリを使用します。このことは弊社のバイナリ ディストリビューションを [/usr/local/mysql](#) と異なるロケーションにインストールするには、[libmysqlclient.so](#) をインストールしたディレクトリのパスを [/etc/ld.so.conf](#) ファイルあるいは [LD_LIBRARY_PATH](#) 環境変数の値のいずれかに追加する必要があります。

[Problems Linking to the MySQL Client Library](#) を参照してください。

SELinux の注釈

RHEL4 は SELinux と一緒に出荷され、プロセスのタイトなアクセスをサポートします。If SELinux が有効になると ([/etc/selinux/config](#) の [SELINUX](#) が [enforcing](#) に設定され、[SELINUXTYPE](#) が [targeted](#) あるいは [strict](#) のいずれかに設定される)、MySQL AB RPM パッケージをインストールする際に問題に遭遇する場合があります。

Red Hat 社にはこの問題を修正する更新版があります。その中にはMySQL AB が提供した RPM のインストール構成を処理する「セキュリティ ポリシー」の仕様が含まれています。詳細は、https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=167551 および <http://rhn.redhat.com/errata/RHBA-2006-0049.html> を参照してください。

Mac OS X に関する注釈

On Mac OS X、[tar](#) は長いファイル名は扱えません。[.tar.gz](#) ディストリビューションの解凍はが必要な場合は、[gnutar](#) を参照してください。

Mac OS X 10.x (Darwin)

MySQL は大きな問題なしに Mac OS X 10.x (Darwin) で動作する必要があります。

既知の問題：

- 高負荷でパフォーマンスに問題がある場合には、[--skip-thread-priority](#) オプションを [mysqld](#) に試してみてください。これによりすべてのスレッドを同じ優先度で実行します。On Mac OS X、これは Apple 社が少なくともそのスレッドスケジューラを修正すれば、優れたパフォーマンスを提供します。
- 接続回数 ([wait_timeout](#)、[interactive_timeout](#) および [net_read_timeout](#)) は保証されていません。

これは多分にスレッドライブラリの信号処理の問題で、信号が未処理の読み込みを終了できないため、今後のスレッドライブラリの更新でこの問題の修正を期待しています。

弊社の Mac OS X 用バイナリは Darwin (ダーウィン) 6.3 に以下の [configure](#) 行でコンパイルしています。

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql \
--with-extra-charsets=complex --enable-thread-safe-client \
```

```
--enable-local-infile --disable-shared
```

Mac OS X に MySQL をインストールする を参照してください。

Mac OS X Server 1.2 (Rhapsody)

現在の Mac OS X サーバには、MySQL をコンパイルする前のオペレーティング システムの変更は必要ありません。サーバ プラットフォームのコンパイルは Mac OS X のクライアント バージョンの場合と同じです。

旧バージョン (Mac OS X Server 1.2、a.k.a. Rhapsody) には、MySQL を設定する前に最初に pthread パッケージをインストールする必要があります。

Mac OS X に MySQL をインストールする を参照してください。

Solaris に関する注釈

PKG ディストリビューションを使用した Solaris への MySQL のインストールに関する情報は、[Solaris に MySQL をインストールする](#) を参照してください。

Solaris の `tar` が長いファイル名を扱えないので、Solaris を使用する際、MySQL のディストリビューションを解凍する前でさえ、問題に遭遇する場合があります。これは MySQL を解凍する時にエラーが表示されることを意味します。

この問題が発生したら、GNU `tar` (`gtar`) を使用してディストリビューションを解凍します。Solaris 用にコンパイルしたコピーは <http://dev.mysql.com/downloads/os-solaris.html> にあります。

Sun のネイティブ スレッドは Solaris 2.5 およびそれ以降でしか動作しません。Solaris 2.4 およびそれ以前には、MySQL 自動的に MIT-pthreads を使用します。[MIT-pthreads ノート](#) 参照。

`configure` で以下のエラーが表示された場合には、コンパイラのインストールに問題があることを意味しています。

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

この場合、コンパイラを新しいバージョンにアップグレードします。この問題を `config.cache` ファイルに以下の行を追加することでも解決できる場合があります。

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

SPARC で Solaris を使用している場合、推奨しているコンパイラは `gcc` 2.95.2 あるいは 3.2 です。これは <http://gcc.gnu.org/> にあります。`egcs` 1.1.1 および `gcc` 2.8.1 は SPARC では信頼性に欠けます。

`gcc` 2.95.2 を使用する際に推奨している `configure` 行は:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
--enable-assembler
```

UltraSPARC システムの場合、`-mcpu=v8 -Wa,-xarch=v8plusa` を `CFLAGS` および `CXXFLAGS` 環境変数に追加すると 4 % パフォーマンスが向上します。

Sun の Forte 5.0 (あるいはそれ以降) コンパイラには、`configure` を以下のように実行します。

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

Sun の Forte コンパイラで 64 ビットのバイナリを作成するには、以下の設定オプションを使用します。

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

`gcc` を使用して 64 ビットの Solaris バイナリを作成するには、`-m64` を `CFLAGS` および `CXXFLAGS` に追加して `--enable-assembler` を `configure` 行から削除します。

MySQL ベンチマークでは、Forte 5.0 を 32 ビットモードで使用すると、gcc 3.2 を `-mcpu` フラグで使用した場合に比べて UltraSPARC 上で速度が 4% 速くなります。

64-bit `mysqld` バイナリは、32 ビット バイナリより速度は 4% 遅くなりますが、さらに多くのスレッドおよびメモリを処理できます。

Solaris 10 を x86_64 で使用する場合、InnoDB ファイルを `forcedirectio` オプションで保持するファイルシステムを実装する必要があります。(デフォルトではこのオプションは実装されていません。)それを実装しないと InnoDB ストレージ エンジンがこのプラットフォームで使用的した場合性能が大幅に劣化します。

`fdatasync` あるいは `sched_yield` で問題があった場合、`LIBS=-lrt` を `configure` 行に追加すると修正できます。

WorkShop 5.3 以前のコンパイラの場合、`configure` スクリプトを編集する必要があります。この行を：

```
#if !defined(__STDC__) || __STDC__ != 1
```

以下のように変更します。

```
#if !defined(__STDC__)
```

`__STDC__` を `-Xc` オプションで起動すると、Sun のコンパイラは Solaris `pthread.h` ヘッダーファイルではコンパイルできません。これじゃ Sun のバグです (破損したコンパイラあるいは破損した include ファイル)。

`mysqld` を実行したときに以下のエラーメッセージが表示された場合、`-mt` マルチ スレッド オプションを有効にしないで MySQL を Sun のコンパイラでコンパイルしてみます。

```
libc internal error: _rmutex_unlock: rmutex not held
```

`-mt` を `CFLAGS` および `CXXFLAGS` 追加して再コンパイルします。

gcc の SFW バージョン (Solaris 8 に同梱) を使用する場合、`/opt/sfw/lib` を環境変数 `LD_LIBRARY_PATH` に `configure` を実行数r前に追加します。

gcc を sunfreeware.com から入手して使用する場合、多くの問題が発生する場合があります。これを避けるには、gcc および GNU `binutils` をそれらを動作しているマシンでコンパイルする必要があります。

MySQL を gcc でコンパイル中に以下のエラーが表示された場合、gcc が 使用している Solaris のバージョンに合っていないことを意味します。

```
shell> gcc -O3 -g -O2 -DDEBUG_OFF -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

この場合の適切な方法は gcc の最新バージョンを入手してそれを現在の gcc コンパイラでコンパイルすることです。少なくとも Solaris 2.5、gcc のほとんどすべてのバイナリ バージョンは古くて、使用できない include ファイルを使用しており、スレッドを使用するすべてのプログラムを破損し、同様に他のプログラムも破損させる可能性があります。

Solaris ではどのシステム ライブラリも静的バージョン (`libpthread` および `libdl`) を提供していませんので、MySQL を `--static` でコンパイルすることはできません。コンパイルしようとすると、以下のエラーが表示されます。

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

お客様ご自身の MySQL クライアント プログラムをリンクさせる場合、ランタイムで以下のエラーが表示される場合があります。

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

この問題は以下のメソッドのいずれかで回避できます。

- クライアントを `-Lpath` ではなく `-Wl,r/full/path/to/libmysqlclient.so` フラグにリンクします。
- `libmysqlclient.so` を `/usr/lib` にコピーします。

- クライアントを起動する前に `libmysqlclient.so` のディレクトリのパス名を `LD_RUN_PATH` 環境変数に追加します。

`zlib` をインストールせずに `-lz` にリンクする際 `configure` に問題がある場合、2 つのオプションがあります。

- 圧縮通信プロトコルを使用するには、`zlib` を ftp.gnu.org から取得してインストールします。
- MySQL をビルドするには `configure` を `--with-named-z-libs=no` オプションで実行します。

`gcc` を使用してユーザー定義関数 (UDFs) の MySQL へのロードで問題がある場合、`-lgcc` を UDF のリンク行に追加してみます。

MySQL 自動的に起動するには、`support-files/mysql.server` を `/etc/init.d` にコピーして、シンボリックリンクを `/etc/rc3.d/S99mysql.server` の名前のそれに作成します。

あまりにも多くのプロセスが急激に `mysqld` に接続を試みた場合、MySQL ログに以下のエラーが記録されます。

```
Error in accept: Protocol error
```

この問題の回避策としてサーバを `--back_log=50` オプションで起動してみます。(MySQL 4 の前に `-O back_log=50` を使用します。)

Solaris は `setuid()` アプリケーションのコア ファイルをサポートしていないので、`--user` オプションを使用している場合、コア ファイルを `mysqld` から取得できません。

Solaris 2.7/2.8 に関する注釈

通常、Solaris 2.6 のバイナリを Solaris 2.7 および 2.8 で使用できます。Solaris 2.6 公開版のほとんどもまた Solaris 2.7 および 2.8 に適用できます。

MySQL は自動的に Solaris の新バージョンを検知して以下の問題に対して回避策を取ります。

Solaris 2.7 / 2.8 の include ファイルにはいくつかバグがあります。`gcc` を使用すると以下のエラーが表示される場合があります。

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

この問題が発生したら `/usr/include/widec.h` を `.../lib/gcc-lib/os/gcc-version/include` にコピーして行 41 を以下から：

```
#if !defined(lint) && !defined(__lint)
```

以下のように変更します。

```
#if !defined(lint) && !defined(__lint) && !defined(getwc)
```

また、`/usr/include/widec.h` を直接編集することもできます。いずれの場合も、修正を加えた後に `config.cache` を削除し `configure` を再度実行します。

`make` を実行した時に以下のエラーが表示された場合、`configure` が `curses.h` ファイルを検知できなかったことを意味します (多分 `/usr/include/widec.h` のエラーによる)。

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `;'
/usr/include/term.h:1081: syntax error before `;'
```

この問題を解決するには以下のいずれかを実行します。

- `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure` で設定する。
- `/usr/include/widec.h` を前述の説明に従って編集し `configure` を再度実行する。
- `#define HAVE_TERM` 行を `config.h` ファイルから削除し `make` を再度実行する。

クライアント プログラムをリンク中にリンクが `-lz` を見つけられない場合、その問題は多分 `libz.so` ファイルが `/usr/local/lib` にインストールされているからです。この問題を解決するには以下のメソッドのいずれかを実行します。

- `/usr/local/lib` を `LD_LIBRARY_PATH` に追加する。
- リンクを `libz.so` に `/lib` から追加する。
- Solaris 8 を使用している場合、オプションの `zlib` を Solaris 8 CD ディストリビューションからインストールできます。
- MySQL をビルドするには `configure` を `--with-named-z-libs=no` オプションで実行します。

Solaris x86 に関する注釈

x86 の Solaris 8 では、`strip` を使用してデバッグ記号を削除すると `mysqld` がコアをダンプします。`strip`。

`gcc` あるいは `egcs` を Solaris x86 で使用中の負荷下でコア ダンプの問題が発生したら、以下の `configure` コマンドを使用します。

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti -DHAVE_CURSES_H" \
./configure --prefix=/usr/local/mysql
```

これにより `libstdc++` ライブラリおよび C++ 例外で問題を回避します。

これで問題が解決できない場合、デバッグ バージョンをコンパイルし、それをトレース ファイルあるいは `gdb` で実行します。[Debugging mysqld under gdb](#) 参照。

BSD に関する注釈

この項では MySQL を異なる BSD Unix で使用する際の情報を提供します。.

FreeBSD に関する注釈

スレッド パッケージよく統合されているため、MySQL の使用に FreeBSD 4.x あるいはそれ以降の使用をお勧めします。セキュアで安定してシステムにするには、`-RELEASE` の印の付いた FreeBSD kernel のみを使用します。

MySQL のインストールで最も容易 (で推奨される) な方法は `mysql-server` ポートおよび `mysql-client` ポートを使用することです。それらは <http://www.freebsd.org/> で入手できます。これらのポートを使用することで以下のメリットがあります。

- FreeBSD バージョンで動作することが知られているすべての最適化を行った MySQL
- 自動設定およびビルド
- `/usr/local/etc/rc.d` にインストールされた自動スクリプト
- どのファイルがインストールされているかを確認するための `pkg_info -L` の使用
- お客様のマシンで MySQL をもはや必要ない場合にそれを削除できる `pkg_delete` の使用

MIT-pthreads を FreeBSD 2.x で、ネイティブ スレッドを FreeBSD 3 およびそれ以降で使用するようお勧めします。ネイティブ スレッドを 2.2.x バージョンで動作できますが、`mysqld` のシャットダウンの問題が発生する場合があります。

不幸にも、FreeBSD のある種の関数のコールがまだ完全にスレッド セーフになっていません。最も注目に値するのは、これには `gethostbyname()` 関数が含まれているので、これによって MySQL でホスト名を IP アドレスに変換できます。ある環境下では、`mysqld` プロセスが突然 100% CPU 負荷になり、応答がなくなります。この問題に遭遇した場合には、`--skip-name-resolve` オプションを使用して MySQL を起動してみます。

あるいは、FreeBSD の MySQL を LinuxThreads ライブラリにリンクすることによって、ネイティブの FreeBSD のスレッド実装による幾つかの問題を回避することができます。LinuxThreads とネイティブ スレッドの比較をよくまとめた Jeremy Zawodny の記事 あなたの MySQL サーバにどちらを選ぶ、FreeBSD それとも Linux ? が <http://jeremy.zawodny.com/blog/archives/000697.html> にあります。

FreeBSD 上の LinuxThreads の既知の問題:

- 接続回数 ([wait_timeout](#)、[interactive_timeout](#) および [net_read_timeout](#)) の値は保証されていません。この問題の兆候としては、接続が執拗に非常に長い間遮断されることなく続き、スレッドの「停止」の効果がなく スレッドが新しいコマンドによって停止されるまで続きます。

これは多分スレッド ライブラリの信号処理の問題で信号が未処理の読み込みを中断できないために起こるのものと考えられます。この問題は FreeBSD 5.0 では修正される予定です。

MySQL ビルド プロセスが機能するには GNU make ([gmake](#)) が必要です。GNU [make](#) が利用できない場合、MySQL をコンパイルする前にインストールする必要があります。

[gcc](#) (2.95.2 and up) の FreeBSD に MySQL をコンパイルしてインストールする方法として推奨しているのは :

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \
CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions \
-felide-constructors -fno-strength-reduce" \
./configure --prefix=/usr/local/mysql --enable-assembler
gmake
gmake install
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
bin/mysql_safe &
```

[configure](#) が MIT-pthreads を使用している場合、MIT-pthreads の注釈を読む必要があります。[MIT-pthreads ノート](#) 参照。

[make install](#) から [/usr/include/pthreads](#) が見つからないとのエラーが表示された場合、[configure](#) が MIT-pthreads に必要なそれを検知しなかったことになります。この問題を修正するには、[config.cache](#) を削除して、次に [configure](#) を [--with-mit-threads](#) オプションで実行します。

名前のリゾルバーが正しく設定されているか確認します。正しく設定されていないと、リゾルバーの遅延あるいは [mysqld](#) への接続に失敗します。また、[localhost](#) の [/etc/hosts](#) ファイルへのエントリが正しいか確認します。このファイルは以下のような行で実行されます。

```
127.0.0.1 localhost localhost.your.domain
```

FreeBSD は非常に小さいデフォルトのファイル処理制限があることで知られています。['File' Not Found and Similar Errors](#) 参照。サーバを [--open-files-limit](#) オプションを使用して [mysqld_safe](#) に起動しするか、あるいは [/etc/login.conf](#) の [mysql](#) ユーザーの制限を上げそれを [cap_mkdb /etc/login.conf](#) で再度ビルドします。また、デフォルト ([chpass mysqld-user-name](#) を使用する) を使用していない場合、パスワード ファイルのこのユーザーに適切なクラスを設定しているか確認します。[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) 参照。

FreeBSD の制限はで、システムに利用できる大きな RAM がある場合でも、プロセスのサイズを 512MB に制限します。それで以下のようなエラーが表示されます。

```
Out of memory (Needed 16391 bytes)
```

現在のバージョンの FreeBSD (最低でも 4.x およびそれ以降) では、この制限を以下のエントリを [/boot/loader.conf](#) ファイルに追加し、マシンをリブートして増やすことができます (これらはランタイム時に [sysctl](#) コマンドで変更できる設定ではありません)。

```
kern.maxdsiz="1073741824" # 1GB
kern.dfdsiz="1073741824" # 1GB
kern.maxssiz="134217728" # 128MB
```

FreeBSD の旧バージョンでは、kernel を再コンパイルしてプロセスの最大データ セグメント サイズを変更する必要があります。この場合の詳細に関しては [LINT](#) 設定ファイルの [MAXDSIZ](#) オプションを参照してください。

MySQL の現在の日付で問題がある場合には、[TZ](#) 変数の設定が役に立ちます。[環境変数](#) 参照。

NetBSD に関する注釈

NetBSD でコンパイルするには、GNU [make](#) が必要です。それがないと、ビルド プロセスは [make](#) が [lint](#) を C++ ファイルで実行する際に失敗します。

OpenBSD 2.5 に関する注釈

OpenBSD 2.5 では、MySQL をネイティブのスレッドで以下のオプションを使用してコンパイルできます。

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

BSD/OS Version 2.x に関する注釈

MySQL をコンパイル中に以下のエラーが表示された場合、仮想メモリの `ulimit` 値が小さすぎます。

```
item_func.h: In method
`Item_func_ge::Item_func_ge(const Item_func_ge &):
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

`ulimit -v 80000` を使用して試し `make` を再度実行します。これで修正できずしかも `bash` を使用している場合には、`csch` あるいは `sh` に切り替えてみます。BSDI のユーザー数社から `bash` と `ulimit` の問題が報告されています。

`gcc` を使用している場合、`--with-low-memory` フラグを `configure` に使用して `sql_yacc.cc` をコンパイルすることもできます。

MySQL の現在の日付で問題がある場合には、`TZ` 変数の設定が役に立ちます。[環境変数](#) 参照。

BSD/OS Version 3.x に関する注釈

BSD/OS 3.1 へのアップグレード。アップグレードできなかった場合、BSDIpatch M300-038 をインストールします。

MySQL を設定する際以下のコマンドを使用します。

```
env CXX=shlcc++ CC=shlcc2 \
./configure \
--prefix=/usr/local/mysql \
--localstatedir=/var/mysql \
--without-perl \
--with-unix-socket-path=/var/mysql/mysql.sock
```

以下が動作することが知られています。

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure \
--prefix=/usr/local/mysql \
--with-unix-socket-path=/var/mysql/mysql.sock
```

任意にディレクトリのロケーションを変更できます。特に指定するロケーションがない場合はデフォルトを使用します。

高負荷でパフォーマンスに問題がある場合には、`--skip-thread-priority` オプションを `mysqld` に試してみてください。これによりすべてのスレッドを同じ優先度で実行します。BSDI 3.1では、少なくとも BSDI がそのスレッドスケジューラを修正すればよい性能が得られます。

コンパイル中にエラー `virtual memory exhausted` が表示されたら、`ulimit -v 80000` を使用して `make` を再度実行します。これで修正できずしかも `bash` を使用している場合には、`csch` あるいは `sh` に切り替えてみます。BSDI のユーザー数社から `bash` と `ulimit` の問題が報告されています。

BSD/OS Version 4.x に関する注釈

BSDI 4.x にはスレッド関連のバグがいくつかあります。これに MySQL を使用する場合には、スレッド関連のすべてのパッチをインストールする必要があります。少なくとも M400-023 はインストールしてください。

BSDI 4.x システム上で、共有ライブラリに問題がある場合もあります。問題の兆候としては、どのクライアントプログラムも、例えば `mysqladmin` を実行できません。この場合、共有ライブラリを `--disable-shared` オプションで設定しないように再設定します。

カスタマーの中には BSDI 4.0.1 で `mysqld` バイナリがしばらくの間テーブルを開かないとの問題が出ています。この問題はライブラリ/システム関連のバグによって `mysqld` が現在のディレクトリの変更要求を出さずに現在のディレクトリを変更するからです。

これを修正するには MySQL を最低でもバージョン 3.23.34 にアップグレードするか、`configure` の実行後に行 `#define HAVE_REALPATH` を `config.h` から `make` を実行する前に削除します。

これはデータベース ディレクトリをシンボリックに別のデータベース ディレクトリにリンクできない、あるいはテーブルを別のデータベースにシンボリックにリンクできないことを意味します。(シンボリックに別のディスクにリンクすることはできます)。

他の Unix に関する注釈

HP-UX バージョン 10.20 に関する注釈

MySQL を HP-UX 上でコンパイルする際に 2, 3 の小さな問題があります。`gcc` がよいコードを生成しますので HP-UX ネイティブ コンパイラではなく `gcc` の使用をお勧めします。

`gcc 2.95` を HP-UX 上で使用することをお勧めしています。高最適化のフラグ (`-O6` など) は HP-UX 上では安全ではないので使用しないでください。

以下の `configure` 行は `gcc 2.95` で作用します。

```
CFLAGS="-I/opt/dce/include -fpic" \
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \
-fno-rtti" \
CXX=gcc \
./configure --with-pthread \
--with-named-thread-libs='ldce' \
--prefix=/usr/local/mysql --disable-shared
```

以下の `configure` 行は `gcc 3.1` で作用します。

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors \
-fno-exceptions -fno-rtti -O3 -fPIC" \
./configure --prefix=/usr/local/mysql \
--with-extra-charsets=complex --enable-thread-safe-client \
--enable-local-infile --with-pthread \
--with-named-thread-libs=ldce --with-lib-ccflags=-fPIC \
--disable-shared
```

HP-UX バージョン 11.x に関する注釈

標準の HP-UX ライブラリに深刻なバグが幾つかあるため、以下のパッチを MySQL を HP-UX 11.0 上で動作させる前にインストールします。

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

これにより `EWOULDBLOCK` を `recv()` から `EBADF` を `accept()` をスレッド化したアプリケーションで取得する問題を解決します。

`gcc 2.95.1` をパッチなしの HP-UX 11.x システムで使用すると、以下のエラーが発生する場合があります。

```
In file included from /usr/include/unistd.h:11,
from ../include/global.h:125,
from mysql_priv.h:15,
from item.cc:19:
/usr/include/sys/unistd.h:184: declaration of C function ...
/usr/include/sys/pthread.h:440: previous declaration ...
In file included from item.h:306,
from mysql_priv.h:158,
from item.cc:19:
```

この問題は HP-UX が `pthread_atfork()` を安定的に定義しないために起こります。`/usr/include/sys/unistd.h:184` および `/usr/include/sys/pthread.h:440` に衝突するプロトタイプがあります。

一つの解決策としては `/usr/include/sys/unistd.h` を `mysql/include` にコピーして `unistd.h` を編集しそれを `pthread.h` の定義に一致させます。以下の行を探します。

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
```

```
void (*child)());
```

それを以下のように変更します。

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                          void (*child)(void));
```

変更すると以下の `configure` 行が作用します。

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

HP-UX コンパイラを使用している場合、以下のコマンド (`cc B.11.11.04` でテスト済み) を使用できます。

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
--with-extra-character-set=complex
```

以下のようなエラーは無視できます。

```
aCC: warning 901: unknown option: `-'3': use +help for online
documentation
```

`configure` で以下のようなエラーが表示された場合、HP-UX C および C++ コンパイラへのパスの前に K&R のパスがないことを確認します。

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

コンパイルできない別の原因としては説明通りに `+DD64` フラグを定義しなかったことが考えられます。

HP-UX 11 の別の可能性としては弊社でビルドしてテストした <http://dev.mysql.com/downloads/> で提供している MySQL バイナリを使用することです。MySQL により提供された HP-UX 10.20 バイナリは HP-UX 11 上で問題なく動作するとの報告が入ってきています。問題が発生した場合には HP-UX のパッチレベルをチェックする必要があります。

IBM-AIX に関する注釈

`xlC` の自動検知が `Autoconf` で不明のため、`configure` を実行する前に多くの変数を設定する必要があります。以下の例では IBM コンパイラを使用しています。

```
export CC="xlC_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CXX="xlC_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS

./configure --prefix=/usr/local \
--localstatedir=/var/mysql \
--sbindir=/usr/local/bin \
--libexecdir=/usr/local/bin \
--enable-thread-safe-client \
--enable-large-files
```

前述のオプションを使用して <http://www.frec.bull.com/> にある MySQL ディストリビューションをコンパイルします。

前述の `configure` 行の `-O3` を `-O2` に変更すると、`-qstrict` オプションも削除できます。これが IBM C コンパイラの制限です。

MySQL のコンパイルに `gcc` あるいは `egcs` を使用する場合、`-fno-exceptions` フラグを使用する必要があります。なぜなら `gcc/egcs` の例外処理はスレッドセーフではないからです!(これは `egcs 1.1` でテスト済み。)IBM のコンパイラには `gcc` を使用すると間違ったコードを生成する未詳の問題があります。

以下の `configure` 行を AIX の `egcs` および `gcc 2.95` にお勧めします。

```
CC="gcc -pipe -mcpu=power -Wa,-many" \
CXX="gcc -pipe -mcpu=power -Wa,-many" \
```

```
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory
```

コンパイルを成功させるには `-Wa,-many` オプションが必要です。IBM はこの問題を認識していますが、回避策があるためにこの問題の修正にはそれほど急いでいません。MySQL は例外を使用しておらず、オプションが高速のコードを生成し、それを常に `egcs` / `gcc` と一緒に使用するよう勧めているため、`-fno-exceptions` が `gcc` 2.95 に必要かは定かではありません。

アセンブラのコードで問題が発生した場合には、`-mcpu=xxx` オプションを CPU に合うように変更してみてください。一般的には `power2`、`power`、あるいは `powerpc` が必要です。`.604` あるいは `604e` が必要になる場合もあります。確かではありませんが `power` はほとんどの時間、`power2` マシン上でも安全だと思われます。

どの CPU を使用しているか分からない場合には、`uname -m` コマンドを実行します。それによって `000514676700` のような文字列を生成し、フォーマットは `xyyyyyymmss` で `xx` および `ss` は常に `00` で、`yyyyyy` は一意のシステム ID で `mm` は CPU Planar の ID です。これらの値のチャートは http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm にあります。

これによりどんな CPU を使用しているかを決定するマシンの種類とマシンのモデルが分かります。

信号の問題 (MySQL が高負荷で突然停止する) がある場合、スレッドおよび信号の OS バグがある場合があります。この場合、以下のように設定して MySQL が信号を使用しないようにします。

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
-DDONT_USE_THR_ALARM" \
./configure --prefix=/usr/local/mysql --with-debug \
--with-low-memory
```

これによって MySQL の性能には影響ありませんが、`mysqladmin kill` あるいは `mysqladmin shutdown` と接続して「sleeping」しているクライアントを切断できなくなる副作用がでます。その代わりに、クライアントはそれが次のコマンドを発行した時に切断します。

AIX バージョンの中には、`libbind.a` とリンクすることによって `getservbyname()` ダンプ コアを作成します。これは AIX のバグですので IBM への報告する必要があります。

For AIX 4.2.1 および `gcc` で、以下の変更が必要です。

設定後、`config.h` および `include/my_config.h` を編集して行を以下のように変更します。

```
#define HAVE_SNPRINTF 1
```

に変更します。

```
#undef HAVE_SNPRINTF
```

最後に、`mysql.cc` で、`initgroups()` にプロトタイプを追加する必要があります。

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

`mysqld` プロセスに大きなメモリを割り当てる必要がある場合、`ulimit -d unlimited` を使用するだけでは十分ではありません。`mysqld_safe` も変更して以下のような行を追加する必要があります。

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

大きなメモリの使用に関する詳細は http://publib16.boulder.ibm.com/pseries/en_US/aixprgpd/genprogclrg_prg_support.htm にあります。

AIX 4.3 のユーザーは AIX にある `make` ユーティリティの代わりに `gmake` を使用する必要があります。

AIX 4.1 では、C コンパイラは個別の製品として AIX とはバンドルしていません。ここで入手できる `gcc` 3.3.2 を弊社としては推奨しています。 <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/gcc/>

AIX 上の `gcc` 3.3.2 で MySQL をコンパイルするステップは `gcc` 2.95 (特に `config.h` および `my_config.h` を `configure` 実行後に編集する必要性) を使用したステップに似ています。しかし、`configure` を実行する前に、`curses.h` ファイルを以下のようにパッチする必要があります。


```

/opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h.ORIG
Mon Dec 26 02:17:28 2005
--- /opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h
Mon Dec 26 02:40:13 2005
*****
*** 2023,2029 ****

#endif /* _AIX32_CURSES */
! #if defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus) || defined
(__STRICT_ANSI__)
extern int delwin (WINDOW *);
extern int endwin (void);
extern int getcurx (WINDOW *);
--- 2023,2029 ----

#endif /* _AIX32_CURSES */
! #if 0 && (defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus)
|| defined
(__STRICT_ANSI__))
extern int delwin (WINDOW *);
extern int endwin (void);
extern int getcurx (WINDOW *);

```

SunOS 4 に関する注釈

SunOS 4 では、MySQL をコンパイルするには MIT-pthreads が必要です。これはつまり GNU [make](#) が必要だということ意味します。

SunOS 4 システムの中には動的ライブラリと [libtool](#) に問題がありものがあります。この問題を回避するには以下の [configure](#) 行を使用します。

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

[readline](#) をコンパイルする際、二重の定義で警告が出る場合があります。これらは無視して構いません。

[mysqld](#) をコンパイルする際、[implicit declaration of function](#) 警告がでます。これらは無視して構いません。

Alpha-DEC-UNIX に関する注釈 (Tru64)

Digital Unix で [egcs](#) 1.1.2 を使用する際、DEC 上の [egcs](#) に幾つかの重大なバグがあるので [gcc](#) 2.95.2 にアップグレードする必要があります。

Digital Unix でスレッド化したプログラムをコンパイルするには、説明資料では [-pthread](#) オプションを [cc](#) および [cxx](#) および [-lmach -lexc](#) ライブラリ ([-lpthread](#) に加えて) 使用することを勧めています。[configure](#) を以下のように実行します。

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

[mysqld](#) をコンパイルする際、以下のような警告が表示される場合があります。

```

mysqld.cc: In function void handle_connections():
mysqld.cc:626: passing long unsigned int ** as argument 3 of
accept(int,sockaddr *, int *)

```

これらの警告は無視しても問題ありません。これは [configure](#) が警告ではなくエラーだけを検知するために起こります。

コマンドラインからサーバを直接起動すると、ログアウトしたときにサーバが停止する場合があります。(ログアウトすると、未処理のプロセスが [SIGHUP](#) 信号を受信します。)その場合、サーバを以下のように起動してみます。

```
nohup mysql [options] &
```

[nohup](#) はそれに続くコマンドに端末から送信された [SIGHUP](#) 信号を無視させます。その代わりに、[mysqld_safe](#) を実行してサーバを起動し、それが [nohup](#) を使用して [mysql](#) を呼び出します。[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) 参照。

`mysys/get_opt.c` をコンパイル中に問題が発生したら、`#define _NO_PROTO` 行をそのファイルに起動から削除します。

Compaq の CC コンパイラを使用している場合、以下の `configure` 行が作用します。

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
-speculate all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
-speculate all -arch host -noexceptions -nortti"
export CC CFLAGS CXX CXXFLAGS
./configure \
--prefix=/usr/local/mysql \
--with-low-memory \
--enable-large-files \
--enable-shared=yes \
--with-named-thread-libs="-lpthread -lmach -lexc -lc"
gnumake
```

以下に示すように共有ライブラリでコンパイル中に `libtool` で問題があった場合、`mysql` のリンク中に、以下のコマンドを発行してこの問題を回避します。

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread -O3 -DDEBUG_OFF \
-O4 -ansi_alias -ansi_args -fast -inline speed \
-speculate all \ -arch host -DUNDEF_HAVE_GETHOSTBYNAME_R \
-o mysql mysql.o readline.o sql_string.o completion_hash.o \
../readline/libreadline.a -lcurses \
../libmysql/.libs/libmysqlclient.so -lm
cd ..
gnumake
gnumake install
scripts/mysql_install_db
```

Alpha-DEC-OSF/1 に関する注釈

コンパイル中の問題が発生し DEC `CC` and `gcc` をインストールしている場合、`configure` を以下のように実行してみます。

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

`c_asm.h` ファイルに問題がある場合、ダミーの `c_asm.h` ファイルを作成してそれを使用します。

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

以下の `ld` プログラムの問題は最新の DEC (Compaq) パッチ キットを次からダウンロードすることで修正できます。
<http://ftp.support.compaq.com/public/unix/>.

OSF/1 V4.0D およびコンパイラ「DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)」ではそのコンパイラはいくつかの予想外の振る舞い (未定義の `asm` 記号) をしました。`/bin/ld` も破損すると思われます (`mysqld` のリンク中の `_exit undefined` エラーの問題)。このシステムでは、MySQL を以下の `configure` 行で `/bin/ld` を OSF 4.0C バージョン置き換えた後に何とかコンパイルできました。

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

With the Digital compiler "C++ V6.1-029," the following should work:

```
CC=cc -pthread
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
-speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
-speculate all -arch host -noexceptions -nortti
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql \
```

```
--with-mysqld-ldflags=-all-static --disable-shared \
--with-named-thread-libs="-lmach -lexc -lc"
```

OSF/1 のバージョンのいくつかでは、`alloca()` 関数が壊れます。この問題は 'HAVE_ALLOCA' を定義する `config.h` の行を削除して修正します。

`alloca()` 関数も不正なプロトタイプを `/usr/include/alloca.h` に持つ場合があります。これに由来するこの警告は無視できます。

`configure` は以下のスレッド ライブラリを自動的に使用します。--with-named-thread-libs="-lpthread -lmach -lexc -lc".

`gcc` を使用する際、`configure` を以下のように実行してみます。

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

信号の問題 (MySQL が高負荷で突然停止する) がある場合、スレッドおよび信号の OS バグがある場合があります。この場合、以下のように設定して MySQL が信号を使用しないようにします。

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

これによって MySQL の性能には影響ありませんが、`mysqladmin kill` あるいは `mysqladmin shutdown` と接続して「sleeping」しているクライアントを切断できなくなる副作用が出ます。その代わりに、クライアントはそれが次のコマンドを発行した時に切断します。

`gcc` 2.95.2 では、以下のコンパイラ エラーが発生する場合があります。

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

この問題を修正するには、`sql` ディレクトリに変更して `gcc` の最後の行を切り取り/貼り付けし、`-O3` を `-O0` (あるいは `-O` オプションがコンパイル行にない場合は `-O0` を `gcc` の直ぐ後に追加する) に変更します。この変更を行った後、ディレクトリの上段に戻り `make` を再度実行します。

SGI Irix に関する注釈

MySQL 5.0 では、Irix にはもはやバイナリを提供していません。

Irix 6.5.3 あるいはそれ以降を使用している場合、`mysqld` はそれを `CAP_SCHED_MGT` 権限 (`root` など) を有すユーザーとして実行した場合あるいは `mysqld` サーバにこの権限を以下のシェルのコマンドを使用して与えた場合にのみスレッドを作成できます。

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

`config.h` のいくつかの記号の定義を `configure` を実行した後のコンパイルする前に外す必要がある場合があります。

Irix 実装の中には、`alloca()` 関数が破損しているものもあります。`mysqld` サーバが `SELECT` ステートメントを実行中に停止した場合、`HAVE_ALLOC` および `HAVE_ALLOCA_H` を定義した `config.h` から行を削除します。`mysqladmin create` が機能しない場合、`HAVE_READDIR_R` を定義した `config.h` から行を削除します。`HAVE_TERM_H` 行を同様に削除する必要があるかもしれません。

SGI では以下のページのすべてのパッチをこの段階でまとめてインストールすることをお勧めしています。 http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

最低限の条件として、最新の kernel ロールアップ、最新の `rd` ロールアップ、および最新の `libc` ロールアップをインストールする必要があります。

スレッドをサポートするには以下のページのすべての POSIX パッチのインストールが必須です。

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

`mysql.cc` をコンパイル中に以下のようなエラーが表示された場合:

```
"/usr/include/curses.h", line 82: error(1084):
invalid combination of type
```

MySQL ソース ツリーの最上段のディレクトリに以下を入力します。

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

プログラムのスケジュールに関する問題も報告されています。1 つのスレッドのみが動作している場合、パフォーマンスは低くなります。別のクライアントを起動してこの問題を回避します。他のスレッドを使用することによって実行速度が 2 倍から 10 倍ほど向上します。これは Irix のスレッドの問題についての認識の甘さによるものです。この問題が修復されるまでは別の方法を考える必要があります。

gcc でコンパイルする際、以下の `configure` コマンドを使用します。

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
--with-named-thread-libs=-lpthread
```

Irix 6.5.11 のネイティブ Irix C および C++ コンパイラ バージョン 7.3.1.2 では、以下の動作が報告されています。

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARG:platform=IP22 -l/usr/local/include \
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARG:platform=IP22 \
-l/usr/local/include -L/usr/local/lib' \
./configure --prefix=/usr/local/mysql --with-innodb \
--with-libwrap=/usr/local \
--with-named-curses-libs=/usr/local/lib/libncurses.a
```

SCO UNIX および OpenServer 5.0.x に関する注釈

現在のポートは [sco3.2v5.0.5](#)、[sco3.2v5.0.6](#)、および [sco3.2v5.0.7](#) のシステムのみでテストされたものです。[sco3.2v4.2](#) へのポートのテストも進行中です。Open Server 5.0.8 (Legend) はネイティブのスレッドで 2GB 以上のファイルが可能です。現在の最大のファイル サイズは 2GB です。

MySQL を以下の `configure` コマンドで gcc 2.95.3 の OpenServer でコンパイルできます。

```
CC=gcc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=gcc CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
--enable-thread-safe-client --with-innodb \
--with-openssl --with-vio --with-extra-charsets=complex
```

gcc は <ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj> にあります。

この開発システムは OpenServer 5.0.6 に関する OpenServer 実行環境補足 oss646B が必要です。oss656B および OpenSource ライブラリは gwxlibs にあります。すべての OpenSource ツールは [opensrc](#) ディレクトリにあります。それらは <ftp://ftp.sco.com/pub/openserver5/opensrc/> で入手できます。

MySQL の最新の量産リリースの使用をお勧めしています。

SCO ではオペレーティング システムのパッチを <ftp://ftp.sco.com/pub/openserver5> で OpenServer 5.0.[0-6] 用、<ftp://ftp.sco.com/pub/openserver5/507> で OpenServer 5.0.7 を提供しています。

SCO では OpenServer 5.0 のセキュリティの修正版に関する情報を <ftp://ftp.sco.com/pub/security/OpenServer> で提供しています。

OpenSever 5.0.x システムの最大ファイル サイズは 2GB です。

OpenServer 5.0.x ではストリーム バッファ、clists、および ロック レコードに割り当てられる総メモリは 60MB を超えることはできません。

ストリーム バッファは 4096 バイト ページ、clists は各 70 バイト、およびロック レコードは各 64 バイトなどの単位で割り当てられます。

```
(NSTRPAGES * 4096) + (NCLIST * 70) + (MAX_FLCKREC * 64) <= 62914560
```

データベースのサービス オプションを設定するには以下の手順に従います。アプリケーションが以下が必要かどうか分からない場合には、アプリケーションで提供された説明資料を参照してください。

1. `root` としてログインします。

2. `/etc/conf/sdevice.d/suds` ファイルを編集して SUDS ドライバを有効にします。2 番目のファイルの **N** を **Y** に変更します。
3. `mkdev aio` あるいは `Hardware/Kernel` マネージャを使用して非同期 I/O のサポートを有効にして kernel を再度リンクします。ユーザーがこの種の I/O にメモリを固定するには、`aiomemlock(F)` ファイルを更新します。このファイルを更新して AIO および固定する最大容量のメモリを使用するユーザー名を含めます。
4. 多くのアプリケーションは単独のユーザーとして指定できるように `setuid` バイナリを使用しています。アプリケーションに付属の説明資料を参照して使用するアプリケーションのこのようになっているか確認します。

このプロセスを完了したら、システムをリブートしてこれらの変更を加えた新しい kernel を作成します。

デフォルトでは、`/etc/conf/cf.d/mtune` へのエントリは以下のように設定されます。

Value	Default	Min	Max
NBUF	0	24	450000
NHBUF	0	32	524288
NMPBUF	0	12	512
MAX_INODE	0	100	64000
MAX_FILE	0	100	64000
CTBUFSIZE	128	0	256
MAX_PROC	0	50	16000
MAX_REGION	0	500	160000
NCLIST	170	120	16640
MAXUP	100	15	16000
NOFILES	110	60	11000
NHINODE	128	64	8192
NAUTOUP	10	0	60
NGROUPS	8	0	128
BDFLUSHR	30	1	300
MAX_FLCKREC	0	50	16000
PUTBUFSZ	8000	2000	20000
MAXSLICE	100	25	100
ULIMIT	4194303	2048	4194303
* Streams Parameters			
NSTREAM	64	1	32768
NSTRPUSH	9	9	9
NMUXLINK	192	1	4096
STRMSGSZ	16384	4096	524288
STRCTLSZ	1024	1024	1024
STRMAXBLK	524288	4096	524288
NSTRPAGES	500	0	8000
STRSPPLITFRAC	80	50	100
NLOG	3	3	3
NUMSP	64	1	256
NUMTIM	16	1	8192
NUMTRW	16	1	8192
* Semaphore Parameters			
SEMAP	10	10	8192
SEMMNI	10	10	8192
SEMMNS	60	60	8192
SEMMNU	30	10	8192
SEMMSL	25	25	150
SEMOPM	10	10	1024
SEMUME	10	10	25
SEVMVMX	32767	32767	32767
SEMAEM	16384	16384	16384
* Shared Memory Parameters			
SHMMAX	524288	131072	2147483647
SHMMIN	1	1	1
SHMMNI	100	100	2000
FILE	0	100	64000
NMOUNT	0	4	256
NPROC	0	50	16000
NREGION	0	500	160000

これらの値を以下のように設定するようお勧めします。

- **NOFILES** は 4096 あるいは 2048 にします。
- **MAXUP** は 2048 にします。

kernel に変更を加えるには `idune name parameter` コマンドを使用します。idune はお客様に代わって `/etc/conf/cf.d/stune` ファイルを変更します。例えば、SEMMS を 200 に変更するには、このコマンドを root として実行します。

```
# /etc/conf/bin/idune SEMMNS 200
```

次に以下のコマンドを発行して kernel を再度ビルドしてリブートします。

```
# /etc/conf/bin/idbuild -B && init 6
```

弊社ではシステムのチューニングを推奨していますが、使用する適切なパラメータ値はアプリケーションあるいはデータベースにアクセスするユーザー数およびデータベースのサイズ (つまり、使用済みバッファプール) に基づきます。以下の kernel パラメータは idune で設定できます。

- **SHMMAX** (推奨設定値:128MB) および **SHMSEG** (推奨設定値:15). これらのパラメータはユーザーのバッファプールを作成する MySQL のデータベース エンジンに影響を及ぼします。
- **NOFILES** および **MAXUP** は最低でも 2048 に設定します。
- **MAXPROC** は最低でも 3000/4000 (ユーザー数による) あるいはそれ以上に設定します。
- 弊社ではまた **SEMMSL**、**SEMMNS**、および **SEMMNU** の値の計算に以下の式の使用を推奨しています。

```
SEMMSL = 13
```

13 は Progress および MySQL の両方に最適な値であることが分かっています。

```
SEMMNS = SEMMSL × number of db servers to be run on the system
```

SEMMNS を一度にシステムで使用するデータベース サーバの倍数 (最大) の **SEMMSL** の値に設定します。

```
SEMMNU = SEMMNS
```

SEMMNU の値を設定して **SEMMNS** の値と同じにします。この値を **SEMMNS** の 75% に設定できますが、これは控えめな値です。

gcc を使用するには少なくとも SCO OpenServer Linker およびアプリケーション開発ライブラリあるいは OpenServer 開発システムをインストールする必要があります。これらのうちどれかをインストールしなければ GCC Dev システムは使用できません。

FSU Pthreads パッケージを取得して最初にそれをインストールする必要があります。これは <http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz> にあります。コンパイル済みパッケージは <ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz> で取得できます。

FSU Pthreads は SCO Unix 4.2 の tcpip、あるいは SCO 開発システムをインストールし GCC 2.5.x のグッドポートを使用した OpenServer 3.0 または Open Desktop 3.0 (OS 3.0 ODT 3.0) 使用してコンパイルできます。ODT あるいは OS 3.0 には、GCC 2.5.x のグッドポートが必要です。グッドポートがなければ多くの問題が発生します。この製品のポートには SCO Unix 開発システムが必要です。それがなければ、ライブラリおよび必要なリンクがみつかりません。また [SCO-3.2v4.2-includes.tar.gz](ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz) も必要です。このファイルは MySQL のビルドに必要な SCO 開発 include files への変更を含んでいます。既存のシステム include files をこれらの変更ヘッダーファイルで置き換える必要があります。それらは <ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz> から入手できます。

お客様のシステム上で FSU Pthreads にビルドに必要なことは、GNU make を実行することだけです。FSU-threads-3.14.tar.gz の Makefile を設定して FSU-threads を作成します。

`./configure` を `threads/src` ディレクトリで実行して SCO OpenServer オプションを選択できます。このコマンドは `Makefile.SCO5` を `Makefile` にコピーします。次に `make` を実行します。

デフォルトの `/usr/include` ディレクトリにインストールするには、root として、次に `cd` から `thread/src` ディレクトリにログインして `make install` を実行します。

MySQL をビルドするには GNU make を使用することを忘れないでください。

注:mysqlid_safe を root として実行しなかった場合、デフォルトでプロセスごとに 110 オープン ファイルしか取得できません。mysqlid はこのことをログ ファイルに記録します。

SCO 3.2V4.2 には、FSU Pthreads バージョン 3.14 あるいはそれ以降を使用します。以下の `configure` コマンドが作ります。

```
CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \
./configure \
--prefix=/usr/local/mysql \
--with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \
--with-named-curses-libs="-lcurses"
```

include files の中には問題が発生するものもあります。この場合、新しい SCO 専用の include files を <ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz> から入手できます。

このファイルを MySQL ソース ツリーの `include` ディレクトリで解凍します。

SCO の開発ノート：

- MySQL は自動的に FSU Pthreads を検知して `mysqld` を `-lgthreads -lsocket -lgthreads` にリンクします。
- SCO 開発ライブラリは FSU Pthreads に再度追加されたものです。SCO によれば、そのライブラリ関数は新たに追加されたもので、FSU Pthreads に追加する必要があるとのことです。OpenServer の FSU Pthreads は SCO スキーマを使用して再追加のライブラリを作成します。
- FSU Pthreads (少なくとも <ftp://ftp.zenez.com> の「バージョン」) には GNU `malloc` が付いています。メモリの使用で問題がある場合には、`gmalloc.o` が `libgthreads.a` および `libgthreads.so` に含まれていることを確認します。
- FSU Pthreads では、次のシステム コールは pthreads-アウェア (認識) です。 `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()`, および `wait()`。
- CSSA-2001-SCO.35.2 (そのパッチはカスタムで `erg711905-dscr_remap` セキュリティ パッチ (バージョン 2.0.0 としてリストされています)) は FSU threads を壊して `mysqld` を不安定にします。`mysqld` を OpenServer 5.0.6 マシンで動作させる場合にはこれを削除する必要があります。
- SCO OpenServer 5 を使用する場合、FSU pthreads を `CFLAGS` の `-DDRAFT7` で再コンパイルする必要があります。さもないと、`InnoDB` は `mysqld` の起動時にハングする場合があります。
- SCO は OpenServer 5.0.x 用のオペレーティング システムのパッチを <ftp://ftp.sco.com/pub/openserver5> で提供しています。
- SCO は OpenServer 5.0.x 用のセキュリティの修正版および `libsocket.so.2` を <ftp://ftp.sco.com/pub/security/OpenServer> および <ftp://ftp.sco.com/pub/security/sse> で提供しています。
- Pre-OSR506 のセキュリティ修正版また、`telnetd` の修正版は <ftp://stage.caldera.com/pub/security/openserver/> あるいは <ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/> その両方 `libsocket.so.2` および `libresolv.so.1` にはその pre-OSR506 システムのインストールの説明書が付いています。

MySQL をコンパイル/使用する前にこれらのパッチをインストールするようお勧めします。

Legend/OpenServer 6.0.0 以降には、ネイティブ スレッドがあり、2GB ファイル サイズの制限はありません。

SCO OpenServer 6.0.x に関する注釈

OpenServer 6 には以下の主な改善が施されています。

- 1 TB までの大きなファイル サポート
- プロセッサのサポートが 4 個から 32 個まで増えたマルチ プロセッサ サポート
- 64GB まで拡張したメモリ サポート
- UnixWare のパワーの OpenServer 6 への拡張
- 大幅なパフォーマンスの改善

OpenServer 6.0.0 のコマンドは以下のような構成になります。

- `/bin` は OpenServer 5.0.x とまったく同じ振る舞いをするコマンド用です。

- [/u95/bin](#) は、例えば大きなファイル システム (LFS) サポートなど標準適合に適したコマンド用です。
- [/udk/bin](#) は UnixWare 7.1.4 上と同様に振る舞いをするコマンド用です。デフォルトは LFS サポートです。

以下は [PATH](#) を OpenServer 6 上で設定するガイドです。ユーザーが従来の OpenServer 5.0.x を希望する場合には [PATH](#) は [/bin](#) が最初になります。ユーザーが LFS サポートを希望する場合、そのパスは [/u95/bin:/bin](#) になります。ユーザーが UnixWare 7 サポートを最初に希望する場合、そのパスは [/udk/bin:/u95/bin:/bin](#) になります。

MySQL の最新の量産リリースの使用をお勧めしています。旧リリースの MySQL を OpenServer 6.0.x に使用される場合、MySQL のバージョンは少なくとも 3.22.13 に近いものでポータビリティおよび OS の問題に対する修正を含んだものを使用する必要があります。

MySQL ディストリビューション ファイルで以下のフォーム名を持つものはメディア画像の [tar](#) アーカイブで SCO OpenServer 上での SCO Software Manager ([/etc/custom](#)) との使用に適しています。

```
mysql-PRODUCT-5.1.15-beta-sco-osr6-i686.VOLS.tar
```

ディストリビューションで [PRODUCT](#) が [pro-cert](#) のものは商業的にライセンス許諾した MySQL プロ認定サーバです。ディストリビューションで [PRODUCT](#) が [pro-gpl-cert](#) のものは MySQL プロ認定サーバで一般利用許諾書 (GPL) の条件に基づいてライセンスを許諾したものです。

インストールする任意のディストリビューションを選択し、ダウンロードしたら、[tar](#) アーカイブを空のディレクトリに取り出します。例えば、

```
shell> mkdir /tmp/mysql-pro
shell> cd /tmp/mysql-pro
shell> tar xf /tmp/mysql-pro-cert-5.1.15-beta-sco-osr6-i686.VOLS.tar
```

インストールの前に、[MySQL のアップグレード](#) の手順に従ってデータのバックアップを取ります。

以前にインストールした MySQL の [pkgadd](#) バージョンを削除します。

```
shell> pkginfo mysql 2>&1 > /dev/null && pkgrm mysql
```

SCO Software Manager を使用して MySQL Pro をメディア画像からインストールします。

```
shell> /etc/custom -p SCO:MySQL -i -z /tmp/mysql-pro
```

SCO Software Manager のデスクトップの [Software Manager](#) アイコンをクリックしてグラフィック表示し、[Software -> Install New](#) を選択し、ホストを選択し、[Media Images](#) をメディア デバイスに選択し、そして [/tmp/mysql-pro](#) をイメージ ディレクトリとして入力します。

インストールしたら、[mkdev mysql](#) を [root](#) ユーザーとして起動し新たにインストールした MySQL プロ認定サーバを設定します。

注:VOLS パッケージのインストール手順ではパッケージがデフォルトとして使用する [mysql](#) のユーザーおよびグループは作成しません。[mysql](#) のユーザーおよびグループを作成するか、または [mkdev mysql](#) のオプションを使用して別のユーザーまたはグループを選択します。

MySQL Pro server を Apache Web サーバに PHP でインターフェースする設定を行うには、PHP の更新版を SCO の [ftp://ftp.sco.com/pub/updates/OpenServer/SCOSA-2006.17/](#) からダウンロードしてインストールします。

MySQL を以下の [configure](#) コマンドで OpenServer 6.0.x 上でコンパイルできます。

```
CC=cc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=CC CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
--enable-thread-safe-client \
--with-extra-charsets=complex \
--build=i686-unknown-sysv5SCO_SV6.0.0
```

[gcc](#) を使用する場合には、[gcc](#) 2.95.3 あるいはそれ以降を使用する必要があります。

```
CC=gcc CXX=g++ ... ./configure ...
```

SCO では OpenServer 6 のオペレーティング システムのパッチを [ftp://ftp.sco.com/pub/openserver6](#) で提供しています。

SCO はセキュリティの修正版の情報を [ftp://ftp.sco.com/pub/security/OpenServer](http://ftp.sco.com/pub/security/OpenServer) で提供しています。

デフォルトでは OpenServer 6.0.0 システムの最大ファイル サイズは 1TB です。オペレーティング システムのユーティリティの中には 2GB の制限のものもあります。UnixWare 7 の最大可能ファイル サイズは VXFS あるいは HTFS で 1TB です。

OpenServer 6 は UNIX kernel をチューニングして大きなファイル サポート用に設定できます (2GB 以上のファイル サイズ)。

デフォルトでは、`/etc/conf/cf.d/mtune` へのエントリは以下のように設定されます。

Value	Default	Min	Max
SVMLIM	0x9000000	0x1000000	0x7FFFFFFF
HVMLIM	0x9000000	0x1000000	0x7FFFFFFF

kernel に変更を加えるには `idune name parameter` コマンドを使用します。idune はお客様に代わって `/etc/conf/cf.d/stune` ファイルを変更します。kernel 値の設定は以下のコマンドを `root` として実行して設定するようお勧めします。

```
# /etc/conf/bin/idune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idune SVMLIM 0x7FFFFFFF
# /etc/conf/bin/idune HVMLIM 0x7FFFFFFF
# /etc/conf/bin/idune SFNOLIM 2048
# /etc/conf/bin/idune HFNOLIM 2048
```

次に以下のコマンドを発行して kernel を再度ビルドしてリブートします。

```
# /etc/conf/bin/ibuild -B && init 6
```

弊社ではシステムのチューニングを推奨していますが、使用する適切なパラメータ値はアプリケーションあるいはデータベースにアクセスするユーザー数およびデータベースのサイズ (つまり、使用済みバッファプール) に基づきます。以下の kernel パラメータは idune で設定できます。

- **SHMMAX** (推奨設定値:128MB) および **SHMSEG** (推奨設定値:15). これらのパラメータはユーザーのバッファプールを作成する MySQL のデータベース エンジンに影響を及ぼします。
- **SFNOLIM** および **HFNOLIM** は最大で 2048 にします。
- **NPROC** は最低でも 3000/4000 (ユーザー数による) に設定します。
- 弊社ではまた **SEMMSL**、**SEMMNS**、および **SEMMNU** の値の計算に以下の式の使用を推奨しています。

```
SEMSL = 13
```

13 は Progress および MySQL の両方に最適な値であることが分かっています。

```
SEMMNS = SEMMSL * number of db servers to be run on the system
```

SEMMNS を一度にシステムで使用するデータベース サーバの倍数 (最大) の **SEMSL** の値に設定します。

```
SEMMNU = SEMMNS
```

SEMMNU の値を設定して **SEMMNS** の値を同じにします。この値を **SEMMNS** の 75% に設定できますが、これは控えめな値です。

SCO UnixWare 7.1.x および OpenUNIX 8.0.0 に関する注釈

MySQL の最新の量産リリースの使用をお勧めしています。MySQL の旧リリースを UnixWare 7.1.x に使用する場合、最低でも MySQL の 3.22.13 に近いものを使用するとポータビリティおよび OS の問題の修正版を使用できます。

以下の `configure` コマンドを使用して UnixWare 7.1.x 上で MySQL をコンパイルできます。

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
```

```
./configure --prefix=/usr/local/mysql \
--enable-thread-safe-client \
--with-innodb --with-openssl --with-extra-charsets=complex
```

gcc を使用する場合には、gcc 2.95.3 あるいはそれ以上を使用する必要があります。

```
CC=gcc CXX=g++ ... ./configure ...
```

SCO ではオペレーティング システムのパッチを UnixWare 7.1.1 は <ftp://ftp.sco.com/pub/unixware7> で、UnixWare 7.1.3 は <ftp://ftp.sco.com/pub/unixware7/713/> で、UnixWare 7.1.4 は <ftp://ftp.sco.com/pub/unixware7/714/> で、OpenUNIX 8.0.0 は <ftp://ftp.sco.com/pub/openunix8> で提供しています。

SCO ではセキュリティの修正版を OpenUNIX は <ftp://ftp.sco.com/pub/security/OpenUNIX> で、UnixWare は <ftp://ftp.sco.com/pub/security/UnixWare> で提供しています。

UnixWare 7 のファイル サイズは VXFS で最大 1 TB です。オペレーティング システム ユーティリティの中には 2GB 制限にものもあります。

UnixWare 7.1.4 では大きなファイル サイズのサポートを取得するのに何もすることはありませんが、UnixWare 7.1.x 以前のバージョンで大きなファイルをサポートするには [fsadm](#) を実行する必要があります。.

```
# fsadm -Fvxfs -o largefiles /
# fsadm /      * Note
# ulimit unlimited
# /etc/conf/bin/ldtune SFSZLIM 0x7FFFFFFF ** Note
# /etc/conf/bin/ldtune HFSZLIM 0x7FFFFFFF ** Note
# /etc/conf/bin/ldbuild -B

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.
```

[shutdown](#) を使用してシステムをリブートします。

デフォルトでは、[/etc/conf/cf.d/mtune](#) へのエントリは以下のように設定されます。

Value	Default	Min	Max
SVMMLIM	0x9000000	0x1000000	0x7FFFFFFF
HVMMLIM	0x9000000	0x1000000	0x7FFFFFFF

kernel に変更を加えるには [ldtune name parameter](#) コマンドを使用します。ldtune はお客様に代わって [/etc/conf/cf.d/stune](#) ファイルを変更します。kernel 値の設定は以下のコマンドを [root](#) として実行して設定するようお勧めします。

```
# /etc/conf/bin/ldtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/ldtune SFNOLIM 2048
# /etc/conf/bin/ldtune HFNOLIM 2048
```

次に以下のコマンドを発行して kernel を再度ビルドしてリブートします。

```
# /etc/conf/bin/ldbuild -B && init 6
```

弊社ではシステムのチューニングを推奨していますが、使用する適切なパラメータ値はアプリケーションあるいはデータベースにアクセスするユーザー数およびデータベースのサイズ (つまり、使用済みバッファ プール) に基づきます。以下の kernel パラメータは [ldtune](#) で設定できます。

- [SHMMAX](#) (推奨設定値:128MB) および [SHMSEG](#) (推奨設定値:15). これらのパラメータはユーザーのバッファ プールを作成する MySQL のデータベース エンジンに影響を及ぼします。
- [SFNOLIM](#) および [HFNOLIM](#) は最大で 2048 にします。
- [NPROC](#) は最低でも 3000/4000 (ユーザー数による) に設定します。
- 弊社ではまた [SEMMSL](#)、[SEMMNS](#)、および [SEMMNU](#) の値の計算に以下の式の使用を推奨しています。

```
SEMMSL = 13
```


13 は Progress および MySQL の両方に最適な値であることが分かっています。

```
SEMMNS = SEMMSL × number of db servers to be run on the system
```

SEMMNS を一度にシステムで使用するデータベース サーバの倍数（最大）の **SEMMSL** の値に設定します。

```
SEMMNU = SEMMNS
```

SEMMNU の値を設定して **SEMMNS** の値と同じにします。この値を **SEMMNS** の 75% に設定できますが、これは控え目な値です。

OS/2 に関する注釈

MySQL は極少数のオープン ファイルを使用しています。このため、以下のようなものを **CONFIG.SYS** ファイルに追加する必要があります。

```
SET EMXOPT=-c -n -h1024
```

これを追加しなかった場合、以下の問題が発生する場合があります。

```
File 'xxxx' not found (Errcode: 24)
```

MySQL を OS/2 Warp 3 で使用するには、FixPack 29 あるいはそれ以降が必要です。OS/2 Warp 4 では、FixPack 4 あるいはそれ以降が必要です。これは Pthreads ライブラリに必要です。MySQL は HPFS、FAT32 などの長いファイル名をサポートしたパーテーションにインストールする必要があります。

INSTALL.CMD スクリプトは OS/2 の **CMD.EXE** で実行する必要があり、**4OS2.EXE** などの代わりのシェルでは動作しません。

scripts/mysql-install-db スクリプトの名前が変わっています。それは **install.cmd** と呼ばれ REXX スクリプトで、デフォルトの MySQL のセキュリティを設定し、WorkPlace Shell のアイコンを MySQL に作成します。

動的モジュールのサポートはコンパイルしましたがまだ完全にテストしていません。動的モジュールは Pthreads のランタイム ライブラリを使用してコンパイルする必要があります。

```
gcc -Zdll -Zmt -Zcrtldl=pthrdrtl -I../include -I../regex -I.. \
-o example udf_example.c -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

注:OS/2 の制限により、UDF のモジュール名は 8 文字を超えることはできません。モジュールは **/mysql2/udf** ディレクトリに保持され、**safe-mysqld.cmd** スクリプトがこのディレクトリを **BEGINLIBPATH** 環境変数に加えます。UDF モジュールを使用する場合、指定された拡張は無視されます **-.udf** と想定されます。例えば、Unix では、共有モジュールは **example.so** と命名され、関数をそれから以下の様にロードします。

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';
```

OS/2 では、モジュールは **example.udf** と命名できますが、モジュール拡張は指定できません。

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

環境変数

この項では直接的あるいは間接的に MySQL で使用されるすべての環境変数について説明します。これらの多くは本マニュアルの別の場所にもあります。

コマンドラインのいかなるオプションもオプション ファイルまたは環境変数で指定された値に優先し、オプション ファイルの値は環境変数の値に優先することにご留意ください。

多くの場合、オプション ファイルを環境変数の代わりに使用して MySQL の振る舞いを変更するほうが好ましいといえます。[オプションファイルの使用](#) 参照。

変数	説明
CXX	C++ コンパイラ名 (configure 実行用)。

CC	C コンパイラ名 (configure 実行用)。
CFLAGS	C コンパイラ用フラグ (configure 実行用)。
CXXFLAGS	C コンパイラ用フラグ (configure 実行用)。
DBI_USER	Perl DBI のデフォルトユーザー名。
DBI_TRACE	Perl DBI のトレース オプション
HOME	mysql 履歴ファイルのデフォルトのパスは <code>\$HOME/.mysql_history</code> 。
LD_RUN_PATH	libmysqlclient.so のロケーション指定に使用。
MYSQL_DEBUG	デバッグ中のデバッグトレース オプション。
MYSQL_GROUP_SUFFIX	オプション グループの接尾辞の値 (<code>--defaults-group-suffix</code> などの指定)。
MYSQL_HISTFILE	mysql 履歴ファイルへのパス。この変数を設定すると、その値は <code>\$HOME/.mysql_history</code> のデフォルトをオーバーライドします。
MYSQL_HOME	サーバ特定の <code>my.cnf</code> ファイルが常駐するディレクトリへのパス (MySQL 5.0.3 として)。
MYSQL_HOST	mysql のコマンドライン クライアントが使用するデフォルトのホスト名
MYSQL_PS1	mysql コマンドライン クライアントで使用するコマンドプロンプト
MYSQL_PWD	mysql に接続する際のデフォルトのパスワード。これを使用することはセキュアされていません。 パスワードのセキュリティ 参照。
MYSQL_TCP_PORT	デフォルトの TCP/IP ポート番号。
MYSQL_UNIX_PORT	デフォルトの Unix ソケット ファイル名。 localhost への接続に使用。
PATH	シェルが MySQL プログラムの検索に使用する。
TMPDIR	一時ファイルが作成されるディレクトリ。
TZ	ローカル タイム ゾーンに設定する必要があります。 Time Zone Problems 参照。
UMASK_DIR	ディレクトリを作成する際のユーザーディレクトリ作成マスク。これは <code>UMASK</code> で <code>AND</code> されます。
UMASK	ファイルを作成する際のユーザーファイル作成マスク。
USER	mysql に接続する際に使用される Windows および NetWare 上のデフォルトのユーザー名

Perl のインストールに関する注釈

MySQL の Perl のサポートは [DBI/DBD](#) クライアント インターフェースによって提供されます。そのインターフェースには Perl 5.6.1 あるいはそれ以降が必要です。旧バージョンの Perl の場合にはそれは機能しません。

トランザクションを Perl DBI で使用する場合、`DBD::mysql` バージョン 1.2216 あるいはそれ以降が必要です。`DBD::mysql` 2.9003 あるいはそれ以降をお勧めします。

MySQL 4.1 あるいはそれ以降の新しいクライアント ライブラリを使用している場合、`DBD::mysql` 2.9003 あるいはそれ以降を使用する必要があります。

MySQL のディストリビューションには Perl のサポートは含まれていません。<http://search.cpan.org>、あるいは Windows の ActiveState [ppm](#) プログラムを使用して Unix に必要なモジュールを取得できます。以下の項でその取得の仕方について説明します。

MySQL のベンチマーク スクリプトを実行するには MySQL に Perl サポートをインストールする必要があります。[MySQL ベンチマークスイート](#) 参照。

Unix に Perl をインストールする

MySQL の Perl のサポートには MySQL のクライアント プログラム サポート (ライブラリおよびヘッダーファイル) をインストールする必要があります。ほとんどのインストール メソッドで必要なファイルをインストールできます。しかし、Linux 上で MySQL を RPM ファイルからインストールした場合、開発者 RPM をインストールしたことを確認

する必要があります。クライアント プログラムはクライアント RPM にありますが、クライアント プログラム サポートは開発者 RPM にあります。

Perl サポートをインストールする場合、必要なファイルは CPAN (包括的な Perl アーカイブ ネットワーク) から <http://search.cpan.org> で入手できます。

Unix に Perl モジュールをインストールするには **CPAN** モジュールを使用するのが一番簡単です。例えば、

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

DBD::mysql インストールには多くのテストを実行します。これらのテストはデフォルトのユーザー名およびパスワードを使用してローカルの MySQL サーバに接続を試みます。(Unix 上のデフォルトのユーザー名はお客様のログイン名で、Windows 上では **ODBC** になります。デフォルトのパスワードは「パスワードではありません。」) サーバにそれらの値 (例えば、アカウントにパスワードを設定している場合)、テストは失敗します。**force install DBD::mysql** を使用して失敗したテストを無視します。

DBI には **Data::Dumper** モジュールが必要です。それはインストールできますが、もしできなかった場合、**DBI** をインストールする前にそれをインストールします。

モジュールのディストリビューションを圧縮 **tar** アーカイブでダウンロードしてモジュールを手動で作成することもできます。例えば、DBI ディストリビューションを解凍してビルドするには、以下のような手順に従います。

1. ディストリビューションを現在のディレクトリに解凍します。

```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

このコマンドが **DBI-VERSION** 名のディレクトリを作成します。

2. ロケーションを解凍したディストリビューションの上段のディレクトリに変更します。

```
shell> cd DBI-VERSION
```

3. ディストリビューションをビルドしてすべてをコンパイルします。

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

make test コマンドはモジュールが動作していることを確認するために重要です。**DBD::mysql** のインストール中にそのコマンドを実行してインターフェースのコードを実行するには、MySQL サーバが動作していなければなりません。さもなければそのテストは失敗します。

MySQL の新しいリリースをインストールする時はいつでも **DBD::mysql** ディストリビューションを再ビルドして再インストールするのがいいでしょう。特にすべての **DBI** スクリプトが MySQL をアップグレードした後に失敗するような場合にはそれはいい考えです。

システムのディレクトリに Perl モジュールをインストールするためのアクセス権限がない場合、あるいはローカルの Perl モジュールをインストールする場合、以下の引用が役に立ちます。 <http://servers.digitaldaze.com/extensions/perl/modules.html#modules>

ヘッディング「ローカルでインストールしたモジュールが必要な新しいモジュールのインストール」の下を見ます。

Windows に ActiveState Perl をインストールする

Windows 上で、MySQL の **DBD** モジュールを ActiveState Perl でインストールするには以下の手順に従います。

1. <http://www.activestate.com/Products/ActivePerl/> から ActiveState Perl を入手してインストールします。
2. コンソール ウィンドウ (「DOS window」) を開きます。
3. 必要に応じて **HTTP_proxy** 変数を設定します。例えば、以下のような設定を試してみます。

```
set HTTP_proxy=my.proxy.com:3128
```

4. PPM プログラムの実行:

```
C:\> C:\perl\bin\ppm.pl
```

5. これまでインストールしたことがない場合、DBI をインストールします。

```
ppm> install DBI
```

6. これが完了したら、以下のコマンドを実行します。

```
install \
ftp://ftp.de.uu.net/pub/CPAN/authors/id/JWIED/DBD-mysql-1.2212.x86.ppd
```

このプロシージャは ActiveState Perl 5.6 あるいはそれ以降で機能します。

使用するプロシージャを取得できない場合には、MyODBC ドライバを代わりにインストールし、ODBC で MySQL サーバに接続します。

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
die "Got error $DBI::errstr when connecting to $dsn\n";
```

Perl DBI/DBD インターフェースを使用した際の問題

Perl が `../mysql/mysql.so` モジュールを見つけることができない場合、その問題は多分 Perl が `libmysqlclient.so` 共有ライブラリの場所が分からないためです。この問題は以下のメソッドの一つで解決できるはずです。

- `DBD::mysql` ディストリビューションを `perl Makefile.PL` の代わりに `perl Makefile.PL -static -config` でコンパイルします。
- `libmysqlclient.so` を他の共有ライブラリがある (多分 `/usr/lib` あるいは `/lib`) ディレクトリにコピーします。
- `DBD::mysql` のコンパイルに使用される `-L` オプションを変更して `libmysqlclient.so` の実際のロケーションに入れます。
- Linux 上で、`libmysqlclient.so` があるディレクトリのパス名を `/etc/ld.so.conf` ファイルに追加します。
- `libmysqlclient.so` があるディレクトリのパス名を `LD_RUN_PATH` 環境変数に追加します。システムの中には `LD_LIBRARY_PATH` を使用しているものもあります。

リンクが見つけられない他のライブラリがある場合 `-L` オプションを変更する必要がある場合があります。例えば、リンクが `libc` が `/lib` にあるため見つけられない場合でリンクのコマンドが `-L/usr/lib` を指定している場合、`-L` オプションを `-L/lib` に変更するかあるいは `-L/lib` を既存のリンクコマンドに追加します。

`DBD::mysql` から以下のエラーが表示された場合、多分 `gcc` (あるいは `gcc` でコンパイルされた旧バイナリを使用している) を使用していることになります。

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

`-L/usr/lib/gcc-lib/...-lgcc` をリンク コマンドに `mysql.so` ライブラリがビルドされた時に追加します (`make` の出力を `mysql.so` に対し Perl のクライアントをコンパイルした時にチェックします)。`-L` オプションはシステム上の `libgcc.a` があるディレクトリのパス名を指定します。

この問題の別の原因は Perl および MySQL が両方とも `gcc` でコンパイルされていない場合です。この場合、このミスマッチをその両方とも `gcc` でコンパイルすることで解決できます。

テストを実施した時に以下の `DBD::mysql` のエラーが表示される場合があります。

```
t/00base.....install_driver(mysql) failed:
Can't load '../lib/arch/auto/DBD/mysql/mysql.so' for module DBD::mysql:
../lib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

このことは `-lz` 圧縮ライブラリをリンク行に含める必要があることを意味しています。それは `lib/DBD/mysql/Install.pm` のファイルに以下の行を変更することでできます。

```
$sysliblist .= "-lm";
```

行を以下に変更します。

```
$sysliblist .= "-lm -lz";
```

変更した後に、`make realclean` を必ず実行して次にインストールはじめから実施します。

SCO に DBI をインストールするには DBI-xxx の `Makefile` および各サブディレクトリを編集する必要があります。以下は `gcc 2.95.2` あるいはそれ以降を想定したものです。

```
OLD:                NEW:
CC = cc              CC = gcc
CCCDLFLAGS = -KPIC -W1,-Bexport    CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport    CCDLFLAGS =

LD = ld              LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib    LDDLFLAGS = -L/usr/local/lib
LDLFLAGS = -belf -L/usr/local/lib    LDLFLAGS = -L/usr/local/lib

LD = ld              LD = gcc -G -fpic
OPTIMISE = -Od       OPTIMISE = -O1

OLD:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

NEW:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

これらの変更は Perl dynaloader が DBI モジュールをそれらが `icc` あるいは `cc` でコンパイルされた場合はロードしないために必要です。

動的リンク (SCO など) をサポートしていないシステムで Perl のモジュールを使用する場合、DBI および `DBD::mysql` を含む Perl の静的バージョンを生成できます。これを可能にするには Perl のバージョンをリンクした DBI コードで生成しそれを現在の Perl の先頭にインストールします。次にそれを使用して新たにリンクされた DBD コードを持つ Perl のバージョンをビルドし、それをインストールします。

SCO では、以下の環境変数が設定されている必要があります。

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

または

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
/usr/skunk/man:
```

最初に、DBI ディストリビューションがあるディレクトリのこれらのコマンド実行して静的にリンクした DBI モジュールを含む Perl を作成します。

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

次に新しい Perl をインストールします。`make perl` の出力がインストールの実施に必要な正確な `make` コマンドです。SCO では、これは `make -f Makefile.apperl inst_perl MAP_TARGET=perl` です。

次に、上記で作成した Perl を使用して `DBD::mysql` ディストリビューションがあるディレクトリのこれらのコマンドを使用して同様に静的にリンクした `DBD::mysql` を含む別の Perl を作成します。

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

最後に、この新しい Perl をインストールします。`make perl` の出力は上記同様使用するコマンドを表しています。

第3章 MySQL プログラムの使用

目次

MySQL プログラム の概要	139
MySQL プログラムの起動	140
プログラム・オプションの指定	141
コマンドラインにおけるオプションの使用	141
オプションファイルの使用	143
オプション指定のための環境変数の使用	147
プログラム変数セットのためのオプション使用	148

ここでは MySQL AB におけるコマンドラインの概要と、プログラムを起動させたときの具体的なオプション基本構造についての解説が述べられています。大半のプログラムにはそれぞれに呼応した特定のオペレーション システムが必要となりますが、オプション構造は全てにおいて似通っています。後半では個々のプログラムについて、どのオプションがサポートされるかという内容も含めたより詳しい解説がなされます。

MySQL AB はまた、3つの GUI のクライアントプログラムを MySQL サーバとして使用しています。

- MySQL アドミニストレータ:このツールでは MySQL サーバ、データベース、テーブル、そしてアカウントが管理されています。
- MySQL クエリブラウザ:このグラフィカルツールでは MySQL AB によって、MySQL データベースにおける作成、実行、そしてディスク最適化が行われます。
- MySQL 移動ツールキット:このツールはスキーマとデータを MySQL と使用するため、関連する他のデータベース管理システムから移動する際に使用されます。

これらの GUI プログラムそれぞれには個々のマニュアルがあり、<http://dev.mysql.com/doc/> からアクセスすることができます。

MySQL プログラム の概要

MySQL AB は様々な種類のプログラムを提供しています。

- MySQL サーバとその起動スクリプト :

- `mysqld` が MySQL サーバです。
- `mysqld_safe`, `mysql.server` と `mysqld_multi` はサーバ起動スクリプトです。
- `mysql_install_db`によってデータディレクトリと初期データベースが初期化されます。
- MySQL インスタンス・マネージャーが MySQL サーバのインスタンスを表示、そして管理します。

4章データベース管理ではこれらのプログラムについてより詳しい解説がなされています。

- サーバにアクセスしているクライアントプログラム :

- `mysql` は SQL ステートメントをインタラクティブに、また一括モードで実行するためのコマンドラインクライアント。
- `mysqladmin` は管理におけるクライアント。
- `mysqlcheck` はテーブルメンテナンスオペレーションを行います。
- `mysqldump` と `mysqlhotcopy` はデータベースのバックアップを行います。
- `mysqlimport` はデータファイルをインポートします。

- `mysqlshow` はデータベースとテーブルについての情報を表示します。

7章クライアントプログラムとユーティリティプログラムではこれらのプログラムについてより詳しい解説がなされています。

- サーバとは独立して作動するユーティリティプログラム：
 - `myisamchk` はテーブルメンテナンスオペレーションを行います。
 - `myisampack` はテーブルを圧縮し、リードオンリーにします。
 - `mysqlbinlog` はバイナリログファイルを処理するためのツールです。
 - `perror` はエラーコードの意味を表示します。

7章クライアントプログラムとユーティリティプログラムではこれらのプログラムについてより詳しい解説がなされています。

MySQL ディストリビューションの殆どがこれらのプログラム全てを含んでいますが、プラットフォームに特有のプログラムはこれに含まれません。(例：サーバ起動スクリプトは Windows では使用されません) 例外として RPM ディストリビューションはより明細かされています。サーバ用の RPM がひとつ、そしてそれとはまた別にクライアントプログラムのためのものがある、といった具合になっています。もしこのプログラムのうちのいくつかが欠けているようであれば、2章MySQL のインストールと更新を参照して、ディストリビューションのタイプとその内容を確認してください。全プログラムを含まないディストリビューションである可能性があり、その際には別のものをインストールする必要があります。

MySQL プログラムの起動

MySQL プログラムをコマンドライン (シェル、またはコマンドプロンプトから) から呼び出すには、そのプログラムに実行したい内容を命令するためのオプション、または引数を含むプログラム名を入力します。下に表示されているコマンドはプログラム実施のサンプルです。ここでの「`shell>`」はコマンドインタプリタへのプロンプトを表していて、ユーザによって入力されたものではありません。このプロンプトは使用されるコマンドインタプリタによって異なります。多く見られるものとしては、`sh` または `bash` として `$`、`csh` または `tcsh` として `%`、そして Windows の `command.com` または `cmd.exe` コマンドインタプリタとして `C:\>` が挙げられます。

```
shell> mysql -u root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump --user=root personnel
```

シングル、またはダブルダッシュ (`-`、`--`) で始まる引数はオプション引数です。これらは一般的にプログラムがサーバに対してとるべき接続タイプを特定するか、その作動モードに影響を与えます。オプション構文については [プログラム・オプションの指定](#) で述べられています。

ノンオプション引数 (ダッシュを伴わない引数) はプログラムに追加の情報を提供します。例えば `mysql` は最初のノンオプション引数をデータベース名に変換し、これによってコマンド `mysql -u root test` が `test` データベースの使用が要求されていることを指示します。

個々のプログラムについてより詳しい解説がなされている後半のセクションでは、どのオプションがプログラムによって認知されるかが明らかにされ、そして追加のノンオプション引数についても説明がされています。

オプションのうちのいくつかは多くのプログラムに共通しているものです。そのうちの最も多くは、`--host` (または `-h`)、`--user` (または `-u`)、そして `--password` (または `-p`) オプションで、コネクションパラメータを特定します。これらは MySQL サーバが作動しているホスト、そして MySQL アカウントのユーザネームとパスワードを表示します。これらのオプションは全ての MySQL クライアント・プログラムによって認知されます。これによって初めて、どのサーバに接続するか、そしてサーバの使用に必要なアカウントが特定されます。

その他の接続オプションとしては、TCP/IP ポート番号を特定する `--port` (または `-P`)、そして Unix 上 (Windows ではパイプ名と呼ばれる) で Unix ソケットファイルを特定する `--socket` (または `-S`) があります。

初期設定時のホスト名は `localhost` です。Unix 上でのクライアントプログラムには、このホストネーム `localhost` は特別な意味を持っています。これによってクライアントが Unix ソケットファイルを通し MySQL サーバに接続するこ

とができるようになります。これはまた、`--port` もしくは `-P` オプションがポート番号を特定するために与えられたとしても実行されます。クライアントによって TCP/IP 接続をローカルサーバに確実に行わせるためには、ホスト名、もしくは IP アドレス、ローカルサーバ名のいずれかを `127.0.0.1` に指定するために `--host` または `-h` を使用します。`--protocol=tcp` オプションを使用することによって、`localhost` に対してまた、プロトコル接続をさらに明確に指定することができます。

プログラムがインストールされている `bin` ディレクトリへのパスネームを使って MySQL を起動させることが必要とされることもあります。この場合は、`bin` ディレクトリ以外の辞書を使用して MySQL の起動を試み、「program not found」というエラーが表示された可能性があります。MySQL をさらに実用的に使用できるように `bin` ディレクトリへのパスネームを `PATH` 環境変数セッティングに加えることも可能です。こうすることによってプログラムを、パスネームではなくプログラム名を入力するだけで実行することが出来るようになります。例えば `/usr/local/mysql/bin` に `mysql` がインストールされている場合、`mysql` としてプログラムを実行することが可能で、`/usr/local/mysql/bin/mysql` として呼び出す必要はありません。

`PATH` を変数にセッティングするの際の手順におけるコマンドインタプリタについては、使用説明を参照してください。環境変数セッティングの構文は各インタプリタごとに特有となっています。(これに関する説明は [オプション指定のための環境変数の使用](#) で述べられています)

プログラム・オプションの指定

MySQL プログラムのオプションを指定するには次の手段があります。

- プログラム名に続くコマンドラインにオプションをリストアップします。これはプログラムを特別な形で立ち上げる際のオプションの中として最も代表的なものです。
- オプションをプログラム起動時に読み込まれるオプション ファイルの中にリストアップします。これはプログラム起動の度、自動的に読み込まれるように設定する際に有効です。
- オプションを環境変数にリストアップします。この方法はプログラム起動の際、その都度設定する場合に便利なオプションです。実際には、オプションファイルの使用理由の多くがこれに当てはまります。しかし、[Unix で複数サーバの実行](#) では、環境変数が大変役立つ一例が取り上げられています。ここでは TCP/IP ポート番号と Unix ソケット ファイルをサーバとクライアント プログラム両用に指定するために便利な手法が述べられています。

MySQL はどのオプションが最初に与えられたかを環境変数内の調査、そしてオプションファイルの読み取り、コマンドラインのチェックといった作業を通して判断します。これはつまり、環境変数は優位性が最も低く、コマンドラインのオプションは優位性が最も高いということを意味しています。

オプションは規則正しく処理されるため、特定のオプションが複数回指定されると、最後のものが最優先されるようになっています。ここで取り上げる次のコマンドは、`localhost` 上のサーバに接続するための `mysql` を立ち上げます。

```
shell> mysql -h example.com -h localhost
```

相反する、または関連するオプションが与えられた場合、後で与えられたオプションが前のものよりも優先されます。次のコマンドは `mysql` を「列名なし」モードで立ち上げます。

```
shell> mysql --column-names --skip-column-names
```

オプションの指定は、完全な形、または不明瞭な点なく頭部が入力されることで行われます。例えば、`--compress` オプションは `mysqldump` に `--compr` として与えられた場合は読み取られますが、不明瞭な形の `--comp` として与えられた場合は作動しません。

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

オプション頭部の入力次第では、プログラムに新たなオプションが実行される手順で問題が生じることもあるという点に注意してください。現在は問題のなくても今後不明瞭とされることも考えられます。

MySQL はオプションファイル内でプログラムのオプションに初期設定値を指定することでオプションを処理していきます。従って、これを利用すれば、プログラムを立ち上げる際毎回手動で入力しなくてもよいだけでなく、必要に応じてはコマンドラインのオプションを使うことで初期設定を塗り替えることもできます。

コマンドラインにおけるオプションの使用

コマンドラインにおけるプログラムのオプションは次の法則に従って作動します。

- オプションはコマンド名の後に置かれます。
- オプション引数はその名前が短いかに応じて、シングル、またはダブルダッシュで始まります。オプションの多くがこの両方を持ち合わせています。例えば、`-?`と`--help`はMySQLでヘルプメッセージの表示を行うオプションのそれぞれのフォームにあたります。
- オプション名は時にとてもデリケートなものでもあります。`-v`と`-V`はどちらも正しく、それぞれ異なる意味を持っています。(`-v`は`--verbose`の、`-V`は`--version`の短縮形としての意味で使われます。)
- オプションの中にはオプション名に続く値をとるものもあります。例えば、`-h localhost`または`--host=localhost`はMySQLがクライアントプログラムをホストしていることを示します。オプションの値はプログラムにMySQLが起動しているホストの名前を示します。
- 値を持つ長いオプションにおいては、オプション名と値を`=`印に従って切り離します。値を持つ短いオプションでその値はオプション名の直後に続いているか、またはスペースが空いていますが、`-hlocalhost`と`-h localhost`は同等の意味を持っています。ここでの例外は、MySQLパスワードを指定するオプションの法則です。このオプションは`--password=pass_val`という長い形でか、`--password`という短い形で与えられます。後者の場合 (パスワード値が与えられていない状態で) には、プログラムによってパスワードを入力するようになるとのプロンプトが表示されます。パスワードオプションはまた、`-ppass_val`、もしくは`-p`という短い形で入力することも可能です。しかしこの場合、もし値が付属していれば間にスペースを入れずオプションの直後に値が来るようにしなければいけません。その理由は、もしオプションの後にスペースが入れば、プログラムはこれに続く引数がパスワード値なのか、または何か別の引数なのかを特定することができなくなるからです。従って、ここで例に挙げるふたつのコマンドはそれぞれが全く異なった意味を持っています。

```
shell> mysql -ptest
shell> mysql -p test
```

初めのコマンドは`test`のパスワード値を使用するための`mysql`を指示しますが、初期設定データベースの特定は行いません。これとは異なり、もう一方は`mysql`プロンプトでパスワード値を求めるように指示し、さらに`test`を初期設定データベースとして使用します。

オプションのうちの数種はオン、またはオフの切り替えをコントロールします。例えば、`mysql`クライアントは、クエリから得られる結果の冒頭にカラム名の行を表示するか否かを決定するオプション`--column-names`をサポートします。この作業は初期設定において可能です。けれども、データだけを読み取り初期ヘッダーラインを読み取らないようなプログラムに`mysql`が出力された場合などは、この機能を無効にすることも可能です。

列名を無効にするには次のオプションのうちのいずれかを指定します。

```
--disable-column-names
--skip-column-names
--column-names=0
```

`--disable`と`--skip`の頭部と末尾`=0`は全て同じ働きをします。つまり、オプションをオフに切り替えます。

オプションを「enabled」に切り替えるためには、次のうちのいずれかを指定する必要があります。

```
--column-names
--enable-column-names
--column-names=1
```

もし頭部が`--loose`のオプションがあれば、もしプログラムがオプションを認知できない場合でもエラーで終了せず、ただ警告を發します。

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

この`--loose`はMySQLを同じコンピュータ上に複数回インストールした状態でプログラムを立ち上げ、オプションファイルにオプションをリストアップする時に役立ちます。全てのバージョンには認知されないようなオプションでも`--loose`またはオプションファイル内では`loose`を入力することによって使用可能になります。オプションを認知できるバージョンでは通常通りに処理され、そうでないバージョンでは警告が表示され、そして無視されます。

その他のオプションの中で場合によっては`mysql`とともに便利に働くのが`--execute`、または`-e`で、SQLステートメントをサーバに送り届けます。ステートメントはシングル、またはダブルの引用符で囲まれている必要があります。もしこの引用符内の値を使用したい場合には、ステートメントにダブルの引用符を使用し、シングルの引用符をステー

トメント内の値のうちのいずれかに使用します。このオプションが使用される際には、[mysql](#)がステートメントを実行し、その後終了します。

ここで例として、次のコマンドをユーザアカウントのリスト取得のために使うことにしましょう。

```
shell> mysql -u root -p --execute="SELECT User, Host FROM user" mysql
Enter password: *****
+-----+
| User | Host |
+-----+
| | gigan |
| root | gigan |
| | localhost |
| jon | localhost |
| root | localhost |
+-----+
shell>
```

長い形のタイプ (`--execute`) の後ろにはイコール (`=`) が続いている事を確認します。

上記の例では[mysql](#)データベースの名前は個別の引数として認められています。しかし、同じステートメントを初期設定データベースをしていしないコマンドを使用して実行することも可能です。

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
```

複数のSQLステートメントはセミコロンによって切り離され、コマンドライン上に認められることもあります。

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: *****
+-----+
| VERSION() |
+-----+
| 5.1.5-alpha-log |
+-----+
+-----+
| NOW() |
+-----+
| 2006-01-05 21:19:04 |
+-----+
```

オプション[--execute](#)、または[-e](#)は、アナログ形式のコマンドをMySQLクラスターへの[ndb_mgm](#)管理クライアントへと届けるために使用されることもあります。例については [安全なシャットダウンと再起動](#)を参照してください。

オプションファイルの使用

MySQLプログラムの多くは、スタートアップのオプションをオプションファイル (時にコンフィグレーションファイルとも呼ばれる) から解読することができます。オプションファイルはよく使用されるオプションを指定するのに便利で、これによって毎回プログラムを起動させるたびにコマンドラインにおける作業を省略することができます。MySQLサーバにはMySQLによって多大な[preconfigured option files](#)が提供されます。

プログラムがオプションファイルを解読しているかを確認するには、[--help](#) (`mysqld`には[--verbose](#)と[--help](#)) を使用してプログラムを起動します。プログラムがオプションファイルを解読している場合には、ヘルプメッセージによって解読中のファイルとどのオプショングループを認識中かが表示されます。

注記 :MySQL クラスタープログラムを伴うオプションファイルについては [MySQL Cluster の設定](#)で述べられています。

Windowsにおいて、MySQLプログラムは次のファイルからスタートアップオプションを解読します。

ファイル名	目的
WINDIR\my.ini	グローバルオプション
C:\my.cnf	グローバルオプション
INSTALLDIR\my.ini	グローバルオプション
defaults-extra-file	<code>--defaults-extra-file=</code> path で指定されたファイル (あれば)

WINDIR ユーザのWindowsディレクトリの場所を表しています。一般的に、**C:\WINDOWS** もしくは **C:\WINNT** になります。以下のコマンドを使用して **WINDIR** 環境変数の値から正確なロケーションを割り出すことができます。

```
C:\> echo %WINDIR%
```

INSTALLDIR はMySQLのインストールディレクトリを表します。これは主に、**C:\PROGRAMDIR\MySQL\MySQL 5.1 Server** です。MySQL 5.1がインストールとコンフィギュレーションウィザードを使用してインストールされた場合、**PROGRAMDIR** がプログラムディレクトリ（通常は英語バージョンWindows上の **Program Files**）を表します。詳細は **my.ini ファイルのロケーション** を参照してください。

Unixにおいて、MySQLプログラムは次のファイルからスタートアップオプションを解釈します。

ファイル名	目的
/etc/my.cnf	グローバルオプション
\$MYSQL_HOME/my.cnf	サーバ固有のオプション
defaults-extra-file	--defaults-extra-file=path で指定されたファイル（あれば）
~/.my.cnf	ユーザ固有のオプション

MYSQL_HOMEはサーバ固有のファイル**my.cnf**を含むディレクトリへのパスを含む環境変数です。

MYSQL_HOME がセットされていない状態で **mysqld_safe** プログラムを使ってサーバをスタートさせると、**mysqld_safe** は次のように **MYSQL_HOME** をセットしようとします。

- **BASEDIR** と **DATADIR** をそれぞれ、MySQLベースディレクトリとデータディレクトリのパスネームの代理に立てます。
- **DATADIR**には**my.cnf**ファイルが存在し、**BASEDIR**には存在しない場合、**mysqld_safe**は**MYSQL_HOME**を**DATADIR**にセットします。
- または、もし**MYSQL_HOME**がセットされておらず、**my.cnf**ファイルが**DATADIR**に存在しない場合、**mysqld_safe**は**BASEDIR**に**MYSQL_HOME**をセットします。

MySQL 5.1では**DATADIR**の**my.cnf**ファイルのロケーションとしての使用は認証されていません。**BASEDIR** の方が良い場所です。

一般的に**DATADIR**は**/usr/local/mysql/data**でバイナリインストールに、または**/usr/local/var**でソースインストールに使用されます。これはコンフィギュレーションタイムに特定されたデータディレクトリロケーションであって、**mysqld**が起動したときの**--datadir**オプションに呼応しているのではないということに注意してください。**--datadir**の起動時の使用はサーバがオプションファイルを探す際に何の影響ももたらしません。これはサーバがオプションを処理する前にオプションファイルを探すからです。

MySQLは指示された通りの順序でオプションファイルを探し、その中で存在するものを解釈します。もし使用したいオプションファイルが存在しなければ、プレーンテキストエディタで新しく作ってください。

与えられたオプションに対して複数のインスタンスが挙げた場合には、最後のものが優先されます。ここでの唯一の例外は**mysqld**に対して、**--user**オプションの最初のインスタンスがセキュリティ上用心のため使用され、オプションファイルのユーザ固有タイプがコマンドラインで優先されるのを防ぎます。

注記:Unixのプラットフォームでは、MySQLは全世界から書き込み・書き取り可能なコンフィギュレーションファイルを認知しません。これは故意に行われており、セキュリティ対策です。

MySQL起動時にコマンドラインにおいて与えられるロングオプションは、その全てがオプションファイルにも送られる可能性があります。プログラムに対して使用可能な存在するオプションのリストを入手するには、**--help**オプションを使用してプログラムを起動してください。

オプションファイルへとオプションを特定するための構文はコマンドラインの構文と似通っており、異なる点は頭のダブルダッシュを省略するということです。例えば、コマンドラインにおける**--quick**または**host=localhost**は、**quick**または**host=localhost**としてオプションファイル内に指定されます。**--loose-opt_name**という形のオプションをオプションファイルに指定するには、**loose-opt_name**という形で入力します。

オプションファイルでは空白のラインは認識されません。空白でないラインは次のいずれかの形を取ります。

- `#comment, ;comment`

コメントラインは `#` または `;` で始まります。`#` コメントはラインの真ん中からスタートすることもできます。

- `[group]`

`group` とはプログラムの名前、またはオプションをセットしたいグループを示します。グループラインの後には、オプションファイル、あるいは別のグループラインの端が与えられるまで、オプションをセットするラインは全て指定されたグループに当てはまります。

- `opt_name`

これはコマンドラインの `--opt_name` と同じです。

- `opt_name=value`

これはコマンドラインの `--opt_name=value` と同じです。オプションファイルでは `=` の周囲にスペースをおくこともできますが、これはコマンドラインでは適切な操作ではありません。シングルの引用符で閉じることができ、これはコメントラインが `#`、またはスペースを含んでいる場合に有効です。

数値の値をとる変数については、値は、 1024 、 1024^2 乗または 1024^3 乗、それぞれの乗数を示すために、**K**、**M** あるいは **G** (大文字か小文字のいずれか) の接尾辞で与えることができます。例えば次のコマンドは `mysqladmin` へサーバに 1024 回 ping を打つように、そして各 ping の間それぞれ 10 秒ずつ間を空けるようにと命令します。

```
mysql> mysqladmin --count=1K --sleep=10 ping
```

ブランクのリードと追跡はオプション名とその値から自動的に削除されます。バックスペース、タブ、復帰改行、キャリッジリターン、バックスラッシュおよびスペース文字を表わすために、オプション値の中でエスケープシーケンス `\b`、`\t`、`\n`、`\r`、`\\`、および `\'` を使用することもできます。

`\'` エスケープシーケンスが単一のバックスラッシュを表わすので、`\\` として各 `\` を書かなければなりません。選択肢としては、パスネームセパレータとして `\` ではなく `/` を使用して、値を指定することができます。

オプショングループ名がプログラム名と同じである場合、グループ中のオプションは特定のそのプログラムに当てはまります。例えば、`[mysqld]` および `[mysql]` グループは、`mysqld` サーバおよび `mysql` クライアントプログラムにそれぞれ当てはまります。

`[client]` オプショングループは、すべてのクライアントプログラム (`mysqld` は除く) によって解釈されます。これによって全てのクライアントに当てはまるオプションを指定することが可能になります。例えば、`[client]` は、サーバに接続するために使用するパスワードを指定するために使用する完全なグループです (しかしオプションファイルはユーザー当事者にのみ判読可能、書き込み可能であることに注意してください。これによって、他のユーザはこのパスワードを見つけ出すことができなくなります)。使用する全てのクライアントプログラムに認識されない限り、`[client]` グループにオプションを入れないようにしてください。オプションを解釈しないプログラムは、オプションを実行しようとすればエラーメッセージを表示した後に中断されます。

次に示されているのは典型的なグローバルオプションファイルです。

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

上のオプションファイルは、`key_buffer_size` と `max_allowed_packet` の変数をセットするラインのために `var_name=value` 構文を使用します。

ここに典型的なユーザオプションファイルがあります。

```
[client]
```

```
# The following password will be sent to all standard MySQL clients
password="my_password"
```

```
[mysql]
no-auto-rehash
connect_timeout=2
```

```
[mysqlhotcopy]
interactive-timeout
```

特定のMySQLリリースシリーズのmysqldサーバのみによって解釈されるオプショングループを作りたいければ、[\[mysqld-5.0\]](#)、[\[mysqld-5.1\]](#)などの名前を備えたグループの使用により可能になります。次のグループは次のことを示します--newオプションは5.1.xバージョン番号を持ったMySQLサーバによってのみ使用される必要があります。

```
[mysqld-5.1]
new
```

MySQL 5.0.4から始め、オプションファイルを求めて特定のディレクトリを探索する他のオプションファイル、および[!includedir](#)を含むようにオプションファイルの中で[include](#)指令を使用することができます。例えば、[/home/mydir/myopt.cnf](#)ファイルを含むためには次の指令を使用することができます。

```
!include /home/me/myopt.cnf
```

[/home/mydir](#)ディレクトリを探し出し、見つかったオプションファイルを解釈するためにはこの指令を使います。

```
!includedir /home/mydir
```

注記:一般的に、Unixオペレーティングシステム上で見つけることができ、[!includedir](#)指令の使用が含まれているファイルの全てが、[.cnf](#)で終わるファイル名を持っていないわけじゃありません。Windowsにおいては、この指令が[.ini](#)か[.cnf](#)拡張を備えたファイルをチェックします。

付属のファイルから解釈されたオプションは、一般的なオプショングループのコンテキストに適用されることに注意してください。[my.cnf](#)に次のラインを書かなければならないと仮定してください。

```
[mysqld]
!include /home/mydir/myopt.cnf
```

この場合、[myopt.cnf](#)ファイルはサーバ用のみ処理されます。また、[!include](#)指令は全てのクライアントアプリケーションによって無視されます。しかし、次のラインを使用したならば、ディレクトリ[/home/mydir/my-dump-options](#)は、サーバによって、あるいは他のクライアントアプリケーションによってではなく、[mysqldump](#)のみによってオプションファイルのチェックが行われます。

```
[mysqldump]
!includedir /home/mydir/my-dump-options
```

ソース分配を持っていれば、[support-files](#)ディレクトリ中に[my-xxxx.cnf](#)という名前のサンプルオプションファイルが見つかります。バイナリ分配の場合は、MySQLインストールディレクトリの下で[support-files](#)ディレクトリの中を見てください。Windowsにおいては、サンプルオプションファイルはMySQLインストールディレクトリに置かれていることがあります(この章の前半、どこのことか分からない場合には[2章MySQLのインストールと更新](#)を参照してください)。一般的に、小さな、ミディアム、大きな、また大規模システムのためのサンプルオプションファイルがあります。これらのファイルのうちのひとつで実験するには、Windows上で[C:\my.cnf](#)に、あるいはUnix上のホームディレクトリ中にある[.my.cnf](#)にコピーしてください。

注記:Windowsにおいては、[.cnf](#)か[.ini](#)オプションファイル拡張が表示されない可能性があります。

オプションファイルをサポートするMySQLプログラムは全て、次のオプションを扱い、オプションファイル取り扱いに影響します。したがって、これらはオプションファイル中ではなくコマンドライン上で与えられる必要があります。適切な動作のためには、これらのオプションの各々は速やかにコマンド名に続かなければなりません。ここでの例外として、[--print-defaults](#)は[--defaults-file](#)または[--defaults-extra-file](#)直後に使用することができます。また、望み通りに解釈されないかもしれないので、ファイル名を指定する場合、```シエルメタ文字の使用は回避するべきです。

- [--no-defaults](#)

どんなオプション・ファイルも読まないでください。

- [--print-defaults](#)

オプションファイルから得るプログラム名、および全てのオプションを印刷してください。

- `--defaults-file=file_name`

与えられたオプションファイルだけを使用してください。`file_name`はファイルへのフルパスネームです。ファイルが存在しないかアクセス不能な場合、プログラムはエラーメッセージとともに終了します。

- `--defaults-extra-file=file_name`

グローバルなオプションファイルの後に、しかし(Unix上では)ユーザオプションファイルの前にこのオプションファイルを読んでください。`file_name`はファイルへのフルパスネームです。ファイルが存在しないかアクセス不能な場合、プログラムはエラーメッセージとともに終了します。

- `--defaults-group-suffix=str`

このオプションが与えられる場合、プログラムは通常のオプショングループだけでなく通常の名前、および`str`の接尾辞を備えたグループも解釈します。例えば、`mysql`クライアントは通常`[client]`および`[mysql]`グループを解釈します。`--default-group-suffix=_other`オプションが与えられた場合には、`mysql`はさらに`[client_other]`および`[mysql_other]`グループも解釈します。

シェルスクリプトでは、オプションファイルを解析する`my_print_defaults`プログラムを使用し、オプションが与えられたプログラムによって何に使用されるか確かめることができます。次の例は、`[client]`および`[mysql]`グループで見つかったオプションを表示するように依頼された時、`my_print_defaults`によって生産される可能性のある出力を示しています。

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

開発者への注意:オプションファイル取り扱いは、単純に適切なグループ中で、または任意のコマンドライン引数の前でオプションを全て処理することにより、Cクライアントライブラリで実行されます。複数回指定されるオプションの最後のインスタンスを使用するプログラムにとって、これは有効な方法です。複数回指定されるオプションをこの方法で扱うが、オプションファイルを解釈しないCまたはC++のプログラムを持っている場合、この機能を使えるようにするためにはたった2行のラインを加えればよいだけです。この方法を確認するには、スタンダードMySQLクライアントのうちのいずれかのソースコードをチェックしてください。

MySQLへの他の言語インターフェースのうちのいくつかはCクライアントライブラリに基づいています。またその中には、オプションファイルのコンテンツへアクセス方法が提供されているものもあります。これらはPerlとPythonを含んでいます。詳細については、手持ちのインターフェースにとって好ましい使用説明を参照してください。

あらかじめ形成されたオプション・ファイル

MySQLは、MySQLサーバを調整する際にベースとして使用することができる、あらかじめ形成されたオプションファイルを数多く提供します。基礎構成ファイルとしての使用のための適切な位置へリネームしコピーすることのできる`my-small.cnf`、`my-medium.cnf`、`my-huge.cnf`、および`my-huge.cnf`といったファイル用のインストールディレクトリの中を見てください。名前、そして適切な位置に関しては[オプションファイルの使用](#)の一般的な情報を参照してください。Windowsにおいてこれらのファイルは、`.cnf`よりも`.ini`拡張を持っている確立が高いです。

オプション指定のための環境変数の使用

環境変数を使用してオプションを指定するためには、コマンド処理プログラムに適切な構文を使用し、変数をセットしてください。例えば、WindowsまたはNetWareにおいては、`USER`変数にMySQLアカウント名を指定させることができます。そのためには、次の構文を使用してください。

```
SET USER=your_name
```

Unixの上の構文はユーザのシェルに依存します。`MYSQL_TCP_PORT`変数を使用してTCP/IPポート番号を指定すると仮定してください。典型的な構文 (`sh`、`bash`、`zsh`など) は、以下の通りです。

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```


最初のコマンドが変数を設定し、また、`export`コマンドがシェル環境に変数を輸出することによって、その値がMySQLおよび他のプロセスにアクセス可能になります。

`csch`と`tcsh`については、シェル変数を環境を利用可能にするために`setenv`を使用してください。

```
setenv MYSQL_TCP_PORT 3306
```

環境変数をセットするコマンドは効果を直ちに現わすため、ユーザのコマンドプロンプトで実行することができます。しかし、このセッティングはログアウトするまでしか続きません。このセッティングがログインごとに効果を発揮するようにするためには、コマンドインタプリタが毎回起動時に解読するスタートアップファイルに適切なコマンド(複数可)を置きます。典型的なスタートアップファイルは、Windows用の`AUTOEXEC.BAT`、`bash`用の`.bash_profile`あるいは`tcsh`用の`.tcshrc`です。詳細についてはコマンドインタプリタについての使用説明を参照してください。

環境変数にはMySQLプログラムオペレーションに影響する環境変数が全てリストアップされています。

プログラム変数セットのためのオプション使用

MySQLプログラムの多くがランタイムでセットすることができる内部変数を持っています。プログラム変数は、値をとる他の長いオプションと同じ方法を課されます。例えば、`mysql`は、そのコミュニケーションバッファの最大サイズをコントロールする`max_allowed_packet`変数を持っています。`mysql`のために16MBの値に`max_allowed_packet`変数をセットするためには、下記コマンドのどちらかを使用してください。

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

第1のコマンドはバイトで値を指定します。第2はメガバイトで値を指定します。数値の値をとる変数については、値は、 1024 、 1024^2 乗または 1024^3 乗、それぞれの乗数を示すために、**K**、**M**あるいは**G**(大文字か小文字のいずれか)の接尾辞で与えることができます。(例えば、`max_allowed_packet`をセットするために使用された時、接尾辞はキロバイト、メガバイトあるいはギガバイトのユニットを示します)。

オプションファイルでは、可変セッティングが主要なダッシュなしで与えられます。

```
[mysql]
max_allowed_packet=16777216
```

または：

```
[mysql]
max_allowed_packet=16M
```

好みによって変数名中の下線はダッシュとして指定することができます。次のオプショングループは等価です。共に512MBにサーバの重要なバッファのサイズをセットします。

```
[mysqld]
key_buffer_size=512M
```

```
[mysqld]
key-buffer-size=512M
```

注記 :MySQL 4.0.2以前は、セットするプログラム変数用の唯一の構文は`--set-variable=option=value` (あるいはオプションファイル中の`set-variable=option=value`) でした。この構文はまだ存在しますが、MySQL 4.0.2でもクレームが多いものです。

サーバシステム変数の多くはまた、ランタイムでもセットすることができます。詳細は、**動的システム変数** をご覧ください。

第4章 データベース管理

目次

サーバ サイド プログラムの概略	150
mysqld — MySQL サーバ	151
オプションと変数のリファレンス	152
コマンド オプション	152
システム変数	162
システム変数の使用	187
ステータス変数	195
SQL モード	204
シャットダウン プロセス	209
サーバ サイド ヘルプ	210
MySQL サーバ スタートアップ プログラム	210
mysqld_safe — MySQL サーバ スタートアップ スクリプト	210
mysql.server — MySQL サーバ スタートアップ スクリプト	213
mysqld_multi — 複数のMySQL サーバ管理	213
mysqlmanager — MySQL Instance Manager	217
MySQL Instance Manager コマンド オプション	217
MySQL Instance Manager の設定ファイル	220
MySQL Instance Manager で MySQL Server の起動	221
Instance Manager のユーザとパスワード管理	222
MySQL サーバ インスタンス ステータスの監視	224
MySQL Instance Manager へ接続	225
MySQL Instance Manager のコマンド	225
インストール関連プログラム	228
make_win_bin_dist — Package MySQL 配布 (ZIP アーカイブ)	228
mysql_fix_privilege_tables — MySQL システム テーブルのアップグレード	229
mysql_install_db — MySQL データ ディレクトリ 初期化スクリプト	230
mysql_upgrade — MySQL アップグレードのテーブル チェック	231
mysql_tzinfo_to_sql — タイム ゾーン テーブルのロード	232
セキュリティ問題	232
セキュリティ ガイドライン	232
MySQL のクラッカー対策	234
セキュリティ関連の mysqld オプション	236
LOAD DATA LOCAL のセキュリティ関連事項	237
ユーザによる MySQL の実行	238
MySQL アクセス権限システム	238
権限システムの役割	239
権限システムの機能	239
MySQL 提供の権限	243
MySQL サーバへの接続	246
アクセス制御の段階 1: 接続確認	246
アクセス制御の段階 2: 接続確認	250
権限の変更が反映するタイミング	252
Access denied エラーの原因	252
MySQL 4.1 のパスワードハッシュ	256
MySQL ユーザ アカウント管理	260
MySQL ユーザ名とパスワード	260
MySQL への新規ユーザの追加	261
MySQL ユーザの削除	264
ユーザ リソースの制限	264
パスワードの設定	265
パスワードのセキュリティ	266
接続安全	267
バックアップとリカバリ	274

データベースのバックアップ	274
バックアップとリカバリ手法の例示	276
任意時点のリカバリ	279
テーブル保守とクラッシュ リカバリ	280
MySQL のローカライズと国際的使用	289
データおよびソート用キャラクタ セット	289
英語以外のエラーメッセージ	290
新しいキャラクタ セットの追加	291
キャラクタ定義配列	292
文字列照合サポート	292
マルチ バイト文字サポート	293
キャラクタ セットに関する問題	293
MySQL サーバのタイム ゾーン サポート	293
MySQL サーバのローケル サポート	295
MySQL サーバ ログ	297
一般クエリとスロー クエリのログ出力先の選択	298
エラー ログ	299
一般クエリ ログ	300
バイナリ ログ	301
スロー クエリ ログ	304
ログ ファイルの保守	305
同じマシン上での複数 MySQL サーバの実行	306
Windows で複数サーバの実行	307
Unix で複数サーバの実行	310
複数サーバ環境でのクライアントプログラムの使用	311
MySQL クエリ キャッシュ	311
クエリ キャッシュの動作	312
クエリ キャッシュでの SELECT オプション	313
クエリ キャッシュの設定	314
クエリ キャッシュのステータスと保守	315

この章では、MySQLのインストールに関わる管理事項について説明します。

- サーバ設定
- ユーザ管理
- バックアップ
- ログ ファイル
- クエリ キャッシュ

サーバ サイド プログラムの概略

MySQL サーバ、[mysqld](#) は、MySQL 設定の中核を担うメイン プログラムです。このプログラムには、MySQL を設置するときの設定方法やサーバの起動と停止の関連スクリプトなどを添付しています。このセクションでは、サーバと関連プログラムの概要について説明します。次のセクションでは、そのプログラムに関する詳細について説明します。

MySQL プログラムには様々なオプションがあります。それぞれのプログラムには、プログラム オプションの詳細などを示す [--help](#) オプションがあります。必要に応じて、[mysqld --help](#) コマンドを試行してください。

MySQL の標準プログラム (デフォルト値) は、コマンドラインまたはオプション ファイルなどで指定して、書き換えることができます。詳細は、[プログラム・オプションの指定](#) を参照してください。

次のリストは、MySQL サーバと関連プログラムの概説です。

- [mysqld](#)

SQL デーモン (MySQL サーバ)。クライアントがサーバへ接続することで、データベースへアクセスするという仕組みであるため、クライアント プログラムを使用するには、[mysqld](#) が稼動していることが条件となる。詳細は [mysqld — MySQL サーバ](#) を参照のこと。

- [mysqld_safe](#)

サーバの起動スクリプト。[mysqld_safe](#) で、[mysqld](#) の起動を試行する。詳細は [mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) を参照のこと。

- [mysql.server](#)

サーバの起動スクリプト。System V の実行ディレクトリを使用しているシステムで使用するスクリプトである。これで 特定の動作レベルのシステム サービスを立ち上げる。このスクリプトは、MySQL サーバを起動するために、[mysqld_safe](#) を呼び出す。詳細は [mysql.server — MySQL サーバ スタートアップ スクリプト](#) を参照のこと。

- [mysqld_multi](#)

サーバの起動スクリプト。複数サーバを使用している場合の起動、停止を行う。詳細は [mysqld_multi — 複数の MySQL サーバ管理](#) を参照のこと。[mysqld_multi](#) の択一的なコマンドには、[mysqlmanager](#) がある。これを MySQL Instance Manager と呼ぶ。詳細は [mysqlmanager — MySQL Instance Manager](#) を参照のこと。

- [mysql_install_db](#)

このスクリプトは MySQL データベースを作成し、デフォルト権限で特権テーブルを初期設定する。MySQL を初めてシステムにインストールするときに一度だけ実行する。詳細は [Unix のインストール後のプロシージャ](#) を参照のこと。

- [mysqlmanager](#)

MySQL Instance Manager は MySQL サーバの監視と管理を行うプログラム。詳細は [mysqlmanager — MySQL Instance Manager](#) を参照のこと。

- [mysql_fix_privilege_tables](#)

MySQL アップグレード操作を行った後にこのプログラムを使う。このプログラムは、新しいバージョンの MySQL で発生した変更にあわせて権限テーブルを更新する。詳細は [mysql_fix_privilege_tables — MySQL システム テーブルのアップグレード](#) を参照のこと。

ノート：MySQL 5.1.7 以降は、[mysql_upgrade](#) よりも優先のコマンドである。

- [mysql_upgrade](#)

MySQL アップグレード操作を行った後にこのプログラムを使う。このプログラムは、テーブルの互換性をチェックし、必要に応じて修正を行う。そして、新しいバージョンの MySQL で発生した変更を特権テーブルで更新する。詳細は [mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照のこと。

- [mysql_tzinfo_to_sql](#)

このプログラムは zoneinfo データベース (タイムゾーンを記述しているファイル セット) のホスト システムの内容を使用して、[mysql](#) のタイムゾーン テーブルをロードする。詳細は [mysql_tzinfo_to_sql — タイムゾーン テーブルのロード](#) を参照のこと。

- [make_win_bin_dist](#)

このプログラムは Windows で使用。ソース ディストリビューションを建てたあとに、インストール用の MySQL ディストリビューションをパッケージ化する。詳細は [make_win_bin_dist — Package MySQL 配布 \(ZIP アーカイブ\)](#) を参照のこと。

サーバ ホストでは他のプログラムも稼動しています。

- [make_binary_distribution](#)

このプログラムでコンパイルした MySQL のバイナリ リリースを行う。これは、他の MySQL ユーザの便宜を図り、FTP を経由して [ftp.mysql.com の/pub/mysql/upload/](ftp://ftp.mysql.com/pub/mysql/upload/) で送ることができる。

mysqld — MySQL サーバ

[mysqld](#) は MySQL サーバです。ここでは、MySQL サーバ設定について説明します。

- サーバ サポートの起動オプション
- サーバのシステム変数
- サーバ ステータス変数
- SQL モードの設定方法
- 停止プロセス

ノート:MySQL サーバのバイナリとコンフィギュレーションでは、すべてのストレージ エンジンをサポートしていません。使用している MySQL サーバでサポートがあるストレージ エンジンを確認するには、[SHOW ENGINES 構文](#)を参照してください。

オプションと変数のリファレンス

次のテーブルに、`mysqld` 内で適用できるすべてのコマンド ライン オプション、サーバ、ステータス変数をリストします。

テーブルには、コマンドライン オプション (Cmd-line)、設定ファイル のオプション (Option file)、サーバのシステム変数 (Server Var)、ステータス変数 (Status var) の一覧と、どのオプションや変数が利用可能であるかということも示しています。コマンドラインやオプション ファイルで設定するサーバ オプションが、対応するサーバシステムやステータス変数の名前と異なる場合には、そのオプションの下に変数名を記しています。ステータス変数に関しては、変数のスコープ (Scope) がグローバル、セッション、あるいはその両方です。それぞれのオプションと変数に関する詳細や設定方法は、それぞれのリンクを参照してください。

注記

現在、このテーブルの情報拡大または簡略化を行っています。テーブルに関する改善と追加情報に関しては、近日中の公開となります。

コマンド オプション

`mysqld` サーバを立ち上げるときに、[プログラム・オプションの指定](#)で説明しているやり方で、プログラム オプションを指定します。オプション ファイルまたはコマンドラインでオプションを使用することが、最も一般的な方法です。作業を開始する前に、オプション ファイルなどを利用して、実行ごとにサーバが常に同じオプションを使用していることを確認してください。詳細は [オプションファイルの使用](#)を参照してください。

`mysqld` は `[mysqld]` と `[server]` のグループからオプションを読み込みます。`mysqld_safe` は `[mysqld]`、`[server]`、`[mysqld_safe]`、`[safe_mysqld]` などのグループからオプションを読み込みます。`mysql.server` は `[mysqld]` と `[mysql.server]` のグループからオプションを読み込みます。

Embedded MySQL サーバは通常、`[server]`、`[embedded]`、および `[xxxxx_SERVER]` からオプションを読み込みます。ここでは、`xxxxx` はこのサーバを組み込んでいるアプリケーション名です。

`mysqld` には、様々なコマンド オプションがあります。`mysqld --help` を実行すると概説を参照できます。完全なオプション一覧を参照するには、`mysqld --verbose --help` を実行してください。

次に、一般的なサーバ オプションについて説明します。別用途のオプションについては、別のセクションで説明します。

- セキュリティに関わるオプションは、[セキュリティ関連の mysqld オプション](#)を参照してください。
- SSL関連オプションは、[SSL コマンド オプション](#)を参照してください。
- バイナリ ログを制御するオプションは[バイナリ ログ](#)を参照してください。
- レプリケーションに関連するオプションは[レプリケーションのオプションと変数](#)を参照してください。
- ストレージ エンジン固有のオプションは[MyISAM スタートアップオプション](#)、[InnoDB 起動オプションとシステム変数](#)、`mysqld` の [MySQL Cluster 関連コマンド オプション](#)などを参照してください。

変数名をオプションに使用して、サーバのシステム変数の値を設定できます。これに関しては、このセクションで後述します。

- `--help, -?`

ショート ヘルプ メッセージを表示、終了。詳細メッセージを参照するには、`--verbose` と `--help` オプションを使用する。

- `--allow-suspicious-udfs`

メイン関数が `xxx` というシンボル (名前) だけのユーザ定義関数をロードするかどうかを制御する。デフォルトはオフ (無効) で、少なくとも一つの追加関数を持ったユーザ定義関数だけをロードできる。つまり、不正なユーザ定義関数をロードする危険性を防ぐ設定になっている。詳細は [User-Defined Function Security Precautions](#) を参照のこと。

- `--ansi`

MySQL 構文ではなく、ANSI 準拠の SQL 文を使用する。SQL モードの精密制御には、`--sql-mode` オプションを使用する。詳細は [ANSIモードでのMySQLの実行](#) および [SQL モード](#) を参照のこと。

- `--basedir=path, -b path`

MySQL をインストールしているディレクトリへの基準パス。パスは通常、このディレクトリに関係する。

- `big-tables`

すべてのテンポラリ セットをファイルに保存して、大きな結果セットにする。ほとんどの「table full」エラーがこのオプションで解決するが、メモリ内テーブルだけで足りるクエリの実行速度も遅くする。MySQL 3.23.2 以降、必要に応じて、小さいテンポラリ テーブルにメモリを使用し、大きな結果セットを自動的にディスク テーブルでの保存に切り替える。

- `--bind-address=IP`

バインドする IP アドレス。

- `--binlog-format={row|statement}`

レプリケーションに、レコード ベースとステートメント ベースのどちらを使用するかを指定する。詳細は [レプリケーション フォーマット](#) を参照のこと。(MySQL 5.1.5 からの導入)

- `--binlog-row-event-max-size=N`

行ベースのバイナリ ログ イベントの最大サイズ (バイト) を指定する。イベントにグループ化する行のサイズはこの値より小さいことが望ましい。値を256の倍数とする。デフォルトは1024。詳細は [レプリケーション フォーマット](#) を参照のこと。(MySQL 5.1.5 からの導入)

- `--bootstrap`

`mysql_install_db` スクリプト実行時に使用するオプション。MySQL サーバを起動せずに MySQL 権限テーブルを作成できる。

このオプションは、MySQL のコンフィギュレーションに `--disable-grant-options` オプションを使用していた場合は使えない。詳細は [典型的な configure オプション](#) を参照のこと。

- `--character-sets-dir=path`

キャラクタ セットを格納しているディレクトリ。詳細は [データおよびソート用キャラクタ セット](#) を参照のこと。

- `--character-set-client-handshake`

クライアント送信のキャラクタ セット情報を無視しない (ようにする) オプション。クライアント情報を無視して、サーバのデフォルトのキャラクタ セットを使用するには、`--skip-character-set-client-handshake` を使用する (MySQL が MySQL 4.0. のように動作するようにする。)

- `--character-set-filesystem=charset_name`

ファイルシステムのキャラクタ セット。[character_set_filesystem](#) のシステム変数を設定する。(MySQL 5.1.6. からの導入)

- [--character-set-server=charset_name, -C charset_name](#)

デフォルトのキャラクタ セットを設定するときに [charset_name](#) を使用する。詳細は [データおよびソート用キャラクタ セット](#) を参照のこと。このオプションを使用する場合には、非デフォルトのキャラクタ セットを指定する。照合順序は [--collation-server](#) で指定する。

- [--chroot=path](#)

MySQL サーバ起動中に [chroot\(\)](#) のシステム コールを使用して、[mysqld](#) の Closed 環境でサーバを起動 (デーモンを配置) する。これは推奨セキュリティ対策。ただし、このオプションを使用すると [LOAD DATA INFILE](#) と [SELECT ... INTO OUTFILE](#) に制約が加わる。

- [--collation-server=collation_name](#)

サーバのデフォルトの照合順序を設定するときに [collation_name](#) を指定する。詳細は [データおよびソート用キャラクタ セット](#) を参照のこと。

- [--console](#)

[--log-error](#) を指定している場合でも、[stderr](#) および [stdout](#) にエラー ログ メッセージを書き込む。Windows のみに適用。このオプションを使用した場合、[mysqld](#) で、コンソール画面を閉じない (開いたままになる)。

- [--core-file](#)

[mysqld](#) が異常終了した場合に、コア ファイルを作成する。システムによっては、[mysqld_safe](#) ([mysqld](#) のラッパ) に [--core-file-size](#) の指定が必要になる。詳細は [mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) を参照のこと。注意: [--user](#) オプションも使用している場合、Solaris などシステムではコア ファイルを作成できない。

- [--datadir=path, -h path](#)

データ ディレクトリへのパス

- [--debug\[=debug_options\], -# \[debug_options\]](#)

[--with-debug](#) でコンパイルした MySQL を使っている場合、このオプションを使用して [mysqld](#) のトレース ファイルを取得できる。[debug_options](#) の文字列は、['d:t:o,file_name'](#) という並び (通常)。デフォルトは ['d:t:i:o,mysqld.trace'](#)。詳細は [Creating Trace Files](#) を参照。

MySQL 5.1.12 以降、デバッグ サポートに [--with-debug](#) を使用して MySQL をコンパイルすると、サーバ起動時に [--debug="d,parser_debug"](#) を使用できる。これは、SQL 文の処理に使用している Bison パーサーが、サーバの標準エラー出力にパーサー トレースをダンプすることを起因します。

- [--default-character-set=charset_name](#) (DEPRECATED)

デフォルトのキャラクタ セットを設定するときに、[charset_name](#) を使用する。このオプションは [--character-set-server](#) をうけて、廃止予定である。詳細は [データおよびソート用キャラクタ セット](#) を参照のこと。

- [--default-collation=collation_name](#)

デフォルトの照合順序を設定するときに [collation_name](#) を使用する。このオプションは [--collation-server](#) をうけて、廃止予定である。詳細は [データおよびソート用キャラクタ セット](#) を参照のこと。

- [--default-storage-engine=type](#)

デフォルトのストレージ エンジン (テーブル タイプ) を設定する。詳細は [13章ストレージエンジンとテーブルタイプ](#) を参照のこと。

- [--default-table-type=type](#)

このオプションは、[--default-storage-engine](#) のシノニム (別名)。

- `--default-time-zone=timezone`

サーバのデフォルト タイム ゾーンを指定する。このオプションで、グローバル スコープの `time_zone` システム変数をセットする。このオプションを使用しない場合、デフォルトのタイム ゾーンは、システムのタイム ゾーンと同じになる。(system_time_zone システム変数の値)

- `--delay-key-write={OFF|ON|ALL}`

遅れたキーの書き込みをどうするかを指定する。キーの書き込みが遅れると、キー バッファを `MyISAM` テーブルへの書き込み間のフラッシュを抑制する。`OFF` で遅れたキーの書き込みを無効にする。`ON` は `DELAY_KEY_WRITE` オプションで作成したテーブルに対して遅れたキーの書き込みを可能にする。`ALL` はすべての `MyISAM` テーブルに対するキー書き込みを遅らせる。詳細は [サーバパラメータのチューニング](#) および [MyISAM スタートアップオプション](#) を参照のこと。

ノート:この変数を `ALL` にセットするときに、別のMySQL サーバ、または `myisamchk` コマンドなどテーブルが使用中の場合は、別のプログラム内から `MyISAM` テーブルを使用しないでください。インデックス破損の原因になります。

- `--des-key-file=file_name`

`DES_ENCRYPT()` および `DES_DECRYPT()` 関数で使用するデフォルトの DES キー をこのファイルから読み取る。

- `--enable-named-pipe`

名前付きパイプ (named pipe) のサポートを有効化する。これは Windows NT、2000、XP、2003 のみで使用可能。`mysqld-nt` または名前付きパイプの接続をサポートしているサーバで使用可能。

- `--event-scheduler`

イベント スケジューラを無効、有効、開始、停止するオプション (MySQL 5.1.6 からの導入) 注意 : 許容値とその動作に関しては MySQL 5.1.11 で変更し、MySQL 5.1.12 でも再度変更している。

詳細は [event-scheduler オプション \[1078\]](#) を参照のこと。

- `--exit-info[=flags], -T [flags]`

`mysqld` サーバのデバッグに使用できる、複数の異なるフラグのビット マスク。このオプションを使用するには、完全に このオプションを理解していることが必要。

- `--external-locking`

外部ロック (システム ロック) を有効にする。MySQL 4.0 のデフォルトでは無効と設定していた。注意 : `lockd` が完全には機能しないシステム (Linux など) でこのオプションを使用すると、簡単に `mysqld` がデッドロックになる。このオプションの旧称は `--enable-locking`。

ノート:MySQL のプロセスにおいて、`MyISAM` テーブルへの更新を有効にするためにこのオプションを使用する場合、次の条件を満たす必要がある。

- 別のプロセスで更新したテーブルを使用するクエリをキャッシュしていない。
- 共有テーブルで `--delay-key-write=ALL` または `DELAY_KEY_WRITE=1` を使用していない。

この条件を定かにする方法は、常に `--external-locking` を `--delay-key-write=OFF` および `--query-cache-size=0` を併用することである。(様々なステップにおいて、前述オプションを組み合わせることが有用であることから、デフォルトでは設定をしていない。)

- `--flush`

それぞれの SQL コマンドの実行後に、ディスクへ変更のすべてをフラッシュ (同期) する。通常、MySQL はそれぞれの SQL コマンドの後に、変更だけをディスクに書き込み、ディスクへの同期処理はオペレーティング システムに任せている。詳細は [What to Do If MySQL Keeps Crashing](#) を参照のこと。

- `--general-log={0|1}`

`--log` または `-l` オプションを与えた場合に、初期のログ状態を指定する。引数がない、または 0 の場合、`--general-log` オプションがログを無効化する。引数を省略または 1 とした場合、ログを有効化する。`--log` または `-l` を指定しない場合、`--general-log` の効果はない。(MySQL 5.1.12 からの導入)

- `--init-file=file_name`

サーバ起動時にこのファイルから読み込んだ SQL コマンドを実行する。それぞれのコマンドはシングルライン (一行) で、コメントを含まないものとする。

このオプションは、MySQL のコンフィギュレーションに `--disable-grant-options` オプションを使用している場合は使えない。詳細は [典型的な configure オプション](#) を参照のこと。

- `--innodb-xxx`

InnoDB オプションは [InnoDB 起動オプションとシステム変数](#) を参照のこと。

- `--language=lang_name, -L lang_name`

クライアントのエラー メッセージに使用する言語。`lang_name` は言語名またはフルパスで指定できる (言語情報を格納するディレクトリ)。詳細は [英語以外のエラーメッセージ](#) を参照のこと。

- `--large-pages`

オペレーティング システムによっては、デフォルト (4KB) よりも大きいメモリ ページをサポートしている。サポートの実装はハードウェアや OS に依存する。大量のメモリ アクセスがあるアプリケーションの場合には、大きなメモリ ページを使用して、パフォーマンスを改善できる可能性がある。つまり トランスレーション・ルックアサイド・バッファ (TLB; Translation Lookaside Buffer) のミスが減る。

現在、MySQL は Linux 実装だけをサポートしている。Linux ではこれを HugeTLB と呼ぶ。FreeBSD や Solaris、その他の OS でのサポートは計画中である。

Linux で大きなページを使用するには、HugeTLB メモリ プールを設定する必要がある。リファレンスとして、Linux カーネル ソースの [hugetlbpage.txt](#) ファイルを参照のこと。

このオプションのデフォルトは無効と設定している。

- `--log[=file_name], -l [file_name]`

一般クエリ ログを有効化する。一般クエリ ログにはクライアントとの接続記録とクライアントから受信した SQL 文のエントリをログしている。MySQL 5.1.6 から、ログ出力先は `--log-output` オプションで選択できる。MySQL 5.1.6 以前は、ログはクエリ ログ ファイルに記録している。このファイル名を省略すると、MySQL でファイル名に `host_name.log` を使用する。詳細は [一般クエリとスロー クエリのログ出力先の選択](#) および [一般クエリ ログ](#) を参照のこと。

- `--log-bin[=base_name]`

バイナリ ログを有効化する。サーバがデータを変更するクエリをすべてバイナリ ログに記録する。これは、バックアップおよびレプリケーション用に使用できる。詳細は [バイナリ ログ](#) を参照のこと。

オプション値を指定すると、それはログ順序のベース名 (basename) になる。サーバはベース名に数値のサフィックスを与えた配列でバイナリ ログ ファイルを作成する。ベース名指定は推奨事項である ([Open Issues in MySQL](#) を参照のこと)。指定がない場合、MySQL で `host_name-bin` をベース名として扱う。

- `--log-bin-index[=file_name]`

バイナリ ログ ファイル名のインデックス ファイルを指定する。詳細は [バイナリ ログ](#) を参照のこと。ファイル名を省略する場合に、`--log-bin` で名前を指定しないときには、MySQL で、`host_name-bin.index` をファイル名として扱う。

- `--log-bin-trust-function-creators[={0|1}]`

引数なし、または 1 で、このオプションは `log_bin_trust_function_creators` システム変数を 1 と設定する。引数 0 の場合は、システム変数を 0 にセットする。`log_bin_trust_function_creators` はMySQL の格納関数 (stored function) の動作に制約を与える。詳細は [ストアドルーチンとトリガのバイナリログ](#) を参照のこと。

- `--log-error[=file_name]`

エラーと起動メッセージをファイルに記録する。詳細は [エラー ログ](#) を参照のこと。ファイル名を省略する場合、MySQL で `host_name.err` を使用する。ファイル名に拡張子がない場合は、サーバで `.err` という拡張子を加える。

- `--log-isam[=file_name]`

すべての MyISAM の変更をファイルに記録する。MyISAM をデバッグするときだけに使用する。

- `--log-long-format` (DEPRECATED)

追加情報をログ ファイルに記録する (更新ログ、バイナリ更新ログ、スロー クエリ ログなど、有効化しているログのすべて)。たとえば、クエリのユーザ名とタイム スタンプを記録する。このオプションは `--log-short-format` の説明と同じ理由で、廃止となり、デフォルト設定になった。一方で、MySQL インデックスを使用しないクエリをスロー クエリ ログに記録するための `--log-queries-not-using-indexes` オプションが利用可能になっている。

- `--log-output[=value,...]`

このオプションで、一般クエリ ログとスロー クエリ ログの出力先を決める。オプション値は `TABLE`、`FILE`、`NONE` など、一つ以上の文字 (words) で与える。このオプションに値がない場合、つまりデフォルトでは、`TABLE` で、mysql データベースの `general_log` および `slow_log` テーブルに記録する。`FILE` は、ログ ファイルに記録する。`FILE` のロギングでは、`--log` および `--slow-log` のオプションでログ ファイルの場所を決定する。`NONE` はロギングを無効にする。`NONE` がオプション値にある場合は、どの文字よりも優位になる。オプションで `TABLE` および `FILE` 両方のログ出力先を選択できる。

ノート： このオプションはログ出力先を選択するが、ログ出力には関与しない。ログ出力を有効にするには、`--log` および `--log-slow-queries` オプションを使用する。詳細は [一般クエリとスロー クエリのログ出力先の選択](#) を参照のこと。

`--log-output` オプションは MySQL 5.1.6 で導入した。

- `--log-queries-not-using-indexes`

このオプションを `--log-slow-queries` として使用する場合、インデックスを使用しないクエリはスロー クエリ ログで記録する。詳細は [スロー クエリ ログ](#) を参照のこと。

- `--log-short-format`

ログ ファイル (更新ログ、バイナリ更新ログ、スロー クエリ ログなど、有効化しているログのすべて) で記録する情報が少なくなる。たとえば、クエリのユーザ名とタイム スタンプを記録しなくなる。

- `--log-slow-admin-statements`

`OPTIMIZE TABLE`、`ANALYZE TABLE`、`ALTER TABLE` など、時間がかかる管理系 SQL 文をスロー クエリ ログに出力する。

- `--log-slow-queries[=file_name]`

スロー クエリ ログを有効化する。実行時間が `long_query_time` 秒以上かかるクエリをすべてスロー ログ ファイルとして記録する。詳細については、`--log-long-format` オプションおよび `--log-short-format` オプションの説明を参照のこと。

MySQL 5.1.6 以降でのログ出力先は `--log-output` で選択できる。MySQL 5.1.6 以前の場合は、ロギングはスロー クエリ ログ ファイルで行なう。このファイル名を省略すると、MySQL で `host_name-slow.log` をファイル名として扱う。詳細は [一般クエリとスロー クエリのログ出力先の選択](#) および [スロー クエリ ログ](#) を参照のこと。

- `--log-tc=file_name`

バイナリ ログを無効にすると、複数のストレージ エンジンに影響を与える XA トランザクションに対する、メモリマップのトランザクション コーディネータのログ ファイルの名前。デフォルト名は `tc.log`。このファイルをフルパスで指定しない場合には、データ ディレクトリ下での作成になる。現段階ではこのオプションは使用していない。

- `--log-tc-size=size`

メモリ マップのトランザクションのコーディネータのログのバイト サイズ。デフォルト サイズは 24 KB。

- `--log-warnings=[level], -W [level]`

`Aborted connection...` などの警告をエラー ログに出力する。このオプションを有効にしておく (推奨) と、レプリケーションを行う場合に、ネットワークの問題や再接続のメッセージに関する詳細を得ることができる。このオプションは、デフォルトで有効化 (1) していて、デフォルトの `level` 値を省略する場合は、1 になる。このオプションを無効化するには、`--log-warnings=0` を使用する。値を 1 より大きくすると、エラー ログで通信エラー (Aborted connections) を記録することになる。詳細は [Communication Errors and Aborted Connections](#) を参照のこと。

- `--low-priority-updates`

テーブル変更操作で、選択よりも優先順位を下げる (`INSERT`、`REPLACE`、`DELETE`、`UPDATE` など)。`{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` を使用して 1 つのクエリだけ優先順位を低くしたり、`SET LOW_PRIORITY_UPDATES=1` を使用して 1 つのスレッド内での優先順位を変更することができる。詳細は [テーブルロック関連の問題](#) を参照のこと。

- `--memlock`

メモリ内の `mysqld` プロセスをロックする。これは、使用しているシステムが `mlockall()` システム コールをサポートしている場合 (Solaris など) に使用可能。たとえば、オペレーティングシステムが原因で `mysqld` のスワップが発生するという問題がある場合、その解決に役立つ。注意: このオプションを使用するには、サーバを `root` として起動することが必要になるが、これは安全ではない。詳細は [ユーザによる MySQL の実行](#) を参照のこと。

- `--myisam-recover=[option[,option]...]`

MyISAM のストレージ エンジン のリカバリ モードをセットする。オプションは、`DEFAULT`、`BACKUP`、`FORCE`、`QUICK` の任意の組み合わせ。複数の値を指定するにはカンマで区切る。このオプションを無効にするには、これを明示的に `""` を使用する。このオプションを使用すると、`mysqld` はテーブルを開く一方で、MyISAM テーブルを開き、テーブルにクラッシュのマークが付いていないか、つまりテーブルを正しく閉じているかどうかをチェックする。(これは外部ロックを無効にした状態で稼働している場合にだけ機能する)。テーブルにクラッシュ マークが付いていた場合、`mysqld` はテーブルをチェックし、テーブルが破損していた場合は、`mysqld` で修復を試みる。

次のオプションで、修復方法を決定する。

オプション	説明
<code>DEFAULT</code>	<code>--myisam-recover</code> でどのオプションも指定しないのと同じ。
<code>BACKUP</code>	修復時にデータ テーブルを変更する場合、 <code>tbl_name.MYD</code> データファイルのバックアップを <code>tbl_name-datetime.BAK</code> として保存する。
<code>FORCE</code>	<code>.MYD</code> ファイルから複数のレコードがなくなる場合でも修復を実行する。
<code>QUICK</code>	削除ブロックがない場合、テーブル内のレコードをチェックしない。

サーバはテーブルを自動的に修復する前に、エラー ログに修復するというノートを書き込む。`BACKUP`、`FORCE` を使用すると、ユーザの介入をなくして、ほとんどすべての問題をリカバリできる。このやり方は、テーブルの修復を強制的に行うことになるため、レコードを若干削除してしまう可能性を伴うが、古いデータ ファイルがバックアップとして残るので後で調べることができる。

詳細は [MyISAM スタートアップオプション](#) を参照のこと。

- `--ndb-connectstring=connect_string`

NDB ストレージ エンジンを使用している場合、接続文字列オプションを設定して、クラスタ コンフィギュレーションを渡しているマネージメント サーバをポイントアウトできる。構文は [クラスタの 接続文字列](#) を参照のこと。

- `--ndbcluster`

NDB Cluster ストレージ エンジンへのサポートをバイナリに組み込んでいる場合、このオプションがエンジンを有効化する。デフォルト設定では無効になっている。詳細は [14章MySQL Cluster](#) を参照のこと。

- `--old-passwords`

新たなパスワードに古いパスワード形式でのハッシュ生成を強制する。(MySQL4.0 以前のパスワードを強制する。) これは、サーバが古いクライアント プログラムをサポートする必要がある場合に有用である。詳細は [MySQL 4.1 のパスワードハッシュ](#) を参照のこと。

- `--one-thread`

(Linux でのデバッグ) でスレッドを一つだけにする。サーバでのデバッグが有効化している場合にだけ使用できる。詳細は [Debugging a MySQL Server](#) を参照のこと。

- `--open-files-limit=count`

`mysqld` で使用可能なファイル記述子の数を変更する。これが設定されていない、または 0 で設定する場合、`mysqld` は、`setrlimit()` で使用している値を使用してファイル記述子をリザーブする。値が 0 の場合、`mysqld` は `max_connections*5` または `max_connections + table_open_cache*2` のどちらか大きい値をファイル数としてリザーブする。`mysqld` で `Too many open files` のエラーが出る場合、この値を大きくする。

- `--pid-file=path`

プロセス ID (PID) ファイルのパス。`mysqld_safe` など、別プログラムでサーバの PID を指定するときに使用するファイルのこと。

- `--port=port_num, -P port_num`

TCP/IP 接続時に使用するポート番号を指定する。ポート番号は 1024 以上にする。サーバが `root` システム ユーザで立ち上がっている場合はこの限りではない。

- `--port-open-timeout=num`

このオプションは TCP/IP ポートが開かない場合に、何秒待機するかを示す。システムによっては、サーバ停止後、すぐには TCP/IP ポートを使用できない。サーバを終了した直後に再起動すると、サーバは失敗する可能性のあるポートを再び開けようとする。デフォルト設定では待機しない、と指定している。(MySQL 5.1.5 からの導入)

- `--safe-mode`

いくつかの最適化ステージをスキップする。

- `--safe-show-database` (DEPRECATED)

詳細は [MySQL 提供の権限](#) を参照のこと。

- `--safe-user-create`

これを有効化した場合、ユーザに `mysql.user` テーブルまたはそのテーブル内のカラムに対する `INSERT` 権限がなければ、`GRANT` コマンドを使用して新規ユーザを作成できなくなる。

- `--secure-auth`

クライアントからの古いパスワード形式 ((MySQL4.1 より前の形式)) による認証を許可しない。

- `--shared-memory`

ローカル クライアントとの共有メモリ接続を有効化する。このオプションは Windows にだけ使用できる。

- `--shared-memory-base-name=name`

共有メモリ接続で使用する共有メモリの名前を指定します。このオプションは Windows にだけ使用できる。デフォルトの名前は `MYSQL`。この名前は大文字と小文字を区別する。

- `--skip-concurrent-insert`

`MyISAM` テーブルに対する `SELECT` 文と `INSERT` 文の同時実行を無効化する。このオプションは、この動作に関してバグの可能性あるときにだけ有効化する。詳細は [同時挿入](#) を参照のこと。

- `--skip-external-locking`

外部ロック (システム ロック) を使わないようにする。外部ロックが無効化している場合に、[myisamchk](#) を使用するときには、サーバをシャットダウンする ([システム、コンパイル時間およびスタートアップパラメータのチューニング](#) 参照)。外部ロックを回避するには。これを不要にするには、[CHECK TABLE](#) および [REPAIR TABLE](#) 文をチェックに使用して、[MyISAM](#) テーブルを修正する。

MySQL 4.0 以降、外部ロックのデフォルト設定では、無効化している。

- [--skip-grant-tables](#)

サーバが一切の権限システムを使用しないようにする。つまり、誰でも すべてのデータベースにフルアクセスできるようになる。サーバ実行中に、権限テーブルの行使を再開するには、[mysqladmin flush-privileges](#) または [mysqladmin reload](#) をシステム シェルから実行するか、またはサーバ接続後に MySQL [FLUSH PRIVILEGES](#) ステートメントを発行する。このオプションはまた、プラグインとユーザ定義関数 (UDF) のロードを抑制する。

このオプションは、MySQL のコンフィギュレーションに [--disable-grant-options](#) オプションを使用している場合は使えない。詳細は [典型的な configure オプション](#) を参照のこと。

- [--skip-host-cache](#)

name-to-IP の解決に、ホスト名のキャッシュを使用せず、接続ごとに DNS サーバに対しクエリを発行する。詳細は [MySQLの DNS の使用](#) を参照のこと。

- [--skip-innodb](#)

[InnoDB](#) ストレージ エンジンが無効にする。メモリとディスク領域の節約および演算処理のスピードアップに役立つ。[InnoDB](#) テーブルを必要とする場合は、このオプションを使用しない。

- [--skip-name-resolve](#)

(MySQLでのDNS逆引きを無効にする。) クライアント接続のチェックに、IPアドレスからホスト名を割り出す。このオプションを使用する場合には、権限テーブルの [Host](#) カラム値すべてが IP アドレスまたは [localhost](#) であることが必要。詳細は [MySQLの DNS の使用](#) を参照のこと。

- [--skip-ndbcluster](#)

[NDB Cluster](#) ストレージ エンジンが無効にする。[NDB Cluster](#) ストレージ エンジン サポートを投じたバイナリではデフォルトである。[--ndbcluster](#) オプションを明示的に与えると、サーバはこのストレージ エンジンのメモリや別リソースを配分する。使用例に関しては [MySQL Cluster の簡単なテストの設定](#) を参照のこと。

- [--skip-networking](#)

TCP ポートを一切使用しない。[mysqld](#) とのやり取りはすべて、名前付きパイプ、共有メモリ (Windows) または Unix ソケット ファイル (Unix) を介して行う必要がある。ローカルからの接続要求のみを許可しているシステムにおいては、特にこのオプションを使用する (推奨)。詳細は [MySQLの DNS の使用](#) を参照のこと。

- [--ssl*](#)

[--ssl](#) で始まるオプションは、SSL経由での接続をクライアントに許可するかどうかを指定し、SSL キーと証明書 がどこにあるかを示す。詳細は [SSL コマンド オプション](#) を参照のこと。

- [--standalone](#)

Windows NT システム専用。MySQL サーバをサービスとして起動せずにスタンドアロンで起動する。

- [--symbolic-links](#), [--skip-symbolic-links](#)

シンボリック リnkのサポートを有効化または無効化する。このオプションは、Windows と Unix では効果が異なる。

- Windows で有効化した場合は、[db_name.sym](#) ファイルを作成してデータベース ディレクトリにシンボリック リnkを設置できる。このファイルには、実ディレクトリへのパスが入る。詳細は [上のデータベースに対するシンボリックリンクの使用](#) を参照のこと。

- Unix で有効にした場合は、[MyISAM](#) インデックス ファイル、またはデータ ファイルを別のディレクトリとリンクできる。これには、[CREATE TABLE](#) 文の [INDEX DIRECTORY](#) または [DATA DIRECTORY](#) などのオプションを使用する。詳細は [Unix 上のテーブルに対するシンボリックリンクの使用](#) を参照のこと。
- [--skip-safemalloc](#)
MySQL を [--with-debug=full](#) で設定する場合、すべてのプログラムが、メモリの割り当て時と解放時に必ずメモリのオーバーランをチェックする。このチェックには時間がかかるため、このチェックが不要のサーバに対しては、[--skip-safemalloc](#) オプションを使用して、チェックをスキップできる。
- [--skip-show-database](#)
ユーザが [SHOW DATABASES](#) 権限を持っていない場合に、[SHOW DATABASES](#) コマンドを無効化する。ステートメントはすべてのデータベース名を表示する。このオプションをセットしない場合、[SHOW DATABASES](#) はすべてのユーザが利用できるようになるが、ユーザが [SHOW DATABASES](#) 権限、またはそのデータベースに関する何らかの権限を持っているときには、それぞれのデータベース名だけを表示する。注意：すべてのグローバル権限がデータベースに対する権限になる。
- [--skip-stack-trace](#)
スタックトレースを書き込まないようにする。このオプションは、デバッガで [mysqld](#) を実行するときに役立つ。システムによっては、コア ファイルを取得するためにこのオプションの使用が必要な場合がある。詳細は [Debugging a MySQL Server](#) を参照のこと。
- [--skip-thread-priority](#)
応答時間を短くするため、スレッド優先度の使用を無効化する。
- [--slow-query-log={0|1}](#)
[--log-slow-queries](#) オプションで、イニシャルのログ状態を指定する。引数がない、または 0 の場合、[--slow-query-log](#) オプションがログを無効化する。省略または 1 を引数とした場合、このオプションはログを有効化する。[--log](#) または [-l](#) を指定しない場合、[--slow-query-log](#) の効果はない。(MySQL 5.1.12 での導入)
- [--socket=path](#)
Unix では、ローカル接続に使用するソケットファイルを指定する。デフォルトは [/tmp/mysql.sock](#)。Windows では、名前付きパイプのローカル接続用パイプ名を指定する。デフォルトは [MySQL](#) (大小文字の区別なし)。
- [--sql-mode=value\[,value\[,value...\]\]](#)
SQL モードを設定する。詳細は [SQL モード](#) を参照のこと。
- [--sysdate-is-now](#)
デフォルトの [SYSDATE\(\)](#) が実行タイムを返す。これは実行を開始するステートメントのタイムではない。[NOW\(\)](#) の動作によって異なる。このオプションを使用すると、[SYSDATE\(\)](#) が [NOW\(\)](#) のエイリアスになる。バイナリロギングやレプリケーションの実装に関しては、[日付時刻関数](#) で [SYSDATE\(\)](#) 用の説明と、[SET 構文](#) で [SET TIMESTAMP](#) 用の説明を参照のこと。
(MySQL 5.1.8 での追加)
- [--tc-heuristic-recover={COMMIT|ROLLBACK}](#)
ヒューリスティックのリカバリ プロセスで使用する決定型 (発見的方法)。現段階では、このオプションは未使用である。
- [--temp-pool](#)
このオプションを使用すると、サーバが作成する大部分のテンポラリ ファイルに、それぞれ一意の名前ではなく、小さな名前のセットを使用する。これで、名前が異なる新規作成ファイルが多い場合に、それを Linux カーネルが処理するときの問題を回避する。Linux では、メモリがディスク キャッシュではなく、ディレクトリ エントリ キャッシュへの割り当てになるため、従来の動作ではメモリの「リーク」が発生しやすい。

- `--transaction-isolation=level`

デフォルトのトランザクション隔離レベルを指定する。`level` 値は、[READ-UNCOMMITTED](#)、[READ-COMMITTED](#)、[REPEATABLE-READ](#)、[SERIALIZABLE](#) などになり得る。詳細は [SET TRANSACTION 構文](#) を参照のこと。

- `--tmpdir=path, -t path`

テンポラリ ファイル作成に使用するディレクトリのパス。テンポラリ ファイルを保存するには小さすぎるパーティション上にデフォルトの `/tmp` ディレクトリがある場合、このオプションが役に立つ。このオプションではラウンドロビン方式のパスを使用できる。Unix ではコロン (':') を、Windows、NetWare、OS/2 ではセミコロン ';' でパスを区切る。MySQL サーバがレプリケーション スレーブとして機能している場合は、メモリベースのファイルシステムのディレクトリや、サーバ ホストのリポートでクリアするディレクトリを指すときには、`--tmpdir` を設定しない。テンポラリ ファイルのストレージ位置に関しては、[Where MySQL Stores Temporary Files](#) を参照のこと。スレーブは、マシンがリポートしてもテンポラリ テーブルのレプリケーション、または [LOAD DATA INFILE](#) のレプリケーション処理を続行するためにテンポラリファイルが必要とする。サーバの再起動時に、テンポラリ ファイル ディレクトリのファイルが消えた場合、レプリケーションは失敗に終わる。

- `--user={user_name|user_id}, -u {user_name|user_id}`

`mysqld` サーバを、`user_name` (名前)、または `user_id` (数字のユーザ ID) を保持するユーザで実行する。ここで「ユーザ」は、権限テーブルにリストしている MySQL ユーザではなく、システム ログイン アカウントのこと。

`mysqld` を `root` アカウントで起動する場合には、必須 のオプションである。起動シーケンス中にサーバがそのユーザ ID を変更し、`root` ではなく、特定のユーザで実行するようになる。詳細は [セキュリティ ガイドライン](#) を参照のこと。

セキュリティ ホールを回避するため、つまりユーザが `--user=root` オプションを `my.cnf` ファイルに追加することが原因で、その結果、サーバが `root` として稼働できないようにするために、`mysqld` で最初に指定した `--user` オプションだけを使用し、複数の `--user` オプションがあった場合に警告を生成する。`/etc/my.cnf` および `$MYSQL_HOME/my.cnf` 内のオプションは、コマンドラインのオプションより先に処理することになるため、`--user` オプションを `/etc/my.cnf` に含めた上で、`root` 以外の値を指定する (推奨)。`/etc/my.cnf` 内のオプションが他の `--user` オプションよりも先に検出することになるので、サーバは確実に `root` 以外のユーザが実行することになり、別の `--user` オプションを検地すると警告を出す。

- `--version, -V`

バージョン情報を表示して、終了する。

`--var_name=value` という配列のオプションを使用して、サーバのシステム変数に値を割り当てることができます。たとえば、`--key_buffer_size=32M` は `key_buffer_size` の変数を 32 MB という値に設定できます。

注意：変数として値を割り当てるときには、MySQL が自動的に範囲内に留まろうとして、値を修正する可能性があります。あるいは、特定の値を許可している場合には、許容値との近似値に調整しようとする可能性もあります。

`SET` のランタイムに設定できる変数の最大値を制限する場合には、`--maximum-var_name=value` をコマンドラインのオプションを使用して、定義できます。

または、`--set-variable=var_name=value` あるいは `-O var_name=value` のシンタックスを使用して、変数を設定することも可能です。このシンタックスは廃止予定です。

`SET` コマンドを使用して、実行中のサーバに対して対部分のシステム変数の値を変更できます。詳細は [SET 構文](#) を参照してください。

すべての変数の詳細に加え、サーバのスタートアップやランタイムで設定する方法に関する追加情報は、[システム変数](#) を参照してください。システム変数を調整したサーバの最適化に関する情報は[サーバパラメータのチューニング](#)を参照してください。

システム変数

`mysql` サーバは、様々なシステム変数を保有しています。そしてその変数は、設定がどのようになっているかを示します。それぞれのシステム変数にはデフォルト値がありますが、サーバ起動時に、コマンドラインまたはオプションファイルなどを使用して変更できます。大抵の場合、`SET` コマンドを使用して実行中のサーバで動的に変更できま

す。つまり、サーバを停止または再起動などで操作を中断しなくても変更することが可能であるということです。システム変数の値は、プログラミング式で指定します。

システム変数の名前や値を確認する方法 (コマンド)

- サーバで使用しているコンパイルのデフォルト値や、読み込むオプション ファイルの所在を確認するコマンド

```
mysql> --verbose --help
```

- サーバが元にするコンパイルのデフォルト値や、オプション ファイルの設定がどうなっているか (無視するかどうか) を確認するコマンド

```
mysql> --no-defaults --verbose --help
```

- 実行中のサーバでカレント値を確認するには、[SHOW VARIABLES](#) ステートメントを使用すること。

このセクションでは、それぞれのシステム変数について説明します。バージョン情報を記載していない変数は、MySQL 5.1 リリースで導入した変数です。実装/導入履歴に関する情報は、MySQL 5.0 Reference Manual または MySQL 3.23, 4.0, 4.1 リファレンス マニュアル をそれぞれ参照してください。

その他のシステム変数に関する詳細は、次のセクションを参照してください。

- [システム変数の使用](#) では、システム変数値に関する構文規則と表示方法について説明します。
- [動的システム変数](#) では、ランタイムで設定できる変数をリストしています。
- システム変数の調整に関する情報は [サーバパラメータのチューニング](#) を参照してください。
- [InnoDB 起動オプションとシステム変数](#) では、InnoDB システム変数をリストしています。

ノート:次に示す変数説明は、変数を「可能にする (有効化)」または「不可能にする (無効化)」ということに言及しています。これらの変数は [SET](#) コマンドを [ON](#) または [1](#) に設定すると実行可能であることを示し、あるいは [OFF](#) または [0](#) に設定すると実行不可能であることを示します。コマンドラインまたはオプション ファイルで変数を設定するには、[1](#) または [0](#) のいずれかを使用してください。[ON](#) または [OFF](#) を使用すると機能しません。たとえば、コマンドラインの場合に、[--delay_key_write=1](#) のオプションは動作しますが、[--delay_key_write=ON](#) では動作しません。

バッファ サイズ、長さ、スタック サイズなどの値は、指定がない限り、バイトで与えます。

- [auto_increment_increment](#)

[auto_increment_increment](#) と [auto_increment_offset](#) はマスタからマスタへのレプリケーションに使用する。 [AUTO_INCREMENT](#) カラムの操作の制御に使用できる。この変数には、グローバルまたはローカルで設定でき、それぞれで 1 から 65,535 までの整数を使用できる。これら 2 種類の変数のどちらかを 0 に設定すると、1 での設定したものとの解釈になる。65,535 より大きな整数、あるいは 0 ではない数字でこれらの変数のどちらかを設定すると、65,535 で設定したものとの解釈になる。 [auto_increment_increment](#) または [auto_increment_offset](#) に、整数以外の値を使用すると、反映できずエラーになり、変数の実際の値 (デフォルト) のままになる。

重要 [auto_increment_increment](#) と [auto_increment_offset](#) は MySQL Cluster レプリケーションには使用しないでください。MySQL Cluster レプリケーションでこの 2 つの変数を使用すると、予期せぬエラーまたは回復不能な状態にある可能性があります。そのため、Cluster レプリケーションでの この 2 つの変数においては、サポートしていません。

これら 2 つの変数は [AUTO_INCREMENT](#) カラムの動作に次のように影響する。

- [auto_increment_increment](#) は自動インクリメントの間隔値を制御する。次はその例示である。

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+----+
| col |
+----+
| 1 |
| 11 |
| 21 |
| 31 |
+----+
4 rows in set (0.00 sec)
```

注意：ここでは `SHOW VARIABLES` を変数値 (カレント値) を取得する目的で使用しています。

- `auto_increment_offset` は `AUTO_INCREMENT` カラム値の開始点を決定する。次の例示は、`auto_increment_increment` 記述の例で、ステートメントを同一のセッション中に実行した場合である。

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc2;
+----+
| col |
+----+
| 5 |
| 15 |
| 25 |
| 35 |
+----+
4 rows in set (0.02 sec)
```

`auto_increment_offset` の値が、`auto_increment_increment` の値よりも大きい場合、`auto_increment_offset` の値は無効になる。

ここで、これら変数のどちらか、あるいは両方が変更して、新規の行をテーブルの `AUTO_INCREMENT` カラムに挿入していなければならない。結果は直感的のようにも思えるが、これは既にカラムに存在している値を無視して、`AUTO_INCREMENT` 値で計算している。つまり、挿入した値が `AUTO_INCREMENT` カラムの最大値よりも大きいシリーズの最小値であることが原因である。つまり、シリーズが次のように計算したことに起因する。

$\text{auto\_increment\_offset} + N \times \text{auto\_increment\_increment}$

N が [1, 2, 3, ...] のシリーズの正の整数。たとえば次の例示の通り。

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset   | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+----+
| col |
+----+
| 1 |
| 11 |
| 21 |
| 31 |
+----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+----+
| col |
+----+
| 1 |
| 11 |
| 21 |
| 31 |
| 35 |
| 45 |
| 55 |
| 65 |
+----+
8 rows in set (0.00 sec)
```

`auto_increment_increment` と `auto_increment_offset` で示している値は、 $5 + N \times 10$ のシリーズを生成する。つまり、[5, 15, 25, 35, 45, ...] である。`INSERT` 前の `col` カラムの最大値は 31 で、`AUTO_INCREMENT` シリーズで次に利用可能な値は 35。そして `col` の挿入値はその時点から始まり、その結果が `SELECT` クエリが表示になる。

ここで重要なことは、1つのテーブルにこれら2つの変数をした場合の効果を制限することはできないということである。ゆえに、別のデータベース管理システムで提供しているシーケンスとは併用できない。この変数は、MySQL サーバのすべてのテーブルにある `AUTO_INCREMENT` カラムのすべての動作を制御する。この変数のどちらかをグローバルでセットした場合、グローバル値を変更するか、またはローカルでこれらを設定して書き直すまで、あるいは `mysqld` が再起動するまで、その効果が続くことになる。ローカルで設定した場合、新しい値はすべてのテーブルの `AUTO_INCREMENT` カラムに影響し、そのセッションのユーザが新たな行を挿入することになる。つまり値がセッション中に変更することになる。

`auto_increment_increment` のデフォルト値は 1。詳細は [Auto-Increment in Multiple-Master Replication](#) を参照のこと。

- `auto_increment_offset`

この変数のデフォルト値は 1。`auto_increment_increment` での記述も参照のこと。

- `automatic_sp_privileges`

この変数が 1 (デフォルト) の場合、サーバが自動的に `EXECUTE` および `ALTER ROUTINE` の権限をストアードルーチンのクリエータに与える。(`ALTER ROUTINE` 権限で、ルーチンをドロップの対象にする。) つまりサーバが、クリエータのルーチンをドロップするときに、自動的にこの権限もドロップする。`automatic_sp_privileges` が 0 の場合、サーバは自動的にドロップしない。権限もドロップの対象にはならない。

- [back_log](#)

MySQL で保持できる未解決の接続要求数。これは、短時間にメインの MySQL スレッドに多くの接続要求が集中したときに機能する (役立つ)。そして、メイン スレッドが接続をチェックするため、新規スレッドの開始には時間 (若干) がかかる。つまり [back_log](#) 値は、MySQL が一時的に新規要求への回答を停止するまでの瞬時に、スタック可能な要求の数を示す。短時間に多くの接続要求数がある場合にだけ、この値を大きくする。

つまり、この値は、TCP/IP 接続のリッスン キューのサイズである。使用しているオペレーティング システムのものにも、このキュー サイズの制限がある。詳細については、Unix [listen\(\)](#) システムコールのマニュアルページを参照のこと。この変数の最大値については、OS のドキュメントを参照する。[back_log](#) を、オペレーティングシステムの制限値より大きくしても、効果はない。

- [basedir](#)

基準パスを指定する。MySQL をインストールしたディレクトリを指す。[--basedir](#) オプションで起動時に設定できる。

- [binlog_cache_size](#)

トランザクション中にバイナリ ログに対する SQL ステートメントを保持するキャッシュのサイズ。(トランザクション間でメモリに保持する SQL 文の最大数。) サーバがトランザクションのストレージ エンジンをサポートする場合、あるいはサーバで [--log-bin](#) オプションを使用して、バイナリ ログを可能にしている場合、バイナリ ログ キャッシュをクライアントに割り当てる。大きな複数ステートメントのトランザクションが頻繁にある場合、この値を大きくすると、パフォーマンスを向上できる。[Binlog_cache_use](#) と [Binlog_cache_disk_use](#) のステータス変数はこの変数のサイズ調整に便利である。詳細は [バイナリ ログ](#) を参照のこと。

- [binlog_format](#)

バイナリ ロギング形式。[STATEMENT](#)、[ROW](#)、[MIXED](#) のどれかになる。[binlog_format](#) は 起動時に [--binlog-format](#) オプションで設定する。または、ランタイムで [binlog_format](#) 変数でも設定できるが、グローバル スcopeでこの変数を設定するには、[SUPER](#) 権限が必要になる ([レプリケーション フォーマット](#) を参照のこと)。(スタートアップ変数の導入: MySQL 5.1.5、ランタイム変数の導入: MySQL 5.1.8、[MIXED](#) の導入: MySQL 5.1.8)

デフォルトでは [STATEMENT](#) を使用。[MIXED](#) を指定して、ステートメント ベースのレプリケーションにすることもできる。ただし、行ベースのレプリケーションで正確な結果を保証する場合 (その必要がある場合)、たとえば、ユーザ定義関数 (UDF)、または [UUID\(\)](#) 関数を含むステートメントの場合には、この限りではない (別の事情がある)。格納ルーチン (stored functions) やトリガなどに、[MIXED](#) を指定して、ステートメント ベースのレプリケーションを行う場合も例外である。

ランタイムでレプリケーションの仕方を切り替えることができない場合

- 格納関数またはトリガ内からの場合
 - [NDB](#) を有効化している場合
 - セッションのレプリケーション モードが行ベースで、テンポラリ テーブルを開けている場合
- この 3 つのどれかに該当する場合に、レプリケーションの仕方を変更すると、エラーになる。

MySQL 5.1.8 前は、行ベース レプリケーションの仕方を変更することは [--log-bin-trust-function-creators=1](#) および [--innodb_locks_unsafe_for_binlog](#) を明示的に設定するとしていた。しかし、MySQL 5.1.8 以降 (MySQL 5.1.8 を含む) は、行ベースのレプリケーションにこのオプションで明示設定しても、通用しない。

- [bulk_insert_buffer_size](#)

空白ではないテーブルにデータを追加する場合に、[MyISAM](#) は特殊なツリー状のキャッシュを使用して、バルクの [INSERT ... SELECT](#)、[INSERT ... VALUES \(...\), \(...\), ...](#)、[LOAD DATA INFILE](#) を高速化する。この変数で、スレッドごとのキャッシュ ツリーのサイズを制限する (バイト単位)。この値を 0 に設定すると、最適化は無効になる。デフォルト値は 8 MB。

- [character_set_client](#)

クライアント送信の文字列のキャラクタ セット (クライアントが送信するキャラクタ セット)。

- [character_set_connection](#)

キャラクタ セット情報が無いリテラルの並びで、数値→文字と変換するときのキャラクタ セット。

- `character_set_database`

デフォルト データベースで使用するキャラクタ セット。デフォルト データベースが変化する度に、サーバがこの変数を変更する。デフォルト データベースが存在しない場合、この値は `character_set_server` と同一。

- `character_set_filesystem`

ファイルシステムのキャラクタ セット。`LOAD DATA INFILE` や `SELECT ... INTO OUTFILE` などのステートメントや `LOAD_FILE()` 関数に対して、この変数でファイル名とリテラルの文字列を読み取る。ファイルを開けようとすると、ファイル名が `character_set_client` から `character_set_filesystem` に変わる。デフォルト値は `binary` (変換が起らない) である。マルチ バイトのファイル名を利用できるシステムでは、異なる値を使用することが好ましい。たとえば、UTF-8 でファイル名を表示しているシステムの場合は、`character_set_filesystem` を `'utf8'` にセットする。(MySQL 5.1.6 実装)

- `character_set_results`

クライアントへ返す文字列 (クエリ結果) のキャラクタ セット。

- `character_set_server`

サーバのデフォルトのキャラクタ セット。

- `character_set_system`

識別子の書き出しにサーバが使用するキャラクタ セット。この値は常に `utf8`。

- `character_sets_dir`

キャラクタ セットを格納しているディレクトリ。

- `collation_connection`

接続キャラクタ セットの照合順序。

- `collation_database`

デフォルト データベースの照合順序。デフォルト データベースが変化する度に、サーバがこの変数を変更する。デフォルトデータベースが存在しない場合、この値は `collation_server` と同一。

- `collation_server`

サーバのデフォルトの照合順序

- `completion_type`

トランザクションのコンプリーション タイプ

- 値が 0 (デフォルト) の場合、`COMMIT` および `ROLLBACK` には影響しない。
- 値が 1 の場合、`COMMIT` は `COMMIT AND CHAIN` に、`ROLLBACK` は `ROLLBACK AND CHAIN` に相当する。(新たなトランザクションが終了したばかりのトランザクションと同一の分離レベルで始まる。)
- 値が 2 の場合、`COMMIT` は `COMMIT RELEASE` に、`ROLLBACK` は `ROLLBACK RELEASE` に相当する。(サーバはトランザクションを終えると切断する。)

- `concurrent_insert`

`ON` (デフォルト) の場合、`INSERT` および `SELECT` ステートメントを `MyISAM` テーブルで同時に実行できる (間にフリー ブロックがない場合)。このオプションを使用しないように設定するには、`mysqld` を `--safe` または `--skip-new` で起動する。

この変数では次の整数値を使う。

値	説明
0	オフ
1	(デフォルト) ホールがない MyISAM テーブルに同時挿入
2	すべての MyISAM テーブルに同時挿入。テーブルにホールがあり、別のスレッドで使用している場合、新規の行はテーブル末尾への挿入となる。テーブルが使用中でなければ、MySQL が通常の読み込みロックを行い、新規の行をホールへ挿入する。

[同時挿入](#) を参照のこと。

- [connect_timeout](#)

mysqld サーバが、**Bad handshake** を返すまで、接続パケットを待つ秒数

- [datadir](#)

MySQL をインストールしたディレクトリ。この変数は **--basedir** オプションで設定できる。

- [date_format](#)

この変数は実装していない。

- [datetime_format](#)

この変数は実装していない。

- [default_week_format](#)

デフォルト モード値。**WEEK()** 関数に使う。詳細は [日付時刻関数](#) を参照のこと。

- [delay_key_write](#)

MyISAM テーブルにだけ使えるオプション。この値は、**CREATE TABLE** ステートメントに使用するとき、**DELAY_KEY_WRITE** テーブル オプションの処理に影響する。次の表で値の詳細を参照のこと。

オプション	説明
OFF	DELAY_KEY_WRITE は無視。
ON	MySQL は CREATE TABLE ステートメント指定の DELAY_KEY_WRITE オプションを優先する。(デフォルト)
ALL	新規の開テーブルすべてを DELAY_KEY_WRITE オプションを実行可能にして作成したかのように処理する。

DELAY_KEY_WRITE をテーブルに対して可能にした場合、インデックス更新でキー バッファのフラッシュではなく、テーブルが閉じたときにだけフラッシュする。これを利用すると、キーの書き込みスピードを速めることが可能である。ただし、これを使用する場合には、**--myisam-recover** オプションでサーバを立ち上げて、すべての **MyISAM** テーブルの自動チェックを設定しておく必要がある (例: **--myisam-recover=BACKUP,FORCE**)。詳細は [コマンド オプション](#) および [MyISAM スタートアップオプション](#) を参照のこと。

ノート: **--external-locking** で外部ロックを有効化した場合、遅延キー書き込み (delayed key write) があるテーブルのインデックス破損に対するプロテクションがなくなる。

- [delayed_insert_limit](#)

delayed_insert_limit レコードを挿入後、**INSERT DELAYED** ハンドラは、保留中の **SELECT** ステートメントがあるかどうかチェックする。ステートメントがある場合、ハンドラは処理を続行する前に保留中のステートメントの実行を許可する。

- [delayed_insert_timeout](#)

INSERT DELAYED ハンドラ スレッドが **INSERT** ステートメントを待機する時間。

- [delayed_queue_size](#)

[INSERT DELAYED](#) を処理時のテーブルあたりのキューの最大値 (レコード単位)。キューが最大値に達すると、[INSERT DELAYED](#) を実行するすべてのクライアントは、キューに空きができるまで待機する。

- [div_precision_increment](#)

この変数は、小数点以下の桁数を示す。DIV 演算子 (/) での演算結果を高める。デフォルト値は 4。最小値は 0、最大値は 30。次の例はデフォルト値を高めた場合の効果を示す。

```
mysql> SELECT 1/7;
+-----+
| 1/7 |
+-----+
| 0.1429 |
+-----+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+-----+
| 1/7 |
+-----+
| 0.142857142857 |
+-----+
```

- [event_scheduler](#)

この変数は Event Scheduler (イベント スケジューラ) のステータス (状態) を示す。MySQL 5.1.12 から予定している値は、[ON](#)、[OFF](#)、[DISABLED](#)。[OFF](#) をデフォルトとする。Event Scheduler 操作における、この変数とその効果は、[リンク \[1078\]](#) (Events 関連の概略) を参照のこと。

(MySQL 5.1.6 での追加)

- [engine_condition_pushdown](#)

この変数は、NDB に適用する。デフォルトでは 0 ([OFF](#))。SELECT * FROM t WHERE mycol = 42 のように mycol でインデックス化していないカラムのクエリを実行する場合、そのクエリは NDB ノード毎にフル テーブル スキャンの対象になる。それぞれのノードで SQL サーバにすべての行 (レコード) を送り、WHERE 条件なる [engine_condition_pushdown](#) を 1 ([ON](#)) にセットすると、条件はストレージ エンジンまで「押し下げられ」、NDB ノードにまで届く。それぞれのノードでスキャンを行う条件を持っているため、MySQL サーバにその条件に適合する行を送り返す。

- [expire_logs_days](#)

バイナリ ログの自動削除の日数を指定する。デフォルトは 0 で「自動削除しない」ことを意味する。MySQL サーバ起動時もしくは ログをローテートするときが、ログを削除するタイミングである。

- [flush](#)

[ON](#) の場合、SQL ステートメントの後で、サーバがすべての変更をデスクにフラッシュ (同期) する。通常、MySQL は、SQL ステートメントで、すべての変更をデスクに書き込むことはなく、オペレーティング システムにディスクとの同期を任せる。[What to Do If MySQL Keeps Crashing](#) を参照のこと。--flush オプションで mysqld を立ち上げる場合には、この変数を [ON](#) に設定する。

- [flush_time](#)

これを 0 以外の値に設定する場合、リソースの解放と未フラッシュ データのディスクへ同期するために、[flush_time](#) 秒ごとにすべてのテーブルを閉じる。Windows 9x、Me など別リソースに限りがあるシステムの場合にだけ、このオプションを使用する (推奨)。

- [ft_boolean_syntax](#)

[IN BOOLEAN MODE](#) を使用しているブーリアン全文検索をサポートする演算子一覧。詳細は [ブール全文検索](#) を参照のこと。

デフォルトの変数値は '+ -><()~*:"'&|' である。この値を変更するときのルールは次の通り。

- 演算子の関数を文字列中の位置で決定している。
- 置換値は 14 文字である。
- 文字に ASCII の非英数字を使用している。
- 最初 (先頭) または2番目の文字がスペース (空白文字) である。
- 重複する文字を使用していない。(11 または 12 番目にクオート文字の重複は可。この2文字は同一である必要はないが、同一でなければならないこともある。)
- 10、13、14番目の文字である。(デフォルトでは `'`、`&`、`|`) はリザーブである。
- `ft_max_word_len`
FULLTEXT インデックスの単語 (word) の最大文字数
 ノート:**FULLTEXT** インデックスは、変数を変更すると、再ビルドしなければならない。**REPAIR TABLE tbl_name QUICK** を使用のこと。
- `ft_min_word_len`
FULLTEXT インデックスの単語の最小文字数
 ノート:**FULLTEXT** インデックスは、変数を変更すると、再ビルドしなければならない。**REPAIR TABLE tbl_name QUICK** を使用のこと。
- `ft_query_expansion_limit`
WITH QUERY EXPANSION を使用した全文検索の最上位マッチ数。
- `ft_stopword_file`
 全文検索のストップワード リストのファイル。ファイル内のすべてのストップワードが使用対象になるが、コメントは使用しない。デフォルトは、ストップワードの組み込みリストを使用する (`storage/myisam/ft_static.c` ファイルの定義)。このパラメータを空白の文字列 (`"`) に設定すると、ストップワードのフィルタリングを無効化する。
 ノート:変数またはストップワードの内容を変更すると、**FULLTEXT** インデックスを再ビルドしなければならない。**REPAIR TABLE tbl_name QUICK** を使用のこと。
- `general_log`
 一般クエリ ログを有効化しているかどうかを示す。値が 0 (または **OFF**) の場合、ログしない。値が 1 (または **ON**) の場合、ログする。デフォルトは `--log` オプションを設定しているかどうかによる。ログ出力先は `log_output` システム変数で制御する。値を **NONE** にした場合、ログできるようにしていても、エントリを書き込まない。(`general_log` は MySQL 5.1.12 での導入)
- `general_log_file`
 一般クエリ ログ ファイルの名前。デフォルトは `host_name.log`。初期値は `--log` オプションで変更可能。(MySQL 5.1.12 から導入)
- `group_concat_max_len`
GROUP_CONCAT() 関数の最大許容結果長さ (返却値の最大文字数) デフォルトは 1024。
- `have_archive`
YES : `mysqld` で **ARCHIVE** テーブルをサポートする場合。**NO** : そうでない場合。
- `have_blackhole_engine`
YES : `mysqld` で **BLACKHOLE** テーブルをサポートしている場合。**NO** : そうでない場合。
- `have_compress`

YES : `zlib` 圧縮ライブラリをサーバで利用できる場合。NO : そうでない場合 (このときは、`COMPRESS()` および `UNCOMPRESS()` 関数は使用できない)。

- `have_crypt`

YES : `crypt()` システム コールをサーバで利用する場合。NO : そうでない場合 (このときは、`ENCRYPT()` 関数は使用できない)。

- `have_csv`

YES : `mysqld` で `ARCHIVE` テーブルをサポートする場合。NO : そうでない場合。

- `have_dynamic_loading`

YES : `mysqld` で動的ロードのプラグインをサポートする場合。NO : そうでない場合。(MySQL 5.1.10 から導入)

- `have_example_engine`

YES : `mysqld` で `EXAMPLE` テーブルをサポートしている場合。NO : そうでない場合。

`have_federated_engine`

YES : `mysqld` で `FEDERATED` テーブルをサポートする場合。NO : そうでない場合。

- `have_geometry`

YES : サーバで空間データ型 (spatial) をサポートしている場合。NO : そうでない場合

- `have_innodb`

YES : `mysqld` で `InnoDB` テーブルをサポートしている場合。DISABLED : `--skip-innodb` を使用している場合。

- `have_ndbcluster`

YES : `mysqld` で `NDB Cluster` テーブルをサポートしている場合。DISABLED : `--skip-ndbcluster` を使用している場合。

- `have_partitioning`

YES : `mysqld` でパーティショニング (領域確保) をサポートしている場合。(MySQL 5.1.1 から導入。MySQL 5.1.6 では、`have_partition_engine` から `have_partitioning` になった (改名)。

- `have_openssl`

YES : `mysqld` で SSL 接続をサポートする場合。NO : そうでない場合。

- `have_query_cache`

YES : `mysqld` で クエリ キャッシュをサポートする場合。NO : そうでない場合。

- `have_row_based_replication`

YES : サーバで行ベースのバイナリ ロギングのレプリケーションを実行できる場合。NO : サーバでステートメントベースのロギングを行う。詳細は [レプリケーション フォーマット](#) を参照のこと。(MySQL 5.1.5 から導入したが、MySQL 5.1.15 で削除している。)

- `have_rtree_keys`

YES : `RTREE` インデックスを利用できる場合。NO : そうでない場合。(MyISAM テーブルの空間インデックスで使用している。)

- `have_symlink`

YES : シンボリック リンク サポートを有効化している場合。NO : そうでない場合。Unix では `DATA DIRECTORY` および `INDEX DIRECTORY` のテーブル オプションを必要とする。Windows では、データ ディレクトリの `symlink` 関数を必要とする。

- [init_connect](#)

クライアント接続でサーバが実行する文字列。この文字列は 1 つ以上の SQL ステートメントから成る。複数のステートメントを指定するには、セミコロンで文字を区切る。たとえば、クライアントで自動コミット (autocommit) モードを有効にしていた場合に、クライアントでデフォルトとして開始する。自動コミットをデフォルトで無効にするグローバル変数は存在しないが、[init_connect](#) を使用すると、同様の効果を期待できる (複数のステートメントを指定する)。

```
SET GLOBAL init_connect='SET AUTOCOMMIT=0';
```

この変数は、コマンドラインまたはオプション ファイルで設定できる。オプション ファイルで変数を設定するには、次のラインを使用する。

```
[mysqld]
init_connect='SET AUTOCOMMIT=0'
```

ノート : [init_connect](#) の内容は [SUPER](#) 権限のあるユーザには通用しない。つまり、[init_connect](#) の誤値がクライアント接続を阻止しないようにしている。たとえば、シンタックス エラーを保持する値がステートメントにあると、クライアント接続に支障をきたす。[SUPER](#) 権限を持つユーザに対して [init_connect](#) を実行しないということは、ユーザとの接続と [init_connect](#) の値に対して害を与えないということである。

- [init_file](#)

サーバ起動時に、[--init-file](#) オプションで指定するファイルの名前。このファイルに起動時に実行する (したい) SQL ステートメントを組み込む。それぞれのステートメントを一行命令文として、コメントは入れないこと。

ノート : [--init-file](#) オプションは、MySQL を [--disable-grant-options](#) オプションでコンフィギュレーションしている場合には利用不可。詳細は [典型的な configure オプション](#) を参照のこと。

- [init_slave](#)

[init_connect](#) に類似する。SQL スレッドを開始するときにスレーブ サーバが実行する文字列。この文字列の形式は [init_connect](#) と同一である。

- [innodb_xxx](#)

InnoDB システム変数は [InnoDB 起動オプションとシステム変数](#) を参照のこと。

- [interactive_timeout](#)

対話式接続を終了する前に、サーバがアクティビティを待機する秒数。対話型クライアントの定義は、[mysql_real_connect\(\)](#) で [CLIENT_INTERACTIVE](#) オプションを使用するクライアントのことである。[wait_timeout](#) も参照のこと。

- [join_buffer_size](#)

完全結合 (インデックスを使用しない結合) に使用するバッファのサイズ。これにより、フル テーブル スキャンを実行できる。一般的には、結合を高速化する最良の方法は、インデックスを追加することである。しかし、インデックスを追加できない場合に、[join_buffer_size](#) の値を大きくすると結合が速くなる (完全結合になる)。つまり、2 つのテーブル間の完全結合ごとにバッファを割り当てる。テーブル間の結合が複雑な場合は、複合バッファを必要とすることもある。

- [key_buffer_size](#)

MyISAM テーブルのインデックス ブロックをバッファし、すべてのスレッドで共有。[key_buffer_size](#) は、インデックス ブロックに使用するバッファのサイズである。キー バッファはキー キャッシュのこと。

[key_buffer_size](#) の最大値は 4 GB。物理 RAM、RAM 制限、オペレーティング システム、プラットフォームの状態などによるが、効果的な設定値としては、4 GB を下回る程度が良い。

環境に応じて、インデックス処理 (すべての読み込みと複数の書き込み) を改善する目的で、この値を大きくできる。一般的には、マシンのメモリ使用率 25 % の値であることが好ましい。使用率の 50 % にすると、値が大き過ぎるため、システム処理が極端に遅くなる。MySQL のデータ読み込みのファイルシステムのキャッシュは OS に依存

しているため、ファイル キャッシュ用にスペースに余裕を持たせることが必要である。別のストレージ エンジンの必要メモリに関しても検討のこと。

同時書き込みが多い場合などに、スピードを上げるには、[LOCK TABLES](#) を使用する。詳細は [INSERTステートメントの速度](#) を参照のこと。

キー バッファのパフォーマンスをチェックするには、[SHOW STATUS](#) ステートメントを発行し、[Key_read_requests](#)、[Key_reads](#)、[Key_write_requests](#)、そして [Key_writes](#) の変数を調べる ([SHOW 構文](#) を参照)。一般的には [Key_reads/Key_read_requests](#) の比率は、0.01 より小さいことが望ましい。操作がほとんど更新と削除だけの場合は [Key_writes/Key_write_requests](#) の比率は 1 に近い。同時に多くの行に影響を与える更新を行う場合や、[DELAY_KEY_WRITE](#) テーブル オプションを使用している場合には、より小さい比率になる。

使用中のキー バッファのフラクシオン (破片) は、バッファ ブロック サイズと、[key_buffer_size](#) に [Key_blocks_unused](#) を併用して決めることができる。これは、[key_cache_block_size](#) システム変数から利用可能。

```
1 - ((Key_blocks_unused × key_cache_block_size) / key_buffer_size)
```

これは近似値である。キー バッファのスペースによっては、管理ストラクチャに対して内部的に割り当てを行っているので、近似とする。

[MyISAM](#) の複数キー キャッシュを作成することが可能である。サイズの上限は、4 GB で、グループごとではなく、キャッシュごとに適用する。詳細は [MyISAMキーキャッシュ](#) を参照のこと。

- [key_cache_age_threshold](#)

この値は、キー キャッシュの hot サブ チェーンから warm サブ チェーンへのバッファ降格を制御する。この値が小さいと降格は急激に起こる。最小値は 100。デフォルトは 300。詳細は [MyISAMキーキャッシュ](#) を参照。

- [key_cache_block_size](#)

キー キャッシュのブロック サイズをバイト単位 (byte)。デフォルトは 1024 。詳細は [MyISAMキーキャッシュ](#) を参照のこと。 .

- [key_cache_division_limit](#)

キー キャッシュのバッファ チェーンにおける hot と warm のサブ チェーン間のディビジョン ポイント (分岐点)。この値は、warm のサブ チェーンに使用するバッファ チェーンのパーセンテージである。許容範囲は 1 から 100 まで。デフォルトは 100 。詳細は [MyISAMキーキャッシュ](#) を参照のこと。

- [language](#)

エラー メッセージに使用する言語。

- [large_file_support](#)

[mysqld](#) を大きなファイルをサポートするオプションでコンパイルしているかどうか (を指す)。

- [large_pages](#)

大きなページをサポートしている場合の戻値。

- [lc_time_names](#)

ロケールを指定する。これは月日、略語などを表示する言語を制御する。[DATE_FORMAT\(\)](#)、[DAYNAME\(\)](#)、[MONTHNAME\(\)](#) などの関数からの出力に影響する。ローケル名は POSIX 標準で、'ja_JP' または 'pt_BR' などとする。デフォルトはシステムのローケル設定を問わず、'en_US' である。詳細は [MySQL サーバのローケル サポート](#) を参照のこと。(MySQL 5.1.12 で導入)

- [license](#)

サーバのライセンス タイプ (種類)

- [local_infile](#)

`LOAD DATA INFILE` ステートメントで、`LOCAL` をサポートしているかどうか (を指す)。詳細は [LOAD DATA LOCAL のセキュリティ関連事項](#) を参照のこと。

- `locked_in_memory`

`--memlock` によるメモリのロックが `mysqld` で有効かどうか (を指す)。

- `log`

すべてのクエリのログを有効にしているかどうか (を指す)。 [一般クエリ ログ](#) を参照のこと。

- `log_bin`

バイナリ ログを有効にしているかどうか (を指す)。 [バイナリ ログ](#) を参照のこと。

- `log_bin_trust_function_creators`

この値はバイナリ ログを有効化しているときに適用。Stored Function (関数) を作成するユーザが、信用できない関数を作成する可能性を制御する。つまり、バイナリ ログへの書き込みに対して危険な関数を発行しないようにする。この値を 0 (デフォルト) にする場合、`CREATE ROUTINE`、`ALTER ROUTINE` 権限のいずれかに加え、`SUPER` 権限を持たないユーザによる関数の作成 (置換) を許可しない。0 にすると、その制限が強まり、`DETERMINISTIC` あるいは、`READS SQL DATA`、`NO SQL` のいずれかの特性を持った関数での宣言が必要になる。この値を 1 にすると、MySQL は関数に対して、このような制限はなくなる。詳細は [ストアドルーチンとトリガのバイナリログ](#) を参照のこと。

- `log_error`

エラー ログの位置。

- `log_output`

一般クエリ ログとスロー クエリ ログの出力先。`TABLE` (テーブルへのログ)、`FILE` (ファイルへのログ)、`NONE` (テーブルまたはファイルにログしない。) などを、複数の単語をカンマ区切りリストにできる。デフォルトは `TABLE`。`NONE` は、どの指定子よりも優先となる。つまり、`NONE` の場合、ログ エントリはログを有効化していても書き込みがない。ログを有効にしていなければ、`log_output` が `NONE` でなくても、ログできない。詳細は [一般クエリとスロー クエリのログ出力先の選択](#) を参照のこと。(MySQL 5.1.6 から導入)

- `log_queries_not_using_indexes`

インデックスしていないクエリをスロー クエリ ログに記録しているかどうか (を指す)。詳細は [スロー クエリ ログ](#) を参照のこと。(MySQL 5.1.11 から導入)

- `log_slave_updates`

スレーブ サーバがマスタ サーバから受け取った更新を、スレーブ サーバ自身のバイナリ ログに記録するかどうかを指す。この効果を得る (スレーブでバイナリ ログする) には、スレーブ上でバイナリ ログを有効にしておく必要がある。 [レプリケーションのオプションと変数](#) を参照のこと。

- `log_slow_queries`

スロー クエリのログするかどうか (を指す)。「Slow」は `long_query_time` の値で決定する。 [スロー クエリ ログ](#) を参照のこと。

- `log_warnings`

警告メッセージに追加情報を表示するかどうか (を指す)。デフォルトでは有効 (1)。0 にすると無効になる。この値が 1 より小さい場合、接続を中断した情報をエラーログに出力できない。

- `long_query_time`

クエリでこの値 (秒単位) より時間がかかると、`Slow_queries` カウンタが増える (increment)。`--log-slow-queries` オプションを使用している場合、クエリはスロー クエリ ログ ファイルでの記録になる。この値は、CPU 時間ではなく、リアルタイムである。したがって、低負荷のシステムではしきい値 (基準値) より下のクエリが、高負荷のシステムでしきい値より上になる場合がある。下限値は 1 で、デフォルトは 10。 [スロー クエリ ログ](#) を参照のこと。

- [low_priority_updates](#)

1 に設定すると、対象テーブルに影響する [SELECT](#) または [LOCK TABLE READ](#) を完了するまで、[INSERT](#)、[UPDATE](#)、[DELETE](#)、[LOCK TABLE WRITE](#) などのステートメントを待機させる。この変数の旧称は [sql_low_priority_updates](#)。

- [lower_case_file_system](#)

データ ディレクトリのファイルシステムの大小文字区別を示す。[OFF](#) の場合は、ファイル名でこの区別をしていることを意味し、[ON](#) の場合は大小文字の区別をしていないことを意味する。

- [lower_case_table_names](#)

この値が 1 の場合、テーブル名を小文字変換で格納している。つまりテーブル名には大小文字の区別がないことを示す。この値が 2 の場合、テーブル名を入力通りに格納するが、小文字の区別をする。このオプションはデータベース名とテーブルのエイリアスで適用。[識別子の大文字/小文字区別](#) を参照のこと。

[InnoDB](#) テーブルを使用する場合、名前を強制的に小文字に変換するには、すべてのプラットフォームでこの値を 1 に設定する。

たとえば、Windows または Mac OS など、システムで MySQL を実行していて、ファイル名に大小文字の区別がない場合は、この値を 0 にしないこと。起動時などでこの値をまだ設定していない状態で、かつデータ ディレクトリのファイルシステムで大小文字の区別をしていないときには、MySQL が自動的に [lower_case_table_names](#) を 2 に設定する。

- [max_allowed_packet](#)

1 パケットの最大サイズ。または生成/中間文字列。

パケット メッセージ バッファは [net_buffer_length](#) バイトで初期化するが、必要に応じて [max_allowed_packet](#) バイトまで大きくできる。デフォルト値は小さいが、大きな (不正) パケットを受けないように設定している。

大きな [BLOB](#) カラムを使用している場合には、この値を大きくする必要がある。使用する最大の [BLOB](#) と同じ大きさにする。[max_allowed_packet](#) のプロトコル制限は 1GB。

- [max_binlog_cache_size](#)

複数ステートメントのトランザクションでこれより多くのメモリが必要になると、[Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage](#) というエラーが発生する。下限値は 4096 で、最大 (デフォルト) は 4 GB。

- [max_binlog_size](#)

バイナリ ログ ファイルへの書き込みがログ ファイル サイズと干渉し、この値を超える場合、バイナリ ログをローテートする。(現行 ファイルを閉じて、次のファイルを開ける。) 設定可能値は、4096 バイト以上 1 GB (デフォルト) 以下。

注意 : トランザクションではバイナリ ログへの書き込みを 1 つのまとまりとして処理するため、トランザクション自体を複数のバイナリ ログに分割することは絶対的にない。したがって、大きなトランザクションがある場合、バイナリログの [max_binlog_size](#) が大きくなることもある。

[max_relay_log_size](#) が 0 の場合、[max_binlog_size](#) の値がリレー ログにも適用となる。

- [max_connect_errors](#)

ホストからの接続中断回数がこの値を越えた場合、それ以降、そのホストからの接続をブロックする。ブロックを解除するには、[FLUSH HOSTS](#) コマンドを使用する。

- [max_connections](#)

MySQL への最大同時接続数。MySQL 5.1.15 以降のデフォルトは 150 (以前は 100)。詳細は [Too many connections](#) を参照のこと。

```
#####MySQL Network Monitoring and Advisory Service #
##max_connections ##### http://www.jp.mysql.com/products/enterprise/
advisors.html () #####
```

この値を増やすと、mysqld のファイルの記述子の数を増やすこととなる。ファイル記述子の制限に関するコメントは [MySQL でのテーブルのオープンとクローズの方法](#) を参照のこと。

- `max_delayed_threads`

`INSERT DELAYED` ステートメント処理に、スレッドがこの数に達すると処理しない。つまり、スレッドの最大数である。使用中の `INSERT DELAYED` スレッド後に、データを新規テーブルに挿入すると、その行は `DELAYED` 属性を指定していない行になる。値を 0 に設定すると、MySQL は `DELAYED` を処理するスレッドを作成しなくなる。事実上、`DELAYED` を完全に無効にする。

- `max_error_count`

`SHOW ERRORS` や `SHOW WARNINGS` で表示するエラーや警告の最大数。

- `max_heap_table_size`

`MEMORY` 型テーブルの最大メモリ サイズを設定する(ヒープ)。この変数の値で `MEMORY` テーブルの `MAX_ROWS` 値を計算する。この変数の設定は、既存の `MEMORY` テーブルには影響しない。ただし、`CREATE TABLE` などのステートメントで再生成したり、`ALTER TABLE` または `TRUNCATE TABLE` で変更した場合は影響する。

```
MySQL Enterprise
MySQL Network Monitoring and Advisory Service ###max_heap_table_size ##### (##) ####
##### http://www.jp.mysql.com/products/enterprise/advisors.html #####
```

- `max_insert_delayed_threads`

`max_delayed_threads` に対するシノニム。

- `max_join_size`

`max_join_size` の値でクエリを制限する。長い時間をかけて百万行を返すような `WHERE` なしの結合を作成するようなクエリをいえる場合にこの変数を設定すると、サーバへの無駄な負荷を軽減させることができる。(`max_join_size` 値を超える行数のレコードを読み取ろうとするとエラーになる。) `SELECT` ステートメントで、レコードの組み合わせ調べ(単一テーブルまたは複数テーブルに対するステートメント)がこの値を超える場合や、またはこの値を超えるディスクークの実行を許可しない。この値を設定すると、キーの使用が不適切で長時間かかるような `SELECT` ステートメントを捕捉できる。

この変数を `DEFAULT` 以外の値で設定すると、`SQL_BIG_SELECTS` の値が 0 にリセットになる。`SQL_BIG_SELECTS` 値を再び設定すると、`max_join_size` 変数は無視の対象になる。

クエリ結果がクエリ キャッシュにある場合は、結果のサイズ チェックすることなく、これは結果がすでに計算済みで、それをクライアントに送信することがサーバの負荷にならないためである。

この変数の旧称は `sql_max_join_size` である。

- `max_length_for_sort_data`

使用する `filesort` アルゴリズムを決定するインデックス値の最大サイズ。[ORDER BY最適化](#) を参照のこと。

- `max_prepared_stmt_count`

サーバの Prepared ステートメントの合計数の最大値。この値で制限する。リクエスト応答を大量に発生させることで、サーバの応答機能の帯域を使い切るなど、いわゆる DoS 攻撃 (denial-of-service attacks) を受ける可能性がある環境で使用する。デフォルト値は 16382。許容値は 0 から 100 万。この値が実行中の Prepared ステートメントの数より低い場合は、既存ステートメントが影響を受けることはなく、そのまま使用できるが、新規のステートメントに関しては、この制限値よりも現行数を低減するまで、準備できない。(MySQL 5.1.10 からの導入)

- `max_relay_log_size`

レプリケーション スレーブがリレー ログに書き込みをすると、カレント ログ ファイル サイズがこの変数値を超える原因になり、スレーブはリレー ログをローテートする。(現行ファイルを閉じ、次のファイルを開ける。)
[max_relay_log_size](#) を 0 とした場合、サーバはバイナリ ログとリレー ログの両方に [max_binlog_size](#) を使用する。[max_relay_log_size](#) が 0 より大きい場合、リレー ログのサイズを抑制し、両ログに異なるサイズを持たせることが可能になる。[max_relay_log_size](#) は 4096 バイトから 1GB 以内で設定するか、または 0 に設定する。デフォルト値は 0。[レプリケーション実装の詳細](#) を参照のこと。

- [max_seeks_for_key](#)

キーでレコード検索の最大回数を制限する。キー スキャンでテーブルからレコードを検索するときに、MySQL オプティマイザでキーの基数とは関係なく(無視して)、キー検索の回数をこの指定値までとする。[SHOW INDEX 構文](#) を参照のこと。この値を低く (100 くらいに) 設定すると、MySQL でのスキャンがテーブルではなくキーを優先するようになる。

- [max_sort_length](#)

BLOB 値または TEXT 値をソートするときに使用するバイト数。各値の最初の [max_sort_length](#) バイトだけを使用し、残りは無視になる。

- [max_sp_recursion_depth](#)

ストアド プロシージャが呼び出す回数。このオプションのデフォルト値は 0 で、ストアド プロシージャの再帰を完全に無効にする。

この変数はグローバル、かつセッションごとの設定が可能。

- [max_tmp_tables](#)

1 つのクライアントが同時に開けたままにできるテンポラリ テーブルの最大数。(このオプションはまだ利用できない。)

- [max_user_connections](#)

単一ユーザ (MySQL アカウント) が同時に接続できる最大数。値 0 は「制限なし」という意味。

この値はグローバル スコープとセッション スコープ (読み込みオンリー) の両方を持つ。セッション値は通常グローバル値と同じ値。ただし、セッション ユーザが 0 以外の [MAX_USER_CONNECTIONS](#) 値を持つ場合には、このセッション値がアカウント制限にも反映する。

- [max_write_lock_count](#)

この回数の書き込みロックをした後に、読み取りロックを許可する。(ロックが必要なほど大量のテーブル書き込みがある場合など)

- [multi_range_count](#)

テーブル ハンドラへ一括送信できる最大許容範囲 (範囲選択時)。デフォルトは 256。複数の値域をハンドラへ 1 回で送れると、一定の選択において劇的なパフォーマンス向上になる。これは NDB Cluster のテーブル ハンドラで非常に有用で、値域要求をすべてのノードへ送るときに役に立つ。要求のバッチを一度に送ることは、通信コストを大幅節減に繋がる。

- [myisam_data_pointer_size](#)

CREATE TABLE 時に、[MAX_ROWS](#) オプションを指定していないときの MyISAM テーブル内部のポインタ サイズを指定する。(テーブルの最大物理サイズの決定。) 変数は 2 以上 7 以下である必要がる。デフォルトでは 6。[The table is full](#) を参照のこと。

- [myisam_max_extra_sort_file_size](#) (廃止)

ノート:この変数は MySQL 5.1 ではサポートしていない。詳細は MySQL 5.0 Reference Manual を参照のこと。

- [myisam_max_sort_file_size](#)

REPAIR TABLE、ALTER TABLE、LOAD DATA INFILEなどを使用して MyISAM インデックスを再生成するとき、MySQL が使用できるテンポラリ ファイルの最大サイズ。ファイル サイズがこれより大きい場合、インデックスはキー キャッシュでの作成になる (時間がかかる)。値の単位はバイト。

デフォルトでは 2GB。MyISAM インデックス ファイルがこのサイズを越え、ディスク容量が要るようになるときには、この値を大きくするとパフォーマンス向上になる。

- [myisam_recover_options](#)

--myisam-recover オプションの値。コマンド オプション を参照のこと。

- [myisam_repair_threads](#)

この値が 1 より大きい場合、Repair by sorting の修復プロセスでの MyISAM テーブル インデックスは並列での作成になる。インデックス毎のスレッド生成になる。

ノート:複数スレッドの修復は ベータ段階 です。

- [myisam_sort_buffer_size](#)

REPAIR TABLE 文実行時に MyISAM テーブルのインデックスをソートする場合、または CREATE INDEX や ALTER TABLE などインデックスを作成する場合に、割り当てるバッファのサイズ。

- [myisam_stats_method](#)

MyISAM テーブルでインデックス分布統計を集計するとき、サーバの NULL 値の扱いを決定する。この変数の値は [nulls_equal](#) または [nulls_unequal](#) のどちらか。[nulls_equal](#) の場合、すべての NULL インデックス値を同等として扱い、NULL 値の数とサイズが同等の単値のグループを生成する。[nulls_unequal](#) の場合、NULL の値同士を同等とは扱わず、それぞれの NULL で、サイズを 1 とする独特のグループを生成する。

この方法で、クエリの実行に対して、オプティマイザがインデックスを選択するときに影響を受けるテーブルの統計を取る。詳細は [MyISAMインデックス統計コレクション](#) を参照のこと。

- [myisam_use_mmap](#)

MyISAM テーブルの読み書き込みで使用するメモリ マッピング。(MySQL 5.1.4 での追加)

- [multi_read_range](#)

範囲指定の SELECT 文を発行するときに、ストレージ エンジンに送ることができる範囲の最大値を指定する。デフォルトでは 256。複数の範囲指定をストレージ エンジンに送ることは 特定の SELECT 文に対して、特に [NDBCLUSTER](#) において、大幅にパフォーマンスを改善する。

- [named_pipe](#)

サーバが名前付きパイプ (named pipes) を経由した接続をサポートするかどうか。Windows 専用。

- [ndb_autoincrement_prefetch_sz](#)

auto_increment カラムにおけるギャップの確率 (probability) を決定する。1 にセットした場合、これを最小限に抑える。最適化を目的として値を大きくする設定すると、— が挿入を高速化するが、バッチ挿入に使用する自動インクリメントの数が減る可能性がある。デフォルトは 32。下限値は 1。

- [ndb_cache_check_time](#)

NDB のクエリ キャッシュをチェックする前に待機するミリ秒 (msec)。この値を 0 (デフォルト) に設定すると、NDB クエリ キャッシュでクエリ毎のバリデーションを行う。

推奨最大値は 1000 で、クエリ キャッシュが一秒間に一度のチェック対象になることを意味する。値が大きくなるということは、NDB のクエリ キャッシュのチェック回数が減り、別の [mysqld](#) での更新が原因で、バリデーションが無効化することを意味する。この値を 2000 以上に設定しないこと。

- [ndb_extra_logging](#)

デバッグやトラブルシューティング用に追加の **NDB** ロギングを行うには、ゼロではない値を設定する。デフォルトは 0。

(MySQL 5.1.6 での追加)

- **ndb_force_send**

NDB へ迅速にバッファを送るよう命令する。他のスレッドを待機しない。デフォルトは **ON**。

- **ndb_index_stat_cache_entries**

統計の精度を設定する。開始キーと終了キーの数を決め、統計メモリ キャッシュに格納する。ゼロはキャッシュしていないことを示し、その場合には、データ ノードが直接的にクエリになる。デフォルトは 32。

- **ndb_index_stat_enable**

NDB インデックス統計。クエリの最適化。デフォルトでは **ON**。

- **ndb_index_stat_update_freq**

統計キャッシュの代わりにデータ ノードにクエリを行うかを示す回数。たとえば、値が 20 の場合、クエリ 20 毎に、データ ノードを渡すという意味。

- **ndb_report_thresh_binlog_epoch_slip**

binlog ステータスをレポートするまでに、遅れるエボック数の閾値。たとえば、値が 3 (デフォルト) であった場合、ストレージ ノードから受けたエボックと 3 以上の binlog に適用したエボックの数が異なるときに、ステータスメッセージをクラスタ ログする。

- **ndb_report_thresh_binlog_mem_usage**

binlog ステータスをレポートするまで残っている空きメモリのパーセンテージの閾値。たとえば、値が 10 (デフォルト) であった場合、データ ノードから受ける binlog データに使うメモリが 10% 降下するという意味。そしてステータス メッセージをクラスタ ログにする。

- **ndb_use_copying_alter_table**

NDB で、オンラインの **ALTER TABLE** 操作で、問題が発生したテーブルをコピーするために使用する。デフォルトは **OFF** になっている。

(MySQL 5.1.12 での追加)

- **ndb_use_exact_count**

SELECT COUNT(*) クエリ的高速化を図るときに、レコードの回数を適用するよう **NDB** に命令する。デフォルトでは **ON** になっている。全体的にクエリを速度化するには、これを無効にする。つまり、**ndb_use_exact_count** を **OFF** にする。

- **ndb_use_transactions**

NDB トランザクションを **OFF** に設定することで無効にできるが、これは極力やらない。デフォルトでは **ON** になっている。

- **net_buffer_length**

クライアント スレッドが接続バッファと結果バッファに関連付けられている。両者は **net_buffer_length** で与えられたサイズで始まるが、必要に応じて、**max_allowed_packet** バイトまで劇的に拡大できる。結果バッファは SQL 文毎に **net_buffer_length** まで縮小する。

この値はできるだけ変更しない。ただし、メモリが非常に限られている環境において、この値をクライアントから送信されるステートメントの長さに合わせて設定できる。ステートメントがこの長さを越えた場合、接続バッファは自動的に拡大する。**net_buffer_length** の最大値は 1MB に設定できる。

- **net_read_timeout**

読み込みを中断するまでデータ追加を待機する秒数。タイムアウトは TCP/IP 接続にだけ適用する。これは Unix のソケット ファイル、または名前付きパイプ、共有メモリなどを経由していない接続のことである。サーバがクライアントからの読み込みを行うとき、[net_read_timeout](#) のタイムアウト値が中断するタイミングを制御する。書き込みを行うときは、[net_write_timeout](#) のタイムアウト値が中断するタイミングを制御する。[slave_net_timeout](#) のセクションも参照のこと。

- [net_retry_count](#)

通信ポートでの読み込みが中断した場合に、実行できる再試行回数。FreeBSD でこの値を大きくすると、内部中断をすべてのスレッドに通知する。

- [net_write_timeout](#)

書き込みを中断するまで、ブロック書き込みを待機する秒数。タイムアウトは TCP/IP 接続にだけ適用する。これは Unix のソケット ファイル、または名前付きパイプ、共有メモリなどを経由していない接続のことである。[net_read_timeout](#) も参照のこと。

- [new](#)

MySQL 4.0 において、MySQL 4.1 と同様の動作 (下位互換性) をするかどうか (を指す)。MySQL 5.1 では、この値は常に **OFF**。

- [old_passwords](#)

サーバが MySQL 4.1 より前で使用しているパスワード形式を採用するかどうか (を指す)。[Client does not support authentication protocol](#) を参照のこと。

- [one_shot](#)

これは変数ではない。しかし、特定の変数を設定するときに使用できる。詳細は [SET 構文](#) を参照のこと。

- [open_files_limit](#)

[mysqld](#) が開けるオペレーティング システムのファイル数。これはシステムで指定されている実際の値であり、起動時のパラメータとして [--open-files-limit](#) オプションで [mysqld](#) または [mysqld_safe](#) に指定したものとは異なる場合がある。MySQL がオープンファイル数を変更できないシステムでは 0 になる。

- [optimizer_prune_level](#)

ヒューリスティックス (経験則) を採用したクエリ最適化を制御し、オプティマイザの検索スペースからあまり見込みのない、部分的なプランをパージ (削除) する。値を 0 にすると、ヒューリスティックスを無効化し、オプティマイザは完全な検索を行う。値を 1 にすると、オプティマイザは中間プラン (intermediate plans) で検索したレコード数に基づいてプランを削除する。

- [optimizer_search_depth](#)

クエリ オプティマイザが実行する検索深さの最大値を指定する。クエリ結果の関係数が大きい値の場合は、プランよりも良いということを示すが、クエリの実行プラン生成に時間がかかる。関係数が小さい値の場合は、より速い実行プランを返すが、結果プランが最適であるとは言えない。値を 0 した場合は、システムは自動的に妥当な値を計算する。値をクエリに使用しているテーブルの最大値に +2 した場合は、オプティマイザが検索を実行するときに、MySQL 5.0.0 (と前バージョン) で使用したアルゴリズムに切り替える。

- [pid_file](#)

プロセス ID (PID) ファイルのパス。[--pid-file](#) オプションで設定する。

- [plugin_dir](#)

プラグイン ディレクトリのパス。(MySQL 5.1.2 での追加)

- [port](#)

[mysqld](#)サーバが利用するTCP/IPポート番号。[--port](#) オプションで設定する。

- [preload_buffer_size](#)

インデックスをプレロードするときに割り当てるバッファ サイズ。

- [prepared_stmt_count](#)

準備されたステートメント (prepared statements) の現在値。[max_prepared_stmt_count](#) システム変数で与えるステートメントの最大値。MySQL 5.1.10 からの追加で、MySQL 5.1.14 では、[Prepared_stmt_count](#) グローバルステータス変数に変換していた。

- [protocol_version](#)

MySQL サーバが使うクライアント/サーバ間のプロトコル バージョン。

- [query_alloc_block_size](#)

クエリの解析や実行で生成するオブジェクトに割り当てるメモリ ブロックの割り当てサイズ。メモリのフラグメントに問題がある場合、これを少し大きくすると改善できる可能性がある。

- [query_cache_limit](#)

この値より大きい結果はキャッシュしない。デフォルトは 1MB。

- [query_cache_min_res_unit](#)

クエリ キャッシュで割り当てるブロックの最小サイズ (バイト単位)。デフォルトは 4096(4KB)。この変数を利用した効果については、[クエリ キャッシュの設定](#) を参照のこと。

- [query_cache_size](#)

古いクエリの結果の保存用に割り当てたメモリ (クエリ キャッシュで確保するメモリ)。この値を 0 (デフォルト) にすると、クエリ キャッシュは無効化する。許容値は 1024 の倍数。値はすべて、近似の倍数で端数を切り捨てる。ノート: [query_cache_size](#) バイトのメモリは、[query_cache_type](#) が 0 で設定してあっても、割り当ての対象となる。詳細は [クエリ キャッシュの設定](#) を参照のこと。

- [query_cache_type](#)

クエリ キャッシュを行う条件を指定する。[GLOBAL](#) 値に設定すると、これ以降に接続するすべてのクライアントの条件が反映する。それぞれのクライアントで [SESSION](#) 値を設定することができ、これはマシン レベルでのクエリ キャッシュに影響する。次のテーブルは数値を示す。

オプション	説明
0 または OFF	キャッシュを使用しない。注意: これはクエリ キャッシュのバッファの割当を解除しない。解除するには query_cache_size を 0 にセットする。
1 または ON	SELECT SQL_NO_CACHE を除くすべての結果をキャッシュする。
2 または DEMAND	SELECT SQL_CACHE で始まるクエリだけをキャッシュする。

この変数のデフォルトは [ON](#)。

- [query_cache_wlock_invalidate](#)

通常、クライアントが [MyISAM](#) テーブルの [WRITE](#) ロックを獲得する場合、別のクライアントはクエリの結果がキャッシュ内にあるときは、そのテーブルへのクエリ発行をブロックしない。この値を 1 にした場合、テーブルに対する [WRITE](#) ロックを得ることとなり、そのテーブルに対するクエリ キャッシュのクエリは無効化する。これにより、テーブルへのアクセスを試みるクライアントに対して、ロック有効中は待機するよう抑制できる。

- [query_prealloc_size](#)

クエリの解析および実行に使用する永続バッファ サイズ。このバッファはクエリ間でも開放しない。頻繁に複雑なクエリを発行する場合、[query_prealloc_size](#)の値を大きくして、クエリ実行時のメモリ割り当て回数を減らすことになるため、パフォーマンスを改善できる可能性がある。

- [range_alloc_block_size](#)

範囲の最適化で割り当てるブロックのサイズ。

- `read_buffer_size`

順次スキャン (全件) を行うときに各スレッドが割り当てるバッファ サイズ。バイトで指定する。このスキャンを何度も行う場合には、この値を大きくする。デフォルトは 131072。

- `read_only`

レプリケーション スレーブ サーバで、この値を `ON` に設定すると、サーバが `SUPER` 権限を持つユーザ以外からの更新ができない。スレーブ サーバにおいては、マスタ サーバからの更新だけを許容し、クライアントからの要求を無視するようになる。この動作は `TEMPORARY` テーブルには使えない。

`read_only` は `GLOBAL` 変数として存在するために、`SUPER` 権限が必要な値への変更になる。マスタ サーバで `read_only` に変更すると、スレーブ サーバで複製できなくなる。この値は、マスタの設定とは独立しているスレーブ サーバで行う。

MySQL 5.1.15 以降は、次のことに注意する。

- `read_only` を有効にしようとするときに、明示的なロック (`LOCK TABLES` などから) がある場合、または未処理のトランザクションがある場合は、エラーになる。
- 別のクライアントに明示的なテーブル ブロックがある場合、または未処理のトランザクションがある場合に、`read_only` を有効にすると、ロックをリリースし、トランザクションが完了するまでブロックする。`read_only` にする試みが進行中である場合、テーブル ロックに対する別のクライアントからの要求、またはトランザクションを開始することへの要求を、`read_only` の設定を完了するまで、ブロックする。
- `read_only` はグローバル読み込みブロック (`FLUSH TABLES WITH READ LOCK` などから) を保持している間に有効化できる。これはテーブル ロックを巻き込まないためである。

- `read_rnd_buffer_size`

ソートしたレコードを読み出すときのバッファサイズを指定する。ディスク検索を行わないように、レコードをこのバッファから読み取る。この値を大きく設定すると、`ORDER BY` のパフォーマンスを大幅に向上できる。しかし、これはスレッド固有の変数であるため、これをグローバル値を大きな値で設定すべきではない。したがって、大量のクエリを実行するときにだけ、クライアント内のセッション値を変更する。

- `relay_log_purge`

リレー ログ ファイルが不必要になったときの自動削除フラグを設定する。デフォルトは 1 (`ON`)。

- `rpl_recovery_rank`

この変数は使用していない。

- `secure_auth`

`--secure-auth` オプションを付けて MySQL サーバを起動する場合、MySQL サーバは古い形式 (4.1 より前) のパスワードを認証しない。このときの値は `ON` (ブロック)。接続をブロックしないようにするには、`OFF` にする。

これによりネットワーク セキュリティが不安定な場合など、古い形式を採用しているパスワードでの接続を許可しないようにするには、このオプションを有効にする。

このオプションを有効にしたときに、権限テーブルが 4.1 よりも前の形式である場合には、起動時にサーバ エラーが発生する。 [Client does not support authentication protocol](#) を参照のこと。

- `server_id`

サーバ ID。`--server-id` オプションで設定する。これはレプリケーションを行うときなどに、マスタとスレーブのサーバ間でお互いを一意的に認識させるために使用する。

- `shared_memory`

共有メモリに付ける識別子を指定する。Windows 専用。

- [shared_memory_base_name](#)

共有メモリ接続で使用する共有メモリの名前。このオプションはWindowsで有効。複数の MySQL インスタンス (サーバ) を一台のマシンで使用している場合に便利。デフォルト名は [MYSQL](#)。この名前は大文字と小文字を区別する。

- [skip_external_locking](#)

OFF : [mysqld](#) が外部ロックを使用している場合。ON : 外部ロックが無効の場合。

- [skip_networking](#)

ON : サーバがローカル接続のみを許可する場合 (非 TCP/IP)。Unix では、ローカル接続に Unix ソケット ファイルを使用。Windows では、名前付きパイプまたは共有メモリを使用。NetWare では、TCP/IP 接続をサポートしている場合のみ。そのため、この変数を ON に設定してはいけない。この変数を ON にするには、[--skip-networking](#) オプションを使用する。

- [skip_show_database](#)

[SHOW DATABASES](#) 権限を持たずして [SHOW DATABASES](#) を使用するユーザにデータベースを表示しない。これは、他人のユーザ権限のデータベースを表示しないため、セキュリティの向上に役立つ。この効果は [SHOW DATABASES](#) 権限の設定オプションに依存する。値を ON にした場合、[SHOW DATABASES](#) ステートメントは [SHOW DATABASES](#) 権限を持つユーザにだけが使用でき、ステートメントにはすべてのデータベース名を表示する。値を OFF にした場合、[SHOW DATABASES](#) はすべてのユーザに対してこのステートメントの発行を許可するが、ユーザが [SHOW DATABASES](#) などの権限を持っているデータベースだけの表示になる。

- [slave_compressed_protocol](#)

マスタとスレーブの両方が圧縮プロトコルをサポートしているかどうか (を指す)。

- [slave_load_tmpdir](#)

スレーブサーバがレプリケーション データ ([LOAD DATA INFILE](#) ステートメント) を読み込むときに 使用するテンポラリ ディレクトリのパスを指定する。

- [slave_net_timeout](#)

読み込みを中断するまで、マスタ/スレーブ接続からのデータを待機する秒数。タイムアウトは TCP/IP 接続にだけ適用。Unix ソケット ファイルを介した接続や、名前付きパイプ、共有メモリには適用しない。

- [slave_skip_errors](#)

スレーブがスキップ (無視) するレプリケーション エラー。

- [slave_transaction_retries](#)

InnoDB デッドロック、InnoDB の [innodb_lock_wait_timeout](#)、NDBCluster の [TransactionDeadlockDetectionTimeout](#) または [TransactionInactiveTimeout](#) を越えた場合、レプリケーション スレーブの SQL スレッドはトランザクションの実行に失敗する。そのときに、エラーで停止する前に、自動試行する回数のことを [slave_transaction_retries](#) 回と定義する。デフォルトでは 10 回。

- [slow_launch_time](#)

スレッドの作成にこの値 (秒単位) より時間がかかると、サーバが [Slow_launch_threads](#) ステータス変数 (カウンタ) をインクリメントする。

- [slow_query_log](#)

スロークエリ ログが有効になっているかどうか (を指す)。値が 0 (または OFF) の場合、ログしない。値が 1 (または ON) の場合、ログする。デフォルトは [--log-slow-queries](#) オプションを設定しているかどうかによる。ログ出力先は [log_output](#) システム変数で制御する。値を NONE にした場合、ログできるようにしていても、ログ エントリを書き込まない。[slow_query_log](#) は MySQL 5.1.12 からの導入。

- [slow_query_log_file](#)

スロー クエリ ログ ファイルの名前。デフォルトは `host_name-slow.log`。初期値は `--log-slow-queries` オプションで変更可能。(MySQL 5.1.12 での追加)

- `socket`

Unix 環境で、ローカル接続に使用するソケット ファイル パス。デフォルトは `/tmp/mysql.sock` になっている。配布用の形式によっては、ディレクトリが異なる場合がある。たとえば、RPM の `/var/lib/mysql` など。

Windows環境では、ローカル接続に使用する名前付きパイプ名のこと。デフォルトは `MySQL`。文字の大小区別なし。

- `sort_buffer_size`

ソートを実行する必要があるスレッドがこのサイズのバッファを割り当てる。`ORDER BY` または `GROUP BY` などの操作を速くするには、この値を大きくする。[Where MySQL Stores Temporary Files](#) を参照のこと。

- `sql_mode`

現在のSQLモード。この値は動的に変更することが可能。[SQL モード](#) を参照のこと。

- `sql_slave_skip_counter`

スレーブ サーバが無視するマスタからのイベント数。`SET GLOBAL SQL_SLAVE_SKIP_COUNTER` 構文 を参照のこと。

- `ssl_ca`

SSL CA 証明をリストしたファイルのパス。(MySQL 5.1.11 での追加)

- `ssl_capath`

SSL CA 証明書 (PEM形式) があるディレクトリへのパス。(MySQL 5.1.11 での追加)

- `ssl_cert`

セキュリティ上、安全に接続を確立するために使用する SSL 証明書。(MySQL 5.1.11 での追加)

- `ssl_cipher`

SSL 暗号化に使用するサイファ (暗号鍵) のリスト。サイファ リストは `openssl ciphers` コマンドと同形式。(MySQL 5.1.11 での追加)

- `ssl_key`

セキュリティ上、安全に接続を確立するために使用する SSL キー ファイル名。(MySQL 5.1.11 での追加)

- `storage_engine`

デフォルトのストレージ エンジン (テーブル タイプ)。サーバ起動時にストレージ エンジンを設定するには、`--default-storage-engine` オプションを使用する。[コマンド オプション](#) を参照のこと。

- `sync_binlog`

この値が正数の場合、MySQL サーバはバイナリ ログへこの `sync_binlog` 回書き込みを行い、それごとにディスクへ同期する (`flush()` を使用)。オートコミットモードでは、一つの SQL 文を発行毎に、ログ書き込みとなる。それ以外のモードでは、トランザクションごとの書き込みになる。デフォルト値は 0 で、ディスクへの同期は行わないことを示す。値を 1 にすると、最も安全な設定となる。これはクラッシュした場合などに失うものが、1 ステートメントあるいは1トランザクションに留まるためである。しかし、これはパフォーマンスを遅くするため、対応策としては、同期を高速化するために、バッテリ バックアップ式キャッシュをディスクに持たせるなどする。

`sync_binlog` 値の 0 (デフォルト) は、追加フラッシュは行わないことを示す。これはフラッシュが OS に依存する。

- `sync_frm`

値を 1 とした場合に、テンポラリ以外のテーブルを作成すると、`.frm` ファイルを ディスクへ同期する (`fdatasync()` を使用)。これは処理スピードを遅くするが、クラッシュに対して安全である。デフォルトは 1。

- `system_time_zone`

サーバシステムのタイムゾーン。サーバが起動するときは、マシンのデフォルトタイムゾーンを継承する。これはサーバを起動したユーザアカウントの環境やスタートアップスクリプトのオプションなどで変更可能。値は `system_time_zone` を設定する。通常、タイムゾーンは `TZ` 環境変数で指定する。または `mysqld_safe` スクリプトでの `--timezone` オプションでも指定可能。

`system_time_zone` と `time_zone` は別物である。両者の値は同値であることもあるが、後者はクライアント接続毎にタイムゾーンを初期化する変数である。[MySQL サーバのタイムゾーンサポート](#) を参照のこと。

- `table_cache`

MySQL 5.1.3 前まで、`table_open_cache` と呼ばれていた。MySQL 5.1.3 以降は `table_open_cache` を使用する。

- `table_definition_cache`

定義キャッシュに保存できるテーブル定義数。テーブル数が多い場合に、テーブルを開けるスピードを速めるために、大きなテーブル定義キャッシュを作成できる。通常のテーブルキャッシュと比較すると、テーブル定義キャッシュが必要とするスペースが小さく、ファイル記述子を使用しない。(MySQL 5.1.3 以降)

- `table_lock_wait_timeout`

テーブルレベルロックで待機する時間(秒)を指定する。デフォルトのタイムアウトは 50 秒。タイムアウトは接続でカーソルを開けるとアクティブになる。`SUPER` 権限を保持していれば、ランタイムにグローバル設定可能。

- `table_open_cache`

すべてのスレッドに対するオープンテーブルの数(キャッシュする最大テーブル数)。この値を大きくするということは、`mysqld` が要求するファイル記述子の数を増やすということ。`Opened_tables` ステータス変数をチェックしてテーブルキャッシュを増やす必要があるかどうかを調べる。[ステータス変数](#) を参照のこと。`Opened_tables` の値が大きく、`FLUSH TABLES` を頻繁に行わない場合(単にテーブルの開閉を強制するだけ)、`table_open_cache` 変数の値を増やす必要がある。テーブルキャッシュに関する詳細は [MySQL でのテーブルのオープンとクローズの方法](#) を参照のこと。MySQL 5.1.3 前には、`table_cache` と呼ばれていた。

- `table_type`

`storage_engine` のシノニム。MySQL 5.1 では `storage_engine` を使用。MySQL 5.2 では、`table_type` は削除されている。

- `thread_cache_size`

再利用のためにキャッシュ可能なスレッド数。クライアントが接続を切断したときに、以前のスレッド数が `thread_cache_size` 以下であれば、そのクライアントのスレッドはキャッシュに入る。新しいスレッドはすべてキャッシュから取り込まれ、キャッシュが空の場合のみ、新しいスレッドが作成される。新しい接続が多く発生する場合、この変数を大きくすることによりパフォーマンスを向上できる(スレッド実装が既に理想的な状態であれば、それほどパフォーマンスは向上しない)。`Connections` および `Threads_created` などのステータス変数の差異を調べると、スレッドキャッシュの効率性を確認できる。詳細は [ステータス変数](#) を参照のこと。

- `thread_concurrency`

Solaris では、`mysqld` がこの値を伴う `thr_setconcurrency()` を呼び出す。アプリケーションに、同時に実行する理想的なスレッド数を提供する。

- `thread_stack`

各スレッドのスタックサイズ。`crash-me` で検出する制限の多くは、この値に依存する。通常の操作ではデフォルト(192 KB)で十分である。[MySQL ベンチマークスイート](#) を参照のこと。

- `time_format`

この変数は実装していない。

- [time_zone](#)

現在のタイムゾーン。この変数はクライアント接続毎にタイムゾーンを初期化する。デフォルトは 'SYSTEM'、つまり [system_time_zone](#)「の値を使う」ということ。サーバ起動時に `--default-time-zone` オプションで明示的に指定できる。[MySQL サーバのタイムゾーンサポート](#) を参照のこと。

- [timed_mutexes](#)

InnoDB ミューテックスをカウントしているかどうかを制御する。この値を 0 または `OFF` (デフォルト) に設定すると、ミューテックス カウントは無効になる。この値を 1 または `ON` に設定すると、ミューテックス カウントは有効になる。このカウントを有効にすると、`SHOW ENGINE INNODB MUTEX` からの出力の `os_wait_times` 値は、OS が待機した回数 (ms) を示す。それ以外では、この値は 0 である。

- [tmp_table_size](#)

メモリ内のテンポラリ テーブルの最大サイズ。実際の限度は [max_heap_table_size](#) や [tmp_table_size](#) の値より小さい値になる。メモリ内テンポラリ テーブルが制限値を超えると、MySQL はこれを自動的にディスク内の [MyISAM](#) テーブルにする。高度な `GROUP BY` クエリを展開する場合にメモリが沢山あるときは、[tmp_table_size](#) (必要に応じて、[max_heap_table_size](#) も) の値を増やす。

- [tmpdir](#)

テンポラリ ファイルとテンポラリ テーブルのディレクトリ。この変数はラウンド ロビン式の複数のパスのリストをセットするとき使用する。Unix で、パスはコロン (':') で区切り、Windows、NetWare、OS/2 などではセミコロン(';') で区切る。

複数ディレクトリ機能は、物理ディスク間で負荷を分担するとき使用する。MySQL サーバが レプリケーション スレーブである場合、[tmpdir](#) を使用して、メモリ ベースのシステムファイルまたはサーバ ホスト (OS) 再起動時に内容が消去されるディレクトリを指定しない。これは、レプリケーション スレーブが、マシンがリブートした場合や `LOAD DATA INFILE` 文の処理中であった場合などに対応するために、テンポラリのテーブルやファイルを必要とするため。もしこのディレクトリからテンポラリ ファイルが消えた場合は、サーバ再起動時にレプリケーション エラーが発生する。しかし、MySQL 4.0.0 以降を使用している場合は、[slave_load_tmpdir](#) 変数を使用して、スレーブのテンポラリ ディレクトリを設定できる。その場合、スレーブは通常の [tmpdir](#) 値を使用しないので、[tmpdir](#) を適切な (非永続的) 位置に設置する (スレーブサーバはこのパスを使用することになる)。

- [transaction_alloc_block_size](#)

メモリ ブロックの割り当てサイズ (byte)。このメモリは、コミットするときにバイナリログへ書き込むための トランザクション内クエリを保存するために使用する。[transaction_prealloc_size](#) の説明を参照のこと。

- [transaction_prealloc_size](#)

永続的バッファの (初期) サイズを[transaction_prealloc_size](#) と呼ぶ。メモリが不十分が原因で、このプール (領域) で割り当てが十分に行えない場合、[transaction_alloc_block_size](#) でプールの値を大きくする。トランザクションが済むと、プールは [transaction_prealloc_size](#) バイトで切り捨てになる

単一トランザクション内のすべてのクエリをバッファ内に収めるように、[transaction_prealloc_size](#)を大きくすると、`malloc()` システム コールを避けることができる。

- [tx_isolation](#)

基準にするトランザクション隔離レベル。デフォルト値は `REPEATABLE-READ`。

この変数は `SET TRANSACTION ISOLATION LEVEL` ステートメントで設定する。[SET TRANSACTION 構文](#) を参照のこと。[tx_isolation](#) を直接、隔離レベル名に設定する場合に、スペースを含むときには、その名前を引用符で囲み、そのスペースをダッシュと置換する。たとえば、

```
SET tx_isolation = 'READ-COMMITTED';
```

- [updatable_views_with_limit](#)

更新の許可するかどうかを制御する。ビューに基準テーブルで定義したプライマリ キーのすべてのカラムが含まれていない場合に、更新ステートメントで `LIMIT` 節を含んでいたら、そのビューを更新するかどうか、ということである。このような更新は GUI ツールなどから生成される。ここでの更新は `UPDATE` または `DELETE` ステートメン

トのこと。ここでのプライマリ キーとは **PRIMARY KEY** または **UNIQUE** インデックスのことで、**NULL** をカラムに含まない。

この変数の値は 2 種類ある。

- **1** または **YES**: エラー メッセージではなく、警告だけを発行。(デフォルト値)
- **0** または **NO**: 更新禁止。

- **version**

サーバのバージョン番号。

- **version_comment**

configure スクリプトには、MySQL を構築するときにコメントを指定できる **--with-comment** オプションがある。そのコメントの値。

- **version_compile_machine**

マシンのタイプ、または MySQL 構築時のアーキテクチャ。

- **version_compile_os**

MySQL 構築時のオペレーティング システムのタイプ。

- **wait_timeout**

対話式ではない接続 (反応の無い接続) を終了する前に、サーバがアクティビティを待機する秒数。このタイムアウトは TCP/IP 接続と Unix のソケット ファイル接続だけに適用。名前付きパイプと共有ファイルの接続には使用できない。

スレッド起動時に、セッション **wait_timeout** 値は、**wait_timeout** グローバル値、または **interactive_timeout** グローバル値で初期化するが、これはクライアントのタイプによる。(**CLIENT_INTERACTIVE** の接続オプションを **mysql_real_connect()** に定義する。) **interactive_timeout** の説明も参照のこと。

システム変数の使用

mysql サーバは、様々なシステム変数を保有し、その変数をどのように設定したかを示します (**システム変数** を参照のこと)。それぞれのシステム変数にはデフォルト値があります。システム変数はサーバ起動時に、コマンドラインまたはオプション ファイルなどを使用してセットできます。大抵の場合、**SET** コマンドを使用して実行中のサーバで動的に変更できます。つまり、サーバを停止または再起動せずに変更できます。プログラミング式でシステム変数の値を参考にしてください。

サーバには 2 種類のシステム変数があります。グローバル変数はサーバの全体的なオペレーションに影響し、セッション変数はそれぞれのクライアント接続でのオペレーションに影響します。与えられた変数はグローバルとセッション、両方の値を持つこともあります。グローバルおよびセッション変数は次のように関係しています。

- サーバが起動するとき、すべてのグローバル変数をデフォルト値に初期化する。このデフォルト値はコマンドラインまたはオプション ファイルなどで指定できる。オプションに関しては、**プログラム・オプションの指定** を参照のこと。
- サーバにはクライアントが接続するセッション変数の組み合わせがある。クライアントのセッション変数は、グローバル変数に呼応するカレント値を使用して接続タイムで初期化する。たとえば、クライアントの SQL モードは、セッション **sql_mode** 値で制御し、クライアントが **sql_mode** のグローバル値に接続するときに初期化する。

システム変数はサーバ起動時にコマンドラインまたはオプションファイルを使用してグローバル設定できます。起動オプションを使用して値を設定するときは、数値を使用し、値には **K** (キロバイト)、**M** (メガバイト)、**G** (ギガバイト) などのサフィックスで与えます (大文字あるは小文字)。これらは、 1024 、 1024^2 または 1024^3 の倍数を示します。これにより、次のコマンドは、クエリ キャッシュ サイズが 16 メガバイト、最大パケット サイズが 1 ギガバイトでサーバが起動することを示します。

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

オプション ファイル内では、次のように設定します。

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

サフィックスの大文字、小文字の区別は問わず、16M と 16m、1G と 1g を同等とします。

SET ステートメントでラインタイムに設定できる変数の最大値を制限する場合には、サーバ起動時に --maximum-var_name=value のオプションで最大値を指定します。たとえば、query_cache_size の値がラインタイムで 32 MB を超えないようにするには、--maximum-query_cache_size=32M とします。

システム変数は動的で、SET ステートメントでサーバ稼動中に変更できます。リストは [動的システム変数](#) を参照してください。SET でシステム変数を変更するには、var_name とし、オプション的に修飾子で先行します。

- 変数がグローバルであることを明示するには、GLOBAL または @@global. で名前を先行する。グローバル変数を設定するには SUPER 権限が必要。
- 変数がセッションであることを明示するには、SESSION、@@session.、@@ など で名前を先行する。セッション値の設定には特別の権限は不要であるが、クライアントだけがそのセッション変数を変更できる。別のクライアントからはできない。
- LOCAL および @@local. は SESSION と @@session. のシノニム。
- 修飾子がない場合は、SET でセッション変数を変更する。

SET ステートメントには複数の変数アサインメントがあり、カンマで区切ります。複数のシステム変数を設定する場合、ステートメント内で最新の GLOBAL または SESSION の修飾子を、後続の指定子のない変数に使用します。

例：

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

システム変数に値を SET で指定するときには、変数にサフィックス文字は使用しません。(起動オプションのときは使用する。) ただし、この値は例示のようにプログラミング形式にできます。

```
SET sort_buffer_size = 10 * 1024 * 1024;
```

システム変数の @@var_name シンタックスは、別のデータベースシステムによっては互換性があります。

セッションシステム変数を変更する場合には、そのセッションが済むまで、または別の値に変更するまで、値の効力を維持します。この変更は別のクライアントには公開しません。

グローバルシステム変数を変更すると、サーバが再起動するまで、値を新規接続で使用するものとして記憶します。(グローバル変数の設定を永続的にするには、オプションファイルで設定する必要があります。) この変更は、そのグローバル変数にアクセスするすべてのクライアントに公開することになりますが、関連しているセッション変数への変更は、変更後に接続するクライアントに対してだけ反映します。グローバル変数の変更は、その時点で接続しているクライアントのセッション変数には反映しません。SET GLOBAL ステートメントを発行するクライアントからのイベントにも反映しません。

SET SESSION とだけ使用できる変数と SET GLOBAL を使用する場合、または GLOBAL (@@global.) をグローバル変数設定時に指定しない場合には、MySQL がエラーメッセージを出し、不正使用を防ぐことができます。

SESSION 変数を GLOBAL 値に、または GLOBAL 値をコンパイルされた MySQL のデフォルト値に設定するには、DEFAULT キーワードを使用します。たとえば、次の二つのステートメントは max_join_size のセッション値をグローバル値に設定するという点において、アイデンティカル (同一) です。

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

すべてのシステム変数において、DEFAULT と設定することはできません。そのような場合には、DEFAULT の使用が、エラーの原因になります。

@@- 修飾子のひとつを使用して、プログラミングにおける特定のグローバルまたはセッションのシステム変数値を照会できます。たとえば、SELECT ステートメントの値を次のように読み出すことができます。

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

プログラミングでのシステム変数を `@@var_name` とすると、つまり、`@@global.` または `@@session.` を指定しないとき、MySQL はセッション値があれば、それを返しますが、ない場合はグローバル値を返します。これは常にセッション値とする `SET @@var_name = value` とは異なります。

ノート:システム変数によっては、`SET` ステートメントで値を `ON` または `1` に設定すると有効化し、`OFF` または `0` にすると無効化します。ただし、このような値をコマンドラインまたはオプション ファイルで設定するには、`1` または `0` で設定します。`ON` または `OFF` での設定はできません。たとえば、コマンドラインにおいて、`--delay_key_write=1` は機能しますが、`--delay_key_write=ON` では機能しません。

システム変数名と値を表示するには、`SHOW VARIABLES` ステートメントを使用します。

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
auto_increment_increment	1
auto_increment_offset	1
automatic_sp_privileges	ON
back_log	50
basedir	/home/mysql/
binlog_cache_size	32768
bulk_insert_buffer_size	8388608
character_set_client	latin1
character_set_connection	latin1
character_set_database	latin1
character_set_results	latin1
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/home/mysql/share/mysql/charsets/
collation_connection	latin1_swedish_ci
collation_database	latin1_swedish_ci
collation_server	latin1_swedish_ci
...	
innodb_additional_mem_pool_size	1048576
innodb_autoextend_increment	8
innodb_buffer_pool_awesome_mem_mb	0
innodb_buffer_pool_size	8388608
innodb_checksums	ON
innodb_commit_concurrency	0
innodb_concurrency_tickets	500
innodb_data_file_path	ibdata1:10M:autoextend
innodb_data_home_dir	
...	
version	5.1.6-alpha-log
version_comment	Source distribution
version_compile_machine	i686
version_compile_os	suse-linux
wait_timeout	28800

`LIKE` 節では、パターンに一致する変数だけを表示します。特定の変数名を得るには、`LIKE` 節を次のように使います。

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

パターンに一致する名前を持つ変数のリストを得るには、`LIKE` 節で、`'%'` のワイルドカード文字を使用します。

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

ワイルドカード文字は、一致させるパターン内に入れます。厳密には、`'` シングル文字と一致するワイルドカードであるため、`\` でエスケープして、文字通りに一致させます。実際には、これほどの厳密さは要求することはありません。

`SHOW VARIABLES` では、`GLOBAL` または `SESSION` のいずれも指定しない場合、MySQL は `SESSION` 値を返します。

`GLOBAL` オンリーの変数を読み取るだけでなく、設定するときに `GLOBAL` キーワードを必要とする理由は、今後の問題を防ぐためです。`GLOBAL` 変数と同じ名前を持つ `SESSION` 変数を取り除いた場合、`SUPER` 権限を持つクライアントが、接続時に `SESSION` 変数だけでなく、偶発的に `GLOBAL` 変数も変更してしまう可能性があります。

す。[SESSION](#) 変数を [GLOBAL](#) と同じ名前を追加する場合、[GLOBAL](#) 変数を意図的に変更するクライアントで、クライアント自体の [SESSION](#) 変数だけが変更したと見なす可能性があります。

構造化システム変数

構造化変数 (structured variable) は通常のシステム変数とは次の 2 点において異なります。

- 値は、コンポーネントを伴うストラクチャであり、このコンポーネントとは、密接に関連している サーバ パラメータを指定するものである。
- 構造化変数が特殊な場合には複数のインスタンスを伴うことがある。それぞれは異なる名前を持ち、サーバで維持しているリソースが異なる。

MySQL 5.1 でサポートする構造化変数は 1 つです。これでキー変更操作のパラメータを指定します。キー キャッシュの構造化変数には次のコンポーネントがあります。

- [key_buffer_size](#)
- [key_cache_block_size](#)
- [key_cache_division_limit](#)
- [key_cache_age_threshold](#)

このセクションでは、構造化変数に関するシンタックスについて説明します。キー キャッシュ変数はシンタックス例で使用しますが、キー キャッシュがどのように操作を行うかに関しては [MyISAMキーキャッシュ](#) を参照してください。

構造化変数のインスタンスのコンポーネントを示すには、[instance_name.component_name](#) 形式でコンパウンド名を使用します。次は、この例示です。

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

それぞれの構造化システム変数には、[default](#) 名のインスタンスを常に事前定義します。インスタンス名なしで構造化変数のコンポーネントを示すと、[default](#) インスタンスを使用することになります。つまり、[default.key_buffer_size](#) と [key_buffer_size](#) の両方が同じシステム変数を指します。

構造化変数インスタンスとコンポーネントには次のネーミング ルールがあります。

- 任意型の構造化変数には、それぞれのインスタンスに、そのタイプの変数内で一意の名前を持たせる。ただし、インスタンス名は構造化変数型 全体 で一意である必要はない。たとえば、それぞれの構造化変数に [default](#) と名付けたインスタンスがある場合、[default](#) は変数型全体では一意ではない。
- それぞれの構造化変数型のコンポーネントの名前は、システム変数名全体を通じて一意である必要がある。そうならない場合、つまり、型の異なる 2 つの構造化変数でコンポーネント メンバ名を共有している場合、インスタンス名で修飾していないメンバ名を参照するときに、どの構造化変数を使用するかを判断できなくなる。
- 単純識別子 (unquoted identifier) としてはリーガルではない 構造化変数インスタンス名が、逆引用符を使用して単純識別子 (quoted identifier) になる (リーガルになる)。たとえば、[hot-cache](#) はリーガルでないが、`'hot-cache'` はリーガルである。
- [global](#)、[session](#)、[local](#) はリーガルのインスタンス名ではない。これは、非構造化システム変数を示すときに [@@global.var_name](#) などのノーテーションとの干渉を防ぐ。

現時点では、現在では、唯一の構造化変数の型がキー キャッシュのものであるため、最初の 2 ルールが違反対象になる可能性はありません。別の構造化変数の型を今後構築できれば、これらのルールはより大きな重要性を持つことになります。

例外として、単純変数名が発生するコンテキストに、コンパウンド名を使用する構造化変数コンポーネントを引き合わせるすることができます。たとえば、コマンドライン オプションで構造化変数に値を指定することができます。

```
shell> mysql -hot_cache.key_buffer_size=64K
```

オプション ファイルで、次のシンタックスを使用します。

```
[mysqlid]
```



```
hot_cache.key_buffer_size=64K
```

サーバをこのオプションで起動する場合、デフォルトの 8 MB のキー キャッシュに加えた、64 KB サイズの `hot_cache` と名づけられたキー キャッシュを生成することになります。

次のようにサーバを起動すると仮定した場合

```
shell> mysqld --key_buffer_size=256K \
--extra_cache.key_buffer_size=128K \
--extra_cache.key_cache_block_size=2048
```

この場合、サーバがデフォルト キー キャッシュを 256 KB に設定します。(`--default.key_buffer_size=256K` と記述することも可能。) さらに、このサーバは、 `extra_cache` と名づけた 128KB のセカンド キー キャッシュとともに、2048 キロバイトのブロック バッファをテーブル インデックス ブロックのキャッシュ用に作成します。

次は、サイズが 3:1:1 の割合の 3 つのキー キャッシュでサーバを起動するときの例です。

```
shell> mysqld --key_buffer_size=6M \
--hot_cache.key_buffer_size=2M \
--cold_cache.key_buffer_size=2M
```

構造化変数値は、ランタイムでの設定、読み取りも可能です。たとえば、 `hot_cache` という名前のキー キャッシュを 10 MB で設定するには、次のステートメントのどちらかを使用します。

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

キャッシュ サイズを読み取るには、次のことをします。

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

ただし、次のようなステートメントは作用しません。この変数はコンパウンド名としての解釈にならず、 `LIKE` 節のパターン一致操作の単純文字列としての解釈になります。

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

これは、構造化変数名を単純変数名が発生する可能性があるところで使用できるようにしておくためです。

動的システム変数

サーバのシステム変数の多くは動的で、 `SET GLOBAL` または `SET SESSION` を使用してランタイムで設定できます。 `SELECT` を使用して、値を取得することも可能です。 [システム変数の使用](#) を参照してください。

次のテーブルは、すべての動的システム変数の完全リストです。一番右のカラムはそれぞれの変数が、 `GLOBAL` または `SESSION`、あるいは両方、のどれで適用するかを示します。テーブルには、 `SET` ステートメントで設定できるセッション オプションをリストしています。 [SET 構文](#) では、これらのオプションについて説明しています。

「string」型の変数は文字列値で、「numeric」型の変数は数値です。「boolean」型の変数には 0、1、 `ON`、 `OFF` で設定します。(コマンドラインまたはオプション ファイルでこれらを指定するときは、数値を使用します。)

「enumeration」(列挙)としてマークしている変数は通常、その変数に対して利用可能な値の 1 つで設定しますが、目的とする値に相当する数字でも設定できます。列挙型では、最初の値は 0 に相当します。これは、最初の列挙値が 1 に相当する `ENUM` カラムとは異なります。

変数名	値のデータ型	タイプ
<code>autocommit</code>	boolean	<code>SESSION</code>
<code>automatic_sp_privileges</code>	boolean	<code>GLOBAL</code>
<code>big_tables</code>	boolean	<code>SESSION</code>
<code>binlog_cache_size</code>	numeric	<code>GLOBAL</code>
<code>binlog_format</code>	string	<code>GLOBAL SESSION</code>
<code>bulk_insert_buffer_size</code>	numeric	<code>GLOBAL SESSION</code>
<code>character_set_client</code>	string	<code>GLOBAL SESSION</code>
<code>character_set_connection</code>	string	<code>GLOBAL SESSION</code>
<code>character_set_filesystem</code>	string	<code>GLOBAL SESSION</code>

character_set_results	string	GLOBAL SESSION
character_set_server	string	GLOBAL SESSION
collation_connection	string	GLOBAL SESSION
collation_server	string	GLOBAL SESSION
completion_type	numeric	GLOBAL SESSION
concurrent_insert	numeric	GLOBAL
connect_timeout	numeric	GLOBAL
default_week_format	numeric	GLOBAL SESSION
delay_key_write	OFF ON ALL	GLOBAL
delayed_insert_limit	numeric	GLOBAL
delayed_insert_timeout	numeric	GLOBAL
delayed_queue_size	numeric	GLOBAL
div_precision_increment	numeric	GLOBAL SESSION
engine_condition_pushdown	boolean	GLOBAL SESSION
error_count	numeric	SESSION
event_scheduler	enumeration	GLOBAL
expire_logs_days	numeric	GLOBAL
flush	boolean	GLOBAL
flush_time	numeric	GLOBAL
foreign_key_checks	boolean	SESSION
ft_boolean_syntax	string	GLOBAL
general_log	boolean	GLOBAL
general_log_file	string	GLOBAL
group_concat_max_len	numeric	GLOBAL SESSION
identity	numeric	SESSION
innodb_autoextend_increment	numeric	GLOBAL
innodb_commit_concurrency	numeric	GLOBAL
innodb_concurrency_tickets	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_purge_lag	numeric	GLOBAL
innodb_support_xa	boolean	GLOBAL SESSION
innodb_sync_spin_loops	numeric	GLOBAL
innodb_table_locks	boolean	GLOBAL SESSION
innodb_thread_concurrency	numeric	GLOBAL
innodb_thread_sleep_delay	numeric	GLOBAL
insert_id	numeric	SESSION
interactive_timeout	numeric	GLOBAL SESSION
join_buffer_size	numeric	GLOBAL SESSION
key_buffer_size	numeric	GLOBAL
last_insert_id	numeric	SESSION
lc_time_names	string	GLOBAL SESSION
local_infile	boolean	GLOBAL

log_output	string	GLOBAL
log_queries_not_using_indexes	boolean	GLOBAL
log_warnings	numeric	GLOBAL
long_query_time	numeric	GLOBAL SESSION
low_priority_updates	boolean	GLOBAL SESSION
max_allowed_packet	numeric	GLOBAL SESSION
max_binlog_cache_size	numeric	GLOBAL
max_binlog_size	numeric	GLOBAL
max_connect_errors	numeric	GLOBAL
max_connections	numeric	GLOBAL
max_delayed_threads	numeric	GLOBAL
max_error_count	numeric	GLOBAL SESSION
max_heap_table_size	numeric	GLOBAL SESSION
max_insert_delayed_threads	numeric	GLOBAL
max_join_size	numeric	GLOBAL SESSION
max_prepared_stmt_count	numeric	GLOBAL
max_relay_log_size	numeric	GLOBAL
max_seeks_for_key	numeric	GLOBAL SESSION
max_sort_length	numeric	GLOBAL SESSION
max_tmp_tables	numeric	GLOBAL SESSION
max_user_connections	numeric	GLOBAL
max_write_lock_count	numeric	GLOBAL
multi_range_count	numeric	GLOBAL SESSION
myisam_data_pointer_size	numeric	GLOBAL
log_bin_trust_function_creators	boolean	GLOBAL
myisam_max_sort_file_size	numeric	GLOBAL SESSION
myisam_repair_threads	numeric	GLOBAL SESSION
myisam_sort_buffer_size	numeric	GLOBAL SESSION
myisam_stats_method	enum	GLOBAL SESSION
myisam_use_mmap	boolean	GLOBAL
ndb_extra_logging	numeric	GLOBAL
net_buffer_length	numeric	GLOBAL SESSION
net_read_timeout	numeric	GLOBAL SESSION
net_retry_count	numeric	GLOBAL SESSION
net_write_timeout	numeric	GLOBAL SESSION
old_passwords	numeric	GLOBAL SESSION
optimizer_prune_level	numeric	GLOBAL SESSION
optimizer_search_depth	numeric	GLOBAL SESSION
preload_buffer_size	numeric	GLOBAL SESSION
query_alloc_block_size	numeric	GLOBAL SESSION
query_cache_limit	numeric	GLOBAL
query_cache_size	numeric	GLOBAL

query_cache_type	enumeration	GLOBAL SESSION
query_cache_wlock_invalidate	boolean	GLOBAL SESSION
query_prealloc_size	numeric	GLOBAL SESSION
range_alloc_block_size	numeric	GLOBAL SESSION
read_buffer_size	numeric	GLOBAL SESSION
read_only	numeric	GLOBAL
read_rnd_buffer_size	numeric	GLOBAL SESSION
rpl_recovery_rank	numeric	GLOBAL
safe_show_database	boolean	GLOBAL
secure_auth	boolean	GLOBAL
server_id	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	numeric	GLOBAL
slave_transaction_retries	numeric	GLOBAL
slow_query_log	boolean	GLOBAL
slow_query_log_file	string	GLOBAL
slow_launch_time	numeric	GLOBAL
sort_buffer_size	numeric	GLOBAL SESSION
sql_auto_is_null	boolean	SESSION
sql_big_selects	boolean	SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION
sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL SESSION
sql_max_join_size	numeric	GLOBAL SESSION
sql_mode	enumeration	GLOBAL SESSION
sql_notes	boolean	SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
updatable_views_with_limit	enumeration	GLOBAL SESSION
sql_warnings	boolean	SESSION
sync_binlog	numeric	GLOBAL
sync_frm	boolean	GLOBAL
storage_engine	enumeration	GLOBAL SESSION
table_definition_cache	numeric	GLOBAL
table_open_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL SESSION
thread_cache_size	numeric	GLOBAL

time_zone	string	GLOBAL SESSION
timestamp	boolean	SESSION
tmp_table_size	enumeration	GLOBAL SESSION
transaction_alloc_block_size	numeric	GLOBAL SESSION
transaction_prealloc_size	numeric	GLOBAL SESSION
tx_isolation	enumeration	GLOBAL SESSION
unique_checks	boolean	SESSION
wait_timeout	numeric	GLOBAL SESSION
warning_count	numeric	SESSION

MySQL Enterprise

#####MySQL Network Monitoring and
Advisory Service ##### [http://www-jp.mysql.com/
products/enterprise/advisors.html](http://www-jp.mysql.com/products/enterprise/advisors.html) #####

ステータス変数

サーバには様々なステータス変数が存在し、オペレーションに関する情報交換をしています。その変数と値は、**SHOW [GLOBAL] STATUS** ステートメントを使用して、閲覧できます。オプションの **GLOBAL** キーワードは全体的な接続において、値を集約します。

```
mysql> SHOW GLOBAL STATUS;
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
| Bytes_sent | 1176560426 |
| ...
| Connections | 30023 |
| Created_tmp_disk_tables | 0 |
| Created_tmp_files | 3 |
| Created_tmp_tables | 2 |
| ...
| Threads_created | 217 |
| Threads_running | 88 |
| Uptime | 1389872 |
+-----+-----+
```

ステータス変数の多くは、**FLUSH STATUS** ステートメントで0にリセットされる。

次に様々なステータス変数を示します。バージョンについて記載のない変数は MySQL 5.1 より前から実装しています。実装履歴については、MySQL 5.0 Reference Manual を参照してください。

- **Aborted_clients**

接続を適切に閉じないままクライアントが終了したことが原因で中断した接続の数。 [Communication Errors and Aborted Connections](#) を参照のこと。

- **Aborted_connects**

MySQLサーバに接続しようとして失敗した回数。 [Communication Errors and Aborted Connections](#) を参照のこと。

- **Binlog_cache_disk_use**

binlog_cache_sizeの値を超えてテンポラリ ログ キャッシュを使用したトランザクションの回数。トランザクションからステートメントを保存するためにテンポラリ ファイルを使用した場合。

- **Binlog_cache_use**

テンポラリ バイナリ ログ キャッシュを使用したトランザクションの回数。

- [Bytes_received](#)

すべてのクライアントから受信したバイト数。

- [Bytes_sent](#)

すべてのクライアントへ送信したバイト数。

- [Com_xxx](#)

[Com_xxx](#) ステートメントのカウント値は、それぞれの [xxx](#) ステートメントを実行した回数を示す。それぞれのステートメント タイプをそれぞれカウントする。たとえば、[DELETE](#) で [Com_delete](#) を 1 とし、[INSERT](#) で [Com_insert](#) を 1 としてカウントする。クエリ結果がクエリ キャッシュから返される場合は、サーバは [Qcache_hits](#) ステータス値の増加になる。[Com_select](#) ではないことに注意が必要。[クエリ キャッシュのステータスと保守](#) を参照のこと。

[Com_stmt_xxx](#) 値のすべては、準備された文 (prepared statement) の引数が未知、または実行中にエラーが発生した場合でも増加する。つまり、この値は、要求発行回数に対応し、要求を完了 (成功) した回数ではない。

[Com_stmt_xxx](#) ステータス変数は次の通り。

- [Com_stmt_prepare](#)
- [Com_stmt_execute](#)
- [Com_stmt_fetch](#)
- [Com_stmt_send_long_data](#)
- [Com_stmt_reset](#)
- [Com_stmt_close](#)

これらの変数は、準備された文 (prepared statement)。名前は ネットワーク レイヤーでし使用した [COM_xxx](#)

- のコマンド セットを示す。つまり、[mysql_stmt_prepare\(\)](#) や [mysql_stmt_execute\(\)](#) など、準備されたステートメントの API コールを実行すると、これらの値は増加する。[PREPARE](#)、[EXECUTE](#)、[DEALLOCATE PREPARE](#) は [Com_stmt_prepare](#)、[Com_stmt_execute](#)、[Com_stmt_close](#) に対応しているので、同様に増加する。さらに、MySQL 4.1.3 から実装しているため、古いステートメントのカウント値は、[PREPARE](#)、[EXECUTE](#)、[DEALLOCATE PREPARE](#) が [Com_prepare_sql](#)、[Com_execute_sql](#)、[Com_dealloc_sql](#) に対応しているため、これも同様に増加する。[Com_stmt_fetch](#) はカーソルからフェッチしたときにネットワーク往復を発行した合計回数のこと。

- [Compression](#)

接続でクライアント/サーバ間の圧縮プロトコルを使用しているかどうか (を指す)。(MySQL 5.1.2での追加)

- [Connections](#)

(成功/不成功に関わらず) MySQL サーバへの接続試行回数。

- [Created_tmp_disk_tables](#)

ステートメント実行中に、ディスク上に作成された暗黙的テンポラリテーブルの数。

- [Created_tmp_files](#)

[mysqld](#) が生成したテンポラリ ファイルの数

- [Created_tmp_tables](#)

ステートメント実行中に、メモリ上に作成された暗黙的テンポラリテーブルの数。[Created_tmp_disk_tables](#) の値が大きい場合には、[tmp_table_size](#) の値も大きくすることによって、テンポラリテーブルをディスクベースではなくメモリベースにすることもできる。

- [Delayed_errors](#)
エラー発生 ([duplicate key](#) の可能性) により、[INSERT DELAYED](#) で書き込まれたレコードの数。
- [Delayed_insert_threads](#)
[INSERT DELAYED](#) ハンドラ スレッドの数。
- [Delayed_writes](#)
[INSERT DELAYED](#) で書き込んだレコードの数。
- [Flush_commands](#)
[FLUSH](#) コマンドの実行回数。
- [Handler_commit](#)
内部 [COMMIT](#) コマンド数。
- [Handler_delete](#)
テーブルからレコードを削除した回数。
- [Handler_discover](#)
MySQL サーバが [NDB Cluster](#) ストレージエンジンに 指定の名前を持ったテーブル認識しているかどうかを問い合わせることができる。この操作をディスカバリー (discovery) と呼ぶ。 [Handler_discover](#) 値は、ディスカバーした回数。
- [Handler_prepare](#)
2 フェーズ コミット操作の準備フェーズのカウンタ。
- [Handler_read_first](#)
インデックスから最初のエントリを読み取った回数。この値が大きい場合、サーバが何回もフル インデックス スキャンを実行している可能性がある。たとえば、[SELECT col1 FROM foo](#) を実行したときに、[col1](#) がインデックスになっている場合など。
- [Handler_read_key](#)
キーに基づいたレコード読み込み要求を受けた回数。この値が大きい場合、クエリをテーブルへ適切にインデックス化していることを示す。
- [Handler_read_next](#)
キー順序で次レコードの読み込み要求を受けた回数。範囲指定をしてインデックス カラムに対してクエリを実行すると、これがインクリメントする。インデックス スキャンを実行した場合もインクリメントする。
- [Handler_read_prev](#)
キー順序で前レコードの読み込み要求を受けた回数。この読み取り方法は主に、[ORDER BY ... DESC](#) の最適化に使用している。
- [Handler_read_rnd](#)
固定位置に基づいたレコード読み込み要求を受けた回数。結果のソートを必要とするクエリを多く実行すると、この値が大きくなる。MySQL でテーブルの全件スキャンや適切にキーを使えない結合を持ったクエリがある可能性がある。
- [Handler_read_rnd_next](#)
データ ファイルで次レコードの読み取り要求を受けた回数。テーブル スキャンの実行が多いと、この値が大きくなる。これは通常、テーブルに適切なインデックス化できない、またはインデックスを利用できないクエリを発行していることを意味する。

- [Handler_rollback](#)

ROLLBACKを内部的 (ストレージ エンジン) に実行した回数。

- [Handler_savepoint](#)

内部的 (ストレージ エンジン) に セーブポイント (savepoint) の配置要求回数。

- [Handler_savepoint_rollback](#)

内部的 (ストレージ エンジン) に セーブポイント (savepoint) へロールバック要求回数。

- [Handler_update](#)

テーブル内のレコードの更新要求回数。

- [Handler_write](#)

テーブルへのレコードの挿入要求回数。

- [Innodb_buffer_pool_pages_data](#)

データがあるページ数 (ダーティ または クリーン)。

- [Innodb_buffer_pool_pages_dirty](#)

ダーティ ページの数。

- [Innodb_buffer_pool_pages_flushed](#)

InnoDB がキャッシュするために使用するメモリ バッファの数。

- [Innodb_buffer_pool_pages_free](#)

空き容量

- [Innodb_buffer_pool_pages_latched](#)

InnoDB のメモリ バッファでラッチした数。データが読み込みまたは書き込みの対象になっていて、フラッシュまたは削除が何らかの理由でできない状態。

- [Innodb_buffer_pool_pages_misc](#)

レコード ロックまたはアダプティブなハッシュ インデックスなどにより、オーバーヘッドの割り当てになったビジー (busy) 状態のデータ。この値は `Innodb_buffer_pool_pages_total - Innodb_buffer_pool_pages_free - Innodb_buffer_pool_pages_data` で算出。

- [Innodb_buffer_pool_pages_total](#)

ページのメモリ バッファの合計サイズ。

- [Innodb_buffer_pool_read_ahead_rnd](#)

InnoDB が開始した 「random (ランダム)」 先読みの数。大型テーブルのクエリ スキャンをランダムに行うと発生する。

- [Innodb_buffer_pool_read_ahead_seq](#)

InnoDB が開始した順次的な先読み数。InnoDBが 順次的にフル テーブル スキャンを行うときに発生する。

- [Innodb_buffer_pool_read_requests](#)

InnoDB が行った論理読み込みの数

- [Innodb_buffer_pool_reads](#)

InnoDBがバッファ プールの内容を利用できず、シングル ページ読み込みを行わなければならなかった論理読み込みの回数

- [Innodb_buffer_pool_wait_free](#)

通常、InnoDB バッファ プールへの書き込みはバックグラウンドで行うが、ページの読み込みまたは作成を行う必要があるのに対して、クリーン ページが得られない場合に、まずそのページがフラッシュするまで待つ必要がある。このカウンタは、その待機回数をカウントする。バッファ プールの値が適切に設定すると、この値は小さくなる。

- [Innodb_buffer_pool_write_requests](#)

InnoDB バッファプールへの書き込み数。

- [Innodb_data_fsyncs](#)

ここまでの `fsync()` 操作数。

- [Innodb_data_pending_fsyncs](#)

現在の `fsync()` 操作保留 (pending) の数。

- [Innodb_data_pending_reads](#)

現在の読み込み保留の数。

- [Innodb_data_pending_writes](#)

現在の書き込み保留の数。

- [Innodb_data_read](#)

ここまでのデータの読み込み量 (単位:バイト)。

- [Innodb_data_reads](#)

データ読み込みの合計数。

- [Innodb_data_writes](#)

データ書き込みの合計数。

- [Innodb_data_written](#)

ここまでのデータの書き込み量 (単位:バイト)。

- [Innodb_dblwr_writes](#), [Innodb_dblwr_pages_written](#)

二重書き込みの実行回数と、二重書き込みが発生したページ数。[InnoDB ディスク I/O](#) を参照のこと。.

- [Innodb_log_waits](#)

ログ バッファが小さすぎるために、作業を継続する前にフラッシュ要求で待機した回数。

- [Innodb_log_write_requests](#)

要求ログ書き込みの回数。

- [Innodb_log_writes](#)

ログ ファイルへの物理的な書き込みの回数。

- [Innodb_os_log_fsyncs](#)

ログ ファイルの `fsync()` 書き込みをした回数。

- `InnoDB_os_log_pending_fsyncs`
`fsync()` 待ちのログ ファイル数。
- `InnoDB_os_log_pending_writes`
ログ ファイルの書き込みの保留回数。
- `InnoDB_os_log_written`
ログ ファイルへの書き込みの回数。
- `InnoDB_page_size`
コンパイル時の `InnoDB` ページ サイズ (デフォルト 16KB)。多くの値がページ カウントの対象になり、ページ サイズは、それらを容易なバイト変換を可能にする。
- `InnoDB_pages_created`
作成したページの数。
- `InnoDB_pages_read`
読み込みしたページの数。
- `InnoDB_pages_written`
書き込みしたページの数。
- `InnoDB_row_lock_current_waits`
現在待機している行ロック (row lock) の数。
- `InnoDB_row_lock_time`
行ロック (row lock)、列の獲得に使用した合計時間 (単位 : ミリ秒)。
- `InnoDB_row_lock_time_avg`
行ロック (row lock)、列の獲得に使用した平均時間 (単位 : ミリ秒)。
- `InnoDB_row_lock_time_max`
行ロック (row lock)、列の獲得に使用した最長時間 (単位 : ミリ秒)。
- `InnoDB_row_lock_waits`
行ロックで待機する必要があった回数。
- `InnoDB_rows_deleted`
`InnoDB` テーブルから削除したレコード数。
- `InnoDB_rows_inserted`
`InnoDB` テーブルへの挿入レコード数。
- `InnoDB_rows_read`
`InnoDB` テーブルからの読み込みレコード数。
- `InnoDB_rows_updated`
`InnoDB` テーブルでの更新レコード数。
- `Key_blocks_not_flushed`
変更後に、まだディスクに未フラッシュのキー キャッシュのキー ブロックの数。

- [Key_blocks_unused](#)

キーキャッシュの未使用ブロックの数。どれだけ使用しているか測定するためにこの値を使用することができません。[システム変数](#)において[key_buffer_size](#)に関する説明を参照のこと。

- [Key_blocks_used](#)

キーキャッシュのブロックの使用数。この値は、MySQLが起動してから現在に至るまでに同時に使用された最大のブロック数を示します。

- [Key_read_requests](#)

キャッシュからキー ブロックを読み込んだリクエスト数。

- [Key_reads](#)

ディスクからのキーブロックの物理的読み込み回数。[Key_reads](#) が大きい場合、[key_buffer_size](#) の値が小さい可能性がある。キャッシュ ミス率は $\text{Key\_reads} / \text{Key\_read\_requests}$ と計算する。

- [Key_write_requests](#)

キャッシュへのキーブロックの書き込んだリクエスト数。

- [Key_writes](#)

ディスクへのキー ブロックの物理的な書き込み回数。

- [Last_query_cost](#)

クエリ オプティマイザが計算し、最後にコンパイルしたクエリの全コスト。同じクエリの異なるクエリ プランのコストを比較するときに使用する。デフォルト値 0 は、クエリがまだ未コンパイルであることを意味する。[Last_query_cost](#) にはセッション スコープがある。

- [Max_used_connections](#)

サーバが起動してから同時使用した接続の最大数。

- [Ndb_cluster_node_id](#)

サーバが MySQL Cluster ノードとして作用している場合、この値はそのクラスタのノード ID。

サーバが MySQL Cluster の一部ではない場合、この値は 0。

- [Ndb_config_from_host](#)

サーバが MySQL Cluster の一部である場合、この値は Cluster マネージメント サーバのホスト名または IP アドレスで、コンフィギュレーション データを取得する。

サーバが MySQL Cluster の一部ではない場合、この値は空文字列。

MySQL 5.1.12 以前では、この変数は [Ndb_connected_host](#) と呼ばれていた。

- [Ndb_config_from_port](#)

サーバが MySQL Cluster の一部である場合、この値はポート番号で、ここから、コンフィギュレーション データを取得する Cluster マネージメント サーバに接続。

サーバが MySQL Cluster の一部ではない場合、この値は 0。

MySQL 5.1.12 以前では、この変数は [Ndb_connected_port](#) と呼ばれていた。

- [Ndb_number_of_data_nodes](#)

サーバが MySQL Cluster の一部である場合、この値はクラスタのデータ ノードの数。

サーバが MySQL Cluster の一部ではない場合、この値は 0。

MySQL 5.1.12 以前では、この変数は `Ndb_number_of_storage_nodes` と呼ばれていた。

- `Not_flushed_delayed_rows`

`INSERT DELAY` 行列への書き込み待ちの行数。

- `Open_files`

開いているファイルの数。

- `Open_streams`

開いているストリームの数。(主に、ログの記録で使用)

- `Open_tables`

開いているテーブルの数。

- `Opened_tables`

これまでに開いたテーブル数。`Opened_tables` の値が多い場合、`table_open_cache` の値が小さい可能性がある。

- `Prepared_stmt_count`

準備されたステートメントの現在の数。`max_prepared_stmt_count` で最大値を与える。(MySQL 5.1.14 での追加)

- `Qcache_free_blocks`

クエリ キャッシュ内の空きメモリ ブロックの数

- `Qcache_free_memory`

クエリ キャッシュ内の空きメモリ ブロック量

- `Qcache_hits`

クエリ キャッシュ ヒットの数。

- `Qcache_inserts`

キャッシュに追加したクエリ数。

- `Qcache_lowmem_prunes`

メモリ不足を解消するために、クエリ キャッシュから削除されたクエリの数。

- `Qcache_not_cached`

キャッシュしないクエリの数。(キャッシュできないか、または `query_cache_type` でキャッシュしない設定)

- `Qcache_queries_in_cache`

クエリ キャッシュに登録したクエリ数。

- `Qcache_total_blocks`

クエリ キャッシュの合計ブロック数。

- `Questions`

サーバに送信したクエリ数。

- `Rpl_status`

フェイル セーフ (安全装備) レプリケーションのステータス。未実装。

- `Select_full_join`

インデックスを使用しない結合の数 (テーブル スキャン)。この値が 0 でない場合、テーブルのインデックスを調べることに。

- [Select_full_range_join](#)

関連テーブルで範囲検索を使用した結合の数。

- [Select_range](#)

ファースト テーブルで範囲指定した部分を使用した結合の数。通常は、この数値が大きくても、それほど致命的な問題にはならない。

- [Select_range_check](#)

キーなしの結合の数。これは、それぞれのレコードのあとにキー使用をチェックする。この値が 0 でない場合、テーブルのインデックスをチェックする必要がある。

- [Select_scan](#)

ファースト テーブルのフル スキャンを行った結合の数。

- [Slave_open_temp_tables](#)

スレーブの SQL スレッドで現在開いているテンポラリ テーブルの数。

- [Slave_running](#)

サーバがマスタに接続しているスレーブの場合は、この値は **ON**。

- [Slave_retried_transactions](#)

起動時から、レプリケーション スレーブの SQL スレッドがトランザクションを再試行した回数のこと。

- [Slow_launch_threads](#)

[slow_launch_time](#) 秒よりも、作成時間を要したスレッドの数。

- [Slow_queries](#)

[slow_launch_time](#) 秒よりも、作成時間を要したクエリの数。[スロー クエリ ログ](#) を参照のこと。

- [Sort_merge_passes](#)

ソート アルゴリズムで必要としたマージ パスの回数。この値が大きい場合は、[sort_buffer_size](#) の値を大きくすることを検討する。

- [Sort_range](#)

範囲指定のソートを行った回数。

- [Sort_rows](#)

ソートしたレコードの数。

- [Sort_scan](#)

テーブル スキャンでソートした回数。

- [Ssl_xxx](#)

SSL接続に使用する変数 (ステータス)。

- [Table_locks_immediate](#)

テーブル ロックを直ちに実行した回数。

- [Table_locks_waited](#)

テーブル ロックをすぐには実行できず、待機が必要だった回数。この値が大きい場合、パフォーマンス上の問題がある。まずクエリを最適化し、次にテーブルを分割するかレプリケーションを行う。

- [Tc_log_max_pages_used](#)

`mysqld` で使用するログのメモリ マップ実装では、それ自体が内部の XA トランザクションのリカバリに対してトランザクション コーディネータとして作用するとき、この値は、サーバが起動してからログに使用したページの最大数を示す。[Tc_log_max_pages_used](#) と [Tc_log_page_size](#) の積が常に、ログ サイズよりも極端に小さい場合は、そのサイズは必要以上に大きいことを示し、減らすことを検討する。このサイズは、[--log-tc-size](#) オプションで指定できる。現在、この変数は未使用。これはバイナリ ログをベースとしたリカバリには不要で、メモリ マップのリカバリ ログ方法は、ストレージ エンジンが 2 フェーズ コミットより大きなものを許容できる場合を除き、使用しない。`InnoDB` は唯一、対応できるエンジンである。

- [Tc_log_page_size](#)

XA リカバリ ログのメモリ マップ実装のページ サイズ。デフォルトには `getpagesize()` を使用して値を決める。現在、この変数は未使用で、その理由は、[Tc_log_max_pages_used](#) に記述したものと同じである。

- [Tc_log_page_waits](#)

リカバリ ログのメモリ マップ実装で、この値は、サーバがトランザクションにコミットできず、ログの空きを待機する必要があると、その度にインクリメント (増加) する。この値が大きい場合、[--log-tc-size](#) オプションでログ サイズの増加を検討する。バイナリ ログをベースとしたリカバリでは、この値は、バイナリ ログが 2 フェーズ コミット中のために閉じることができない場合に、その度にインクリメントする。これは、対象となるすべてのトランザクションが完了するまで待機となる。

- [Threads_cached](#)

スレッド キャッシュ内のスレッド数。

- [Threads_connected](#)

現在開いている接続の数。

- [Threads_created](#)

接続を処理で作成されたスレッドの数。[Threads_created](#) の値が大きい場合、[thread_cache_size](#) の値を大きくしてキャッシュ サイズを増やす。キャッシュヒット率の計算は [Threads_created/Connections](#) とする。

- [Threads_running](#)

スリープ状態になっていないスレッドの数。

- [Uptime](#)

サーバ起動時からの経過秒数。

SQL モード

MySQL サーバは様々な MySQL モードで動作し、モードをクライアント毎に別々に設定できます。この機能により、各アプリケーションがそれぞれの要件に応じてサーバのオペレーティングモードを指定することができるようになります。

MySQL での SQL モードに関する FAQ は、[MySQL 5.1 FAQ — Server SQL Mode](#) を参照してください。

モードとは、どの SQL シンタックスを MySQL がサポートし、どのようなデータ バリデーション チェックを実行すべきかを定義するものです。これにより、異なる環境で MySQL を使用したり、MySQL を他のデータベースサーバと併用したりするのが容易になります。

デフォルトの SQL モードを指定するには、[--sql-mode="modes"](#) オプションで `mysqld` を立ち上げます。または、Unix では `my.cnf` に、Window では `my.ini` に、[sql-mode="modes"](#) を記述します。`modes` とは、カンマ (「,」) で区切られた各モードのリストです。デフォルトは空の状態、モード設定がないことを意味します。明示したい場合は、`modes` の値は空にできません。それには、コマンドラインで [--sql-mode=""](#) オプションを使用するか、あるいは、Unix では、`my.cnf` の、Windows では `my.ini` の [sql-mode=""](#) オプションを使用します。

実行中に SQL モードを変更することもできます。それには、`SET [GLOBAL|SESSION] sql_mode='modes'` 文で `sql_mode` の値を指定します。`GLOBAL` 値を指定するには、`SUPER` 権限が必要になり、また、それ以降に接続するすべてのクライアント操作に影響します。`SESSION` 値を設定するには、現在設定を行った クライアントだけに影響します。クライアントは自身の `sql_mode` セッション値をいつでも変更できます。

次のステートメントで、現行の `sql_mode` のグローバル値とセッション値を読み取ることができます。

```
SELECT @@global.sql_mode;
SELECT @@session.sql_mode;
```

次に、最も重要な `sql_mode` 値を示します。

- **ANSI**

このモードは、標準 SQL とより同調できるように、シンタックス (構文) と動作を変更する。

- **STRICT_TRANS_TABLES**

指定された値をトランザクション テーブルに挿入できない場合、クエリの実行を中断する。非トランザクション テーブルでは、値が 1 行ステートメントの場合、または複数行ステートメントの最初の行である場合に、クエリを中断する。詳細は、このセクションでの後述を参照のこと。

- **TRADITIONAL**

MySQL を「従来型の」SQL データベース システムのように動作させる。このモードの最もシンプルな特徴は、カラムに不正値を挿入するときに、「警告ではなく、エラー メッセージ」を返すことである。注意：`INSERT/UPDATE` は、エラーを認識すると即座に中断する。そのため、非トランザクション式のストレージ エンジンを使用している場合は、使用しない (推奨)。エラー前に変更したデータがロール バックしないため、更新が「部分的に」完了してしまうことがある。

このマニュアルで、「Strict Mode」と示すときは、`STRICT_TRANS_TABLES`、`STRICT_ALL_TABLES` の少なくともどちらかが有効化しているモードである、という意味です。

次のリストはサポートしている全モードの説明です。

- **ALLOW_INVALID_DATES**

日付の完全チェックを行わずに、月は 1 から 12 まで、日は 1 から 31 までであることだけをチェックする。これは、たとえば、Web アプリケーションなどで年月日をそれぞれのフィールドで取得し、(日付に関するバリデーションなしで) ユーザが指定した通りに保存する場合に便利である。このモードは `DATE` と `DATETIME` のカラムに適用する。`TIMESTAMP` カラムは常に有効な日付を必要とするため、対象外である。

サーバでは月日の値は正当である必要があり、単に 1 から 12 そして 1 to 31 の範囲であるだけではいけない。Strict Mode を無効化した場合、`'2004-04-31'` のような入力は無効とみなし、`'0000-00-00'` と修正される。そのときには、警告が出される。Strict Mode を有効にした場合、無効とみなす日はエラーとなる。`'2004-04-31'` を許すには、`ALLOW_INVALID_DATES` を有効にする。

- **ANSI_QUOTES**

引用文字 ```` のように、`""` を識別子として扱う。文字列の引用文字ではない。このモードを有効にした場合には、```` を識別子として使用できる。`ANSI_QUOTES` を有効にした場合には、識別子として解釈されるため、リテラル文字列の引用には二重引用符を使用できなくなる。

- **ERROR_FOR_DIVISION_BY_ZERO**

`INSERT` または `UPDATE` 中に、0 で除算 (または `MOD(X,0)`) すると、Strict Mode でエラーを生成する (そうでなければ、警告)。このモードを有効にしない場合、MySQL は 0 除算 に対して、`NULL` を返す。`INSERT IGNORE` または `UPDATE IGNORE` では、MySQL が 0 除算 に対して警告を生成するが、操作結果は `NULL` になる。

- **HIGH_NOT_PRECEDENCE**

`NOT` 演算子の優先順位は、`NOT a BETWEEN b AND c` を `NOT (a BETWEEN b AND c)` として構文解析するようなプログラム式である。古い MySQL バージョンでは、`(NOT a) BETWEEN b AND c` として構文解析している。以前の優先順位動作は、SQL モードの `HIGH_NOT_PRECEDENCE` を使用すると可能になる。

```
mysql> SET sql_mode = "";
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 1
```

- **IGNORE_SPACE**

関数名と '(' 文字のスペースを許可する。これは、組込み関数名を予約語として扱うようにする。そのため、関数名と同一の識別子を引用符で囲む必要がある。詳細は [識別子](#) を参照のこと。たとえば、[COUNT\(\)](#) という関数があった場合、[count](#) をテーブル名として使用すると、次のようなステートメントでエラーが発生する。

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

テーブル名は次のようにする。

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

IGNORE_SPACE モードは、ユーザ定義関数 (UDF) または格納された関数ではなく、組み込み関数に適用する。**IGNORE_SPACE** を有効化しているかどうかに関わらず、ユーザ定義関数または格納された関数の後ろにスペースを入れることは常に可能である。

IGNORE_SPACE に関する詳細は、[関数名の構文解析と名前解決](#) を参照のこと。

- **NO_AUTO_CREATE_USER**

GRANT 文が自動的に新規ユーザを作成しないようにする。空でないパスワードも指定した場合を除く。

- **NO_AUTO_VALUE_ON_ZERO**

NO_AUTO_VALUE_ON_ZERO は **AUTO_INCREMENT** カラム処理に影響する。通常、次のシーケンス番号をカラムに生成するときには、**NULL** または **0** を挿入する。**NO_AUTO_VALUE_ON_ZERO** は **0** の動作を抑制するため、**NULL** が次のシーケンス番号を生成する。

このモードでは、**0** をテーブルの **AUTO_INCREMENT** カラムに格納する。しかし、**0** を保存するということは、推奨できる方法ではない。たとえば、[mysqldump](#) コマンドでテーブルにダンプして、リロードする場合に、MySQL は、**0** という値に遭遇すると、新たなシーケンス番号を生成し、テーブルにダンプしたものと異なる内容になるという結果を生む。ダンプ ファイルをリロードする前に **NO_AUTO_VALUE_ON_ZERO** を有効にすると、この問題を回避できる。現在、[mysqldump](#) は、自動的にその出力に、**NO_AUTO_VALUE_ON_ZERO** を有効にするステートメントを含むようになったので、これまでの問題を回避できる。

- **NO_BACKSLASH_ESCAPES**

文字列内のエスケープ文字としてのバックスラッシュ ('\') を無効にする。このモードを有効にすると、バックスラッシュが特殊文字でなく通常の文字として扱われる。

- **NO_DIR_IN_CREATE**

テーブル作成時に、**INDEX DIRECTORY** と **DATA DIRECTORY** の命令をすべて無視する。このオプションはスレーブのレプリケーション サーバで役立つ。

- **NO_ENGINE_SUBSTITUTION**

デフォルトのストレージ エンジンの自動置換 (substitution) を防ぐ。これは、**CREATE TABLE** のようなステートメントが、無効化した、またはコンパイルしたストレージ エンジンを指定するときのこと。エラーで知らせる。

- **NO_FIELD_OPTIONS**

MySQL 独自のカラムオプションを **SHOW CREATE TABLE** の出力に印字しない。このモードは [mysqldump](#) コマンドのポータビリティモードで使用される。

- **NO_KEY_OPTIONS**

MySQL 独自のインデックスオプションを `SHOW CREATE TABLE` の出力に印字しない。このモードは `mysqldump` コマンドのポータビリティモードで使用される。

- `NO_TABLE_OPTIONS`

MySQL 特化のテーブル オプション (`ENGINE` など) を `SHOW CREATE TABLE` の出力に印字しない。ポータビリティ モードで `mysqldump` コマンドを使用する。

- `NO_UNSIGNED_SUBTRACTION`

整数引き算操作で、オペランド (演算数) の一つに符号がない場合には、結果を `UNSIGNED` としてマークしない。つまり、引き算の結果はモードに効力がある場合は常に符号があり、オペランドの一つに符号がない場合でも同様。たとえば、テーブル `t1` のカラム `c2` のタイプと、テーブル `t2` の `c2` のタイプを比較すると、次のようになる。

```
mysql> SET SQL_MODE="";
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| c2    | bigint(21) unsigned |      |      | 0        |       |
+-----+-----+-----+-----+-----+

mysql> SET SQL_MODE='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| c2    | bigint(21)    |      |      | 0        |       |
+-----+-----+-----+-----+-----+
```

ノート：これは `BIGINT UNSIGNED` はすべての文脈で完全には使用できないことを意味する。[キャスト関数と演算子](#) を参照のこと。

```
mysql> SET SQL_MODE = "";
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
| 18446744073709551615 |
+-----+

mysql> SET SQL_MODE = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
| -1 |
+-----+
```

- `NO_ZERO_DATE`

Strict Mode では、`'0000-00-00'` は有効な日として扱わない。`IGNORE` オプションで「ゼロ」の日付を挿入する。Strict Mode ではない場合は、この日付は有効になるが、警告が出る。

- `NO_ZERO_IN_DATE`

Strict Mode では、月日に 0 があつた場合は、許可されない。`IGNORE` オプションで併用した場合は、MySQL が `'0000-00-00'` と入れ替える。Strict Mode ではない場合は、日付は有効になるが、警告が出る。

- `ONLY_FULL_GROUP_BY`

`SELECT` リストのクエリを許可しない。このリストは `GROUP BY` 節で名付けていない未集約のカラムを指す。次のクエリは、`address` が `GROUP BY` 節で名付けていないので、このモードを有効にした場合は無効になる。

```
SELECT name, address, MAX(age) FROM t GROUP BY name;
```

MySQL 5.1.11 以後、このモードは、[GROUP BY](#) 節で名付けていない [HAVING](#) の未集約カラムへのリファレンスを制限する。

- [PIPES_AS_CONCAT](#)

`||` を連結演算子として扱う。([CONCAT\(\)](#) と同様に。) [OR](#) のシノニムとしては扱わない。

- [REAL_AS_FLOAT](#)

[REAL](#) を [FLOAT](#) のシノニムとして扱う。デフォルトでは、MySQL が [REAL](#) を [DOUBLE](#) のシノニムとして扱う。

- [STRICT_ALL_TABLES](#)

すべてのステータス エンジンに対して、Strict Mode を有効にする。無効データは排除の対象になる。詳細は、次のパラグラフを参照のこと。

- [STRICT_TRANS_TABLES](#)

トランザクションのストレージ エンジンに対して、Strict Mode を有効にする。非トランザクションのストレージ エンジンに対しても可能な場合がある。詳細は次のパラグラフを参照のこと。

Strict Mode は、MySQL が無効または不明な入力値をどのように処理するかを制御します。何らかの理由で値は無効になることがあります。たとえば、カラムに対して違うデータの入力、範囲を超える入力などがあった場合です。値が不明であるということは、挿入する新規のレコードに、カラム値がない場合で、定義に、明示的な [DEFAULT](#) 節がないときです。

トランザクション テーブルでは、無効または不明な値がクエリにある場合はエラーになります。これは、[STRICT_ALL_TABLES](#) または [STRICT_TRANS_TABLES](#) のどちらかのモードが有効になっている場合に起こります。クエリは中断、そしてロールバックの対象になります。

最初の行で挿入または更新するときに、bad 値が発生する場合、非トランザクションのテーブルでは、どちらのモードでも同様に動作します。クエリ処理は中断し、テーブルはそのまま変更なしの状態になります。クエリを挿入または複数行を変更した場合に、bad 値が2行目以降に発生する場合は、結果はどの制限オプションを有効に設定しているかに依存します。

- [STRICT_ALL_TABLES](#) では、MySQL はエラーを返し、残りの行を無視する。そのような場合は、最初の段階の行がまだ挿入中または更新中であり、部分的な更新を得ることになり、それが予期していたものとは異なる可能性がある。これを回避するには、単行ステートメントを使用すると、テーブルを変更することなく、中断できる。
- [STRICT_TRANS_TABLES](#) では、MySQL は無効値を、カラムに対して最も近い有効値に置換し、調整した値を挿入する。値が不明である場合は、MySQL は、カラムのデータ タイプに明示的なデフォルト値を挿入する。どちらの場合でも、MySQL はエラーではなく、警告を出し、クエリ処理を続行する。明示的なデフォルトに関しては、[データタイプデフォルト値](#) を参照のこと。

Strict Mode では、`'2004-04-31'` を無効な日付として扱います。`'2004-04-00'` または「ゼロ」日など、0 が部分的に含まれた日付は無効ではない。これらも無効として扱うには、Strict Mode の他に、[NO_ZERO_IN_DATE](#) と [NO_ZERO_DATE](#) の SQL モードも追加する。

Strict Mode を使用しない場合、つまり、[STRICT_TRANS_TABLES](#) または [STRICT_ALL_TABLES](#) のどちらかを有効にした場合、MySQL は無効または不明な値の代わりに調整した値を挿入し、警告を出す。Strict Mode では、[INSERT IGNORE](#) または [UPDATE IGNORE](#) を使用して、このような動作を導くことが可能。[SHOW WARNINGS](#) 構文を参照のこと。

次に示すモードは特化型のモードで、前述したモード値のコンビネーションの省略表現でもあります。

説明にあるモード値は、最新の MySQL バージョンで利用できます。古いバージョンの場合は、コンビネーションモードで新しいバージョンでだけ使えるモードを使用しているため、利用できません。

- [ANSI](#)

[REAL_AS_FLOAT](#)、[PIPES_AS_CONCAT](#)、[ANSI_QUOTES](#)、[IGNORE_SPACE](#) に相当。[ANSIモードでのMySQLの実行](#) を参照のこと。

- [DB2](#)

PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS に相当。

- MAXDB

PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_AUTO_CREATE_USER に相当。

- MSSQL

PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS に相当。

- MYSQL323

NO_FIELD_OPTIONS、HIGH_NOT_PRECEDENCE に相当。

- MYSQL40

NO_FIELD_OPTIONS、HIGH_NOT_PRECEDENCE に相当。

- ORACLE

PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_AUTO_CREATE_USER に相当。

- POSTGRESQL

PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS に相当。

- TRADITIONAL

STRICT_TRANS_TABLES、STRICT_ALL_TABLES、NO_ZERO_IN_DATE、NO_ZERO_DATE、ERROR_FOR_DIVISION_BY_ZERO、NO_AUTO_CREATE_USER に相当。

シャットダウン プロセス

サーバのシャットダウン プロセスには次のことが行われます。

1. シャットダウン プロセスの開始

サーバのシャットダウン (システム終了) には様々な方法があります。たとえば、**SHUTDOWN** 権限を持つユーザは、`mysqladmin shutdown` コマンドを実行できます。MySQL がサポートしているプラットフォームであれば `mysqladmin` コマンドを使用します。OS に対応したシャットダウンの開始方法として、Unix 環境のシャットダウンでは、サーバが **SIGTERM** を受けるとシャットダウンし、Windows 環境では、サービス マネージャの指示でシャットダウンします。

2. サーバによるシャットダウン スレッドの作成 (必要に応じて)

シャットダウンの開始方法によっては、サーバがシャットダウン プロセスを処理するスレッドを作成することがあります。シャットダウンがクライアントからの要求によるのであった場合に、シャットダウン スレッドが作成されます。シャットダウンが **SIGTERM** を受信したことによるものである場合、シャットダウン処理をグナル スレッド (**SIGTERM** に対して) が行うことがあります。または、別のスレッドを作成して、その処理を行うことがあります。サーバがシャットダウン スレッドを作成しようとして、作成できない場合には、サーバは診断メッセージをエラー ログに表示します。これは、メモリ不足などの場合に発生します。

```
Error: Can't create thread to kill server
```

3. サーバによる新規接続の拒否

シャットダウン中に新たなアクティビティを開始しないようにするために、サーバは新たなクライアントからの接続を拒否します。これは、P ポート、Unix のソケット ファイル、Windows の名前付きパイプや共有ファイルなどの接続に対して、ネットワーク接続を閉じる、という方法を取ります。

4. サーバによる現行のアクティビティの停止

クライアント接続に関連しているスレッドでは、クライアントへの接続を止め、そのスレッドはバイタルを失った（無くなった）ものとしてマークします。スレッドはそうようにマークされたことを認識し、消滅します。アイドル接続のスレッドは、簡単に消滅します。クエリを処理中のスレッドはその程度を定期的に把握しているために、消滅するまでに時間がかかります。スレッド停止に関する詳細は、[KILL 構文](#) を参照してください。このリンクでは、[MyISAM](#) テーブルでの [REPAIR TABLE](#) または [OPTIMIZE TABLE](#) オペ中の消滅に関して特記しています。

オープン トランザクションのスレッドは、トランザクションがロールバックします。ノート： スレッドが非トランザクションのテーブルを更新中の場合には、[UPDATE](#) または [INSERT](#) などのオペレーションでテーブルを部分的に更新した状態にすることがあります。これはオペレーションが完了する前に停止する可能性があるためです。

サーバがマスタのレプリケーション サーバである場合は、スレッドは接続中のサーバに関連するスレッドを別のクライアントからのスレッドのように扱います。そのため、レプリケーションに関係しているスレッドはバイタルを失い、状態のチェックが済むと終了します。

サーバがスレーブのレプリケーション サーバである場合、I/O と SQL のスレッドがアクティブなときには、クライアント スレッドが失われる前に中断します。SQL スレッドは現行クエリを完了してから、中断します。これにより、レプリケーションで問題が発生しないようにします。SQL スレッドがその時点でトランザクションの最中であった場合は、トランザクションがロールバックします。

5. ストレージ エンジンのシャットダウンまたはクローズ

この段階で、テーブル キャッシュはフラッシュが済み、オープン テーブルのすべてを閉じます。

それぞれのストレージ エンジンでは、関係しているテーブルに対して必要なアクションを行います。たとえば、[MyISAM](#) では、テーブルへのインデックスの書き込みで保留中のものをフラッシュします。[InnoDB](#) では、ディスクにバッファ メモリをフラッシュします。その時点での LSN をテーブルスペースへ書き込み、内部スレッドを停止します。[innodb_fast_shutdown](#) で 2 と設定していた場合はこの限りではありません。

6. サーバの終了

サーバ サイド ヘルプ

MySQL サーバは [HELP](#) ステートメントをサポートしています。これは、MySQL リファレンス マニュアルからオンライン情報を表示します。（[HELP 構文](#) を参照のこと。）適切な操作を行うには、[mysql](#) データベースのヘルプ テーブルにヘルプのトピック情報が必要です。ここでは、[fill_help_tables.sql](#) スクリプトの内容を処理して行います。

Unix における MySQL の バイナリ 配布は、[mysql_install_db](#) を実行するとヘルプ テーブルのセットアップが出てきます。Linux の RPM 配布、または Windows の バイナリ 配布は、ヘルプ テーブルのセットアップは MySQL をインストールする過程で行います。

MySQL のソース配布は、[scripts](#) ディレクトリで、[fill_help_tables.sql](#) ファイルを探してください。このファイルを手動でロードするときには、[mysql_install_db](#) コマンドを実行して、[mysql](#) データベースを初期化し、次のように [mysql](#) クライアントでファイルをプロセします。

```
shell> mysql -u root mysql < fill_help_tables.sql
```

BitKeeper または MySQL の開発ソース ツリーで作業をしている場合、ツリーには [fill_help_tables.sql](#) ファイルが含まれていません。そのため、<http://dev.mysql.com/doc/> から、使用している MySQL バージョンに必要なファイルをダウンロードしてください。ダウンロードが済んだら、そのファイルを解凍して、記述のように [mysql](#) でそのファイルを処理してください。

MySQL サーバ スタートアップ プログラム

このセクションでは、MySQL サーバ、つまり [mysqld](#) を立ち上げるために使用するプログラムについて説明します。

[mysqld_safe](#) — MySQL サーバ スタートアップ スクリプト

[mysqld_safe](#) は Unix や NetWare などの環境で、[mysqld](#) サーバ（デーモン）を起動するときに推奨しているコマンドです。[mysqld_safe](#) は、エラー発生時にサーバを再起動したり、ランタイム情報をログ ファイルに記録するなどのセキュリティ機能が加わります。NetWare に特化した動作に関しては、このセクションの後方で説明します。

`mysqld_safe` は、`mysqld` という名前の実行可能ファイルを立ち上げます。デフォルトの動作を書き換えて、特定のサーバを指定するには、`mysqld_safe` で `--mysqld` または `--mysqld-version` のオプションを指定します。`--ledir` オプションで、`mysqld_safe` がサーバとして使用するディレクトリを指定できます。

`mysqld_safe` でのオプションは、`mysqld` のオプションと同じです。詳細は [コマンド オプション](#) を参照してください。

コマンドラインから `mysqld_safe` に指定したオプションのすべては、`mysqld` に渡ります。`mysqld_safe` へ特定のオプションを使用し、それを `mysqld` がサポートしない場合は、コマンドラインでの指定はしないでください。その代わりに、オプション ファイルの `[mysqld_safe]` グループでそれらをリストしてください。詳細は、[オプションファイルの使用](#) を参照してください。

`mysqld_safe` は、オプション ファイルの `[mysqld]`、`[server]`、`[mysqld_safe]` のセクションからすべてのオプションを読み取ります。下位互換の場合でも、`[safe_mysqld]` セクションから読みますが、MySQL 5.1 をインストールするときに、該当するセクションを `[mysqld_safe]` へとリネームする必要があります。

`mysqld_safe` は次のオプションをサポートします。

- `--help`

ヘルプメッセージを表示し、終了。

- `--autoclose`

NetWare 専用。NetWare では、`mysqld_safe` がスクリーン表示を行う。`mysqld_safe` NLM をアンロード (シャットダウン) するときは、デフォルトでスクリーンが消えることはなく、代わりにユーザ入力をプロンプトする。

```
*<NLM has terminated; Press any key to close the screen>*
```

NetWare で、自動的にスクリーンを閉じるようにするには、`mysqld_safe` で `--autoclose` オプションを使用する。

- `--basedir=path`

基準パス。MySQL をインストールしているディレクトリを指す。

- `--core-file-size=size`

`mysqld` で作成するコア ファイルのサイズ。このオプション値は `ulimit -c` へ渡る。

- `--datadir=path`

データ ディレクトリへのパス。

- `--defaults-extra-file=path`

通常のオプションファイルのほかに、読み込むオプション ファイルの名前。このオプションを使用するときは、これをコマンドラインで最初のオプションにする。このファイルが存在しない、またはアクセスできないという場合には、サーバがエラーを出して終了する。

- `--defaults-file=file_name`

通常のオプション ファイルの代わりに読み込むオプション ファイルの名前。このオプションを使用するときには、これをコマンドラインで最初のオプションにする。

- `--ledir=path`

`mysqld_safe` でサーバを見つけることができない場合、このオプションでサーバがあるディレクトリへのパスを探す。

- `--log-error=file_name`

任意のファイルにエラー ログを書き込む。[エラー ログ](#) を参照のこと。

- `--mysqld=prog_name`

`ledir` ディレクトリにある (起動する) サーバ プログラムの名前。このオプションは、MySQL バイナリ配布を使用するが、バイナリ配布外にデータ ディレクトリがある場合に、このオプションを使用する。`mysqld_safe` でサーバをみつけないことができない場合は、`--ledir` オプションを使用して、サーバ ディレクトリのパスを探す。

- `--mysqld-version=suffix`

`--mysqld` オプションと類似のオプション。ここではサーバのプログラム名 (`mysqld`) のサフィックスだけを指定する。ベース名を `mysqld` とする。たとえば、`--mysqld-version=debug` を使用すると、`mysqld_safe` は `ledir` ディレクトリの `mysqld-debug` プログラムを起動する。`--mysqld-version` の引数が空白の場合、`mysqld_safe` では、`ledir` ディレクトリの `mysqld` を使用する。

- `--nice=priority`

`nice` プログラムは任意の値に対してサーバのスケジュール優先順位を設定する。

- `--no-defaults`

オプション ファイルを読まない。このオプションを使用するときは、コマンドラインの最初に置く。

- `--open-files-limit=count`

`mysqld` が開くファイル数。オプション値は `ulimit -n` へ渡る。ノート：正確に機能させるには、`mysqld_safe` を `root` として立ち上げる。

- `--pid-file=file_name`

ID ファイルのパス名

- `--port=port_num`

TCP/IP 接続時に使用するポート番号。ポート番号は 1024 以上にする。サーバが `root` システム ユーザで立ち上がっている場合はこの限りではない。

- `--socket=path`

サーバがローカル接続に使用する Unix のソケット ファイル。

- `--timezone=timezone`

`TZ` タイム ゾーンの変数を任意のオプション値に設定する。正規 (リーガル) のタイム ゾーン形式に関しては、オペレーティング システムのマニュアルを参照のこと。

- `--user={user_name|user_id}`

`mysqld` サーバを、`user_name` (名前)、または `user_id` (数字のユーザ ID) を保有するユーザとして実行する。ここでの「ユーザ」は、権限テーブルにリストしている MySQL ユーザではなく、システム ログイン アカウントを指す)。

`mysqld_safe` を実行するときに、`--defaults-file` または `--defaults-extra-option` のオプションをオプション ファイルとすると、このオプションをコマンドラインの最初に置く。そうしないと、オプション ファイルは使用できません。たとえば、次のコマンドはオプション ファイルとして使用することはできません。

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

代わりに、次のコマンドを使用します。

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

`mysqld_safe` スクリプトを書くと、通常はソースまたは MySQL のバイナリ配布からインストールしたサーバを立ち上げることができ、これは、このようなタイプの配布を通常とは異なる場所にインストールする場合も同様です。(インストールのレイアウトを参照のこと。) `mysqld_safe` では、次の条件が `true` である必要があります。

- サーバとデータベースは作業中のディレクトリと関連する。つまり `mysqld_safe` から呼び出したディレクトリであること。バイナリ配布の場合、`mysqld_safe` は、`bin` と `data` のディレクトリにあり、ソース配布の場合

は、`libexec` と `var` のディレクトリにある。この条件が一致していると、MySQL をインストールしたディレクトリから `mysqld_safe` を実行できる。たとえば、バイナリ配布の場合には、`/usr/local/mysql` である。

- サーバとデータベースが作業中のディレクトリと関連しない場合は、`mysqld_safe` は絶対パスで探そうとする。通常は、`/usr/local/libexec` と `/usr/local/var` に位置する。実際の場合は、構築したとき設定した値で決定する。MySQL を設定時に指定の場所へインストールしていれば、正確に動作する。

`mysqld_safe` は作業中のディレクトリに関連しているサーバとデータベースを探すため、基本的には MySQL のバイナリ配布をどこにでもインストールできますが、`mysqld_safe` の実行は、MySQL をインストールしたディレクトリである必要があります。

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

MySQL をインストールしたディレクトリから呼び出しても、`mysqld_safe` で失敗する場合、`--ledir` または `--datadir` オプションで指定して、システム内のサーバとデータベースがあるディレクトリを指してください。

通常、`mysqld_safe` のスクリプトは編集してはいけません。その必要がある場合には、`mysqld_safe` を `my.cnf` オプション ファイルの `[mysqld_safe]` セクションにあるコマンドライン オプションを使用します。稀なケースでは、サーバを正確に起動するために、`mysqld_safe` を編集する必要があります。これを行う場合、`mysqld_safe` を修正しても、MySQL をアップグレードするときに、上書きされてしまうため、その修正バージョンのコピーを再インストールように控えておくことをお勧めします。

NetWare では、`mysqld_safe` が NetWare Loadable Module (NLM) であり、オリジナルの Unix シェル スクリプトからポートしています。この場合には、サーバは次のような経緯で立ち上がります。

1. 数々のシステムを実行し、オプション チェックを行う。
2. `MyISAM` テーブルのチェックをする。
3. MySQL サーバにスクリーンのプレゼンスを規定する。
4. `mysqld` を立ち上げ、それを監視し、再起動するときに、エラーで終了するかどうかをみる。
5. データ ディレクトリに `mysqld` から `host_name.err` へエラー メッセージを送る。
6. `mysqld_safe` のスクリーン出力をデータ ディレクトリの `host_name.safe` ファイルへ送る。

mysql.server — MySQL サーバ スタートアップ スクリプト

Unix の MySQL 配布には、`mysql.server` というスクリプトがあります。これは、System V-style が実行するディレクトリで使用するシステム、たとえば、Linux や Solaris など、システム サービスを起動または停止するために使用します。MySQL 用の Mac OS X Startup Item でも使用できます。

`mysql.server` コマンドは、MySQL をインストールしたディレクトリ、または MySQL のソース配布の `support-files` にあります。

Linux の RPM パッケージ (`MySQL-server-VERSION.rpm`) の場合は、`mysql.server` スクリプトを `/etc/init.d` のディレクトリに `mysql` という名前でインストールします。手動でインストールする必要はありません。Linux RPM パッケージに関する詳細は、[Linux に MySQL をインストールする](#) を参照してください。

ベンダーによっては、`mysqld` などの異なる名前でスタートアップスクリプトをインストールする RPM パッケージを提供しています。

自動的に `mysql.server` をインストールしないバイナリ配布形式、またはソース配布から MySQL をインストールする場合は、手動でインストールすることもできます。手順は [MySQL を自動的に起動・停止する](#) を参照してください。

`mysql.server` はオプション ファイルの `[mysql.server]` や `[mysqld]` のセクションからオプションを読みます。下位互換の場合には、`[mysql_server]` から読みますが、MySQL を使用するのであれば、`[mysql.server]` とセクションをリネームする必要があります。

mysqld_multi — 複数のMySQL サーバ管理

`mysql_multi` は複数の `mysqld` プロセス管理を目的とした、Unix ソケット ファイルや TCP/IP ポートからの接続を処理するプログラムです。これは、サーバを起動、停止、そしてステータスを報告します。MySQL Instance Manager は複数サーバのマッピングの選択的な方法です。MySQL Instance Manager に関する詳細は、[mysqlmanager — MySQL Instance Manager](#) を参照してください。

`mysql_multi` は `[mysqldN]` と名付けられたグループを `my.cnf` から検索します。(または `--config-file` オプションで指定しているファイル。) ここで、`N` は正の整数です。この番号は次の説明でのオプション グループ番号、あるいは `GNR` とします。グループ番号はオプション グループを識別、起動、停止、またはステータス レポートを取得するなど、サーバに指定する `mysql_multi` の引数として使用します。(例は、[MySQL を自動的に起動・停止する](#) を参照のこと。) 複数のサーバを使用する場合には、それぞれのサーバで、Unix ソケット ファイルや TCP/IP ポート番号への別々のオプションを使用する必要があります。複数サーバの環境で、どのオプションを一意とする必要があるのかについては、[同じマシン上での複数 MySQL サーバの実行](#) を参照してください。

`mysql_multi` を呼び出すには、次のシンタックスを使用します。

```
shell> mysql_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

`start`、`stop`、`report` は、どのオペレーションを実行するかを指します。サーバが 1 つの場合、または複数の場合でも指定したオペレーションを実行できますが、これは、オプションに従う `GNR` リストによります。リストがない場合は、`mysql_multi` がオプション ファイル内ですべてのサーバのオペレーションを実行します。

`GNR` はそれぞれに、オプション グループ番号、またはグループ番号の範囲を表します。値は、オプション ファイルのグループ名の末尾の数字になります。たとえば、`[mysqld17]` というグループ名の `GNR` は `17` です。番号の範囲を指定するには、最初と最後の番号をダッシュで区切ります。たとえば、`GNR` 値を `10-13` とするときは、`[mysqld10]` から `[mysqld13]` のグループを表します。複数のグループや範囲を指定するには、コマンドラインで、カンマ区切りをして指定します。`GNR` リストには空白文字 (スペース、タブなど) を使用しないでください。空白文字から後にあるものは無視の対象になります。

次のコマンドは `[mysqld17]` というオプション グループを使用したシングル サーバを起動します。

```
shell> mysql_multi start 17
```

次のコマンドは、`[mysqld8]` と `[mysqld10]` から `[mysqld13]` までのオプション グループを使用しているサーバ (複数) を停止します。

```
shell> mysql_multi stop 8,10-13
```

オプション ファイルの設定例には、次のコマンドを使用します。

```
shell> mysql_multi --example
```

`mysql_multi` では、次のオプションをサポートします。

- `--help`

ヘルプメッセージを表示し、終了。

- `--config-file=file_name`

代替オプション (設定) ファイル。これは、`mysql_multi` が `[mysqldN]` オプション グループを探すときに影響する。このオプションがない場合、すべてのオプションは通常の `my.cnf` ファイルから読み取る。このオプションは、`mysql_multi` が独自のオプションを読むときには影響しない。つまり常に、`[mysql_multi]` グループを通常の `my.cnf` ファイルから読み取る。

- `--example`

サンプルのオプション ファイルを表示する。

- `--log=file_name`

ログ ファイルを指定する。このファイルが存在していれば、すべてがログ ファイルへの記録になる。

- `--mysqladmin=prog_name`

サーバ停止 (シャットダウン) に使用する `mysqladmin` バイナリ。

- `--mysqld=prog_name`

使用する `mysqld` バイナリ。ノート：このオプションに `mysqld_safe` を値として指定することもできる。`mysqld_safe` をサーバの起動に使用する場合は、`[mysqldN]` オプショングループに相当する `mysqld` または `ledir` などのオプションを含めることができる。これらのオプションは、`mysqld_safe` が起動するサーバの名前とサーバが置かれているディレクトリのパスを指す。(これらのオプションに関する説明は、[mysqld_safe — MySQL サーバスタートアップスクリプト](#) を参照のこと。) 次は例示。

```
[mysqld38]
mysqld = mysqld-debug
ledir = /opt/local/mysql/libexec
```

- `--no-log`

ログファイルではなく、`stdout` にログを書き込む。デフォルトはログファイルへの出力。

- `--password=password`

`mysqladmin` を呼び出すときに使う MySQL アカウント ユーザのパスワード。ノート：MySQL プログラムの場合とは異なり、パスワード値はこのオプションではオプションではない。

- `--silent`

サイレントモード、警告を無効にする。

- `--tcp-ip`

MySQL サーバを TCP/IP ポートまたは Unix ソケット ファイルを経由して接続する。ソケット ファイルがない場合でも、サーバは実行可能であるが、TCP/IP ポートからだけアクセスできる。デフォルトでは、Unix ソケット ファイルを使用した接続。このオプションは `stop` と `report` の操作に影響する。

- `--user=user_name`

`mysqladmin` を呼び出すときの MySQL アカウントのユーザ名。

- `--verbose`

冗長にする。(出力をより詳細にする。)

- `--version`

バージョン情報を表示して、終了する。

次は、`mysqld_multi` に関するノートです。

- 重要：`mysqld_multi` コマンドを使用する前には、`mysqld` サーバへ渡すオプションの意味と、なぜ、`mysqld` プロセスを区切る必要があるのかを十分に理解してください。同じデータディレクトリで複数の `mysqld` サーバを使用することには危険が伴います。これを複数のサーバを使用するときは、データディレクトリを別々に分けてください。同じデータディレクトリで複数のサーバを使用することが、スレッドシステムのパフォーマンス改善にはなりません。[同じマシン上での複数 MySQL サーバの実行](#) を参照してください。
- 重要：それぞれのサーバのデータディレクトリが Unix アカウントから完全にアクセスできるかどうかを確かめてください。このアカウントは特定の `mysqld` プロセスを起動するものです。これに、状況を完全に理解していない場合は、Unix `root` アカウントを使用しないでください。[ユーザによる MySQL の実行](#) を参照してください。
- `mysqld` サーバを (`mysqladmin` プログラムで) 終了するときに使用する MySQL アカウントが、それぞれのサーバで同一のユーザ名とパスワードであることを確かめてください。そのアカウントには `SHUTDOWN` 権限があることも確かめてください。管理するサーバで、管理アカウント用に異なるユーザ名とパスワードがある場合は、それぞれのサーバで同一のユーザ名とパスワードのアカウントを作成してください。たとえば、共通の `multi_admin` アカウントをセットアップします。これには、次のコマンドをそれぞれのサーバで実行します。

```
shell> mysql -u root -S /tmp/mysql.sock -p
Enter password:
mysql> GRANT SHUTDOWN ON *.*
-> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
```

[権限システムの機能](#) を参照してください。これは、それぞれの [mysqld](#) サーバで行ってください。それぞれにアクセスするときに、接続パラメータを適宜に変更します。ノート： アカウント名のホスト名の部分は、[mysqld_multi](#) を実行する所のホストから [multi_admin](#) として接続できるようにします。

- Unix ソケット ファイルと TCP/IP ポート番号は、すべての [mysqld](#) とは異なる必要があります。
- `--pid-file` オプションはとても重要です。たとえば、`--mysqld=mysqld_safe` などのように、[mysqld](#) を起動するときに [mysqld_safe](#) を使用している場合と特に重要です。どの [mysqld](#) コマンドにも独自のプロセス ID ファイルがあります。[mysqld](#) ではなく、[mysqld_safe](#) を使用するということには、[mysqld_safe](#) は [mysqld](#) のプロセスを監視しているため、`kill -9` を使用したシグナル送信や、segmentation fault のようなことがあると、プロセスを終了し、再起動するという利点があります。[mysqld_safe](#) スクリプトは特定の場所から開始しなければならないことがあります。これは、[mysqld_multi](#) を実行する前に、特定のディレクトリに位置を変更しなければならない可能性がある、ということです。起動時に問題がある場合は、[mysqld_safe](#) スクリプトを調べ、特に次のラインをチェックください。

```
-----
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release)
if test -d $MY_PWD/data/mysql -a \
  -f ./share/mysql/english/errmsg.sys -a \
  -x ./bin/mysqld
-----
```

これらのラインでテストすると成功しますが、そうでない場合、つまり問題がある場合は、[mysqld_safe — MySQL サーバ スタートアップ スクリプト](#) を参照してください。

- `--user` オプションを [mysqld](#) に使用する場合は、Unix `root` ユーザで [mysqld_multi](#) スクリプトを実行する必要があります。このオプションがオプション ファイルにあるかどうかは問題ではなく、もしスーパーユーザではない人が、[mysqld](#) プロセスを Unix アカウントで立ち上げると、警告が出ます。

次の例は、[mysqld_multi](#) と使用するオプション ファイルの設定方法について示します。[mysqld](#) プログラムが開始または終了する順番は、オプション ファイルで指定する順番によります。グループ番号が完全なシーケンスである必要はありません。例では、最初と 5 番目の [\[mysqldN\]](#) グループは内部的に省略しています。これは、オプション ファイルで「gaps」があっても構わないというイラストレーションです。これによって、柔軟性が出ます。

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld  = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user     = multi_admin
password = multipass

[mysqld2]
socket   = /tmp/mysql.sock2
port     = 3307
pid-file = /usr/local/mysql/var2/hostname.pid2
datadir  = /usr/local/mysql/var2
language = /usr/local/share/mysql/english
user     = john

[mysqld3]
socket   = /tmp/mysql.sock3
port     = 3308
pid-file = /usr/local/mysql/var3/hostname.pid3
datadir  = /usr/local/mysql/var3
language = /usr/local/share/mysql/swedish
user     = monty

[mysqld4]
socket   = /tmp/mysql.sock4
port     = 3309
pid-file = /usr/local/mysql/var4/hostname.pid4
datadir  = /usr/local/mysql/var4
language = /usr/local/share/mysql/estonia
user     = tonu

[mysqld6]
```

```

socket = /tmp/mysql.sock6
port   = 3311
pid-file = /usr/local/mysql/var6/hostname.pid6
datadir = /usr/local/mysql/var6
language = /usr/local/share/mysql/japanese
user    = jani

```

詳細は [オプションファイルの使用](#) を参照してください。

mysqlmanager — MySQL Instance Manager

mysqlmanager は MySQL Instance Manager (IM) / MySQL インスタンス マネージャです。このプログラムは、MySQL Database Server のインスタンスを監視、管理します。MySQL Instance Manager は Unix のようなオペレーティングシステムで利用可能です。Windows でも利用可能で、TCP/IP ポートを使用するデーモンのように動作します。Unix では、Unix ソケット ファイルも監視します。

MySQL Instance Manager は **mysqld_safe** スクリプトの代わりに使用し、MySQL Server の 1 以上のインスタンスの開始や停止を行います。Instance Manager は、複数のサーバ インスタンスを管理するため、**mysqld_multi** スクリプトの代わりとしても使用できます。Instance Manager には次の機能があります。

- Instance Manager でインスタンスの開始と終了、そしてステータスの報告
- サーバ インスタンスにはガードあり、またはガードなしで扱える
 - Instance Manager の起動で、ガード インスタンスが立ち上がる。インスタンスがクラッシュすると、Instance Manager がそれを認識し、再起動する。Instance Manager の停止で、インスタンスが終了する。
 - ガードなしのインスタンスは、Instance Manager の起動または、それによる監視が行われている場合は立ち上がらない。起動してからインスタンスがクラッシュする場合、Instance Manager は再起動しない。Instance Manager が停止してもインスタンスが動いていれば、終了しない。

インスタンスはデフォルトでガードされている。インスタンスは 設定ファイル (コンフィギュレーション) で **nonguarded** オプションを含めてガードなしと指定できる。

- Instance Manager で、相互作用インターフェイスを設定インスタンスに与える。それにより、設定ファイルの手動修正の手間が省ける、または削減できる。
- Instance Manager で、インスタンス管理を遠隔操作できる。これは、MySQL Server のインスタンスを制御したいホストで実行できるという意味であるが、インスタンス管理のオペレーションを実行するリモート ホストから、接続できるという意味でもある。

次のセクションでは、MySQL Instance Manager の操作をより詳しく説明します。

MySQL Instance Manager コマンド オプション

MySQL Instance Manager は数多くのコマンド オプションをサポートしています。簡単な例として、**--help** オプションで、**mysqlmanager** を呼び出します。オプションはコマンドラインまたは Instance Manager 設定ファイルで指定します。Windows では、Instance Manager をインストールしたディレクトリの **my.ini** が標準設定ファイルです。Unix では、**/etc/my.cnf** が標準設定ファイルです。異なる設定ファイルを指定するには、Instance Manager を **--defaults-file** オプションで立ち上げてください。

mysqlmanager は次のリストで説明しているオプションをサポートします。パスワード ファイルでエントリを管理するオプションに関しては、[Instance Manager のユーザとパスワード管理](#) で説明しています。

- **--help, -?**
ヘルプ メッセージを表示し、終了。
- **--add-user**
新規ユーザをパスワード ファイルに追加する。**--username** オプションで指定する。(MySQL 5.1.12 での追加)
- **--angel-pid-file=file_name**

エンジェル プロセス (angel process) がプロセス ID に記録するファイル。mysqlmanager がデーモン モードで実行になるときのこと。(すなわち、`--run-as-service` オプションを与えた場合。) デフォルトのファイル名は `mysqlmanager.angel.pid`。

`--angel-pid-file` オプションを与えない場合、デフォルトのエンジェル PID ファイルは、PID ファイルと同じ名前。ただし、PID ファイルの拡張子は、`.angel.pid` の拡張子と置換。(例: `mysqlmanager.pid` は `mysqlmanager.angel.pid` となる。)

(MySQL 5.1.11 での追加)

- `--bind-address=IP`

バインドする IP アドレス。

- `--check-password-file`

パスワード ファイルの正当性 (validity) と整合性 (consistency) をチェックする。(MySQL 5.1.12 での追加)

- `--clean-password-file`

パスワード ファイルからすべてのユーザをドロップする。(MySQL 5.1.12 での追加)

- `--debug=debug_options, -# debug_options`

デバッグ ログを書き込む。debug_options 文字列は、`'d:t:o,file_name'` というのが通常。(MySQL 5.1.10 での追加)

- `--default-mysqld-path=path`

MySQL Server バイナリのパス。このパスは、設定ファイルのサーバ インスタンス セクションすべてに使用する。この設定ファイルには `mysqld-path` オプションがない。デフォルトではコンパイル時のパス。これは、MySQL 配布がどのように設定されたかによる。次はその例。`--default-mysqld-path=/usr/sbin/mysqld`

- `--defaults-file=file_name`

Instance Manager と MySQL Server のセッティングを任意のファイルから読み込む。Instance Manager による設定変更のすべてがこのファイルに書き込まれる。これを使用するときには、コマンドラインで最初のオプションにし、ファイルは存在している必要がある。

このオプションを使用しない場合、Instance Manager は標準の設定ファイルを使用する。Windows では、Instance Manager をインストールしたディレクトリの `my.ini` ファイル。Unix では、`/etc/my.cnf` が標準ファイル。

- `--drop-user`

新規ユーザをパスワード ファイルからドロップする。`--username` オプションで指定する。(MySQL 5.1.12 での追加)

- `--edit-user`

パスワード ファイルの既存ユーザ エントリを変更する。`--username` オプションで指定する。(MySQL 5.1.12 での追加)

- `--install`

Windows では、Instance Manager を Windows のサービスとしてインストールする。サービス名は `MySQL Manager`。

- `--list-users`

パスワード ファイルのユーザをリストする。(MySQL 5.1.12 に追加)

- `--log=file_name`

Instance Manager のログ ファイルのパス。`--run-as-service` オプションを与えない限り、このオプションの無効。このオプションで指定するファイル名が相対的である場合、ログ ファイルは Instance Manager を立ち上げたディレクトリ下での作成となる。特定のディレクトリで作成できたかどうかを確かめるには、これをフル パスで指す。

`--run-as-service` を `--log` なしで与えた場合、ログ ファイルはデータ ディレクトリの `mysqlmanager.log` になる。

`--run-as-service` を与えない場合、ログ メッセージは標準出力へ行く。ログ 出力を獲得するには、Instance Manager 出力をファイルにリダイレクトする。次はその例。

```
mysqlmanager > im.log
```

- `--monitoring-interval=seconds`

サーバ インスタンス監視のインターバル (秒)。デフォルトでは、20 秒。Instance Manager は監視対象 (ガード) のインスタンスのそれぞれとの接続を試行するときに、存在していない `MySQL_Instance_Manager` ユーザ アカウントを使用して、バイタルがあるかどうかをチェックする。接続試行の結果がインスタンスにバイタルがないことを示す場合、Instance Manager はインスタンスを再起動する前に、この試行を何度か実行する。

通常、`MySQL_Instance_Manager` アカウントは存在しないため、Instance Manager の接続試行は、監視しているインスタンスから次のようなメッセージをクエリ ログに生成する。

```
Access denied for user 'MySQL_Instance_M'@'localhost' »
(using password: YES)
```

適切なサーバ インスタンス セクションにある `nonguarded` オプションが特定のインスタンス監視を無効にする。起動してからインスタンスが動かない場合は、Instance Manager はそれを再起動しません。`SHOW INSTANCES`などで、インスタンスのステータスを要求しない限り、Instance Manager はガードなしのインスタンスとの接続を試行します。

詳細は、[MySQL サーバ インスタンス ステータスの監視](#)を参照してください。

- `--mysqld-safe-compatible`

`mysqld_safe` に準拠するマナーで実行。詳細は、[MySQL Instance Manager で MySQL Server の起動](#)を参照してください。(MySQL 5.1.12 での追加)

- `--password=password, -p password`

パスワード ファイルに、エントリを追加する、また変更するときのパスワードを指定する。これまでの MySQL プログラムの`--password/-P` オプションとは異なり、パスワードは必須です。オプションではありません。(MySQL 5.1.12 での追加)

- `--password-file=file_name`

Instance Manager がユーザとパスワードを探すファイルの名前。Windows では、Instance Manager をインストールしたディレクトリの `mysqlmanager.passwd` がデフォルト。Unix では、`/etc/mysqlmanager.passwd` がデフォルトファイル。

- `--pid-file=file_name`

プロセス ID ファイル。Windows では Instance Manager をインストールしたディレクトリの `mysqlmanager.pid` ファイルがデフォルト。Unix では、データ ディレクトリの `mysqlmanager.pid` がデフォルト。

- `--port=port_num`

クライアントからのTCP/IP 接続に使用するポート番号。デフォルトのポート番号は、IANA 割り当ての 2273。

- `--print-defaults`

現在のデフォルトを出力し、終了。このオプションを使用するときは、コマンドラインの最初に置く。

- `--print-password-line`

パスワード ファイルへのエントリを準備し、標準出力を表示し、終了する。Instance Manager の出力をファイルへリダイレクトして、そのファイルに保存できる。

MySQL 5.1.12 前のこのオプションの旧称は `--passwd` である。

- `--remove`

Windows 環境で、Windows サービスとしての Instance Manager を取り除く。これは、Instance Manager が以前に `--install` オプションで稼動していたことを前提とする。

- `--run-as-service`

Unix 環境で、デーモン化して、エンジェル プロセスを開始する。エンジェル プロセスは Instance Manager を監視し、クラッシュしたときに、再起動する。エンジェル プロセス自体はシンプルで、クラッシュすることはあまりない。

- `--socket=path`

Unix 環境で、着信する接続に使用するソケット ファイル。デフォルトは `/tmp/mysqlmanager.sock`。このオプションは Windows では無意味。

- `--standalone`

Windows で、Instance Manager をスタンドアローン型として稼動するときに使用する。これを指定するときは、Instance Manager をコマンドラインから立ち上げる。

- `--user=user_name`

Unix 環境で、`mysqlmanager` コマンドを起動実行するときに使用するシステム アカountのユーザ名。このオプションは警告を出す、`root` または名前をつけたユーザで `mysqlmanager` を立ち上げる場合 (有効なユーザ ID に変更する) を除いて、効果がない。`mysqld` サーバを実行するときと同じアカウントを使用している場合に、`mysqlmanager` を組み込むことを推奨。(ここの「ユーザ」とは、システム ログインのときのアカウントのことで、権限テーブルの MySQL ユーザのことではない。)

- `--username=user_name, -u user_name`

パスワード ファイルに追加または変更するエントリのユーザ名を指定する。(MySQL 5.1.12 に追加)

- `--version, -V`

バージョン情報を表示して、終了する。

- `--wait-timeout=N`

着信 (incoming) 接続でのアクティビティを閉じる前に待機する秒数。デフォルトでは 28800 秒 (8 時間)。

MySQL 5.1.7 で追加した。以前は、このタイムアウトは 30 秒で、変更不可だった。

MySQL Instance Manager の設定ファイル

MySQL Instance Manager は、`--defaults-file` オプションで別のファイルを指定して起動する場合を除き、標準の設定ファイルを使用します。Windows では、Instance Manager をインストールしたディレクトリの `my.ini` が標準ファイルです。Unix では、`/etc/my.cnf` が標準ファイルです。

Instance Manager で 設定ファイル (configuration file) の `[manager]` セクションからオプションと、`[mysqld]` または `[mysqldN]` のセクションのオプションを読み取ります。`[manager]` セクションには、[MySQL Instance Manager コマンド オプション](#) でリストするオプションがあります。コマンドラインのファースト オプション (最初のコマンド) として与えられているものに関しては別です。次は `[manager]` オプションのサンプルです。

```
# MySQL Instance Manager options section
[manager]
default-mysqld-path = /usr/local/mysql/libexec/mysqld
socket=/tmp/manager.sock
pid-file=/tmp/manager.pid
password-file = /home/cps/mysqlmanager.passwd
monitoring-interval = 2
port = 1999
bind-address = 192.168.1.5
```

それぞれの `[mysqld]` または `[mysqldN]` インスタンス セクションでは、サーバ インスタンスのスタートアップ用に Instance Manager で与えるオプションを指定します。さらに、`[mysqldN]` セクションでは、Instance Manager を指定

するオプションを含むことができ、これを次にリストしています。このオプションは Instance Manager が解釈し、サーバの起動を試行するときに、サーバへは渡しません。

警告

Instance Manager に特化したオプションは、`[mysqld]` では使用できません。サーバを Instance Manager を使用せずに立ち上げると、サーバではそのオプションを認識しません。そのため、サーバは正確に起動しません。

- `mysqld-path = path`

`mysqld` サーバ バイナリのパス。サーバ インスタンスに使用する。

- `nonguarded`

このオプションは、サーバ インスタンスに対する Instance Manager の監視機能を無効化する。デフォルトでは、インスタンスをガードしています。Instance Manager の起動時に、インスタンスも起動する。インスタンス ステータスを監視し、失敗したときは再起動を試行する。Instance Manager の終了時には、インスタンスを停止します。ガードなしのインスタンスの場合には、これらの動作はない。

- `shutdown-delay = seconds`

Instance Manager がシステム終了する前に、サーバ インスタンスを待機する秒数。デフォルトは 35 秒。この待機値を超えると、Instance Manager は、インスタンスにバイタルがないとみなし、システム終了を試行する。大きなテーブルで `InnoDB` を使用している場合、この値を大きくする。

インスタンス セクションのサンプルは次の通りです。

```
[mysqld1]
mysqld-path=/usr/local/mysql/libexec/mysqld
socket=/tmp/mysql.sock
port=3307
server_id=1
skip-stack-trace
core-file
log-bin
log-error
log=mylog
log-slow-queries

[mysqld2]
nonguarded
port=3308
server_id=2
mysqld-path= /home/cps/mysql/trees/mysql-5.1/sql/mysqld
socket  = /tmp/mysql.sock5
pid-file = /tmp/hostname.pid5
datadir= /home/cps/mysql_data/data_dir1
language=/home/cps/mysql/trees/mysql-5.1/sql/share/english
log-bin
log=/tmp/fordel.log
```

MySQL Instance Manager で MySQL Server の起動

このセクションでは、サーバ インスタンスが起動するときに、Instance Manager がどのように起動するかについて説明します。Instance Manager を起動する前に、パスワード ファイルを設定する必要があります。これにより、Instance Manager 起動後の制御ができなくなります。Instance Manager アカウントの作成に関する詳細は [Instance Manager のユーザとパスワード管理](#) を参照してください。

Unix では、`mysqld` MySQL データベース サーバは、通常、`mysql.server` スクリプトで起動します。これは、`/etc/init.d/` フォルダにあります。このスクリプトはデフォルトで `mysqld_safe` を呼び出します。Instance Manager を使用して、`[mysql.server]` セクションに `use-manager` を追加して、`/etc/my.cnf` 設定ファイルを変更することもできます。

```
[mysql.server]
use-manager
```

MySQL 5.1.12 前は、Instance Manager ではサーバ インスタンスの一つだけの起動を試行していました。Instance Manager は起動時に設定ファイルがあればそれ読み取り、サーバ インスタンスを検索し、インスタンスのリストを準

備します。インスタンス セクションには `[mysqld]` または `[mysqldN]` という形式の名前があり、この `N` は符号なしの整数です。(例: `[mysqld1]`、`[mysqld2]` など。).

インスタンスのリストが整うと、Instance Manager がカードされたインスタンスをリストから起動します。インスタンスがない場合には、Instance Manager は `mysqld` というインスタンスを作成し、これをデフォルト (コンパイル) の設定値で立ち上げます。これは、Instance Manager をデフォルトの場所でインストールしていないと、`mysqld` プログラムを検出できないということです。(インストールのレイアウトで、MySQL 配布のコンポーネントに関するデフォルトの場所を説明しています。) MySQL を標準ではないところへインストールした場合、Instance Manager の設定ファイルを作成する必要があります。

起動時の動作は `mysqld_safe` で説明したものと同様に、サーバを立ち上げようとします。しかし、サーバ インスタンスの起動を抑制するような方法で、Instance Manager を実行することは不可能なため、オペレーションによっては柔軟性に欠ける場合があります。たとえば、MySQL のインストーラを実行するときのような、インスタンスを開始せずに設定するという目的で、Instance Manager を呼び出すことはできません。MySQL 5.1.12 での変更事項は、次の通りです。

- 新オプション: `--mysqld-safe-compatible`、MySQL 5.1.12 以前と同様に起動時の動作を Instance Manager にさせる。Instance Manager は `[mysqld]` インスタンス セクションを設定ファイルで検索し、起動する。Instance Manager が `[mysqld]` セクションを見つけないことができない場合、設定ファイルにアクセス可能であれば、新たにデフォルトの設定値で `[mysqld]` セクションを書き込む。そして `mysqld` インスタンスを起動する。Instance Manager は設定ファイル内の別のガード付きインスタンスを立ち上げることもできる。
- `--mysqld-safe-compatible` がない場合、Instance Manager はその設定ファイルを読み、ファイルがあれば、そのファイルのガード付きインスタンス セクションでインスタンスを起動する。ファイルがない場合、インスタンスを起動できない。

Instance Manager は、シャットダウン時にすべてのガード付きサーバ インスタンスを終了します。

`[mysqldN]` サーバ インスタンス セクションで使えるオプションは、MySQL Instance Manager の設定ファイルで説明しています。そこでは、特別な `mysqld-path=path-to-mysqld-binary` オプションを使用します。これは Instance Manager だけが認識します。このオプションを使用して、Instance Manager に `mysqld` バイナリがどこにあるかを知らせます。複数のインスタンスがある場合には、`datadir` と `port` のような別のオプションをセットする必要があります。これは、それぞれのインスタンスに異なるデータ ディレクトリと TCP/IP ポート番号があることを確かめるために行います。同じマシン上での複数 MySQL サーバの実行では、同一のマシンで複数のインスタンスを稼働するときなど、それぞれのインスタンスで、設定値が異なる必要がある場合について説明しています。

警告

`[mysqld]` インスタンス セクションがある場合は、Instance Manager 特有のオプションを含めないでください。

MySQL Instance Manager を使用する MySQL サーバでの Unix のスタートアップおよびシャットダウンのサイクルは、次の通りです。

1. `/etc/init.d/mysqld` スクリプトで MySQL Instance Manager を起動する。
2. Instance Manager がガード付きサーバ インスタンスを立ち上げ、それらを監視する。
3. サーバ インスタンスの立ち上げに失敗すると、Instance Manager が再起動する。
4. Instance Manager を `/etc/init.d/mysqld stop` などのコマンドでシステム終了するときに、すべてのサーバ インスタンスが終了になる。

Instance Manager のユーザとパスワード管理

Instance Manager はユーザ情報をパスワード ファイルに保存します。Windows 環境でのデフォルトは、Instance Manager をインストールしたディレクトリの `mysqlmanager.passwd` です。Unix 環境でのデフォルトは、`/etc/mysqlmanager.passwd` です。パスワード ファイルを別の場所に指定するには、`--password-file` オプションを使用します。

パスワード ファイルがない場合、またはパスワードのエントリがない場合、Instance Manager と接続することはできません。

注記

サーバインスタンスを監視中の Instance Manager は、パスワード ファイルでの変更を認識しません。そのため、システム終了して、パスワード エントリでの変更を行ってから、再起動してください。

パスワード ファイルへのエントリには次の形式があります。ユーザ名と暗号化パスワードの 2 つのフィールドをコロンで区切ります。

```
petr:*35110DC9B4D8140F5DE667E28C72DD2597B5C848
```

Instance Manager のパスワード暗号化は、MySQL Server と同じです。一方向操作です。暗号化パスワードの複合化は禁止です。

Instance Manager アカウントは、MySQL Server アカウントとは若干異なります。

- MySQL Server アカウントは、ホスト名、ユーザ名、パスワード。(MySQL ユーザ名とパスワード を参照のこと。)
- Instance Manager はユーザ名とパスワードだけ。

これは、クライアントがどのホストからでもユーザ名で Instance Manager に接続可能ということです。この接続を制限し、クライアント接続をローカル ホストだけにするには、`--bind-address=127.0.0.1` オプションで、Instance Manager を立ち上げます。これにより、ローカルのネットワーク インターフェースだけを使用するようになります。リモート クライアントからは接続できません。ローカル クライアントは次のように接続します。

```
shell> mysql -h 127.0.0.1 -P 2273
```

MySQL 5.1.12 前のパスワード ファイル生成に関するオプションは、`--passwd` だけで、これは、Instance Manager でユーザ名とパスワードを指し、その結果を表示するというものです。そして、出力を `/etc/mysqlmanager.passwd` のパスワード ファイルに保存しています。たとえば、次の通りです。

```
shell> mysqlmanager --passwd >> /etc/mysqlmanager.passwd
Creating record for new user.
Enter user name: mike
Enter password: mikepass
Re-type password: mikepass
```

プロンプトで、新規ユーザのユーザ名とパスワードを新たな Instance Manager ユーザとします。そのときは、パスワードを 2 回入力します。画面ではエコーしませんが、2 回入力することで、意図したものとは異なるパスワードの入力を防ぎます。つまり、入力するパスワード、2つが異なる場合、エントリは生成されないということです。

前述のコマンドは、`/etc/mysqlmanager.passwd` に次のラインを付加します。

```
mike:*BBF1F551DD9DD96A01E66EC7DDC073911BAD17BA
```

MySQL 5.1.12 当初、`--passwd` オプションは、`--print-password-line` という名前に変更になり、コマンドラインからユーザ アカウント管理のオプションを操作できるようになりました。たとえば、`--username` や `--password` などのオプションは、アカウント エントリに対するユーザ名とパスワードを指定するときに、コマンドラインで使用できます。これらを使用して、エントリを生成します。シングル ラインのコマンドを入力するなどのプロンプトは不要です。

```
shell> mysqlmanager --print-password-line
--username=mike --password=mikepass >> /etc/mysqlmanager.passwd
```

`--username` または `--password` オプションを省略した場合、Instance Manager が必要な値を指します。

`--print-password-line` オプションで、Instance Manager の出力はアカウント エントリの結果として送信し、パスワード ファイルへ付加できます。パスワード ファイルで直接操作できるように `account-management` オプションを Instance Manager に与える方法を次のリストで説明します。オプションは、`account-management` を目的とした Instance Manager でのスクリプタブル (記述可) です。パスワード ファイル操作を正確に行うには、ファイルが存在し、Instance Manager でアクセスできるようにする必要があります。ただし、`--clean-password-file` は例外です。これは、存在に関わらず、ファイルを作成します。択一的に、パスワード ファイルがあれば、手動で空ファイルを作成し、Instance Manager だけが読み書き込みするようにアクセス モードと権限を制限します。指定がなければ、デフォルトのパスワード ファイルを使用し、指定するときは、`--password-file` オプションで指定します。

パスワード ファイル操作での整合性を確認するには、サーバ インスタンスを管理するために使用する Instance Manager のシステム アカウントで ファイルを作成してください。そして、パスワード ファイルのアカウントを管理するときには、そのファイルをそのシステム アカウントから呼び出してください。

- 新規ユーザの作成

```
mysqlmanager --add-user --username=user_name [--password=password]
```

任意のユーザ名とパスワードをパスワード ファイルに新たなエントリとして加える。--username (または -u) オプション が必要。mysqlmanager でパスワードを指し、これは、--password (または -p) のコマンドラインで与えていないパスワードのこと。ユーザが既に存在していれば、追加できない。

- 既存ユーザの削除

```
mysqlmanager --drop-user --username=user_name
```

パスワード ファイルのユーザ名のエントリを削除する。ユーザ名が必要。ユーザが存在しなければ、削除できない。

- 既存ユーザのパスワード変更

```
mysqlmanager --edit-user --username=user_name [--password=password]
```

パスワード ファイルのユーザのパスワードを削除する。ユーザ名が必要。コマンドラインを使用しない場合は、mysqlmanager で呼び出す。ユーザが存在しなければ、変更できない。

- 既存ユーザのリスト

```
mysqlmanager --list-users
```

このコマンドで、パスワード ファイルのアカウント ユーザ名をリストする。

- パスワード ファイルのチェック

```
mysqlmanager --check-password-file
```

このコマンドで、パスワード ファイルの整合性と妥当性チェックを行う。このコマンドの失敗は、ファイルに異常があることを示す。

- パスワード ファイルの空化

```
mysqlmanager --clean-password-file
```

パスワード ファイルを空にする。ファイルにリストされているすべてのユーザを削除する。パスワード ファイルが存在しない場合、このオプションで作成。つまり、このオプションは、別の account-management 操作で使用する新しいパスワード ファイルを作成する。誤ってアカウントを削除しないように、このオプションを使用するときは注意が必要。

MySQL サーバ インスタンス ステータスの監視

ガードしているサーバ インスタンスのステータスを監視では、MySQL_Instance_Manager@localhost というユーザ アカウントで、MySQL_Instance_Manager@localhost というパスワードを使用して、MySQL Instance Manager で接続を試行します。

MySQL Server でこのアカウントを作成する必要はありません。もともと、存在しないものとして扱われます。Instance Manager では、サーバが操作できる状況にあるかどうかを、サーバの接続できるかどうかで確認し、このアカウントで接続を試行すると、アクセスを拒否し、ログイン エラーを返します。この接続試行はサーバの一般クエリ ログで記録に記録されます。詳細は [一般クエリ ログ](#) を参照してください。

SHOW INSTANCES または SHOW INSTANCE STATUS コマンドを使用すると、ガードがないサーバ インスタンスへの接続試行が Instance Manager で行われます。これは、ガードしていないインスタンスでのみ行うステータス監視です。

サーバ インスタンスが起動に失敗すると、その状態が Instance Manager へ送信られ、それを認識します。インスタンスが起動した後に、クラッシュするという場合は、Instance Manager は、インスタンスの親プロセスであるため、そのサイン (signal) を受信します。

MySQL 5.1.12 から、Instance Manager でインスタンス ステータスを追跡できるようになり、どのコマンドをそれぞれのインスタンスに使用できるかを決定します。たとえば、インスタンスの設定を変更するコマンドは、インスタンスがオフラインのときだけ使用可能です。

次のテーブルは、インスタンスのステータスの説明です。ガードしているインスタンスには、どのステータスも該当します。ガードしていないインスタンスは、オンラインまたはオフラインの状態です。ステータス情報は、[SHOW INSTANCES](#) または [SHOW INSTANCE STATUS](#) コマンドで、[status](#) カラムに表示されます。

状態	説明
offline	未起動、未稼働
starting	起動中 (初期化)。ガードなしの場合は、直接オフラインからオンラインになる。
stopping	停止中。ガードなしの場合は、直接オフラインからオンラインになる。起動に失敗すればオフラインのまま。
online	起動。稼働中。
failed	クラッシュしたために、オフラインで Instance Manager が再起動中。またはサーバがまったく起動せず、Instance Manager での再起動が試行中の状態。ガードなしのサーバにこの状態は発生しない。
crashed	Instance Manager が接続を試行したが、起動できない。(しばらくしてから、Instance Manager による接続試行が開始される。) ガードなしのサーバにこの状態は発生しない。
abandoned	Instance Manager で起動できず、指示があるまで接続試行しない。Instance Manager に再開を指示するには、 STOP INSTANCE でオフラインにして、それから START INSTANCE を指示する。設定変更 (configuration) が必要な場合には、サーバをオフラインにしてから、作業を行う。(Instance Manager ではサーバがオフラインのときにだけ、設定変更のコマンドを使用できる。) ガードなしのサーバにこの状態は発生しない。

MySQL Instance Manager へ接続

MySQL Instance Manager では、まずパスワード ファイルをセットアップを行い、それに続いて稼働、接続を行います。MySQL クライアント サーバ プロトコルで、Instance Manager と通信します。たとえば、標準の [mysql](#) クライアント プログラムを使用して接続します。

```
shell> mysql --port=2273 --host=im.example.org --user=mysql --password
```

Instance Manager での MySQL クライアント サーバ プロトコルは、MySQL 4.1 以降の配布したライブラリとクライアント ツールを使用します。そのため、MySQL C API を使用しているプログラムなどで接続できません。

MySQL Instance Manager のコマンド

MySQL Instance Manager と接続後、コマンドを発行します。コマンドの実行には、次のような原則があります。

- インスタンス名が有効でない場合、動作しない。
- インスタンス名が無い (存在しない) 場合、動作しない。([CREATE INSTANCE](#) は別。)
- インスタンスが適切なステート (状態) であるよう要求する。オフラインではないインスタンスを設定または起動することはできない。(MySQL 5.1.12 以降)
- ファイルが存在しない、または Instance Manager からアクセスできない場合は、設定ファイルの変更はできない。Instance Manager では、内部キャッシュ (メモリ) でインスタンス設定 (コンフィギュレーション) に関する情報を保管している。基本的には、設定ファイルが存在すれば、情報はそこから送られ、コマンドによってはインスタンスの設定を変更できる。

コマンドで、内部キャッシュ とサーバの設定ファイルのインスタンス セクションの整合性を維持するために、両方を変更 (修正) する。(MySQL 5.1.12 以降。) これを行うには、まずインスタンスがオフラインであること、設定ファイルがアクセス可能であり、不正形式ではないことを確認する。設定ファイルを更新できない場合に、コマンドに失敗すると、キャッシュへの変更がないままとなる。

- Windows 環境では、Instance Manager をインストールした ディレクトリの [my.ini](#) が標準ファイル。Unix 環境では、[/etc/my.cnf](#) が標準の設定ファイル。異なる設定ファイルを指定するには、[--defaults-file](#) オプションで Instance Manager を起動する。

- `[mysqld]` というインスタンス セクションが設定ファイルには、Instance Manager特有のオプションを入れない。(MySQL Instance Manager の設定ファイル を参照のこと。) つまり、`mysqld` という名前のインスタンスの設定変更には、これらのオプションを付加しないこと。

次のリストは、使用例とともに、Instance Manager で使えるコマンドを説明します。

- `CREATE INSTANCE instance_name [option_name[=option_value], ...]`

新たなインスタンスを設定する。設定ファイルに、`[instance_name]` セクションを作成する。インスタンス名が `instance_name` に対して有効ではない場合、または存在しない場合は、コマンドに失敗する。

オプションを付けない場合は、インスタンスのセクションを空にする。付ける場合は、オプション ファイルで記述するときと同じ形式にする。使用可能なシンタックスに関しては、[オプションファイルの使用](#) を参照のこと。複数のオプションを指定する場合は、それらをカンマで区切る。

例： `[mysqld98]` という名前のインスタンス セクションを作成するには、変更しようとしている設定ファイルで次のようにする。

```
[mysqld98]
basedir=/var/mysql98
```

`CREATE INSTANCE` 経由で同様の効果を与えるには、Instance Manager で次のコマンドを発行する。

```
mysql> CREATE INSTANCE mysqld98 basedir="/var/mysql98";
Query OK, 0 rows affected (0,00 sec)
```

`CREATE INSTANCE` でインスタンスを作成するが、起動はしない。

インスタンス名が、(廃止された) 名前 `mysqld` である場合には、Instance Manager に対して指定するオプションをオプション リストに含めることはできない。たとえば、`nonguarded` など。(MySQL Instance Manager の設定ファイル を参照のこと。)

(MySQL 5.1.12 に追加)

- `DROP INSTANCE instance_name`

設定ファイルから `instance_name` の設定を削除する。

```
mysql> DROP INSTANCE mysqld98;
Query OK, 0 rows affected (0,00 sec)
```

`instance_name` が有効なインスタンス名でない場合、あるいは、インスタンスが存在しない、またはオフラインの場合は、コマンドに失敗する。.

(MySQL 5.1.12 に追加)

- `START INSTANCE instance_name`

オフラインのインスタンスの起動を試行する。非同期 (asynchronous) で、インスタンス起動を待機する。

```
mysql> START INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- `STOP INSTANCE instance_name`

インスタンス停止を試行する。同期 (synchronous) で、インスタンス停止を待機する。

```
mysql> STOP INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- `SHOW INSTANCES`

ロードされたすべてのインスタンスの名前とステータスを表示する。

```
mysql> SHOW INSTANCES;
+-----+-----+
| instance_name | status |
```



```

+-----+-----+
| mysqld3 | offline |
| mysqld4 | online  |
| mysqld2 | offline |
+-----+-----+

```

- **SHOW INSTANCE STATUS instance_name**

ステータスとバージョン情報を表示する。

```

mysql> SHOW INSTANCE STATUS mysqld3;
+-----+-----+-----+
| instance_name | status | version |
+-----+-----+-----+
| mysqld3      | online | unknown |
+-----+-----+-----+

```

- **SHOW INSTANCE OPTIONS instance_name**

インスタンスで使用しているオプションを表示する。

```

mysql> SHOW INSTANCE OPTIONS mysqld3;
+-----+-----+-----+-----+
| option_name | value |
+-----+-----+-----+-----+
| instance_name | mysqld3 |
| mysqld-path  | /home/cps/mysql/trees/mysql-4.1/sql/mysqld |
| port        | 3309 |
| socket       | /tmp/mysql.sock3 |
| pid-file     | hostname.pid3 |
| datadir      | /home/cps/mysql_data/data_dir1/ |
| language     | /home/cps/mysql/trees/mysql-4.1/sql/share/english |
+-----+-----+-----+-----+

```

- **SHOW instance_name LOG FILES**

インスタンスで使用しているすべてのログ ファイルをリストする。結果セットはログ ファイルのパスとサイズ。設定ファイルのインスタンス セクションで、ログ ファイルのパスを指していない場合、(例 : [log=/var/mysql.log](#))、Instance Manager が代替するものを検索する。Instance Manager でログ ファイルに代わるものを見つけれないときは、設定ファイルで該当するインスタンス セクションのログ オプションを使用して、明示的にログ ファイルの位置を指定する。

```

mysql> SHOW mysqld LOG FILES;
+-----+-----+-----+
| Logfile | Path | Filesize |
+-----+-----+-----+
| ERROR LOG | /home/cps/var/mysql/owlet.err | 9186 |
| GENERAL LOG | /home/cps/var/mysql/owlet.log | 471503 |
| SLOW LOG | /home/cps/var/mysql/owlet-slow.log | 4463 |
+-----+-----+-----+

```

SHOW ... LOG FILES はログ ファイルに関する情報だけを表示する。サーバ インスタンスでログ ファイルを使用する場合には、テーブルに関する情報に関しては表示しない。[一般クエリとスロー クエリのログ出力先の選択](#) を参照のこと。

ログ オプションに関する詳細は [コマンド オプション](#) を参照こと。

- **SHOW instance_name LOG {ERROR | SLOW | GENERAL} size[,offset_from_end]**

指定したログ ファイルを部分的に取り出す。多くのユーザは最新のログ メッセージに関心があるため、**size** パラメータは最後のログから読み出すバイト数を定義する。ログ ファイルの途中からデータを読み出すには、任意の **offset_from_end** パラメータを指定する。次の例では、21 バイトのデータを読み出し。ログ ファイルの終わりから 23 バイト前で始まり、2 バイト前で終わる。

```

mysql> SHOW mysqld LOG GENERAL 21, 2;
+-----+
| Log |
+-----+
| using password: YES |
+-----+

```

- `SET instance_name.option_name[=option_value]`

特定インスタンスの設定セクションを変更する。インスタンス オプションを変更または追加するため。このセクションに追加したオプションは、まだ現在していないオプションのこと。それ以外では新たなセッティングが既存のものとの置き換えになる。

```
mysql> SET mysql2.port=3322;
Query OK, 0 rows affected (0.00 sec)
```

MySQL 5.1.12 以降、カンマ区切りで、複数のオプションを指定することができるようになり、オフラインのインスタンスに `SET` を使用できる。それぞれのオプションがインスタンス名を示すようにする。

```
mysql> SET mysql2.port=3322, mysql3.nonguarded;
Query OK, 0 rows affected (0.00 sec)
```

MySQL 5.1.12 前は、指定できるオプションが 1 つだけで、設定ファイルへの変更は、MySQL サーバが再起動するまで効果を発揮しない。また、変更はインスタンス管理をするローカル キャッシュには保存されず、`FLUSH INSTANCES` コマンドを実行するまで有効にならない。

- `UNSET instance_name.option_name`

インスタンスの設定セクションからオプションを削除する。

```
mysql> UNSET mysql2.port;
Query OK, 0 rows affected (0.00 sec)
```

MySQL 5.1.12 以降、カンマ区切りで、複数のオプションを指定することができるようになり、オフラインのインスタンスに `UNSET` を使用できる。それぞれのオプションがインスタンス名を示すようにすること。

```
mysql> UNSET mysql2.port, mysql4.nonguarded;
Query OK, 0 rows affected (0.00 sec)
```

MySQL 5.1.12 前は、指定できるオプションは 1 つだけで、設定ファイルへの変更は、MySQL サーバが再起動するまで効果を発揮しない。また、変更はインスタンス管理をするローカル キャッシュには保存しないため、`FLUSH INSTANCES` コマンドを実行するまで有効にならない。

- `FLUSH INSTANCES`

MySQL 5.1.12 以降、`FLUSH INSTANCES` をすべてのインスタンスをオフラインにしてから使用する。このコマンドで Instance Manager に設定ファイルを再読み込みさせ、内部メモリのコンフィギュレーション キャッシュを更新 (update) し、ガード インスタンス (ガード有り) を起動する。

MySQL 5.1.12 前では、Instance Manager に設定ファイルの読み込みと内部ストラクチャのリフレッシュを強制している。設定ファイルを編集してからこのコマンドを実行するが、インスタンスの再起動はできない。

```
mysql> FLUSH INSTANCES;
Query OK, 0 rows affected (0.04 sec)
```

`FLUSH INSTANCES` は廃止。MySQL 5.2 で削除。

インストール関連プログラム

`make_win_bin_dist` — Package MySQL 配布 (ZIP アーカイブ)

実行可能なプログラム作成で、ソースから MySQL 配布を構築した後の Windows 環境でのスクリプトです。バイナリとサポート ファイルを ZIP ファイルのパッケージです。これを MySQL をインストールする場所で解凍します。

`make_win_bin_dist` はシェル スクリプトですので、使用するには Cygwin をインストールしておく必要があります。

このプログラムは変更する可能性があります。現在は、ソース配布の root ディレクトリから、次のように呼び出します。

```
shell> make_win_bin_dist [options] package_basename [copy_def ...]
```

`package_basename` 引数が ZIP ファイルのベース名になります。ファイルを解凍したときに、ディレクトリの名前になります。

別のビルドからファイルを取り込む場合は、次に示すスクリプトを使用して、対象ファイルをコピーします。これは `relative_dest_name=source_name` の `copy_def` 引数経由です。

例:

```
bin/mysqld-max.exe=../my-max-build/sql/release/mysqld.exe
```

ディレクトリを指定するときは、ディレクトリ全体がコピーの対象になります。

`make_win_bin_dist` でオプションを認識します。そのオプションは次の通りです。

- `--debug`

デバッグ バイナリを梱包し、構成できなかった場合にはエラーを生成。

- `--embedded`

埋め込みサーバを梱包し、構成できなかった場合にはエラーを生成。デフォルトは構成したら梱包という設定。

- `--exe-suffix=suffix`

`mysql` バイナリのベース名にサフィックスを付ける。例: `-abc` というサフィックスで、`mysqld-abc.exe` というバイナリになる。

- `--no-debug`

構成できていたとしても、デバッグ バイナリを梱包しない。

- `--no-embedded`

構成できていたとしても、埋め込みサーバを梱包しない。

- `--only-debug`

構成したものが `Debug` を対象としているときに使用するオプション。通常のバイナリをデバッグ バージョンと交換したい場合などに使用する。従って、分けている `debug` ディレクトリ には使用しない。

mysql_fix_privilege_tables — MySQL システム テーブルのアップグレード

MySQL のリリースによっては、新たに権限を追加するとき、または新たな機能をサポートするときに、`mysql` データベースのシステム テーブルのストラクチャを変更できます。新しいバージョンの MySQL にアップグレードするときは、システム テーブルも同様に更新し、ストラクチャが最新であることを確かめる必要があります。これをしないと、この利点を活用できません。まず、`mysql` データベースをバックアップし、次の手順に従います。

ノート:MySQL 5.1.7 以降は `mysql_upgrade` を使用してください。`mysql_fix_privilege_tables` と `mysql_upgrade` と置き換わりました。[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照こと。

Unix または Unix のようなシステム環境では、`mysql_fix_privilege_tables` スクリプトでシステム テーブルを更新します。

```
shell> mysql_fix_privilege_tables
```

このスクリプトはサーバが稼働しているときに実行してください。ローカル ホストで `root` で稼働しているサーバに接続するためです。`root` アカウントでパスワードが必要な場合には、コマンドラインで次のようにパスワードを指します。

```
shell> mysql_fix_privilege_tables --password=root_password
```

`mysql_fix_privilege_tables` スクリプトは、必要に応じて現行のフォーマットに合わせてシステム テーブルを変換するアクションを行います。実行中に `Duplicate column name` という警告がでることがありますが、これは無視します。

スクリプトを実行後、サーバをシステム終了し、再起動します。

Windows システム環境では、MySQL 配布に、`mysql_fix_privilege_tables.sql` という SQL スクリプトが含まれていて、これは、`mysql` のクライアントを使用して実行します。たとえば、MySQL を `C:\Program Files\MySQL\MySQL Server 5.1` でインストールした場合のコマンドは次のようになります。

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 5.1"
C:\> bin\mysql -u root -p mysql
mysql> SOURCE scripts/mysql_fix_privilege_tables.sql
```

mysql コマンドで、root のパスワード入力を指示されたら、パスワードを入力します。

インストールした場所が別のディレクトリである場合は、適宜パスを調整します。

Unix の手順では、mysql で mysql_fix_privilege_tables.sql のステートメントの処理中に、Duplicate column name という警告が出ますが、これは無視します。

スクリプトを実行後、サーバをシステム終了し、再起動します。

mysql_install_db — MySQL データ ディレクトリ 初期化スクリプト

mysql_install_db は MySQL データを初期化し、システム テーブルを作成します (システム テーブルがない場合)。MySQLサーバとしての mysqld は起動後にデータ ディレクトリのデータにアクセスする必要があります。そのため、mysqld を実行するのと同じアカウントで mysql_install_db を起動するか、または root で立ち上げて、--user オプションを使用して、mysqld を実行するユーザ名を指します。

mysql_install_db を呼び出すには、次のシンタックスを使用します。

```
shell> mysql_install_db [options]
```

mysql_install_db で mysqld を呼び出します。これには、--bootstrap と --skip-grant-tables を使用します。(典型的な configure オプション 参照。) MySQL を --disable-grant-options オプションで構成した場合、--bootstrap と --skip-grant-tables のオプションは無効化します。この処理には、すべてのオプションを有効化したフル パスとサーバに、MYSQLD_BOOTSTRAP 環境変数をセットします。mysql_install_db はそのサーバを使うようになります。

mysql_install_db では次のオプションをサポートします。

- --basedir=path

基準パス。MySQLをインストールしているディレクトリを指す。

- --force

DNS では機能しない場合に、mysql_install_db で実行します。この場合、通常使用している権限テーブルのエントリのホスト名は IP アドレスになります。

- --datadir=path, --ldata=path

MySQL データ ディレクトリへのパス。

- --rpm

内部使用。MySQL インストール プロセス中の、RPM ファイルのオプション。

- --skip-name-resolve

権限テーブルのエントリで、ホスト名ではなく、IP アドレスを使用する。DNS が機能しない場合に使用するオプション。

- --srcdir=path

内部使用。mysql_install_db で、エラー メッセージやペルプ テーブルのファイルなど、サポート ファイルを検索するときのディレクトリ。(MySQL 5.1.14 での追加)

- --user=user_name

mysqld を実行するときのログイン ユーザ名。このユーザで、mysqld で作成するファイルやディレクトリの操作を行う。このオプションを使用するときは、root であることが必要。デフォルトでは、ログイン中の名前が mysqld が動作し、ファイルやディレクトリはそのログイン ユーザの所有となる。

- `--verbose`

冗長モード。プログラム実行に関する情報を出力する。

- `--windows`

内部使用。Windows 配布を作成するときに使用するオプション。

mysql_upgrade — MySQL アップグレードのテーブル チェック

MySQL のアップグレードでは、その度に、`mysql_upgrade` を実行します。これにより、データベース内のテーブルにおける最新の MySQL Server との互換性をチェックすることができます。該当テーブルが非互換の場合は、チェックの対象になり、問題があれば、そのテーブルを修正します。`mysql_upgrade` コマンドは、システム テーブルのアップグレードも行うため、新たな権限と追加機能を使用できるようになります。

チェックと修正が済んだテーブルは、最新の MySQL バージョン番号とマーク (特徴付け) される。これにより、次に同じバージョンのサーバで `mysql_upgrade` を立ち上げるときに、そのテーブルをチェックして修正する必要があるかどうかを確かめる必要がなくなります。

`mysql_upgrade` では、データ ディレクトリの `mysql_upgrade.info` というファイルに MySQL のバージョン番号も保存します。これは、すべてのテーブルがチェック済みであるかどうかを簡単に調べます。これにより、リリースでテーブル チェックをスキップできるようになります。ファイルを無視するには、`--force` オプションを使用します。

テーブル チェックおよび修復、システム テーブルのアップグレードを行なうには、`mysql_upgrade` で次のコマンドを実行します。

```
mysqlcheck --check-upgrade --all-databases --auto-repair
mysql_fix_privilege_tables
```

`mysql_upgrade` コマンドは、古い方、つまり `mysql_fix_privilege_tables` より優先です。MySQL 5.1.7 では、シェル スクリプト として `mysql_upgrade` が加えられ、Unix システムだけで機能します。MySQL 5.1.10 以降は、`mysql_upgrade` は実行可能なバイナリとして、すべてのシステムで使用できます。`mysql_upgrade` をサポートしているものより古いシステムでは、手動で `mysqlcheck` コマンドを実行し、システム テーブルのアップグレードを行いません。詳細は `mysql_fix_privilege_tables` — MySQL システム テーブルのアップグレード を参照してください。

何がチェックされるか、に関する詳細は、`CHECK TABLE` ステートメントの `FOR UPGRADE` オプションに関する説明を参照してください。(CHECK TABLE 構文)

`mysql_upgrade` を使用するには、サーバが起動していることを確認し、次のように呼び出します。

```
shell> mysql_upgrade [options]
```

`mysql_upgrade` はコマンドライン、およびオプション ファイルの `[mysql_upgrade]` グループからオプションを読み取ります。これは次のオプションをサポートします。

- `--help`

ショート ヘルプ メッセージを表示し、終了。

- `--basedir=path`

基準パス。MySQL がインストールされているディレクトリを指す。

- `--datadir=path`

データ ディレクトリへのパス。

- `--force`

`mysqlcheck` を強制実行する。最新の MySQL バージョンで `mysql_upgrade` を既に実行している場合も。つまり、このオプションは、`mysql_upgrade.info` を無視するようにする。

- `--user=user_name, -u user_name`

サーバに接続するときの MySQL ユーザ名。デフォルト名は `root`。

- `--verbose`

冗長モード。プログラム実行に関する情報を出力する。

他のオプションは、`mysqlcheck` および `mysql_fix_privilege_tables` へ渡される。たとえば、`--password[=password]` オプションは指定しなければならない可能性がある。

mysql_tzinfo_to_sql — タイムゾーンテーブルのロード

`mysql_tzinfo_to_sql` は、`mysql` データベースに、タイムゾーンテーブルをロードするプログラムです。zoneinfo データベース (タイムゾーンを記述するファイルのセット) があるシステムで使います。たとえば、Linux、FreeBSD、Sun Solaris、Mac OS X などです。一般的なファイル位置は `/usr/share/zoneinfo` です。zoneinfo データベースがないシステムの場合には、[MySQL サーバのタイムゾーンサポート](#) で説明するパッケージをダウンロードします。

`mysql_tzinfo_to_sql` はいくつかの方法で呼び出せます。

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

最初の起動シNTAX (invocation) では、`mysql_tzinfo_to_sql` に zoneinfo ディレクトリのパスを渡し、出力を `mysql` プログラムに送ります。次はその例です。

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` はシステム内のタイムゾーンファイルを読み、そこから SQL ステートメントを生成します。`mysql` でそのステートメントが処理され、タイムゾーンテーブルにロードされます。

2 番目のシNTAX は `mysql_tzinfo_to_sql` を タイムゾーン名 `tz_name` に関連するシングル タイムゾーン ファイル `tz_file` にロードします。

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

タイムゾーンで、リープ秒のカウントが必要な場合は、`mysql_tzinfo_to_sql` を 3 番目のシNTAX を使用して呼び出して、リープ秒情報を初期化します。`tz_file` は使用しているタイムゾーンファイルの名前です。

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

セキュリティ問題

このセクションでは、MySQL のインストールすることによる外部攻撃、または不正使用に対するセキュリティを高めるために理解が必要なセキュリティ事項に関して説明します。MySQL を使用したユーザアカウントのセットアップやデータベースアクセスへのチェックなど、コントロールシステムへのアクセスに関する詳細は [MySQL アクセス権限システム](#) も参照してください。

MySQL での SQL サーバのセキュリティに関する QA は、[MySQL 5.1 FAQ — Security](#) を参照してください。

セキュリティ ガイドライン

インターネットに接続しているコンピュータで MySQL を使用するには、このセクションの内容を十分に理解し、セキュリティ面での誤操作を起こさないように注意してください。

セキュリティに関する説明では、様々な攻撃に対して、MySQL サーバだけでなく、サーバ ホスト全体を完全に守る必要があることを強調しています。この攻撃とは、聴傍受、改ざん、再生、サービス妨害などのことです。ここでは、可用性および耐障害性のすべてについては触れていません。

MySQL では、接続、クエリ、そしてユーザが行なう別のオペレーションに対して、アクセス制御リスト (ACL, Access Control Lists) に基づくセキュリティ保護を行ないます。MySQL クライアントとサーバ間での SSL 暗号化接続をサポートしています。ここで説明するコンセプトの多くは、MySQL だけに該当するものではなく、様々なアプリケーションにも該当します。

MySQL を実行するときには、常に次のガイドラインに従います。

- MySQL `root` ユーザ以外のだれにも `mysql` データベースの `user` テーブルへのアクセス権を与えない。これは特に重要なことです。
- MySQL のアクセス権限システムについて学び、理解する。MySQL へのアクセスを制御するには、`GRANT` および `REVOKE` のステートメントを使用します。必要以上の権限をユーザに設定しないこと。すべてのホストに対する権限は決して与えないこと。

チェックリスト

- `mysql -u root` を試す。パスワードなしでサーバに正常に接続できるよと問題がある。この場合、すべての権限を持つ `root` ユーザとして、MySQL サーバに接続できるということである。特に `root` パスワードの設定に関する項目に注意して、MySQL インストール手順を見直す。最初の MySQL アカウントの確保を参照のこと。
- `SHOW GRANTS` コマンドを使用して、だれが何にアクセスできるかをチェックする。`REVOKE` コマンドを使用して不要な権限を削除する。
- データベースには、テキスト形式のパスワードを保存しないこと。コンピュータがクラックされたとき、パスワードの完全なリストが奪われて不正使用される結果になる。`MD5()`、`SHA1()`、または別の一方方向ハッシング関数を使用する。
- 辞書を使用してパスワードを選択しない。特殊プログラムで解読されてしまう。「xfish98」のようなパスワードも良くない。標準 QWERTY キーボードで「fish」を 1 列左でタイプした「duag98」などを推奨。また、「Mary had a little lamb」の頭文字を取って「Mhall」とするのも良い。この方法だと覚えやすく、また部外者が類推することも困難。
- ファイアウォールに投資する。これにより、少なくとも 50% の攻撃を防ぐことができる。MySQL をファイアウォール内つまり非武装地帯 (DMZ) に配置する。

チェックリスト

- `nmap` などのツールを使用して、インターネットから自分のポートをスキャンしてみる。MySQL はデフォルトでポート 3306 を使用する。このポートは、信頼されていないホストからアクセスできてはいけない。他にも、MySQL ポートが開いているかどうか確かめる簡単な方法として、リモート コンピュータから以下のコマンドを実行するという方法がある。ここで、`server_host` は MySQL サーバのホスト名、または IP アドレスである。

```
shell> telnet server_host 3306
```

接続できて意味不明な文字が返ればポートが開いている。開いておく正当な理由がない限り、ファイアウォールまたはルータで閉じておく。`telnet` がハングするか、接続が拒否されれば正常である。つまり、ポートは閉じている。

- ユーザが入力するデータを信頼しないこと。Web フォーム、URL、その他のアプリケーションから特殊文字またはエスケープ文字シーケンスが入力され、不正操作が行われる可能性がある。ユーザが「`;` `DROP DATABASE mysql;`」などのような入力をして、アプリケーションが安全に保たれるようにしなければならない。これは極端な例であるが予防策を講じておかないと、クラッカーによる同様の手段によって、重大なセキュリティリークやデータの紛失が発生する可能性がある。

また、数値データもチェックすること。数値定数をアポストロフで囲むこと。`SELECT * FROM table WHERE ID=234` ではなく、`SELECT * FROM table WHERE ID='234'` のようにする。(ユーザが 234 という値を入力したときに、`SELECT * FROM table WHERE ID=234` というようなクエリをアプリケーションで生成する場合、そのユーザが `234 OR 1=1` という値を入力して、そのアプリケーションで `SELECT * FROM table WHERE ID=234 OR 1=1` というクエリを生成させることができる。その結果、サーバがテーブルの行をすべて読み出してしまい、サーバの負荷が増える。) MySQL は自動的にこの文字列を数値に変換し、非数値記号を削除する。

文字列だけを保護するのは、よくあるミスである。公開データのためのデータベースは保護する必要がないという考えもよくある誤りである。そのようなデータベースにも、サービス妨害タイプの攻撃 (DoS 攻撃) が可能であり、前述の例は、そのようなテクニックを不正使用したケースであり、これによるサーバの脆弱性をつかれ、システムダウンに繋がる。(正当なユーザに対してサービスを提供できなくなる。)

チェックリスト

- すべての Web フォームで、`"` および `'` を入力してみる。何らかの MySQL エラーが発生した場合は、すぐにその問題を調べる。
- URL で `%22` (`"`)、`%23` (`#`)、および `%27` (`'`) を追加して URL の動的修正を試みる。
- 前述の文字を含む URL の動的 データを数値型から文字型に変更する。この類の攻撃があってもアプリケーションに影響しないようにする。
- 数値フィールドに数値ではなく、文字、スペース、および特殊記号を入力してみる。MySQL に渡すことなくアプリケーションがそれらの値を削除するか、エラーを生成することが必要である。値がチェックされずに MySQL に渡ると非常に危険である。
- MySQL に渡す前にデータサイズをチェックする。
- 管理目的で使用するユーザ名とは別のユーザ名で、アプリケーションをデータベースに接続させる。アプリケーションに、必要以上のアクセス権を設定しないこと。
- 様々なアプリケーション プログラムのインターフェースは、データ値に特殊文字をエスケープする手段になる。適切に使用しなければ、アプリケーションのユーザによる入力の原因で、意図したものとは異なる効果を与えるステートメントの生成を回避できる。
- MySQL C API : `mysql_real_escape_string()` API コールを使用する。
- MySQL++ : クエリのストリームには `escape` または `quote` などの修飾子を使用する。
- PHP : PHP 4.3.0以降 は、`mysql_real_escape_string()` 関数を使用する。PHP 4.3.0前の PHP バージョンでは、`mysql_escape_string()` を使用して、PHP 4.0.3 以前は、`addslashes()` を使用する。ノート : `mysql_real_escape_string()` 関数だけが認識文字セット。そのほかの関数は、無効なマルチ バイトの文字セットを使用すると、「擦り抜ける (bypassed)」。PHP 5 では、`mysqli` という拡張モジュールを使用する。これは、改善を施した MySQL の認識プロトコルとパスワード、プレースホルダの準備されたステートメント (prepared statements) をサポートする。
- Perl DBI : プレースホルダ、または `quote()` メソッドを使用する。
- Ruby DBI : プレースホルダ、または `quote()` メソッドを使用する。
- Java JDBC : `PreparedStatement` オブジェクトとプレースホルダ を使用する。

別のプログラミングのインターフェイスでも上記と同様の役割がある可能性がある。

- 暗号化していないプレーンのデータをインターネット経由で送らない。インターネット経由の情報は、時間と技術があれば誰にでもアクセスでき、第三者によって悪用される可能性がある。MySQL では、バージョン 4.0.1以降、内部 SSL 接続をサポートしている。SSH ポート転送を使用して、暗号化 (および圧縮) された通信トンネルを作成できる。
- `tcpdump` や `strings` などのユーティリティの使用法を理解すること。ほとんどの場合、以下のようなコマンドで、MySQL データ ストリームが暗号化されているかどうかをチェックできる。

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

これは、Linux システムで有効。他のシステムでも、少し修正するだけで使用できる。警告 : データを見ることができなくても、必ずしも実際に暗号化されているとは限らない。高度なセキュリティが必要な場合は、専門家に相談する。

MySQL のクラッカー対策

MySQL サーバに接続するときは、パスワードを使用します。MySQL 4.1.1 以降、クライアント接続中のパスワード処理に関して、より高度なセキュリティでグレードアップしています。パスワードは平文テキストでは送信していませんが、MySQL 4.1 より前のバージョンでの暗号化アルゴリズムは、新バージョンで採用しているパスワード形式ほど強力なものではありません。そのため、古いパスワード形式を使用している場合は、クライアントとサーバ間のトラフィックを盗聴できれば、クラッカーは少しの努力でパスワードを探り当てることができるようになります。パスワードのメカニズムに関しては、[MySQL 4.1 のパスワードハッシュ](#) を参照してください。

MySQL Enterprise

The MySQL Network Monitoring and Advisory Service ##### <http://www.jp.mysql.com/products/enterprise/advisors.html> #####

パスワード以外の情報はテキスト形式で送信されているため、第三者によって内容を読み盗られる可能性があります。クライアントとサーバ間の接続が信用できないネットワークを経由する場合に、この可能性を懸念するときは、解読が困難な圧縮プロトコルを使用します。接続時のセキュリティをさらに高めるには、MySQL の内部 SSL サポートを使用します。(接続安全を参照のこと。) 別の方法としては、MySQL サーバと MySQL クライアント間で暗号化 TCP/IP 接続に SSH を利用することも可能です。Open Source SSH クライアントは <http://www.openssh.org/> を、商用の SSH クライアントは、<http://www.ssh.com/> をそれぞれ参照してください。

MySQL システムのセキュリティを確保するためには、以下の事項を強く推奨します。

- すべての MySQL ユーザにパスワードを使用する。クライアント プラグラム側では、そのプログラムを実行してる者が誰であるのかを知る必要はなく、クライアント/サーバ型のアプリケーションにおいては、クライアント側で任意のユーザ名を指定できるのが一般的である。たとえば、別の者が接続に `mysql` プログラムを簡単に使用することができ、つまり、`other_user` にパスワードが設定されていなければ、だれでも `mysql -u other_user db_name` として簡単に他人になりすましてログインできる。すべてのユーザにパスワードを使用すれば、他人のユーザ名で接続することが困難になる。

パスワードのセッティング方法に関する記述は、[パスワードの設定](#) を参照のこと。

- MySQL サーバ (デーモン) を Unix `root` アカウントで実行しないこと。これを行なうと、`FILE` 権限のあるユーザであれば誰でも `root` (例: `~root/.bashrc`) としてファイルを作成できてしまうので、非常に危険である。これを防ぐために、`--user=root` オプションを使用して直接指定された場合を除き、`mysqld` は `root` として実行することを拒否するようになっている。

`mysqld` は、普通のユーザ、つまり権限なしのユーザで実行できる。新しい Unix アカウント `mysql` を作成してさらにセキュリティを高めることができ、これを、MySQL 管理専用のアカウントとする。別の Unix アカウントで `mysqld` を開始するには、サーバのデータディレクトリにある `my.cnf` オプション設定ファイルの `[mysqld]` グループに、アカウント名を指定する `user` 行を追加する。次に例を示す。

```
[mysqld]
user=mysql
```

これで、サーバを手動で起動した場合も、`mysqld_safe` または `mysql.server` を使用して起動した場合でも、指定のアカウントでサーバが起動する。詳細は [ユーザによる MySQL の実行](#) を参照のこと。

`mysqld` を `root` ではない Unix ユーザで実行するということは、`user` テーブルで `root` というユーザ名を変更する必要がある、という意味ではない。MySQL アカウントのユーザ名と、Unix アカウントにはお互い何の関係もない。

- テーブルへのシンボリック リンクをサポートしない。(これは `--skip-symbolic-links` オプションで無効にできる)。このことは、`root` で `mysqld` を実行する場合、`mysqld` サーバ データ ディレクトリへの書き込み権限があるユーザであれば誰でも、システムのすべてのファイルを削除できることになるので、特に重要である。[Unix 上のテーブルに対するシンボリックリンクの使用](#) を参照のこと。
- `mysqld` を実行する Unix アカウントだけに、データベース ディレクトリの読み取り権限と書き込み権限があることを確認する。
- `PROCESS` または `SUPER` の権限を管理側ユーザには以外には与えない。`mysqladmin processlist` と `SHOW PROCESSLIST` の出力には、現在実行中のクエリのテキストが表示される。そのため、これらコマンドを実行できるユーザは、`UPDATE user SET password='PASSWORD('not_secure')` などのクエリを実行して、サーバ プロセス リストを開示してしまう可能性がある。

`mysqld` は、`SUPER` 権限を持つユーザ用に特別接続枠を予約しているので、すべての通常接続が使用中の場合でも、MySQL `root` ユーザはログインしてサーバの状態をチェックできる。

`SUPER` 権限はクライアント接続を中断でき、システム変数を変更することでサーバのオペレーションを変更し、さらに、レプリケーション サーバもコントロールする。

- `FILE` 権限をすべてのユーザには与えない。この権限を持つユーザは、`mysqld` デーモンの権限でファイルシステムのどの場所にもファイルを書き込める。安全対策として、`SELECT ... INTO OUTFILE` で生成されるファイルについてはだれでも書き込み可能だが、既存のファイルには書き込めないようになっている。

`FILE` 権限は、サーバを実行している Unix ユーザがアクセスできるすべての読み取り可能ファイルを読む場合にも使用できる。また、ユーザが何らかの権限を持っているカレントデータベースに任意のファイルを読み込むことができる。つまり、不正使用される可能性がある。たとえば、`LOAD DATA` を使用して `/etc/passwd` をテーブルにロードし、`SELECT` で読むことができる。

- DNS を信頼しない場合、権限テーブルで、ホスト名ではなく IP アドレスを使用すること。いずれにしても、ワイルドカードを含むホスト名を使用して権限テーブルを作成する際には特に注意が必要である。
- 単一ユーザの接続数を制限する場合は、`mysqld` で `max_user_connections` 変数を設定する。`GRANT` ステートメントでも、ユーザへのサーバ接続を制限するリソースコントロール オプションがある。`GRANT` 構文 を参照のこと。

セキュリティ関連の `mysqld` オプション

次の `mysqld` オプションはセキュリティに影響します。

- `--allow-suspicious-udfs`

メイン関数しか持っていないユーザ定義関数をロードすることが出来るかどうかを制御する。(xxxという名前のユーザ関数を定義するとき。)デフォルトでは、このオプションはオフで、少なくとも一つの追加関数を持ったユーザ定義関数だけをロードできる。これによって、不正なユーザ定義関数がロードされる危険性を防ぐ。詳細は [User-Defined Function Security Precautions](#) を参照。

- `--local-infile[={0|1}]`

`--local-infile=0` を使用すると、クライアントで `LOAD DATA` ステートメントの `LOCAL` を使用できなくなる。`LOAD DATA LOCAL` の [セキュリティ関連事項](#) を参照のこと。

- `--old-passwords`

新たなパスワードに古いパスワード形式でのハッシュの生成を強制する。(MySQL 4.0 以前のパスワードを強制する。)これは、サーバが古いクライアント プログラムをサポートする必要がある場合に有用である。詳細は [MySQL 4.1 のパスワードハッシュ](#) を参照。

- `--safe-show-database` (OBSOLETE)

前バージョンの MySQL では、このオプションが `SHOW DATABASES` ステートメントに影響し、どのユーザにどのような権限が与えられているかを示すデータベース名を表示していた。MySQL 5.1 では、このオプションはデフォルトで使用不可としています。ユーザ毎にデータベース名へのアクセスを制御することができるものは `SHOW DATABASES` 権限が存在します。`GRANT` 構文 を参照のこと。

- `--safe-user-create`

このオプションが有効になっている場合、ユーザに `mysql.user` テーブルへの `INSERT` 権限がなければ、そのユーザは `GRANT` コマンドを使用して新規 MySQL ユーザを作成できない。事前設定された権限で新規ユーザを作成できるようにするには、対象ユーザに以下の権限を設定する。

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

これで、ユーザは権限カラムを直接変更できないが、`GRANT` コマンドを使用して他のユーザに権限を与えることができるようになる。

- `--secure-auth`

4.1 より前のパスワードでのアクセスを認証しない。

`mysql` クライアントには、`--secure-auth` オプションもあり、これは、サーバがそのクライアントに対して古い形式のパスワードを要求する場合に、サーバへの接続を拒否する。

- `--skip-grant-tables`

サーバが一切の権限システムを使用しないようにする。これによりすべての人が、すべてのデータベースにアクセスできるようになる。実行中のサーバに権限テーブルの使用を再開するには、`mysqladmin flush-privileges` または

mysqladmin reload をシステム シェルから実行する。またはサーバ接続後に MySQL [FLUSH PRIVILEGES](#) ステートメントを発行する。このオプションは、プラグインとユーザ定義関数 (UDF) のロードを抑圧する。

- [--skip-merge](#)

[MERGE](#) ストレージ エンジンが無効化する (MySQL 5.1.12 での追加)。ユーザに [MyISAM](#) テーブル [t](#) にアクセスできる場合に、そのユーザが、[t](#) へアクセスする [MERGE](#) テーブル [m](#) を作成できる。しかし、[t](#) でのこのユーザ権限が連続して呼び出だすときは、[m](#) を経由して、[t](#) へアクセスを継続できる。

- [--skip-name-resolve](#)

ホスト名を決定できない。権限テーブルの [Host](#) カラム値すべてが、IP アドレスまたは [localhost](#) である必要がある。

- [--skip-networking](#)

TCP/IP 経由の接続を認めない。[mysqld](#) への接続をすべて Unix ソケットで行う。

- [--skip-show-database](#)

ユーザが [SHOW DATABASES](#) 権限を持っていない場合に、[SHOW DATABASES](#) コマンドを無効にする。ステートメントはすべてのデータベース名を表示する。このオプションをセットしない場合、[SHOW DATABASES](#) はすべてのユーザが利用できるようなるが、ユーザが [SHOW DATABASES](#) 権限、またはそのデータベースに関する何らかの権限を持っているときには、それぞれのデータベース名だけが表示される。注意：すべてのグローバル権限がデータベースに対する権限として扱われる。

- [--ssl*](#)

[--ssl](#) で始まるオプションで、SSL経由での接続をクライアントに許可するかどうかを指定し、SSL キーと証明書がどこにあるかを示す。詳細は [SSL コマンド オプション](#) を参照。

LOAD DATA LOCAL のセキュリティ関連事項

[LOAD DATA](#) ステートメントはサーバ ホストに置かれているファイルをロードできます。または [LOCAL](#) キーワードが指定された場合に、クライアント ホストに位置するファイルをロードできます。

[LOAD DATA](#) ステートメントの[LOCAL](#) バージョンのサポートに関しては、潜在的な問題が 2 つあります。

- ファイルの読み取りがサーバ側から開始される。そのため、理論的には、改悪した MySQL サーバを作成しておけば、クライアントがテーブルに対してクエリを実行した時に、クライアント コンピュータ上に存在する全てのファイル (カレントユーザが読み取り権を持つ) を、[LOAD DATA](#) ステートメントの改悪した MySQL サーバが読み取れるということになります。
- クライアントが Web サーバから接続する Web 環境では、ユーザは [LOAD DATA LOCAL](#) を使用して、Web サーバ プロセスが読み取りアクセス権を持つどのファイルでも読み取ることができます。これは、ユーザが SQL サーバに対してすべてのコマンドを実行できる場合です。この環境では、MySQL サーバに対するクライアントは実際には Web サーバであり、Web サーバに接続するユーザによるリモート プログラムのことではありません。

この問題を解決するには、MySQL 3.23.49 と MySQL 4.0.2 (Windows では 4.0.13) 以降の [LOAD DATA LOCAL](#) の扱い方を変更しました。

- デフォルトで、MySQL クライアントとバイナリ配布のすべてを、[--enable-local-infile](#) オプションでコンパイルしました。これは MySQL 3.23.48 以前の MySQL との互換性のためです。
- MySQL をソースから組み、[--enable-local-infile](#) オプションで [configure](#) を呼び出していない場合、[mysql_options\(... MYSQL_OPT_LOCAL_INFILE, 0\)](#) を明示的に呼び出すと指さなければ、クライアントは [LOAD DATA LOCAL](#) を使用できません。[mysql_options\(\)](#) を参照のこと。
- [--local-infile=0](#) オプションで [mysqld](#) を起動すると、サーバ側からすべての [LOAD DATA LOCAL](#) コマンドを無効にできます。
- [mysql](#) のコマンドライン クライアントでは、[--local-infile\[=1\]](#) オプションを指定すると、[LOAD DATA LOCAL](#) を有効化し、[--local-infile=0](#) オプションで無効化します。同様に、[mysqlimport](#) では、[--local](#) または [-L](#) のオプションで、

ローカルのデータ ファイル ロードを有効にします。どのような場合でも、ローカルでのロード操作は、サーバがそれを許可するかどうかによります。

- オプション ファイルから `[client]` グループを読むような、Perl スクリプトまたはその他のプログラムで、`LOAD DATA LOCAL` を使用する場合、`local-infile=1` をそのグループに追加できます。しかし、`local-infile` を認識しないプログラムで問題が発生しないようにするために、例示のように、`loose-` プレフィックスを使います。

```
[client]
loose-local-infile=1
```

- `LOAD DATA LOCAL INFILE` を有効にする場合、サーバまたはクライアントのどちらかで、そのようなステートメントを発行しようとするクライアントは、次のようなエラー メッセージを受け取ります。

```
ERROR 1148: The used command is not allowed with this MySQL version
```

```
MySQL Enterprise
MySQL Network Monitoring and Advisory Service ##### --local-infile #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

ユーザによる MySQL の実行

Windows 環境では、通常のユーザ アカウントで、Windows サービスとしてサーバを実行できます。

Unix 環境では、MySQL サーバの `mysqld` をどのユーザでも起動できます。しかし、セキュリティ面への配慮から、Unix `root` ユーザでサーバを実行することは推奨していません。`mysqld` を変更し、普通の権限なし Unix ユーザ、`user_name` で実行するには、次のことを行ないます。

1. サーバ実行中であれば、終了する。(`mysqladmin shutdown` コマンドを使用)
2. `user_name` でファイルへの読み書き込み権限を与え、データベース ディレクトリとファイルを変更する。場合によっては、この作業を、Unix `root` ユーザで行なう必要がある。

```
shell> chown -R user_name /path/to/mysql/datadir
```

この作業を行なわない場合、`user_name` で実行するときに、サーバからデータベースまたはテーブルへのアクセスができない。

MySQL データ ディレクトリ内のディレクトリまたはファイルがシンボリック リンクである場合は、そのリンクに従い、ディレクトリとファイルを指しているところを変更する。場合によって、`chown -R` では、シンボリック リンクできないことがある。

3. `user_name` というユーザでサーバを起動する。MySQL 3.22 以降を使用している場合は、別の方法として、Unix `root` で `mysqld` を起動し、`--user=user_name` オプションを使用する。`mysqld` が起動したら、接続を開始する前に、Unix ユーザの `user_name` に切り替えて、実行する。
4. システムの起動時に自動的に任意のユーザでサーバを起動するには、`user` オプションを `/etc/my.cnf` オプション ファイルの `[mysqld]` グループに追加するか、サーバのデータ ディレクトリの `my.cnf` オプション ファイルを使用して、ユーザ名を指定する。

```
[mysqld]
user=user_name
```

Unix マシン自体が安全ではない場合、権限テーブルの MySQL `root` アカウントにパスワードを割り当てる。これをしないと、そのマシンのログイン アカウントを持つユーザであれば誰でも、`--user=root` で `mysql` を実行でき、どのような操作をも行なうことができる。つまり、常に MySQL アカウントにパスワードを割り当てるのが賢明である。そのサーバ ホストで別のログイン アカウントが存在する場合は特にそうである。[インストール後の設定とテスト](#) を参照のこと。

MySQL アクセス権限システム

MySQL では、高度化し、かつ非標準のセキュリティおよび権限システムを採用しています。ここでは、それらがどのように動作するかを説明します。

権限システムの役割

MySQL 権限システムの主な役割は、ホストから接続するユーザの認証と、[SELECT](#)、[INSERT](#)、[UPDATE](#)、[DELETE](#) などのデータベースにおける権限があるユーザを関連付けることです。

アノニマス ユーザ (不特定多数のユーザー) を持つこと、そして、管理操作や [LOAD DATA INFILE](#) などのMySQL 特有の関数での権限を与えるといった追加の機能性を備えています。

権限システムの機能

MySQL 権限システムでは、すべてのユーザが許可がある操作だけを行います。ユーザは MySQL サーバに接続すると、接続元のホストと指定するユーザ名でそのアイデンティティ (ID) を認識します。接続後のリクエスト発行では、権限システムが、ユーザ ID とそれが行なう操作に応じて権限を設定します。

MySQL では、ホスト名とユーザ名を使用してユーザを認証します。1 つのユーザ名がインターネット上のどこでも同じユーザを示しているという保証がないためです。たとえば、[office.example.com](#) から接続したユーザ [joe](#) は、[home.example.com](#) から接続した [joe](#) と同一人物とは限りません。MySQL では、同じユーザ名でも異なるホストから接続するユーザ間で区別して、これを処理します。[office.example.com](#) から接続した [joe](#) と、[home.example.com](#) から接続した [joe](#) には別々の権限セットを設定します。

MySQL のアクセス制御には、サーバに接続するクライアント プログラムを実行するときに、2 段階を踏みます。

- 段階 1: ユーザに接続する権限があるかどうかサーバがチェックする。
- 段階 2: 接続できた場合、要求ごとにそれを実行できる権限があるかどうかサーバがチェックする。たとえば、データベースのテーブルからレコードを [SELECT](#) したり、データベースのテーブルを [DROP](#) しようとする、ユーザにそのテーブルの [SELECT](#) 権限があるかどうか、あるいはデータベースの [DROP](#) 権限があるかどうかをサーバがチェックする。

接続中に権限を変更した場合 (ユーザ自身または第三者によって)、必ずしもその変更は次のクエリに反映するとは限りません。詳細は、[権限の変更が反映するタイミング](#) を参照してください。

サーバは、[mysql](#) データベース ([mysql](#) と名付けられたデータベース) の権限テーブルに権限情報を保管します。MySQL サーバは、起動するとこのテーブルの内容をメモリに読み込み、[権限の変更が反映するタイミング](#) で示するような状況においては、それらを再読み込みします。アクセス制御の決定は、権限テーブルの内部コピーを基に行います。

通常、[GRANT](#) や [REVOKE](#) などのクエリを使用して、権限テーブルの内容を間接的に操作し、アカウントのセットアップや権限のコントロールを行ないます。[アカウント管理ステートメント](#) も参照してください。ここでは、権限テーブルの根本的なストラクチャと、サーバがクライアントとのやりとりでどのようにそれを使用するかについて説明します。

サーバでは、両方の段階でのアクセス制御で、[mysql](#) データベースの [user](#)、[db](#)、そして [host](#) テーブルを使用します。[user](#) と [db](#) のフィールドをここで示します。[host](#) テーブルは、[db](#) テーブルと類似しますが、[アクセス制御の段階 2: 接続確認](#) で説明するように特別な使い方をします。

テーブル名	user	db
スコープ フィールド	Host	Host
	User	Db
	Password	User
権限 フィールド	Select_priv	Select_priv
	Insert_priv	Insert_priv
	Update_priv	Update_priv
	Delete_priv	Delete_priv
	Index_priv	Index_priv
	Alter_priv	Alter_priv

	Create_priv	Create_priv
	Drop_priv	Drop_priv
	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Trigger_priv	Trigger_priv
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
セキュリティ フィールド	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
リソース制御フィールド	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

Event_priv と Trigger_priv のフィールドは MySQL 5.1.6 で追加しました。

アクセス制御の 2 段階目 (要求確認) で、要求がテーブルに関連する場合、サーバはそれぞれのクライアントに適切な権限があるかどうかを確認します。それに加えて、user、db、host の権限テーブルで、サーバが tables_priv と columns_priv テーブルを参照します。tables_priv と columns_priv テーブルで、テーブルとカラムの適切な権限制御を行なうことができます。これには次のフィールドがあります。

テーブル名	tables_priv	columns_priv
スコープ フィールド	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
権限フィールド	Table_priv	Column_priv
	Column_priv	

その他のフィールド	Timestamp	Timestamp
	Grantor	

Timestamp と Grantor のフィールドは現在未使用です。.

ストアド ルーチンに関わる要求確認では、サーバは `procs_priv` テーブルを参照します。このテーブルには次のようなフィールドがあります。

テーブル名	procs_priv
スコープ フィールド	Host
	Db
	User
	Routine_name
	Routine_type
権限フィールド	Proc_priv
その他のフィールド	Timestamp
	Grantor

Routine_type は、'FUNCTION' または 'PROCEDURE' の値を伴う ENUM フィールドであり、その行が示すルーチンのタイプを指します。このフィールドでは、関数とプロシージャが同じ名前である場合に、別々に権限を与えます。

Timestamp と Grantor のフィールドは現在未使用です。.

それぞれの権限テーブルには、スコープ フィールドと権限フィールドがあります。

- ・ スコープ フィールドは、テーブルの各登録 (エントリ) の範囲を特定します。たとえば、Host と User の値が 'thomas.loc.gov' および 'bob' である user テーブル エントリは、thomas.loc.gov ホストからサーバに接続しようとする bob を認証します。同様に、Host、User、Db のそれぞれのフィールドの値が 'thomas.loc.gov'、'bob' 'reports' である db テーブル エントリは、thomas.loc.gov ホストから reports データベースに接続しようとする bob を認証します。tables_priv テーブルおよび columns_priv テーブルには、それぞれのエントリを許可しているテーブルまたはテーブルとカラムの組み合わせを示すスコープフィールドがあります。procs_priv スコープ フィールドはエントリに適用するストアド ルーチンを示します。
- ・ 権限フィールドは、テーブル内のエントリごとに設定している権限、つまり何の操作を実行できるかを示します。サーバはさまざまな権限テーブルの情報を組み合わせて、ユーザの権限についての完全な記述を生成します。この動作に適用できるルールについては [アクセス制御の段階 2：接続確認](#) を参照してください。

スコープフィールドは文字列です。ここで示すように、それぞれのデフォルト値は空文字列です。

フィールド名	型
Host	CHAR(60)
User	CHAR(16)
Password	CHAR(16)
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

アクセスをチェックでは、Host 値の比較には大文字と小文字を区別します。User、Password、Db、および Table_name の値については大文字と小文字を区別します。Column_name と Routine_name の値は、大文字と小文字の区別は不要です。

`user`、`db`、`host` のテーブルでは、それぞれの権限を別々のカラムにリストしています。これは、`ENUM('N','Y')` `DEFAULT 'N'` と宣言しています。つまり、それぞれの権限は無効化、および有効化が可能です。

`tables_priv`、`columns_priv`、`procs_priv` のテーブルでは、権限フィールドは `SET` フィールドとして宣言しています。これらのフィールド値はテーブルでコントロールしている権限の組み合わせを含みます。

テーブル名	フィールド名	設定可能な要素
<code>tables_priv</code>	<code>Table_priv</code>	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'
<code>tables_priv</code>	<code>Column_priv</code>	'Select', 'Insert', 'Update', 'References'
<code>columns_priv</code>	<code>Column_priv</code>	'Select', 'Insert', 'Update', 'References'
<code>procs_priv</code>	<code>Proc_priv</code>	'Execute', 'Alter Routine', 'Grant'

ここで、サーバが権限テーブルをどのように使用するかを簡潔に説明します。

- `user` テーブルの `Scope` フィールドで、着信した接続を許可または拒否のどちらかを決定する。接続を許可すると、`user` テーブルに設定している権限はいずれも、そのユーザのグローバル（スーパーユーザ）権限になる。これらの権限は、サーバのすべてのデータベースに適用となる。

ノート：グローバル権限はいずれも、すべてのデータベースに対する権限として扱うため、グローバル権限を持つユーザは、`SHOW DATABASES` を使用したり、あるいは `INFORMATION_SCHEMA` の `SCHEMATA` テーブルを調べたりすると、すべてのデータベース名を閲覧できるようになる。

- `db` テーブルの `Scope` フィールドで、どのユーザがどのホストからどのデータベースにアクセスできるかを決定する。権限フィールドから、何の操作を許可しているか判断する。データベースのレベルで権限があると、データベースとそのすべてのテーブルにアクセスできる。
- `host` テーブルを、任意の `db` テーブル エントリをいくつかのホストに適用する場合、`db` テーブルで補完するものとして使用する。たとえば、1人のユーザに対して、ネットワーク内の複数のホストからデータベースへアクセスすることを許可する場合、ユーザの `db` テーブル エントリの `Host` 値を空白のままにして、それらのホストのそれぞれのエントリを `host` テーブルに入力する。このメカニズムの詳細については、[アクセス制御の段階 2：接続確認](#) を参照のこと。

ノート：`host` テーブルは、`INSERT`、`UPDATE`、`DELETE` などのステートメントで直接、変更する必要がある。`GRANT` や `REVOKE` などのステートメントは、間接的に権限テーブルを修正し、これらのステートメントからは効果がない。MySQL のインストールでは、このテーブルは、例外を除き、使用しない。

- `tables_priv` と `columns_priv` のテーブルは、`db` テーブルと類似するが、これらはより詳細な設定が可能。データベース レベルではなく、テーブルおよびカラム レベルでの適用となる。テーブルで権限を与えた場合、それはテーブル、およびすべてのカラムでの適用となる。カラム レベルで権限を与えた場合、特定のカラムにだけの適用となる。
- `procs_priv` テーブルはストアド ルーチンへの適用となる。ルーチン レベルで権限を与えた場合、単一のルーチンにだけの適用となる。

注意：管理者権限（`RRELOAD`、`SHUTDOWN` など）は、`user` テーブルで指定します。管理操作はデータベース固有ではなく、サーバそのもので行う操作です。そのため、この権限を他の権限テーブルで設定する必要はありません。実際に、管理操作を実行できるかどうか決定するときには、`user` テーブルを参照するだけで済みます。

`FILE` 権限は `user` テーブルで指定します。これは管理者権限ではありません。サーバ ホスト上でのファイルの読み書きは、アクセスしているデータベースからは独立しています。

`mysqld` サーバは権限テーブルの内容を、起動時に 1 回メモリへ読み取ります。`FLUSH PRIVILEGES` ステートメントを発行するか、または `mysqladmin flush-privileges` あるいは `mysqladmin reload` のコマンドを実行して、このテーブルの再読み込みを行なうことも可能です。権限テーブルへの変更してから反映できるまでの詳細は [権限の変更が反映するタイミング](#) を参照してください。

権限テーブルの内容を変更するときは、その変更で、目的とする権限を適切に設定したかどうかを確認してください。任意のアカウントへの権限をチェックするには、`SHOW GRANTS` ステートメントを使用します。（`SHOW GRANTS`

構文を参照のこと。)たとえば、Host が `pc84.example.com` で、User が `bob` のアカウントに与えた権限を確認するには、次のようにステートメントを発行します。

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

権限関連問題の追加的な診断ヘルプに関しては、[Access denied エラーの原因](#) を参照してください。その他、セキュリティに関連のアドバイスに関しては、[セキュリティ問題](#) を参照してください。

MySQL 提供の権限

ユーザ権限に関する情報は、`mysql` データベース (`mysql` という名前のデータベース) の `user`、`db`、`host`、`tables_priv`、`columns_priv`、`procs_priv` などのテーブルにあります。MySQL サーバは起動時、および [権限の変更が反映するタイミング](#) で説明する状況下において、これらのテーブルの内容をメモリへ読み取ります。アクセス制御に関する決定は、権限テーブルの内部メモリを基に行ないます。

`GRANT` や `REVOKE` などのステートメントで権限を指すときに使用する名前を、次の一覧に表示します。この一覧には、権限テーブルのそれぞれの権限に関連するカラム名とその権限を適用する内容 (適用範囲) も示します。それぞれの権限に関する意味などに関しては、[GRANT 構文](#) を参照してください。

権限	カラム	適用範囲
<code>CREATE</code>	<code>Create_priv</code>	データベース、テーブル、またはインデックス
<code>DROP</code>	<code>Drop_priv</code>	データベースまたはテーブル
<code>GRANT OPTION</code>	<code>Grant_priv</code>	データベース、テーブル、またはストアドルーチン
<code>REFERENCES</code>	<code>References_priv</code>	データベースまたはテーブル
<code>EVENT</code>	<code>Event_priv</code>	データベース
<code>ALTER</code>	<code>Alter_priv</code>	テーブル
<code>DELETE</code>	<code>Delete_priv</code>	テーブル
<code>INDEX</code>	<code>Index_priv</code>	テーブル
<code>INSERT</code>	<code>Insert_priv</code>	テーブル
<code>SELECT</code>	<code>Select_priv</code>	テーブル
<code>UPDATE</code>	<code>Update_priv</code>	テーブル
<code>TRIGGER</code>	<code>Trigger_priv</code>	テーブル
<code>CREATE VIEW</code>	<code>Create_view_priv</code>	ビュー
<code>SHOW VIEW</code>	<code>Show_view_priv</code>	ビュー
<code>ALTER ROUTINE</code>	<code>Alter_routine_priv</code>	ストアドルーチン
<code>CREATE ROUTINE</code>	<code>Create_routine_priv</code>	ストアドルーチン
<code>EXECUTE</code>	<code>Execute_priv</code>	ストアドルーチン
<code>FILE</code>	<code>File_priv</code>	サーバ上のファイル アクセス
<code>CREATE TEMPORARY TABLES</code>	<code>Create_tmp_table_priv</code>	サーバ管理
<code>LOCK TABLES</code>	<code>Lock_tables_priv</code>	サーバ管理
<code>CREATE USER</code>	<code>Create_user_priv</code>	サーバ管理
<code>PROCESS</code>	<code>Process_priv</code>	サーバ管理
<code>RELOAD</code>	<code>Reload_priv</code>	サーバ管理
<code>REPLICATION CLIENT</code>	<code>Repl_client_priv</code>	サーバ管理
<code>REPLICATION SLAVE</code>	<code>Repl_slave_priv</code>	サーバ管理
<code>SHOW DATABASES</code>	<code>Show_db_priv</code>	サーバ管理
<code>SHUTDOWN</code>	<code>Shutdown_priv</code>	サーバ管理

SUPER	Super_priv	サーバ管理

MySQL のリリースによっては、権限テーブルのストラクチャへの変更に、新たな権限または特徴を追加することになっています。新しいバージョンの MySQL にアップデートするときは、常に、権限テーブルの更新も同時に行い、カレントストラクチャであることを確認し、新しい機能の利点を扱えるようにします。[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照してください。

EVENT および TRIGGER の権限は MySQL 5.1.6 に追加しました。

バイナリ ログを有効にした場合に格納関数の作成または置換するには、SUPER 権限が必要な場合があります。[ストアドルーチンとトリガのバイナリログ](#) を参照してください。

CREATE や DROP などの権限で、新たなデータベースやテーブルの作成が行なえます。既存のデータベースやテーブルを削除することも可能です。MySQL 5.1.10 初期では、ALTER TABLE ... DROP PARTITION のステートメントを分割したテーブルに使用するとき DROP 権限を必要としています。MySQL 5.1.16 初期では、DROP 権限は TRUNCATE TABLE に使用します。(TRUNCATE TABLE には DELETE 権限が必要。) ユーザに対して、mysql データベースの DROP 権限を与えると、そのユーザで MySQL のアクセス権限があるデータベースの削除が行なえます。

SELECT、INSERT、UPDATE、DELETE などは、データベースの既存テーブルのレコード操作ができるようにする権限です。ANALYZE TABLE、OPTIMIZE TABLE、REPAIR TABLE などのテーブル保守に関するステートメントには、INSERT 権限が必要です。

SELECT ステートメントで、テーブルから直接にレコードを読み取る場合には SELECT 権限が必要です。SELECT ステートメントはテーブルへのアクセスはできませんが、データベースへのアクセス権がなくても実行できます。たとえば、mysql クライアントをシンプルな計算機として使用するような場合です。

```
SELECT 1+1;
SELECT PI()*2;
```

INDEX は、インデックスを作成または破棄 (削除) できる権限です。INDEX 権限は既存テーブルにでの適用となります。テーブルに対して CREATE 権限がある場合、CREATE TABLE ステートメントにインデックス定義を含めることも可能です。

ALTER 権限があると、ALTER TABLE でテーブルのストラクチャを変更したり、テーブルの名前を変更することができます。

CREATE ROUTINE はストアド ルーチンの作成に必要な権限です。ALTER ROUTINE はストアド ルーチンの置換や削除に必要な関数です。EXECUTE はストアド ルーチンの実行に必要な権限です。

TRIGGER はテーブルのトリガの作成と削除に必要な権限です。MySQL 5.1.6 以前では、この操作には、SUPER 権限が必要です。

EVENT はイベント スケジューラのイベント セットアップに必要な権限です。

GRANT はユーザが自分の権限を他のユーザに与える権限です。これはデータベース、テーブル、ストアド ルーチンに使用できます。

FILE は、LOAD DATA INFILE および SELECT ... INTO OUTFILE ステートメントを使用してサーバ上でファイルの読み書きを行う権限です。FILE 権限を持つユーザは、サーバホストのすべてのファイル、つまり MySQL サーバの読み込み可能ファイルを読み込めます。(これは、この権限を持つユーザであれば、サーバがファイルへのアクセスを許可するために、データベースディレクトリのファイルを読むことができるということです。) さらに、FILE 権限があるユーザは、MySQL サーバが書き込みアクセスのあるディレクトリに、新しいファイルを作成できます。セキュリティ対策として、サーバは既存ファイルの上書きは行ないません。

その他の権限は、管理者用の権限です。この多くは、mysqladmin プログラムや SQL ステートメントを発行して実行できます。次の一覧テーブルは、どの管理権限で、mysqladmin コマンドで実行できるかを示します。

権限	実行可能なコマンド
RELOAD	flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, reload
SHUTDOWN	shutdown
PROCESS	processlist

SUPER

kill

reload コマンドは、権限テーブルを再読み込みするよう、サーバに命令します。**refresh** コマンドは、すべてのテーブルをフラッシュし、ログファイルを開いて閉じます。**flush-privileges** は **reload** のシノニムです。他の **flush-xxx** コマンドも **refresh** と同様の役割を果たしますが、範囲が限られているため、状況によって使い分けます。たとえば、ログファイルだけをフラッシュするときは、**refresh** の代わりに **flush-logs** を使用します。

shutdown コマンドでサーバをシャットダウンします。これに関連する SQL ステートメントはありません。

processlist コマンドは、サーバで実行中のスレッドに関する情報を表示します。**kill** コマンドはサーバスレッドを強制終了します。ユーザはいつでも自分のスレッドを表示および強制終了することができますが、他のユーザで開始したスレッドを表示するには **PROCESS** 権限が必要です。強制終了するには **SUPER** 権限が必要です。**KILL** 構文を参照してください。

CREATE TEMPORARY TABLES は **CREATE TABLE** ステートメントの **TEMPORARY** のキーワード使用を可能にする権限です。

LOCK TABLES は、**SELECT** 権限でテーブルをロックするために、明示的な **LOCK TABLES** の使用を許可する権限。この権限で書き込みロックを制御して、不正読み込みからテーブルを守ります。

REPLICATION CLIENT は、**SHOW MASTER STATUS** および **SHOW SLAVE STATUS** の使用を可能にする権限です。

REPLICATION SLAVE は、スレーブサーバがマスタとしてカレントサーバに接続するときに使用するアカウントに対して与える権限です。この権限がないと、スレーブは、マスタサーバのデータベースに対して行なわれた更新を要求できません。

SHOW DATABASES はデータベース名の閲覧を可能にする権限です。**SHOW DATABASE** 権限がないアカウントでは、限定範囲でデータベースの内容を閲覧することができます。**--skip-show-database** オプションでサーバを起動している場合には、このステートメントを使用できません。ノート：グローバル権限があると、データベースに対するすべての権限があることを意味します。

必要がない限り、この権限をアカウントに与えないでください。**FILE** 権限の付与と管理権限に関しては、次の事柄に対する十分な注意が必要です。

- **FILE** 権限の悪用で、MySQL サーバの読み取り可能ファイル、またはカレントデータベースディレクトリのファイルをデータベーステーブルに対して、**SELECT** を使用してその内容にアクセス可能になる。
- **GRANT** 権限は、ユーザが自分の権限を他のユーザに与えることができる。2人のユーザが異なる権限を持ち、両方とも **GRANT** 権限がある場合、お互いの権限を組み合わせることができる。
- **ALTER** 権限は、テーブルの名前を変更して、権限システムを崩壊することができる。
- **SHUTDOWN** 権限の悪用で、サーバがシステム終了し、他のユーザへのサービス妨害となる。
- **PROCESS** 権限は、パスワードの設定や変更を行うクエリなどを含め、現在実行中のクエリの平文テキストを表示することができる。
- **SUPER** 権限は、クライアントの終了と、サーバ動作方法を変更できる。
- **mysql** データベースに対する権限があれば、パスワードおよびその他のアクセス権限情報を変更することができる。(パスワードを暗号化で保護しているため、悪意のあるユーザが単に読み取ってもテキスト形式のパスワードを知ることはできない)。user テーブルの **Password** カラムにアクセスできれば、それを悪用して特定ユーザのMySQLサーバにログインできる。(必要権限があれば、そのユーザはパスワードを書き換えることもできる)。

MySQL Enterprise

```
##### MySQL Network Monitoring and Advisory Service #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

次に、MySQL の権限システムでは不可能なことを示します。

- 特定のユーザに対してアクセス拒否を明示的に指定することはできない。つまり、特定のユーザを明確に割り出し、そこからの接続を拒否することができない、という意味。

- データベース内のテーブルの作成および破棄の権限をユーザに与え、それと同時にデータベースの作成および破棄をできないようには指定できない。
- アカウントのパスワードはグローバルで適用する。つまり、データベース、テーブル、ルーチンなど特定のオブジェクトに対するパスワードは付けられない。

MySQL サーバへの接続

MySQL クライアント プログラムでは、通常、MySQL サーバにアクセスするときに、接続先のホスト、ユーザ名、パスワードといった接続パラメータを指定する必要があります。

- MySQL サーバを実行しているホスト名
- ユーザ名
- パスワード

`mysql` クライアントで次で示すように、コマンドライン プロンプト (ここでは `shell>`) で起動する場合

```
shell> mysql -h host_name -u user_name -p your_pass
```

`-h`、`-u`、`-p` オプションの別の形は、`--host=host_name`、`--user=user_name`、`--password=your_pass` です。`-p` または `--password=` に後続するパスワードの間には、スペースを入れません。

`-p` または `--password` オプションを使用するけれども、パスワード値を指定しない場合、クライアント プログラムがパスワード入力を指示します。このパスワードは入力しても表示しません。このやり方は、パスワードをコマンドラインで与えるよりは安全です。システムを使用するユーザによっては、`ps auxw` などのコマンドを実行して、コマンドラインで指定するパスワードを見る可能性があります。[パスワードのセキュリティ](#) を参照してください。

MySQL クライアント プログラムでは、指定のない限り、接続パラメータにデフォルト値を使用します。

- 初期設定時のホスト名は `localhost`。
- デフォルトのユーザ名は Windows では `ODBC`、Unix では、Unix のログイン名。
- `-p` または `--password` を指定しなければ、パスワード無しになる。

したがって、Unix ユーザ `joe` では、以下のコマンドがいずれも同じ意味になります。

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
shell> mysql -u joe
shell> mysql
```

他の MySQL クライアントでも同様です。

Unix システムでは、接続時に別のデフォルト値を使用するように設定することができます。これにより、クライアント プログラムを起動するたびにコマンドラインで指定する必要がなくなります。設定方法は 2 つあります。

- ホームディレクトリ内にある `.my.cnf` 設定ファイルの `[client]` セクションで接続パラメータを指定する。このセクションは次のようになる。

```
[client]
host=host_name
user=user_name
password=your_pass
```

[オプションファイルの使用](#) でオプション ファイルの詳細を記述しています。

- 環境変数を使用して接続パラメータを指定する。`mysql` のホストは、`MYSQL_HOST` で指定できる。MySQL ユーザ名は、`USER` を使用して指定できる (Windows と NetWare のみ)。パスワードは、`MYSQL_PWD` を使用して指定できる。(ただし、これは安全ではない。[パスワードのセキュリティ](#) を参照のこと)。変数の一覧は [環境変数](#) を参照してください。

アクセス制御の段階 1： 接続確認

ユーザが MySQL サーバに接続しようとした場合、そのユーザ ID、およびそれを正しいパスワードで裏付けできるかどうかによって、サーバは接続の許可または拒否を行います。パスワードが正しくない場合、サーバはアクセスを完全に拒否します。正しければ、サーバは接続を許可し、段階 2 に進んで要求を待ちます。

ユーザ ID は、2 つの情報に基づきます。

- 接続元のホスト
- MySQL ユーザ名

ID チェックには、`user` テーブルの 3 つのスコープフィールド (`Host`、`User`、`Password`) を使用します。`user` テーブルエントリが、指定した `Host` と `User` と一致し、入力したパスワードが正しい場合のみ、接続許可になります。

`user` テーブルの `Host` 値 (スコープ フィールド) には次の方法で値を指定できます。

- `Host` 値にはホスト名または IP アドレスを使用できる。ローカルホストを指定する場合は `'localhost'` を使用する。
- ワイルドカード文字 `'%'` および `'_'` を `Host` フィールドに使用できる。`LIKE` 演算子で実行するマッチング操作と同様の効果がある。たとえば、`Host` 値としての `'%'` はすべてのホスト名を意味する。これに対して、`'%.mysql.com'` という値は `mysql.com` ドメインのすべてのホスト名と一致する。

MySQL

```
'%' ##### MySQL Network Monitoring and Advisory Service #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

- IP アドレスとして指定する `Host` 値に、ネットワークアドレスを示すためのネットマスクを使用できます。次はその例です。

```
GRANT ALL PRIVILEGES ON db.* TO david@'192.58.197.0/255.255.255.0';
```

これは、`david` が `client_ip` という IP アドレスのクライアント ホストから接続できます。これには次の条件があります。

```
client_ip & netmask = host_ip
```

つまり、`GRANT` ステートメント表示です。

```
client_ip & 255.255.255.0 = 192.58.197.0
```

この条件に該当し、MySQL サーバに接続できる IP アドレスは、`192.58.197.0` から `192.58.197.255` までの範囲のもということになります。

ノート: ネットマスクの使用は 8、16、24、または 32 ビットのいずれかのアドレスを使用するようにサーバへ知らせます。たとえば、次の通りです。

- `192.0.0.0/255.0.0.0`: 192 クラスの A ネットワークのすべて
- `192.168.0.0/255.255.0.0`: 192.168 クラスの B ネットワークのすべて
- `192.168.1.0/255.255.255.0`: 192.168 1 クラスの C ネットワークのすべて
- `192.168.1.1`: 特定の IP のみ。

次に示すネットマスク (28 ビット) は使用できません。

```
192.168.0.1/255.255.255.240
```

- `db` テーブル行 (エントリ) の空白の `Host` 値

は、その権限が、クライアントのホスト名と一致する `host` テーブルの行のものと組み合わせるという意味です。この権限は、OR (ユニオン) ではなく、AND (インターセクション) 操作を使用して組み合わせられています。`host` テーブルの使用については、[アクセス制御の段階 2：接続確認](#) を参照してください。

別の権限テーブルの空白の `Host` 値は、`'%'` と同じである。

たとえば、'[144.155.166.%](#)' がサブネットのすべてのホストと一致する、というように、[Host](#) カラムの IP ワイルドカードの値を使用できるということは、ホストを [144.155.166.somewhere.com](#) と命名するなどして、誰かがこの機能 (セキュリティ上の弱点) を悪用できるということです。これを阻止するために、MySQL では、数字やドットで始まるホスト名との一致しないようになっています。つまり、[1.2.foo.com](#) のようなホスト名の場合、この名前は、権限テーブルの [Host](#) カラムとは一致しないということです。IP ワイルドカード値は、IP アドレスとだけ一致し、ホスト名とは一致しません。

[User](#) フィールドには、ワイルドカード文字は使用できません。空白の値は使用でき、これは任意のユーザ名と一致します。[user](#) テーブル エントリに空白のユーザ名があり、接続で空白ユーザにマッチする場合、そのユーザはクライアントが実際に指定した名前のユーザではなく、匿名ユーザ (名前なしのユーザ) との解釈になります。この場合、接続中 (段階 2 の間) のフル アクセス チェックをすべて、空白のユーザ名で行うということです。

[Password](#) フィールドは空白にできます。これはどのパスワードにも一致するという意味ではなく、そのユーザがパスワードを指定せずに接続する必要があるという意味です。

[user](#) テーブルの空白ではない [Password](#) 値は、暗号化パスワードを表します。MySQL では、パスワードを平文テキストでは保存しません。接続しようとするユーザが入力したパスワードを暗号化します ([PASSWORD\(\)](#) 関数を使用)。クライアントおよびサーバがパスワードの確認を行うときは、その暗号化パスワードを使用します。(このとき、その接続を経由して、暗号化パスワードを転送することはありません)。注意：MySQL から見ると暗号化パスワードが実際のパスワードであるため、暗号化パスワードにだれにもアクセスできないようにしてください。特に、一般のユーザに [mysql](#) データベース内のテーブルの読み取りアクセス権を与えないでください。

MySQL 5.1 では (最初の実装は MySQL 4.1 から)、MySQL は従来とは異なるパスワードおよびログインメカニズムを採用しています。万が一、TCP/IP パケットの傍受や [mysql](#) データベースのキャプチャが行なわれた場合でも安全確保できるようになっています。パスワードの暗号化に関する詳細は [MySQL 4.1 のパスワードハッシュ](#) を参照してください。

次の一覧は、[user](#) テーブルエントリの [Host](#) 値および [User](#) 値のさまざまな組み合わせがどのように着信の接続に適用するかを示します。

Host 値	User 値	正当な接続
' thomas.loc.gov '	'fred'	thomas.loc.gov から接続する fred
' thomas.loc.gov '	"	thomas.loc.gov から接続するすべてのユーザ
'%'	'fred'	任意のホストから接続するfred
'%'	"	任意のホストから接続するすべてのユーザ
'%.loc.gov'	'fred'	loc.gov ドメイン内の任意のホストから接続する fred
'x.y.%'	'fred'	x.y.net 、 x.y.com 、 x.y.edu などから接続する fred (実用的ではない)
' 144.155.166.177 '	'fred'	IP アドレス 144.155.166.177 のホストから接続する fred
' 144.155.166.% '	'fred'	144.155.166 クラス C サブネットの任意のホストから接続する fred
' 144.155.166.0/255.255.255.0 '	'fred'	1 つ上の例と同じ

クライアントのホスト名と着信するユーザ名が、[user](#) テーブルのエントリー一つ以上と、一致する可能性があります。前述の例で説明すると、たとえば、[thomas.loc.gov](#) からの [fred](#) の接続は、前述の一覧のいくつかのエントリと一致します。

複数のエントリが一致した場合、サーバは該当するものを選択するために、次のことを行ないます。

- サーバが [user](#) テーブルをメモリに読み込むときに、エントリのソートをする。
- クライアントが接続しようすると、サーバはソート順でエントリを照会する。
- サーバはクライアントのホスト名とユーザ名が一致する最初のエントリを使用する。

[user](#) テーブルが次の内容であると仮定して、これがどのように作用するかを説明します。

```
+-----+-----+
```

```

+-----+-----+
| Host   | User   | ...
+-----+-----+
| %      | root   | ...
| %      | jeffrey | ...
| localhost | root   | ...
| localhost |       | ...
+-----+-----+

```

サーバがテーブルをメモリに読み込むと、最も具体的な **Host** 値を最初にもってきます。ここでは、**Host** カラムの '%' は「任意のホスト」を意味し、具体性が最も低いものとなります。同じ **Host** 値のエントリ間で、最も具体的な **User** 値を最初にもってきます。ここでは、空白の **User** 値は「任意のユーザ」を意味し、具体性が最も低いものとなります。ソートした **user** テーブルは次のようになります。

```

+-----+-----+
| Host   | User   | ...
+-----+-----+
| localhost | root   | ...
| localhost |       | ...
| %      | jeffrey | ...
| %      | root   | ...
+-----+-----+

```

クライアント接続が試みられると、サーバはソートしたエントリで突き合わせ (マッチング) を行い、最初に一致したものを使用します。**localhost** からの **jeffrey** による接続では、2 つのエントリが一致します (空白ユーザ名のエントリが接続ホスト名とユーザ名の両方で一致)。**Host** と **User** のカラム値をみると、'**localhost**' と '' の値が一致します。そして、'%' と '**jeffrey**' の値が一致します。ここで、ソートしたときに、'**localhost**' が最も具体的な値であるため、これが最初に来ます。よってサーバは、表示の順序で選択します。

次に、別例を示します。**user** テーブルが次の内容である場合

```

+-----+-----+
| Host   | User   | ...
+-----+-----+
| %      | jeffrey | ...
| thomas.loc.gov |       | ...
+-----+-----+

```

ソート後のテーブルは次のようになります。

```

+-----+-----+
| Host   | User   | ...
+-----+-----+
| thomas.loc.gov |       | ...
| %      | jeffrey | ...
+-----+-----+

```

thomas.loc.gov からの **jeffrey** による接続は、最初のエントリを一致とし、**whitehouse.gov** からの **jeffrey** による接続は 2 番目のエントリとなります。

サーバが接続の突き合わせを行うとき、ユーザ名とは、明示的にそのユーザを名付けている (ユーザであると定義している) すべてのエントリが最初に来ると、考えることができますが、これは間違っています。上記の例でもわかるように、**thomas.loc.gov** からの **jeffrey** による接続は、'**jeffrey**' が **User** フィールド値であるエントリではなく、ユーザ名なしのエントリと最初に一致します。つまり、**jeffrey** で接続する、とユーザ名を指定したにもかかわらず、彼は匿名ユーザとして認証されています。

接続したが、権限が予期したものと違うという場合、別のユーザで認証している可能性があります。サーバがどのユーザで認識が行なわれたかを突き止めるには、**CURRENT_USER()** 関数を (情報関数 参照) 使用します。(その接続に実際に一致しているユーザとホストの組み合わせを確認できます。) そのときに、**user_name@host_name** という形の値を返します。これは、**User** と **Host** のカラムの値が、**user** テーブル行でどのように一致となっているのかを示します。たとえば、**jeffrey** で接続して、次のようなクエリを発行したとします。

```

mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| @localhost     |
+-----+

```


ここでの結果は、`user` テーブル行で一致したものが、空白の `User` であることを示します。つまり、サーバは `jeffrey` を匿名ユーザとして扱っています。

このような認証に関わる診断をする、別の方法として、`user` テーブルを出力して、それをマニュアル (手動) でソートし、最初の組み合わせをどのような経緯でしか調べます。

アクセス制御の段階 2：接続確認

接続を確立すると、サーバは段階 2 に移行します。その接続での要求ごとに、サーバはユーザに実行できる権限があるかどうかを、操作タイプに基づいてチェックします。ここで、権限テーブルの権限フィールドが関係してきます。権限は、`user`、`db`、`host`、`tables_priv`、`columns_priv`、`procs_priv` のテーブルで設定できます。各権限テーブルのフィールドのリストについては、[権限システムの機能](#) を参照してください。

`user` テーブルは、デフォルト (カレント) データベースに関係なく、ユーザに対してグローバル権限を設定します。たとえば、`user` テーブルに `DELETE` 権限があるユーザは、そのサーバ ホストのどのデータベースのレコードでも削除できます。つまり、`user` テーブルに対する権限はスーパーユーザ権限ということです。そのため、`user` テーブルで権限を設定になる対象は、サーバ管理者やデータベース管理者などのスーパーユーザだけにしておくのが賢明です。他のユーザについては、`user` テーブルの権限を 'N' に設定しておき、権限の付与は適切なレベルで行なうこととしてください。権限の付与には、データベース、テーブル、カラム、ルーチンなど、データベース依存で設定してください。

`db` テーブルおよび `host` テーブルでは、データベース依存の権限を設定します。スコープ フィールドには次の方法で値を指定します。

- ワイルドカード文字の '`%`' および '`'`' を、両テーブルの `Host` フィールドと `Db` フィールドで使用する。これには、`LIKE` 演算子で実行するマッチングと同様の意味があります。権限付与に、どちらかの文字を使用するときに、それをバックスラッシュでエスケープします。たとえば、'`'` 文字 (アンダーライン) をデータベース名の一部として使用するには、`GRANT` コマンドでそれを `★'★` として指定する。
- `db` テーブルの '`% Host`' 値は、「任意のホスト」を意味する。`db` テーブルの空白の `Host` 値は、「`host` テーブルを参照すること」を意味する。(このセクションでプロセスについて後述)
- `host` テーブルの '`%`' または空白の `Host` 値は、「任意のホスト」を意味する。
- 両テーブルにおいて `host` テーブルの '`%`' または空白の `Db` 値は、「任意のデータベース」を意味する。
- 両テーブルにおいて空白の `User` 値は、匿名ユーザを意味する。

`db` テーブルと `host` のテーブルは、サーバ起動時に読み取りとソートを行ないます (`user` テーブル読み込みと同時)。 `db` テーブルは `Host`、`Db`、`User` のそれぞれのスコープ フィールドでソートし、`host` テーブルは `Host` と `Db` のスコープ フィールドでソートします。 `user` テーブルでは、最も具体的な値が最初に、最も抽象的な値が最後の順でソートし、サーバによるエントリの突き合わせでは、最初に一致したエントリを使用します。

`tables_priv`、`columns_priv`、`procs_priv` のテーブルではそれぞれ、テーブル依存およびカラム依存の権限を設定します。スコープ フィールドには次の方法で値を指定します。

- ワイルドカード文字の '`%`' および '`'`' を両テーブルの `Host` フィールドで使用する。これらには、`LIKE` 演算子で実行するマッチングと同義です。
- 両テーブルにおいて '`%`' または空白の `Host` 値は、「任意のホスト」を意味する。
- 両テーブルの `Db`、`Table_name`、`Column_name` のそれぞれのフィールドで、ワイルドカードおよび空白は使用できない。

`tables_priv`、`columns_priv`、`procs_priv` などのテーブルは、`Host`、`Db`、`User` のフィールドでソートが行なわれます。これは `db` テーブルのソートと同様ですが、ワイルドカードの使用が `Host` フィールドだけの限定になるため、よりシンプルです。

サーバは、受け取る要求の妥当性をソートしたテーブルで確認します。`SHUTDOWN` または `RELOAD` などの管理側の権限が必要な要求では、サーバは `user` テーブル エントリをチェックします。これは、このテーブルが唯一、管理権限の指定を行なうテーブルであるためです。要求操作を許可していれば、アクセスを認め、そうでない場合は拒否

します。たとえば、`mysqladmin shutdown` コマンドを実行するときに、`user` テーブル エントリで `SHUTDOWN` 権限の設定がなければ、`db` テーブルや `host` テーブルをチェックすることなくアクセスを拒否します。これらのテーブルには `Shutdown_priv` カラムが存在しないため、チェックしません。

`INSERT`、`UPDATE` などデータベース関連の要求では、サーバは最初に、`user` テーブル エントリでユーザのグローバル (スーパーユーザ) 権限をチェックします。ここで、要求の操作が許可があれば、アクセスを認めます。`user` テーブルのグローバル権限が十分ではない場合には、サーバはユーザのデータベースに関する権限を `db` テーブルおよび `host` テーブルでチェックします。

1. サーバは、`db`
2. テーブルの `Host`、`Db`、`User` のそれぞれのフィールドをチェックする。`Host` フィールドと `User` フィールドでは、接続ユーザのホスト名と MySQL ユーザ名がチェックの対象になる。`Db` フィールドでは、ユーザがアクセスしようとしているデータベースをチェックする。`Host` および `User` に該当するエントリがない場合には、アクセスを拒否する。
3. `db` テーブル エントリが一致し、その `Host` フィールドが空白ではない場合、そのエントリがユーザのデータベース依存の権限を定義する。
4. 一致している `db` テーブル エントリの `Host` フィールドが空白の場合、これは `host` テーブルが、データベースにアクセスできるホストを指定することを意味する。このとき、`host` テーブルの `Host` フィールドと `Db` フィールドがチェックの対象になる。`host` テーブル内に一致するエントリがない場合は、アクセスを拒否する。一致するエントリがあれば、ユーザのデータベースに対する権限が、`db` および `host` テーブル エントリを、和集合ではなく、共通集合として計算する。つまり、両方のエントリの 'Y' が権限であることを指す。このように、`db` テーブル エントリで全般的な権限を設定し、`host` テーブル エントリでホストごとに権限を制限することができる。

サーバは、`db` および `host` テーブルエントリからデータベースに対する権限の特定が済むと、それらの権限を `user` テーブルのグローバル権限と足し合わせます。その結果が要求の操作を許可するものであれば、アクセスを認めます。そうでない場合、サーバは `tables_priv` と `columns_priv` のテーブルでユーザのテーブル権限とカラム権限をチェックし、これらのテーブルに対するユーザの権限を追加します。その結果に基づき、アクセスの許可または拒否を行ないます。ストアド ルーチンの操作には、サーバは `tables_priv` や `columns_priv` ではなく、`procs_priv` に重点を置きます。

ブール値で表すと、上記のユーザ権限算出法は次のような総括になります。

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

`user` 内のグローバル権限が十分ではない場合に、サーバがその十分ではないグローバル権限を、データベース、テーブル、およびカラム権限と足し合わせることは不可解です。しかし、これには、要求に複数の権限が必要な場合があるという事情があります。たとえば、`INSERT INTO ... SELECT` ステートメントを実行するには、`INSERT` 権限および `SELECT` 権限の両方が必要になります。この権限のうち 1 つの権限が `user` テーブル内のエントリにあり、もう 1 つの権限が `db` テーブル内のエントリにあるという場合には、要求を実行するために必要な権限をユーザを持っているにも関わらず、サーバがどちらか一方のテーブルだけでは判断できないため、両テーブルのエントリにある権限を組み合わせた結果で、許可/拒否の判断を行ないます。

`GRANT` または `REVOKE` などのステートメントは、`host` テーブルには影響しません。そのため、MySQL インストールではあまり、このステートメントを使用することはありません。直接に変更しなければならない場合、たとえば、セキュリティで保護したサーバリストの保守などを行なうときなど、稀に使用します。その例として、TcX (TcX DataKonsult AB) で、`host` テーブルには、ローカル ネットワークのすべてのマシンをリストに記述して、すべての権限をここで設定しています。

`host` テーブルを使用して、セキュリティ保護のない ホストを示すこともできます。セキュリティ保護のない公開エリア内に、コンピュータ `public.your.domain` があるとします。ここで、次のように、`host` テーブルのエントリを使用して、ネットワーク内のそのコンピュータ以外のホストすべてへのアクセスを許可することができます。

```
+-----+-----+
| Host      | Db | ...
+-----+-----+
| public.your.domain | % | ... (all privileges set to 'N')
| %.your.domain      | % | ... (all privileges set to 'Y')
```

ノート： 常に、権限テーブルをテストして ([SHOW GRANTS](#) などを使用)、アクセス権限を目的どおりに設定していることを確認してください。

権限の変更が反映するタイミング

[mysql](#) を起動すると、メモリにすべての権限テーブルが読み込まれます。この時点で、内部メモリのテーブルがアクセス制御の効力を施行します。

サーバが権限テーブルをリロードするときは、既存のクライアント接続の権限は次のように発効します。

- テーブルとカラムの権限での変更はクライアントが次に要求を行なうときに発効する。
- データベースの権限での変更は、次の [USE db_name](#) ステートメントを発行するときに発効する。

ノート： クライアント アプリケーションは、データベース名をキャッシュしていることがあるため、実際に別のデータベースを変更するか、[FLUSH PRIVILEGES](#) ステートメント を実行しないと、権限効力を発揮できない可能性がある。

- グローバル権限とパスワードでの変更は、次にクライアントが接続するときに発効する。

[GRANT](#)、[REVOKE](#)、or [SET PASSWORD](#) などのステートメントを使用して、間接的に権限テーブルを変更する場合は、サーバがこれらの変更を認識し、その変更があった直後に権限テーブルをメモリへリロードします。

[INSERT](#)、[UPDATE](#)、[DELETE](#) などのステートメントを使用して、直接に権限テーブルを変更する場合は、サーバを再起動するか、またはテーブルのリロードを行なうまでその権限チェックは施行しません。手動で権限テーブルをリロードするには、[FLUSH PRIVILEGES](#) ステートメントを発行するか、[mysqladmin flush-privileges](#) または [mysqladmin reload](#) コマンドを実行します。

権限テーブルを直接変更したけれど、リロードを忘れたという場合、単に、サーバを再起動するまで、その効果は発揮しません。変更したにも関わらず、その効果がないようなときは、再起動することをお勧めします。

Access denied エラーの原因

ここでは、MySQL サーバに接続しようとして Access denied エラーが発生した場合の対処法について説明します。

- まず、サーバが起動していることを確認します。起動していなければ接続はできません。たとえば、サーバに接続しようとして、次のようなメッセージが出るときには、サーバが立ち上がっていない可能性があります。

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

サーバは起動しているが、TCP/IP ポート、名前付きパイプ、Unix ソケット ファイルで接続しようとしている場合に、サーバが使用しているものが異なる場合があります。これを修正するには、クライアント プログラムを呼び出すときに、[--port](#) オプションを適切なポート番号を指すように指定します。または [--socket](#) で適切な名前付きパイプまたは Unix ソケット ファイルを指定します。ソケット ファイルの場所を検索するには、次のコマンドを使用します。

```
shell> netstat -ln | grep mysql
```

- サーバがアクセス制御に使用する権限テーブルは正確にセットアップする必要があります。Windows のバイナリ配布、または Linux の RPM 配布など、配布の種類によっては、インストールのプロセスで、権限テーブルがある [mysql](#) データベースを初期化します。これをしない種類の配布では、手動で権限テーブルを初期化する必要があります。これには、[mysql_install_db](#) スクリプトを実行します。詳細は [Unix のインストール後のプロシージャ](#) を参照してください。

権限テーブルを初期化する必要があるかどうかを調べるには、データ ディレクトリの [mysql](#) をチェックします。通常、データ ディレクトリは、[data](#) または [var](#) と名前で、MySQL をインストールしたディレクトリ下にあります。[mysql](#) データベース ディレクトリに [user.MYD](#) ファイルがあることを確認してください。これをしない場合

は、[mysql_install_db](#) スクリプトを実行してください。このスクリプトを実行してから、サーバを起動し、次のコマンドを実行して、イニシャル権限をテストします。

```
shell> mysql -u root test
```

これで、エラーなしでサーバに接続できます。

- 問題なくインストールできたら、サーバに接続して、ユーザとアクセス制限についてセットアップします。

```
shell> mysql -u root mysql
```

MySQL [root](#) には最初からパスワードがありません。そのため、サーバへの接続は問題なくできます。これには、セキュリティ面でのリスクが伴うため、[root](#) アカountのパスワードは、MySQL アカountをセットアップするときに、セットすることをお勧めします。初期パスワードの設定手順に関しては、[最初の MySQL アカountの確保](#) を参照してください。

MySQL

MySQL Network Monitoring and Advisory Service #####
#####<http://www.jp.mysql.com/products/enterprise/advisors.html> #####

- MySQL のバージョンを更新するときには、[mysql_upgrade](#) スクリプトを実行する必要があります。新バージョンの機能によっては、権限テーブルのストラクチャを変更することがあります。そのため、アップグレードした後は、常に、テーブルがカレント ストラクチャであるかどうかを確かめてください。この手順に関しては [mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照してください。
- クライアント プログラムで接続時に次のようなエラー メッセージが出る場合、そのクライアントが生成する形式よりも新しい形式でのパスワードをサーバが必要としている、ということを示します。

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

これに関する対処法に関しては、[MySQL 4.1 のパスワードハッシュ](#) および [Client does not support authentication protocol](#) を参照してください。

- [root](#) で接続をしようとする場合に、次のようなエラーが出るときには、['root'](#) の [User](#) フィールドに、[user](#) テーブルのエントリがないことを示します。そのため、[mysqlid](#) がクライアントのホスト名を識別できない状態です。

```
Access denied for user '@'unknown' to database mysql
```

この場合、[--skip-grant-tables](#) オプションでサーバを立ち上げ、[/etc/hosts](#) ファイル、または [\windows\hosts](#) で、ホストのエントリを付加します。

- クライアント プログラムは、環境変数またはオプション ファイルで指定する接続パラメータを使用します。そのため、コマンドラインからデフォルトの接続パラメータを指定していない場合には、適切ではないパラメータを送ることがあります。そのときは、環境変数、および該当するオプション ファイルを調べます。たとえば、オプションなしでクライアントを実行するときに、[Access denied](#) となる場合、オプション ファイルのどこかで古いパスワードを指定している可能性があります。

使用しているオプション ファイルをクライアント プログラムで抑制することができます。これには、[--no-defaults](#) を行使します。

```
shell> mysqladmin --no-defaults -u root version
```

クライアントが使用するこのオプション ファイルの一覧は、[オプションファイルの使用](#) を参照してください。環境変数の一覧は、[環境変数](#) を参照してください。

- 次のエラーが出る場合は、使用している [root](#) のパスワードが誤っていることを示します。

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

パスワードを指定していない場合に、このようなエラーが出るときには、オプション ファイルのどこかに誤ったパスワードがあることを示します。前述の項目で説明したように、[--no-defaults](#) で確かめてください。

パスワードの変更に関する情報は、[パスワードの設定](#) を参照してください。

root パスワードを忘れた場合、`--skip-grant-tables` で `mysqld` を再起動して、パスワードを変更します。詳細は [How to Reset the Root Password](#) を参照してください。

- パスワードの変更に、`SET PASSWORD`、`INSERT`、`UPDATE` を使用する場合は、`PASSWORD()` 関数でそのパスワードを暗号化します。これらのステートメントを使用するときに、`PASSWORD()` 関数を使用しないと、パスワードは機能しません。たとえば、次のようなステートメントでパスワードをセットしても、それを暗号化していない場合、そのユーザはそれ以降の接続ができません。

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

そのため、パスワードを次のようにセットします。

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

`GRANT` または `CREATE USER` ステートメント、あるいは `mysqladmin password` コマンドでパスワードを指定するときは、`PASSWORD()` 関数は不要です。これらの方法では、自動的に、`PASSWORD()` をパスワードの暗号化に適用します。詳細は [パスワードの設定](#) および [CREATE USER 構文](#) を参照してください。

- `localhost` はローカル ホスト名のシノニムです。ホストを明示的に指定しないときには、これがクライアント接続でのデフォルトのホストとなります。

このような問題を回避するには、`--host=127.0.0.1` オプションをサーバ ホストを明示的に指名します。その場合、TCP/IP で、ローカルの `mysqld` サーバに接続します。TCP/IP 接続には、`--host` オプションで実際のローカルホストのホスト名を使用できます。この場合、そのサーバの同じホストでクライアント プログラムを実行している場合でも、ホスト名をサーバホストの `user` テーブル エントリで指定する必要があります。

- `mysql -u user_name` でデータベースに接続しようとするときに、`Access denied` エラーが出る場合は、`user` テーブルに問題がある可能性があります。これをチェックするには、`mysql -u root mysql` の実行し、次の SQL ステートメントを発行します。

```
SELECT * FROM user;
```

結果には、コンピュータのホスト名と MySQL ユーザ名と一致する `Host` と `User` のカラムのエントリが出ます。

- `Access denied` というエラー メッセージは、ログインしようとしているユーザ名、接続元のホスト、およびパスワードを使用したかどうかを通知します。通常、`user` テーブルに、ホスト名とユーザ名に完全にマッチするエントリが 1 つあり、これが このエラーメッセージに出ます。たとえば、エラーメッセージに `using password: NO` と出る場合、パスワードなしでログインしようとしたことを示します。
- MySQL サーバを実行しているホストではないホストから接続しようとして以下のエラーが発生する場合、クライアント ホストと一致する `Host` 値では、レコードが `user` テーブルにないということを示します。

```
Host ... is not allowed to connect to this MySQL server
```

これは、接続時に使用するクライアント ホスト名とユーザ名を組み合わせたアカウントをセットアップすることで解決します。

接続元のコンピュータの IP アドレスまたはホスト名がわからない場合、`user` テーブルの `Host` カラムに `'%'` を使用します。そして、そのクライアント コンピュータから接続しようとするときに、`SELECT USER()` クエリを使用して、実際にどのように接続したか確認します。そのときに、`user` テーブル エントリの `'%'` をログにある実際のホスト名と置き換えます。この作業を行わない場合、どのホストからでもそのユーザ名での接続ができることになるので、セキュリティ上の問題になります。

Linux 上で、このエラーが発生する別の理由として、使用しているバイナリ MySQL バージョンが、`glibc` ライブラリが異なるバージョンを使用してコンパイルしている可能性があります。この場合、OS または `glibc` をアップグレードするか、または MySQL のソース配布をダウンロードして（自分で）コンパイルします。ソース RPM は通常、コンパイルおよびインストールが簡単です。そのため、これは大した問題ではありません。

- ホスト名で接続しようとしたにもかかわらず、エラーメッセージにホスト名がない、またはホスト名が IP アドレスである場合は、MySQL サーバで、クライアント ホストの IP アドレスを解決しようとしてエラーになったことを示します。

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@" (using password: YES)
```


これは、DNS に問題があることを示します。これを修正するには、`mysqladmin flush-hosts` を実行して、内部 DNS ホスト名キャッシュをリセットします。[MySQLの DNS の使用](#) を参照してください。

この解決策としては、次のことを検討します。

- DNS サーバの問題を割り出し、それを修正する。
- ホスト名ではなく、IP アドレスを MySQL の権限テーブルで指定する。
- `/etc/hosts` または `\windows\hosts` にクライアントのマシン名のエントリを入力する。
- `mysqld` を `--skip-name-resolve` オプションで起動する。
- `mysqld` を `--skip-host-cache` で起動する。
- Unix では、サーバとクライアントを同じマシンで実行している場合、`localhost` に接続する。`localhost` への Unix 接続は、TCP/IP ではなく Unix ソケット ファイルを使用する。
- Windows では、サーバとクライアントを同じマシンで実行して、サーバが名前付きパイプ接続をサポートしている場合、ホスト名 `.` (ピリオド) に接続する。`.` (ピリオド) への接続には、TCP/IP ではなく、名前付きパイプを使用する。
- `mysql -u root test` は機能するけれども、`your_hostname` がローカル ホストの実際のホスト名であるにも関わらず、`mysql -h your_hostname -u root test` で `Access denied` が出る場合、`user` テーブルにホストへの正確な名前がない可能性があります。この場合によくある問題として、`user` テーブル エントリの `Host` 値が適切ではないホスト名を指定して、システムでの名前解決ルーチンが完全に適切であるドメイン名を返すことがあります (またはこの逆の場合もあります)。たとえば、ホストを `'tcx'` とするエントリが `user` テーブルにあるにも関わらず、DNS で MySQL にホスト名は `'tcx.subnet.se'` であると伝えてしまうと、そのエントリが適用になりません。そのため、`user` テーブルに、ホストの IP アドレスを `Host` のカラム値として付加します。または、`user` テーブルに、`'tcx.%'` というように、ワイルドカード文字を `Host` 値に使用します。しかし、`'%'` をホスト名の終わりに付けるのは、セキュリティ面での安全を確保できないという理由から 推奨はしていません。
- `mysql -u user_name test` は機能するけれども、`mysql -u user_name other_db_name` が機能しない場合、そのユーザには、`other_db_name` のデータベース アクセスを許可していないことを意味します。
- `mysql -u user_name` はサーバ ホストで実行したときには機能するけれども、`mysql -h host_name -u user_name` がリモートのクライアント ホストから実行したときに機能しない場合、リモート ホストからはそのユーザ名でサーバにアクセスできないようになっていることを意味します。
- `Access denied` エラーの原因がわからない場合、`user` テーブルのエントリから、`'%'` または `'.'` などのワイルドカード文字が付いている `Host` 値をすべて削除します。ここで、`Host = '%'` および `User = 'some_user'` という新しいエントリを挿入して、これで同じマシンから接続するために `localhost` を指定することができるようになると考えることは誤りです。これは機能しません。その理由は、デフォルト権限で、`Host = 'localhost'` および `User = ''` というエントリを含むためです。このエントリには、`Host` カラムに `'localhost'` という `'%'` よりも明確な値があるため、`localhost` から接続するときに新しいエントリよりもデフォルトのエントリが優先になります。そのため、正確な手順としては、`Host = 'localhost'` および `User = 'some_user'` と改めて指定するか、または `Host = 'localhost'` および `User = ''` のエントリを削除します。このエントリを削除したら、`FLUSH PRIVILEGES` ステートメントを発行して、権限テーブルのリロードを必ず行なってください。
- 次のようなエラーが出る場合、`db` または `host` のいずれかのテーブルで問題がある可能性があります。

Access to database denied

`db` テーブルで選択するエントリに空白の `Host` カラムがある場合、`db` テーブルのエントリで適用するホストを指定するときに、対応するエントリが `host` テーブルに、1 つ以上あることを確認してください。

- MySQL サーバに接続はできるけれども、`SELECT ... INTO OUTFILE` または `LOAD DATA INFILE` を発行すると、`Access denied` が出る場合は、`user` テーブルのエントリで、`FILE` 権限がないことを意味します。
- `INSERT`、`UPDATE` または `DELETE` などのステートメントを使用して、直接に権限テーブルを変更するときに、それが効力を発揮していない場合は、`FLUSH PRIVILEGES` ステートメント、または `mysqladmin flush-privileges` コマンドを実行して、サーバへのその権限テーブルの再読み込みを行ないます。これをしないと、変更した内容が次にサーバを起動するまで発効しません。`UPDATE` コマンドで `root` パスワードを変更した場合は、その権限をフラッ

シュするまで、新たなパスワードを使用する必要はありません。その時点 (変更しただけ) では、サーバがまだパスワードを変更したことを認識していません。

- セッション中に権限が変更された可能性がある場合、MySQL の管理者がそれを変更した可能性があります。権限テーブルのリロードは、それ以降のクライアント接続に影響します。これは、既存の接続に対しても影響します。これに関しては、[権限の変更が反映するタイミング](#) を参照してください。
- Perl、PHP、Python、ODBC プログラムでアクセスに問題がある場合は、`mysql -u user_name db_name` または `mysql -u user_name -p your_pass db_name` で、サーバへの接続を試行します。mysql クライアントを使用して接続できる場合は、問題の根源は、アクセス権限ではなく、プログラムにあります。`-p` とパスワードの間には、空白はありませんが、`--password=your_pass` シンタックスを使用して、パスワードを指定することもできます。パスワードなしで `-p --password` オプションを使用すると、MySQL でパスワードの入力を指示します。
- テストするときは、`mysqld` サーバを `--skip-grant-tables` オプションで起動します。そのとき、MySQL 権限テーブルを変更でき、`mysqlaccess` スクリプトを使用して、変更内容に目的の効力があるかどうかをチェックできます。その内容が意図していたものである場合、`mysqladmin flush-privileges` を実行して、`mysqld` サーバに新たな権限テーブルを使用するよう指示できます。権限テーブルのリロードは、`--skip-grant-tables` オプションを上書きします。そのため、この上書きでサーバにリロードした権限テーブルをサーバのシステム終了や再起動をしなくても使い始めることができます。
- 前述の事柄を試しても、解決しない場合、`--debug=d,general,query` などの `mysqld` サーバをデバッグ オプションで立ち上げます。つまり、接続試行に関するホストとユーザの情報と、使用したコマンドも関する情報を出力します。[Creating Trace Files](#) を参照してください。
- MySQL 権限テーブルについて、ここで示したこと以外の問題がある場合は、メーリングリストで問題提起してください。そのときには、MySQL 権限テーブルのダンプも添付してください。テーブルのダンプは、`mysqldump mysql` コマンドでします。バグのレポートは、[質問またはバグの報告](#) の手順に従ってください。場合によっては、`mysqldump` を実行するためには、`--skip-grant-tables` オプションで、`mysqld` を再起動する必要があります。

MySQL 4.1 のパスワードハッシュ

MySQL ユーザ アカунトは、`mysql` データベースの `user` テーブルでリストしています。それぞれの MySQL アカунトにはパスワードを割り当てますが、`user` テーブルの `Password` カラムで保存になるのは、平文テキストのパスワードではなく、パスワードで計算したハッシュ値です。パスワード ハッシュ値は、`PASSWORD()` 関数で計算しています。

MySQL は、クライアントとサーバ間通信の 2 つのフェーズでパスワードを使用します。

- フェーズ 1：クライアントがサーバに接続しようとするとき、初期認証ステップとして、クライアントがパスワードを提示する。このパスワードは、クライアントが使用するアカウントの `user` テーブルで保存するハッシュ値と一致している必要がある。
- フェーズ 2：クライアントは接続した後、クライアントで、`user` テーブルにあるパスワード ハッシュを設定または変更することができる (適切な権限がある場合)。これには、クライアントが、`PASSWORD()` 関数を使用してパスワード ハッシュを生成するか、`GRANT` または `SET PASSWORD` ステートメントを使用する。

つまり、クライアントが最初の接続を試行するとき、サーバが認証にハッシュ値を使用します。接続したクライアントが `PASSWORD()` 関数を呼び出したり、`GRANT` または `SET PASSWORD` ステートメントを使用してパスワードの設定または変更を行うと、サーバがハッシュ値を生成します。

セキュリティ面での向上と、パスワード盗難の危険性に対応するパスワード ハッシュ メカニズムを MySQL 4.1 で更新しました。しかし、この新しいメカニズムはサーバとクライアント同士が相互に MySQL 4.1 以上での使用を必要とするため、それ以外のバージョンを使用している場合には、互換性に問題があります。つまり、MySQL 4.1 以上のクライアントは、パスワード ハッシュの新旧メカニズムの両方を理解するため、MySQL 4.1 より前のサーバにも接続できます。MySQL 4.1 より前のクライアントで MySQL 4.1 以上のサーバに接続する場合、問題が発生する可能性があります。たとえば、MySQL 3.23 の `mysql` のクライアントが 5.1 サーバに接続を試行すると、次のようなエラーメッセージ表示を伴う問題があります。

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

別の例として、MySQL 4.1 以上にアップグレードしてから、古いバージョンの PHP で `mysql` 拡張モジュールを使用するときにも、このような問題が発生します。([MySQL と PHP に対する共通問題](#) を参照のこと。)

次に、新旧のパスワード メカニズムでの違いと、MySQL 4.1 前の古いクライアントとの下位互換性の維持を必要とする場合のアップグレードについて説明します。[Client does not support authentication protocol](#) で、追加情報の参照もしてください。ここでの内容は、PHP で MySQL データベースを 4.0 以前から 4.1 以上にする場合に特に重要です。

注意：ここでの内容は、MySQL 4.1 を境とする動作について説明しています。ここで、4.1 の動作として説明している内容は、実際には 4.1.1 で導入しています。MySQL 4.1.0 は「特異的なリリース」が行なわれたため、4.1.1 以降に実装したメカニズムとは若干異なります。4.1.0 以降のバージョン間での違いに関する詳細は、MySQL 5.0 Reference Manual を参照してください。

MySQL 4.1 より前のバージョンでは、`PASSWORD()` 関数で計算するパスワード ハッシュは 16 バイト長です。次にその例を示します。

```
mysql> SELECT PASSWORD('mypass');
+-----+
| PASSWORD('mypass') |
+-----+
| 6f8c114b58f2ce9e |
+-----+
```

MySQL 4.1 より前のバージョンでは、`user` テーブルの `Password` カラム (ハッシュを保存する場所) も 16 バイト長です。

MySQL 4.1 からは、`PASSWORD()` 関数で、それまでより長い、41 バイトのハッシュ値を生成します。

```
mysql> SELECT PASSWORD('mypass');
+-----+
| PASSWORD('mypass') |
+-----+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-----+
```

それに伴い、このバイト幅に合わせて、`user` テーブルの `Password` カラムも 41 バイト幅です。

- MySQL 5.1 の新規インストールで、`Password` カラムは自動的に 41 バイト幅になる。
- MySQL 4.1 (4.1.1 を含む 4.1 シリーズ) から MySQL 5.1 へのアップグレードでは、パスワード ハッシュ メカニズムが同一であるため、このアップグレードでは、ここに挙げるパスワード ハッシュに関する問題は発生しません。これ以外のアップグレード、たとえば、MySQL 4.1 よりも古いバージョンから 5.1 にする場合は、まず、MySQL 4.1 へアップグレードしてから、5.1 へのアップグレード (インストール) を行ないます。

拡張後の `Password` カラムには、新旧どちらの形式でもパスワード ハッシュを保存できます。パスワード ハッシュ値の形式を次の 2 つの基準で判断します。

- バイト幅 (文字列の長さが 16 バイトまたは 41 バイトのどちらか)。
- パスワード ハッシュの開始文字。新しい形式のパスワードは `*` 文字で始まる。旧形式のパスワードはこれ以外の文字で始まる。

パスワードハッシュ形式は、長い方が暗号化に優れ、クライアント認証においても、旧形式の短いハッシュよりもセキュリティ面での安全性が高まります。

パスワードハッシュのバイト幅の違いは、サーバでのパスワード認証方法と、接続クライアントのパスワード変更に対するサーバでのパスワード ハッシュの生成方法に影響します。

サーバでのパスワード認証方法では、`Password` カラムの幅で、次のように影響します。

- カラム幅が短いと、ハッシュ認証が短くなる。(制限への影響)
- カラムが長いと、長短両方のハッシュを保てるため、サーバ認証では、次のどちらかの方法を使用する。
 - 4.1 より前でのクライアント接続は可能。しかし、旧ハッシュ メカニズムで理解するため、短いハッシュのアカウントだけを認証する。
 - 4.1 以降のクライアントは、長短どちらのハッシュのアカウントも認証する。

ハッシュのアカウントが短くても、実際は認証プロセスが 4.1 以降のクライアントであれば、古いクライアントを使用しているよりは、セキュリティ面での安全を確保しています。認証のセキュリティは以下の順で高くなります。

- 短いパスワード ハッシュでアカウントに対して、4.1 より前のバージョンのクライアント認証
- 短いパスワード ハッシュでアカウントに対して、4.1 以降のクライアント認証
- 長いパスワード ハッシュのアカウントに対して、4.1 以降のクライアント認証

サーバが接続クライアントに対してパスワード ハッシュを生成する方法には、`Password` カラムの幅と `--old-passwords` オプションが影響を与えます。つまり、4.1 以降のサーバでは、`Password` カラムがハッシュ値(長さ)に対応する、そして、`--old-passwords` オプションを指定していない、という条件を満たすと、長いハッシュを生成します。ここでは、次のことに注意します。

- `Password` カラムが 41 バイト のハッシュを保存できる長さであること。4.1 へのアップグレード後に、`mysql_fix_privilege_tables` スクリプトを実行して、`Password` カラムが長くなったことを更新して知らせる。ここで、カラムの更新を行わず、4.1 より前の長さの 16 バイトのままの状態にしておくと、クライアントが `PASSWORD()`、`GRANT`、`SET PASSWORD` を使用してパスワードの変更操作を行うときに、サーバはまだハッシュが長くなったことを知らされていないため、長いハッシュは収まらないと認識し、短いハッシュを生成する。
- `Password` カラムの長さ (幅) が十分であれば、長短どちらのパスワードハッシュも保存できる。この場合、サーバを `--old-passwords` オプションで起動していないときは、`PASSWORD()`、`GRANT`、`SET PASSWORD` で、長いハッシュを生成する。このオプションを指定すると、強制的に短いパスワードハッシュを生成する。

`--old-passwords` オプションを使用する目的には、サーバが長いパスワード ハッシュを生成する環境において、4.1 より前のクライアントと下位互換性を保てるようにすることがあります。このオプションは認証に影響するのではなく、4.1 以降のクライアント、つまり新しいメカニズムのクライアントでは長いパスワード ハッシュのアカウントを使用できるようになっているため、それがパスワードの変更操作の結果として、サーバの `user` テーブルに、長いパスワード ハッシュを保存しないようにします。長いパスワード ハッシュを保存してしまうと、4.1 より前のクライアントが、このアカウントを使用できなくなります。たとえば、`--old-passwords` オプションを指定しないで接続を行なうと、次に示すようなシナリオが想定できます。

- 古いバージョンのクライアントが、短いパスワード ハッシュのアカウントで接続する。
- このクライアントがアカウントのパスワードを変更する。`--old-passwords` オプションをかけていなければ、アカウントに長いパスワード ハッシュの生成が可能になる。
- そして、古いバージョンのクライアントがアカウントに長いパスワード ハッシュをセットしていた場合、長いパスワードは新しいメカニズムでの認証を行なうため、このクライアントが改めて、その長いパスワードをセットしたアカウントからは接続ができなくなります。つまり、このアカウントに長いパスワード ハッシュがあることを、テーブルに読み込ませてしまうと、そのクライアントが古いバージョンであるために、長いハッシュを認識することができません。(4.1 以降の新しいバージョンのクライアントは長いパスワードを認識します。)

このシナリオでは、4.1 より前のバージョンのクライアントをサポートする必要がある場合には、4.1 以降のサーバを稼動するのは危険であり、これを回避するには、`--old-passwords` オプションが必要であることを示しています。`--old-passwords` オプションでサーバを立ち上げていれば、パスワード変更の操作を行なっても、長いパスワード ハッシュを生成することはありません。これにより、古いバージョンのクライアントでアカウントにアクセスできなくなるということはありません。つまり、クライアントの不注意で、パスワード変更が長いパスワード ハッシュの生成を起因して、アクセスできなくなるということはありません。

`--old-passwords` オプションの短所は、どのようなパスワードを作成しても、短いハッシュになることです。これは 4.1 を使用しているクライアントにも該当します。そのため、長いパスワード ハッシュによるセキュリティのメリットを活用できません。4.1 を使用しているクライアントに、長いハッシュでアカウントを作成する必要がある場合などは、`--old-passwords` オプションを使用しないで、サーバが実行中のときに、その操作を行なう必要があります。

4.1 以降のサーバが実行中のときに考えられるケースとしては、次のようなシナリオがあります。

シナリオ 1: ユーザ テーブルの `Password` カラム値の幅が短い場合

- `Password` カラムには、短いハッシュだけが保存の対象になる。
- クライアント認証に、サーバが短いハッシュだけをし使用する。
- 接続クライアントでは、`PASSWORD()`、`GRANT`、`SET PASSWORD` などを使用するパスワード ハッシュの生成操作では、完全に短いハッシュを使用する。アカウントのパスワードの変更で、長くで設定してもパスワード ハッシュは短くなる。

- `--old-passwords` を使用するが、これには何の効果もない。その理由は、`Password` カラムが短いハッシュだけを受け入れるため、サーバは短いパスワード ハッシュだけを生成しているからである。

シナリオ 2: `Password` カラムに長いハッシュを保存できるが、サーバを `--old-passwords` オプションで起動しない場合

- `Password` カラムには、長短、両方のハッシュが保存の対象になる。
- 4.1 以降のクライアントは、長短どちらのハッシュのアカウントも認証する。
- 4.1より前のクライアントでは、短いハッシュのアカウントだけを認証する。
- 接続クライアントでは、`PASSWORD()`、`GRANT`、`SET PASSWORD` などを使用するパスワード ハッシュの生成操作では、完全に長いハッシュを使用する。アカウントのパスワードの変更で、長いパスワード ハッシュのアカウントになる。

前述したように、このシナリオでの問題点は、4.1 より前のクライアントが短いパスワード ハッシュのアカウントでアクセスができなくなることです。`GRANT`、`PASSWORD()`、or `SET PASSWORD` を使用して該当アカウントのパスワードを変更することは、そのアカウントに長いパスワード ハッシュを与えることになります。この時点から、4.1 より前のクライアントを、4.1 にアップグレードしなければ認証ができません。

この問題の対処するには、パスワードを特別の方法で変更します。たとえば、通常、アカウントのパスワード変更には、`SET PASSWORD` を次のように使用しています。

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

パスワードを変更するけれども短いハッシュで作成する場合には、`OLD_PASSWORD()` 関数を使用します。

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` 関数は、明らかに短いハッシュを生成する必要があるような場合に使用します。

シナリオ 3: `Password` カラムに長いハッシュを保存できるが、サーバを `--old-passwords` オプションで 4.1 以降のサーバ起動する場合

- `Password` カラムには、長短、両方のハッシュが保存の対象になる。
- 4.1 以降のクライアントは、長短どちらのハッシュのアカウントも認証する。(注意：長いハッシュは、`--old-passwords` オプションを使用しないでサーバを起動したときにだけ作成できる。)
- 4.1より前のクライアントでは、短いハッシュのアカウントだけを認証する。
- 接続クライアントでは、`PASSWORD()`、`GRANT`、`SET PASSWORD` などを使用するパスワード ハッシュの生成操作では、完全に短いハッシュを使用する。アカウントのパスワードの変更で、長くで設定してもパスワード ハッシュは短くなる。

このシナリオでは、`--old-passwords` オプションが長いハッシュの生成を抑制するため、長いパスワード ハッシュのアカウントを作成することはできません。さらに、`--old-passwords` オプションを使用する前に長いハッシュでアカウントを作成する場合、`--old-passwords` を行使中にアカウントのパスワードを変更すると、短いパスワードのアカウントになるため、長いハッシュのセキュリティ面でのメリットを活用できなくなります。

次に、これらのシナリオの問題を概括します。

シナリオ 1 では、セキュリティ上の安全をより確保できる長いハッシュの長所を活用できません。

シナリオ 2 では、`OLD_PASSWORD()` を明示的に使用しないで、パスワードを変更する場合、4.1 より前のバージョンのクライアントが短いハッシュのアカウントではアクセスできなくなる。

シナリオ 3 では、`--old-passwords` オプションが、短いハッシュのアカウントのアクセスを抑制していますが、パスワード変更操作が長いハッシュを短いハッシュに戻します (操作で長いハッシュに変更してもその操作が取り消しになる)。この短くなったハッシュは、`--old-passwords` を施行している間は、元 (長いハッシュ) に戻せません。

アプリケーション プログラムに対するパスワードハッシュ変更の影響

アプリケーション側でも、`PASSWORD()` を使用してパスワードを生成している場合、MySQL 4.1 にアップグレードすると、アプリケーションとの互換性に問題が生じます。本来、`PASSWORD()` は MySQL ユーザのパスワード管理

専用であるため、アプリケーションではこの関数を実行すべきではありません。しかし現状では、いくつかのアプリケーション側でも、それぞれの目的で `PASSWORD()` を使用しています。

MySQL バージョンを 4.1 以降にアップグレードして、長いパスワード ハッシュが生成できる状態でサーバを実行すると、アプリケーションで `PASSWORD()` を使用している場合は、アプリケーションが壊れます。推奨の対処法として、アプリケーションを修正して `SHA1()` または `MD5()` など、別の関数を使用してハッシュ値を生成するように設定します。この別の関数を使用することが不可能な場合は、`OLD_PASSWORD()` 関数を使用しますが、これは、旧形式の短いハッシュを生成するための関数です。(注意: `OLD_PASSWORD()` は将来サポートしなくなる可能性があります)。

短いハッシュを生成するような状況下でサーバを実行している場合には、`OLD_PASSWORD()` を利用できますが、これは、`PASSWORD()` と同等です。

PHP ユーザは、使用している MySQL データベースをバージョン 4.0 以前から、バージョン 4.1 以降にするときには、[MySQL PHP API](#) を一読してください。

MySQL ユーザ アカウント管理

このセクションでは、MySQL サーバのクライアントのアカウントのセットアップ方法を説明します。次の項目に関して説明します。

- MySQL で使用するアカウント名とパスワードの定義と OS で使用している名前とパスワードとの違い
- 新規アカウントの追加と既存アカウントの削除
- パスワードの変更
- パスワードの安全使用に関するガイドライン
- SSL 接続の安全確保

MySQL ユーザ名とパスワード

MySQL アカウントをサーバ接続が可能なユーザからのユーザ名とクライアント ホスト、またはホスト、と定義します。アカウントにはパスワードがあります。MySQL で使用するユーザ名とパスワードは、オペレーティング システムで使用するものとは特徴的な違いがあります。

- MySQL で認証目的に使用するユーザ名は、Windows や Unix などで使用しているユーザ名 (ログイン名) とは関係がありません。Unix の場合は、MySQL クライアントがデフォルトで、現在使用している Unix のユーザ名を MySQL のユーザ名としてログインを行なうことがあります。これは利便性との兼ね合わせです。クライアント プログラムで `-u` または `--user` のオプションでユーザ名を指定することができるため、デフォルト値は簡単に書き換えることができます。しかし、これは、誰でも適当なユーザ名を使用してサーバ接続を試行できるという意味でもあるため、すべての MySQL アカウントにパスワードをつけることで、データベースの安全を確保します。つまり、パスワードなしのアカウントのユーザ名を指定できる者であれば、サーバへの接続に成功します。
- MySQL ユーザ名には、最大 16 文字まで使用できます。これは、MySQL のサーバとクライアントでハードコード (決め打ち) しています。[mysql](#) データベースのテーブル定義を変更するなどして、この文字制限を回避しないでください。

注意: [mysql](#) データベースのテーブルは、MySQL AB による指示がない限り、絶対に改造しないでください。([mysql_upgrade](#) — MySQL アップグレードのテーブル チェック を参照のこと。) MySQL のシステム テーブルの再定義を試行すると、予期しない結果を招く恐れがあります。

オペレーティング システムのユーザ名は、MySQL ユーザ名とは完全に無関係です。最大文字数も異なります。たとえば、Unix ユーザ名の最大文字数は、8 文字です。

- MySQL パスワードは、オペレーティング システムのログイン パスワードとは別物です。Windows や Unix のマシンにログインするパスワード、およびそのマシンにある MySQL サーバへのアクセスに使用するパスワードとは一切無関係です。
- MySQL では、Unix ログインプロセスのアルゴリズムとは別のアルゴリズムで、パスワードを暗号化する。MySQL のパスワード暗号化には、`PASSWORD()` SQL 関数、Unix のパスワード暗号化には、`ENCRYPT()` SQL 関数で

す。 [PASSWORD\(\)](#) 関数と [ENCRYPT\(\)](#) 関数の詳細については、[暗号化関数と圧縮関数](#) を参照してください。注意：バージョン 4.1 以降の MySQL は、接続時のパスワード保護に、より高度な認証方式を採用しています。これは、TCP/IP パケットの盗聴や、[mysql](#) データベースのキャプチャが行なわれた場合でも、セキュリティを保持できるようになっています。初期バージョンでは、パスワードは [user](#) テーブルに暗号化して保存していますが、暗号化パスワード値に関する知識は MySQL サーバ接続に通用します。

MySQL をインストールするとき用に、権限テーブルには初期設定のアカウント セットを投入しています。そのアカウントには名前とアクセス権限があります。パスワードの割り当て方法に関しては、[最初の MySQL アカウントの確保](#) を参照してください。その後、セットアップを行い、[GRANT](#) や [REVOKE](#) などのステートメントで MySQL アカウントの変更や削除などを行ないます。詳細は、[アカウント管理ステートメント](#) を参照してください。

コマンドライン クライアントで MySQL サーバに接続するときは、ユーザ名とパスワードを使用するアカウントに指定します。

```
shell> mysql --user=monty --password=guess db_name
```

短文のオプションを使用すると、コマンドは次のようになります。

```
shell> mysql -u monty -pguess db_name
```

[-p](#) オプションと後続のパスワード値の間には、空白はありません。[MySQL サーバへの接続](#) を参照してください。

前述のコマンドには、コマンドラインでパスワードを入れています。これは、セキュリティ面でのリスクを伴います。詳細は [パスワードのセキュリティ](#) を参照してください。この問題を解決するには、パスワード値を入れずに、[--password](#) または [-p](#) オプションを指定します。

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

オプションにパスワード値がない場合、クライアントプログラムがプロンプトを出力し、パスワードの入力を待機します。この例示では、パスワード オプションに空白 (スペース) を入れて区切っているため、[db_name](#) はパスワードとは解釈しません。

オペレーティングシステムによっては、MySQL がパスワードを呼び出すときに使用するライブラリ ルーチンで、入力するパスワードは最大 8 文字であることを自動的に制限します。これは、システム ライブラリでの問題で、MySQL とは関係ありません。MySQL では内部的にパスワードの文字数を制限することはありません。この問題を回避するには、MySQL パスワードを変更しますが、そのときは、8 文字よりも少ない文字数にするか、またはオプション ファイルにパスワードを入力します。

MySQL への新規ユーザの追加

MySQL アカウント作成方法は、2 通りあります。

- [CREATE USER](#) または [GRANT](#) などのステートメントをアカウント作成に使用する。
- [INSERT](#)、[UPDATE](#)、[DELETE](#) などのステートメントで直接、MySQL の権限テーブルを操作する。

推奨方法は、[CREATE USER](#) や [GRANT](#) などのアカウント作成のステートメントを使用する方法です。これは、的確であり、エラーを防ぎます。詳細は [CREATE USER 構文](#) および [GRANT 構文](#) を参照してください。

アカウント作成の別の方法としては、MySQL アカウント管理の機能を提供している、[phpMyAdmin](#) などのサードパーティ プログラムを使用することもできます。

次に、新規ユーザのセットアップする [mysql](#) クライアント プログラムの使用方法を例示します。この例示では、[最初の MySQL アカウントの確保](#) で説明するように、デフォルト設定の権限でセットアップしています。これは、変更するには、MySQL [root](#) ユーザとして MySQL サーバに接続する必要があることを示します。この [root](#) アカウントには、[mysql](#) データベースの [INSERT](#) 権限と [RELOAD](#) 管理権限も必要です。

まず、[mysql](#) プログラムを使用してサーバに MySQL [root](#) ユーザとして接続します。

```
shell> mysql --user=root mysql
```

[root](#) アカウントにパスワードを割り当てた場合には、[mysql](#) コマンドで [--password](#) または [-p](#) のいずれかのオプションも使用します。

root でサーバに接続した後に、新規アカウントを追加します。次のステートメントでは、**GRANT** で新規アカウントを 4 つ追加しています。

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
-> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
-> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> GRANT USAGE ON *.* TO 'dummy'@'localhost';
```

GRANT ステートメントで追加したアカウントは、次のような属性を持ちます。

- **monty** というユーザ名と **some_pass** というパスワードのアカウントが 2 つ存在します。どちらもフル権限を持つスーパーユーザのアカウントです。'monty'@'localhost' というアカウントは、ローカル ホストから接続するときだけに使用できます。一方の 'monty'@'%' というアカウントは、どのホストからでも接続できます。注意： **monty** というアカウントは両方とも、**monty** としてどこからでも接続できる必要があります。この **localhost** でアカウントを持っていない場合、**monty** でローカル ホストから接続したときに、**mysql_install_db** で作成している **localhost** のエントリで、匿名ユーザのアカウントとして優先になります。つまり、**monty** が匿名ユーザとして扱われます。この理由は、'monty'@'%' よりも、匿名ユーザの方が具体的な **Host** カラム値にあるため、匿名の方が、**user** テーブルのソート順で先にくきます。(**user** テーブルのソートに関しては、[アクセス制御の段階 1：接続確認](#) を参照してください。)
- **admin** というユーザ名でパスワードがないアカウントがあります。このアカウントは、ローカル ホストから接続するときだけに使用できます。そして、**RELOAD** と **PROCESS** の管理権限があります。この権限は、**admin** ユーザが、to execute the **mysqladmin reload**、**mysqladmin refresh**、**mysqladmin flush-xxx**、**mysqladmin processlist** などのコマンドを実行できます。データベースへのアクセスに関する権限はありません。必要に応じて、追加の **GRANT** ステートメントを発行して、そのような権限を後から追加することができます。
- 4 番目には、**dummy** というユーザ名でパスワードなしのアカウントがあります。このアカウントは、ローカル ホストから接続するときだけに使用できます。権限は一切ありません。**GRANT** ステートメントで **USAGE** 権限を使用すると、全く権限のないアカウントを作成できます。これには、すべてのグローバル権限を '**N**' でセッティングする効果があります。(ここでは、このアカウントには後から特定の権限を付与するものとしています。)

GRANT の択一的な方法として、**INSERT** ステートメントを発行して、**FLUSH PRIVILEGES** でサーバに権限テーブルをリロードさせるという方法で、直接に、前述と同じ内容のアカウントを作成することができます。

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
-> VALUES('localhost','monty',PASSWORD('some_pass'),
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
-> VALUES('%','monty',PASSWORD('some_pass'),
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
-> Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User>Password)
-> VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

INSERT でアカウントを作成するときに、**FLUSH PRIVILEGES** を使用する理由には、その権限テーブルの再読み込みをサーバに行なわせるという目的があります。これをしないと、サーバを再起動するまで、変更内容が反映しません。**GRANT** でアカウントを作成するときは、**FLUSH PRIVILEGES** は不要です。

INSERT を伴う **PASSWORD()** 関数を使用する理由には、パスワードの暗号化という目的があります。**GRANT** ステートメントはパスワードの暗号化を自動的に行なうため、**PASSWORD()** は不要になります。

'Y' はアカウントに対する権限を有効にします。MySQL のバージョンによっては、**INSERT** ステートメントの最初の 2 つのエントリで、'Y' の数が異なる場合があります。**admin** アカウントでは、**SET** を使用した読み込みやすい拡張 **INSERT** シンタックスを採用する場合があります。

dummy アカウントの **INSERT** ステートメントには、**user** テーブルのエントリの **Host**、**User**、**Password** のコラムだけに対して、値を割り当てています。どの権限も具体的にはセットしていません。そのため、MySQL がデフォルト値として、'**N**' を割り当てています。これは、**GRANT USAGE** で行なうことと同じものです。

ノート：スーパーユーザのアカウントをセットアップするには、'Y' にセットした権限コラムで、**user** テーブル エントリを作成します。**user** テーブルの権限はグローバルであるため、別の権限テーブル エントリは不要です。

次の例示は、3つのアカウントと作成し、それに特定のデータベースにアクセスできるようにします。それぞれに、**custom** というユーザ名と、**obscure** というパスワードがあります。

GRANT でこれらのアカウントを作成するには、次にステートメントを使用します。

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON bankaccount.*
-> TO 'custom'@'localhost'
-> IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON expenses.*
-> TO 'custom'@'whitehouse.gov'
-> IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON customer.*
-> TO 'custom'@'server.domain'
-> IDENTIFIED BY 'obscure';
```

この3つのアカウントを次のように使用します。

- 最初のアカウントは、ローカル ホストからのみ、**bankaccount** データベースにアクセスできます。
- 2番目のアカウントは、**whitehouse.gov** というホストからのみ、**expenses** データベースにアクセスできます。
- 3番目のアカウントは、**server.domain** というホストからのみ、**customer** データベースにアクセスできます。

GRANT を使用しないで、**custom** アカウントをセットアップするには、**INSERT** ステートメントを次のように使用して、権限テーブルを直接変更します。

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('whitehouse.gov','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('localhost','bankaccount','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('whitehouse.gov','expenses','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('server.domain','customer','custom',
-> 'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

最初から3つ目までの **INSERT** ステートメントは、**user** テーブル エントリを追加します。このエントリは、**custom** というユーザが、様々なホストから指定のパスワードで接続できますが、グローバル権限の記述はありません。(すべての権限はデフォルトの 'N' でセットになります。) 次の **INSERT** ステートメントの3つは、**db** テーブル エントリを追加します。このエントリは、**custom** に権限を与え、適切なホストからだけ **bankaccount**、**expenses** そして **customer** というデータベースのデータベースにアクセスできます。ここでも、権限テーブルを直接変更するときは、変更内容を反映させるために、**FLUSH PRIVILEGES** でサーバにリロードするよう指示してください。

特定のユーザで、**mydomain.com** など特定のドメインにすべてのマシンからアクセスできるようにする場合は、**GRANT** ステートメントを使用します。このステートメントは、アカウント名のホストの部分で '**%**' ワイルドカード文字を使用します。

```
mysql> GRANT ...
-> ON *.*
-> TO 'myname'@'%mydomain.com'
-> IDENTIFIED BY 'mypass';
```

直接、権限テーブルを変更して、同じことを行なうには、次のようにします。

```
mysql> INSERT INTO user (Host,User,Password,...)
-> VALUES('%mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

MySQL ユーザの削除

アカウントを削除するには、**DROP USER** ステートメントを使用します。これは、**DROP USER** 構文 を参照してください。

ユーザ リソースの制限

MySQL サーバ リソースの使用を制限することの一つの方法として、スタートアップ変数の **max_user_connections** をゼロ以外の値に設定することがあります。しかし、この方法は完全にグローバルに適用するため、個別アカウントの管理はできません。これに加えて、この方法は、単一アカウントによる同時接続の数を制限するものであり、これはクライアントが制限できることではありません。このようなタイプの制御は、インターネット サービス プロバイダ業界などでの MySQL 管理者に高い関心をよせる点です。

MySQL 5.1 では、個別ユーザレベルで 3 つのサーバ リソースを制限できます。

- 時間単位の全クエリ数：1 ユーザが実行できるクエリ
- 時間単位の全更新数：テーブルまたはデータベースを変更するクエリ
- 時間単位の接続数：1 時間で新しく開ける接続

クライアントが発行できるステートメントはクエリ制限に対してカウントします。データベースまたはテーブルを変更するステートメントは更新制限に対してカウントします。

アカウント ベースでサーバへの同時接続の数を制限することも可能です。

ここでのアカウントは、**user** テーブル エントリの 1 レコード (ユーザ) です。このエントリは **User** と **Host** のカラム値で識別します。

ここでの機能を使用するための前提条件として、**mysql** データベースの **user** テーブルには、リソース関連のコラム (フィールド) が必要です。リソース制限は、**max_questions**、**max_updates**、**max_connections**、**max_user_connections** のカラムで保存します。**user** テーブルにこれらのカラムがない場合は、**mysql_upgrade** — MySQL アップグレードのテーブル チェック を参照して、アップグレードしてください。

GRANT ステートメントでリソース制限を設定するには、**WITH** 節を使用します。この節は制限するリソースと制限値の時間単位の回数を指定します。たとえば、**customer** データベースにアクセスできる新規アカウントを作成するときに、制限をつける場合は、次のようなステートメントを発行します。

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
-> IDENTIFIED BY 'frank'
-> WITH MAX_QUERIES_PER_HOUR 20
-> MAX_UPDATES_PER_HOUR 10
-> MAX_CONNECTIONS_PER_HOUR 5
-> MAX_USER_CONNECTIONS 2;
```

制限する種類をすべて **WITH** 節で指定する必要はありませんが、順番がバラバラになります。時間単位の制限値は時間単位を表す整数にします。**GRANT** ステートメントに **WITH** 節がない場合は、この制限はデフォルト値、ゼロでの設定になります。つまり、制限がないことになります。**MAX_USER_CONNECTIONS** の整数は、アカウントが一度にできる同時接続の最大回数を表します。この制限を、デフォルト (ゼロ) にした場合、**max_user_connections** システム変数でアカウント同時接続回数を決定します。

既存アカウントの制限をセットまたは変更するには、グローバル レベル (**ON *.***) で **GRANT USAGE** ステートメントを使用します。**francis** に対するクエリ制限を 100 に変更するステートメントは、次のようになります。

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
-> WITH MAX_QUERIES_PER_HOUR 100;
```

このステートメントでは、アカウントにある既存の権限には影響しません。制限値の指定を行なうだけです。

既存の制限を削除するには、その値をゼロにセットします。たとえば、**francis** が時間当たり接続できる回数の制限を削除するには、次のステートメントを使用します。

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
-> WITH MAX_CONNECTIONS_PER_HOUR 0;
```

リソース使用のカウン트는、アカウントの対象制限でリソースの値がゼロではないときに行なわれます。

サーバを実行すると、それぞれのアカウントのリソース使用回数のカウン트가始まります。前の接続時間内で接続の制限値に到達すると、それ以後の接続はその時間が過ぎるまで接続できません。同様に、そのアカウントでクエリまたは更新の制限回数に到達すると、それ以後のクエリや更新はその時間が過ぎるまでできません。制限値に到達すると、それぞれでエラーが出ます。

リソースのカウン트는、アカウント単位に行います。クライアント単にはありません。たとえば、アカウントのクエリ制限が 50 である場合、サーバへの接続を同時に 2 つのクライアントから行なっても、制限が 100 になるという具合に、制限値が上がることはありません。この 2 つの接続からのクエリは一緒にカウントします。

クエリ キャッシュからのクエリ結果は、`MAX_QUERIES_PER_HOUR` のカウンにはなりません。

現行の時間単位のリソース利用のカウン트는、すべてのアカウントに対して、または別々にグローバルでリセットできます。

- すべてのアカウントに対して現行のカウンをゼロにリセットするには、`FLUSH USER_RESOURCES` ステートメントを発行します。また、このリセット操作は `FLUSH PRIVILEGES` ステートメントや `mysqladmin reload` コマンドを使用して、権限テーブルのリロードを行なうことでもできます。
- アカウント単位でカウンを別々にゼロにリセットするには、制限値を再セットします。これを行なうには、前述の方法のように、`GRANT USAGE` を使用して、その時点でアカウントにある値と同等の別の値を指定します。

カウンのリセットを行なっても、`MAX_USER_CONNECTIONS` 制限には影響しません。

すべてのカウンはサーバ起動時にゼロで始まりますが、再起動した場合に、そのカウンが持ち越しになることはありません。

パスワードの設定

パスワードの設定には、コマンドラインで `mysqladmin` を使用します。

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

このコマンドでリセットするパスワードのアカウントは、`user` テーブル エントリにあるアカウントのことです。これは、`User` カラムの `user_name` と、`Host` カラムにある接続クライアント ホストとに一致します。

`SET PASSWORD` ステートメントを発行して、アカウントにパスワードを設定する方法もあります。

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

`root` など、`mysql` データベースへのアクセス権限があるユーザだけが、別のユーザのパスワードを変更することができます。匿名ユーザでなければ、自分のパスワードを `FOR` 節を省略することでパスワードの変更ができます。

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

アカウントにある現在の権限に影響を与えることなく、アカウントのパスワードを設定するには、`GRANT USAGE` ステートメントをグローバル レベル (`ON *.*`) で使用します。

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

ここに前述した方法が、パスワードを設定するときの推奨のやり方ですが、`user` テーブルを直接に変更する方法を取ることも可能です。

- 新規アカウント作成のパスワード指定方法 (`Password` カラムに値を指定)

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

- 既存アカウントのパスワード変更方法 (`UPDATE` で `Password` カラム値のセット)

```
shell> mysql -u root mysql
```

```
mysql> UPDATE user SET Password = PASSWORD('bagel')
-> WHERE Host = '%' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```

SET PASSWORD、INSERT、UPDATEなどで、空白ではないパスワードのアカウントに設定する場合は、PASSWORD()関数で暗号化します。userテーブルは、平文テキストではなく、暗号化形式でパスワードを保存するため、PASSWORD()の使用は不可欠です。これを忘れた場合には、パスワードを次のように設定します。

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

ここでは、'biscuit'というリテラルの値をuserテーブルにパスワードとして保存するという結果になっています。暗号化した値ではありません。ここで肝心なことは、jeffreyがそのパスワードでサーバ接続を試行したときに、そのパスワードの値が暗号化されるということです。つまり、userテーブルにある値とは異なった文字列で照会されるということです。'biscuit'というリテラルの文字列で保存しているところへ、暗号化された別の文字列で入ってくるため、サーバは接続を拒否します。

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

パスワード設定をGRANT ... IDENTIFIED BYステートメントまたはmysqladmin passwordコマンドで行なうときは、どちらのスクリプトでもパスワードの暗号化が自動的に行なわれます。そのため、この場合には、PASSWORD()関数は不要です。

注意：PASSWORD()の暗号化は、Unixのパスワード暗号化とは別物です。MySQL ユーザ名とパスワードを参照してください。

パスワードのセキュリティ

user権限テーブルへのアクセスを一般ユーザには与えてはいけません。

MySQLサーバに接続するクライアントプログラムを実行するときには、別のユーザにそのパスワードを知られない最大の努力をしてください。ここでは、クライアントプログラムを実行するときに指定するパスワードの使用法とともに、それぞれに付随するリスクについて説明します。

- コマンドラインで-pyour_passまたは--password=your_passなどのオプションを使用する。

```
shell> mysql -u francis -pfrank db_name
```

この方法は簡単ですが、安全ではありません。psなどのシステムステータスプログラムでパスワードが可視的になります。このようなプログラムは別のユーザがコマンドラインを呼び出す可能性があります。MySQLクライアントでは通常、初期シーケンス中にコマンドラインのパスワード引数をゼロで書き換えます。しかし、これには値が可視的になる微妙なインターバルがあります。SystemV Unixなど、システムによっては、psでパスワードが可視状態になる問題があるので、この方法はお勧めしません。

- パスワード指定なしで、-pまたは--passwordオプションを使用する。(your_pass値の指定を行わない。)この場合、クライアントプログラムが端末からのパスワード入力を要求する。

```
shell> mysql -u francis -p db_name
Enter password: *****
```

※ 文字はパスワードの入力場所です。パスワードは入力時には表示しません。

コマンドラインからパスワードを指定するよりも、この方法でパスワードを入力の方が安全です。これは別のユーザに対して可視的ではありません。しかし、このパスワード入力方法は、相互的に実行するプログラムだけで使用することをお勧めします。相互的ではない場合に、スクリプトからクライアントを呼び出すと、端末からパスワードを入力することができません。システムによっては、スクリプトの最初のラインが、読み込んだパスワードが不正になる場合があります。

- オプションファイルにパスワードを保存する。Unix例：ホームディレクトリにある.my.cnfファイルの[client]にパスワードをリストする。

```
[client]
```

```
password=your_pass
```

`.my.cnf` にパスワードを保存する場合、ファイルを自分以外の誰からにもアクセスができないようにします。ファイルのアクセス モードを `400` または `600` に設定して確認します。

```
shell> chmod 600 .my.cnf
```

[オプションファイルの使用](#)で、オプション ファイルに関する記述を参照してください。

- **MYSQL_PWD** 環境変数でパスワードを保存する。この方法で MySQL パスワードを指定することは、非常に危険です。`ps` のバージョンによっては、実行プロセスの環境を表示するオプションがあるため、**MYSQL_PWD** でパスワードを設定すると、`ps` を実行している別のユーザに露呈することになります。`ps` がないシステムでも、処理環境を別のユーザに露呈する可能性があります。[環境変数](#)を参照してください。

結論としては、クライアント プログラムのプロンプトでパスワード確認するか、セキュリティを確保したオプション ファイルに適切にパスワードを指定することが、最も安全な方法です。

接続安全

MySQLでは、MySQL クライアントと Secure Sockets Layer (SSL) プロトコルを使用してるサーバ間での暗号化接続をサポートしています。このセクションでは、SSL 接続の使用方法について説明します。さらに、Windows での SSH セットアップ方法についても説明します。ユーザへの SSL 接続の要求方法については、[GRANT 構文](#)で **GRANT** ステートメントの **REQUIRE** 節に関する記述を参照してください。

MySQL の標準設定では、高速化に重点を置いています。そのため、データの暗号化はクライアントとサーバ間での接続プロトコルを遅くするという理由から、デフォルト設定はしていません。暗号化データは、CPU を大幅に消費して、これはコンピュータに負荷を与えるため、MySQL タスクを遅らせる原因となります。しかし、セキュリティ上、暗号化接続を必要とするアプリケーションでは、暗号化してください。

MySQL では接続単位の暗号化接続が可能です。アプリケーションに応じて、通常の暗号化なしの接続、または SSL による安全な接続を使い分けることができます。

OpenSSL API を元にしたセキュリティ上、安全な接続には、MySQL C API を利用します。レプリケーションでは、この C API を使用して、マスタとサーバ間での接続においてセキュリティを確保しています。

SSL の基本概念

MySQL での SSL 使用方法を理解するために、まず、基本的な SSL と X509 概念について説明します。この基本概念をよく理解している場合は、このセクションを読み飛ばしてください。

MySQL のデフォルト設定では、クライアントとサーバ間の接続を暗号化していません。これは、ネットワークにアクセスできる者がデータの送受信を傍受する可能性があることを示します。場合によっては、送受信中にデータの変更(改ざん)にまで及びます。そのため、クライアント プリグラムを呼び出すときには、`--compress` オプションを使用するなどして、クライアントとサーバ間のセキュリティを少しでも高めることをお勧めします。しかし、これはクラッカー対策としては十分ではありません。

公開ネットワークでの情報のやり取りでも安全性が求められ、基本的に暗号化なしの接続は危険です。暗号化とは、データを読めないようにするというものであり、現在でも暗号化アルゴリズムには様々なセキュリティ対策が投げられています。暗号化メッセージの入れ替えやデータ再生の繰り返しなど、様々な攻撃に対応していく必要があります。

SSL とは、複数の異なる暗号化アルゴリズムを使用して、公開ネットワークで受信するデータの信頼性を保証するためのプロトコルです。SSL にはデータの変更、消失、および再生を検知するメカニズムがあります。さらに、X509 規格の ID 認証方式のアルゴリズムも組み込まれています。

X509 とは、インターネット上で ID 認証を可能にする規格です。これは、電子商取引アプリケーションで最も一般的に使用されています。基本的には、「認証局 (Certificate Authority)」という組織が電子証明書が必要とする者に割り当てるという方法を取ります。証明書には、2 つの暗号化キー (公開キーと秘密キー) がある非対称暗号化アルゴリズムを使用しています。証明書の所有者は、他者に証明書を提示して自分の ID を証明します。証明書には、所有者の公開キーが含まれています。この公開キーで暗号化するデータは、これに対応している秘密キーがなければ解読できません。秘密キーは証明書の所有者が保持しています。

SSL、X509、および暗号化に関する詳細は、インターネット検索エンジンなどで情報検索してください。

SSL接続

MySQL サーバとクライアント プログラムの間で SSL 接続を行うには、まずシステムが OpenSSL または yaSSL のいずれかに対応しているか、そして、使用中の MySQL バージョンが SSL に対応しているかどうかを確認してください。

MySQL は、セキュリティを確保した接続を簡単に行うために、yaSSL とのバンドルになっています。(MySQL と yaSSL は同一のライセンス モデルを採用、OpenSSL は Apache のライセンス。) yaSSL 対応のプラットフォームには限りがありましたが、現在では、MySQL AB サポートのプラットフォームすべてで利用できます。

MySQL と SSL を扱うときの接続安全を確保するには、次の手順に従います。

1. SSL 対応の MySQL のバイナリ配布を使用していない環境で、バンドルの yaSSL ライブラリではなくて、OpenSSL を使用するという場合は、まず OpenSSL をインストールする。(MySQL では OpenSSL 0.9.6 でテスト済。) OpenSSL は、<http://www.openssl.org> からインストールする。
2. SSL 対応の MySQL のバイナリ配布を使用していない場合、MySQL のソース配布で SSL を使用できるように設定する。MySQL を設定するときは、[configure](#) スクリプトを次のように呼び出す。

```
shell> ./configure --with-ssl
```

ここでは、バンドルの yaSSL ライブラリを使用できるようにソース配布を設定している。OpenSSL を使用する場合には、OpenSSL ヘッダ ファイルとライブラリがあるディレクトリのパスで `--with-ssl` オプションを指定する。

```
shell> ./configure --with-ssl=path
```

MySQL 5.1.11 より前のバージョンを使用している場合は、適切なオプションを使用して、使用する SSL ライブラリを選択する。

yaSSL:

```
shell> ./configure --with-yassl
```

OpenSSL:

```
shell> ./configure --with-openssl
```

ノート： Unix 対応の yaSSL では、真の乱数を読み出すために、`/dev/urandom` または `/dev/random` のどちらかを用意する。Solaris 2.8 や HP-UX より前のバージョンでの yaSSL に関する追加情報などは、Bug #13164 を参照のこと。

3. 権限テーブルのアップグレードに、`mysql.user` テーブルの SSL 関連カラムを権限テーブルに含めていることを確認する。MySQL 4.0 より古いバージョンの権限テーブルでは、この作業が必要です。アップグレード手順は [mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照のこと。
4. SSL 対応でサーバ バイナリをコンパイルしていることを確認するには、`--ssl` オプションで呼び出す。サーバが SSL 非対応の場合は、エラーが出る。

```
shell> mysql --ssl --help
060525 14:18:52 [ERROR] mysql: unknown option '--ssl'
```

`mysql` サーバが SSL 対応していることを確認するには、`have_openssl` システム変数を調べる。

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_openssl  | YES   |
+-----+-----+
```

SSL 接続対応であれば、値は **YES**。値が **DISABLED** である場合は、`--ssl-xxx` オプションで起動すると SSL 対応になるということ (このセクションの後述を参照のこと)。SSL 接続対応であれば、値は **YES**。

SSL 接続を有効にするには、適切な SSL 関連コマンド オプションを使用します。(SSL コマンド オプション を参照のこと。)

MySQL サーバを起動して、クライアントが SSL 経由で接続できるようにするには、キーを識別するオプションとサーバの接続確立に必要な証明ファイルを使用します。

```
shell> mysqld --ssl-ca=cacert.pem \
--ssl-cert=server-cert.pem \
--ssl-key=server-key.pem
```

- `ssl-ca` で CA 証明書 を認識する。
- `ssl-cert` で、サーバのパブリック キーを認識する。これをクライアントに送信すると、そこにある CA 証明書を認証する。
- `ssl-key` がサーバ プライベート キーを認識する。

SSL 対応の MySQL サーバとの接続安全を確立するには、クライアント指定のオプションが、クライアントで使用するユーザ アカウントの SSL 条件に依存します。[GRANT 構文](#) で [REQUIRE](#) 節に関する記述を参照してください。

アカウントに特別の SSL 条件がない場合、または [REQUIRE SSL](#) オプションを含む [GRANT](#) ステートメントでアカウントを作成している場合は、`--ssl-ca` オプションで、クライアント接続が安全に行えます。

```
shell> mysql --ssl-ca=cacert.pem
```

クライアント証明書の指定も必要な場合は、アカウントを [REQUIRE X509](#) オプションを使用して作成します。そのとき、そのクライアントでも適切なクライアント キーと証明ファイルを指定する必要があります。これをしないと、サーバが接続を拒否します。

```
shell> mysql --ssl-ca=cacert.pem \
--ssl-cert=client-cert.pem \
--ssl-key=client-key.pem
```

これは、サーバに使用するものと同様のオプションであることを示します。ノート：CA 証明書が同じである必要があります。

クライアントで、サーバとの現在の接続で SSL を使用しているかどうかを決定します。ここでは、`Ssl_cipher` ステータス変数の値をチェックします。SSL を使用していない場合、`Ssl_cipher` の値は空白です。SSL を使用してる場合、値は空白ではありません。例示のようになります。

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+-----+-----+
| Variable_name | Value          |
+-----+-----+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+-----+-----+
```

`mysql` クライアントでは、`STATUS` または `★\s★` コマンドを使用して、SSL のラインをチェックします。

```
mysql> \s
...
SSL:          Not in use
...
```

または

```
mysql> \s
...
SSL:          Cipher in use is DHE-RSA-AES256-SHA
...
```

クライアント プログラム内で接続安全を確立するには、`mysql_ssl_set()` C API 関数を使用し、`mysql_real_connect()` 関数を呼び出す前に、適切な証明オプションをセットします。(`mysql_ssl_set()` を参照のこと。) 接続を確立したら、`mysql_get_ssl_cipher()` を使用して、SSL が使える状態になっているかどうかを確認します。戻値が `NULL` ではない場合は、接続が安全であることを示し、SSL 暗号鍵を指します。戻値が `NULL` である場合は、SSL が使用できていないことを示します。`mysql_get_ssl_cipher()` を参照してください。

SSL コマンド オプション

SSL 使用、証明ファイル、キー ファイル を指定するときに使用するオプションについて説明します。コマンドラインまたはオプション ファイルを使用します。このオプションは、SSL 対応の MySQL でのみ利用できるオプションで

す。(SSL接続を参照のこと。) `--master-ssl*` オプションは、レプリケーションを行うときに、スレーブ サーバからマスタ サーバ間での安全接続を確保するときに使用します。(レプリケーションのオプションと変数を参照のこと。)

- `--ssl`

サーバに SSL 接続の許可を指定するオプション。クライアント プログラムに対しては、SSL を使用してサーバに接続することを許可する。このオプションだけでは SSL 使用の接続を行うには不十分。 `--ssl-ca`、`--ssl-cert`、および `--ssl-key` オプションも指定する必要がある。

このオプションは、別の SSL オプションを上書きをするときなど、別の使い方をするのが一般的である。たとえば、SSL を使用しないと示すときなど。この使い方をするときは、`--skip-ssl` または `--ssl=0` として指定する。

注意: `--ssl` オプションの使用には、SSL 接続を必要としない。たとえば、サーバまたはクライアントが SSL サポートをコンパイルしていない場合、通常の暗号化なしの接続になるだけである

SSL を使用した接続を安全に指定するには、`GRANT` ステートメントに `REQUIRE SSL` 節を含めてアカウントをサーバに作成する。そして、そのアカウントでサーバに接続すると、サーバとクライアントの両方で SSL サポートの接続になる。

`REQUIRE` 節は、別の SSL 関連オプションも有効にする。`GRANT` 構文では、`REQUIRE` に関する記述を参照のこと。そこでは、`REQUIRE` オプションで作成したアカウントを使用して接続するクライアントで指定する必要がある SSL のコマンド オプションに関する詳細について記述している。

- `--ssl-ca=file_name`

信頼された SSL 認証局 (trusted SSL CA) の一覧があるファイルのパス。

- `--ssl-capath=directory_name`

PEM 形式の信頼された CA 証明書を保存しているディレクトリのパス。

- `--ssl-cert=file_name`

接続安全を確立するために使用する SSL 証明書ファイルの名前。

- `--ssl-cipher=cipher_list`

SSL 暗号化に使用できるサイファ (暗号) の一覧。 `cipher_list` は、`openssl ciphers` コマンドと同じ形式。

例: `--ssl-cipher=ALL:-AES:-EXP`

- `--ssl-key=file_name`

接続安全を確立するために使用する SSL キー ファイルの名前。

- `--ssl-verify-server-cert`

クライアント プログラム用のオプション。サーバに接続するときに使用するホスト名に対して、サーバ証明書の Common Name 値を検証するようにするオプションで、一致しない場合には接続却下になる。この機能は、中間者攻撃対策として使用できる。この検証のデフォルトは無効。(MySQL 5.1.11 追加のオプション)

SSL 証明のセットアップ

このセクションでは、MySQL サーバとクライアントで使用する SSL 証明書とキー ファイルののセットアップ方法について説明します。最初の例では、コマンドラインから使用可能な簡略化した手順を示します。2 番目の例では、より詳細なスクリプトで表示しています。どちらの例でも、OpenSSL の一部の `openssl` コマンドを使用しています。

その次の例は、MySQL サーバとクライアントの証明書とキー ファイルを作成するときのコマンド セットです。 `openssl` コマンドでいくつかのプロンプトに対応する必要があります。テストするには、Enter キー を使用します。プロダクション仕様には、空ではないレスポンスを用意します。

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts

# Create CA certificate
```

```

shell> openssl genrsa 2048 > ca-key.pem
shell> openssl req -new -x509 -nodes -days 1000 \
    -key ca-key.pem > ca-cert.pem

# Create server certificate
shell> openssl req -newkey rsa:2048 -days 1000 \
    -nodes -keyout server-key.pem > server-req.pem
shell> openssl x509 -req -in server-req.pem -days 1000 \
    -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem

# Create client certificate
shell> openssl req -newkey rsa:2048 -days 1000 \
    -nodes -keyout client-key.pem > client-req.pem
shell> openssl x509 -req -in client-req.pem -days 1000 \
    -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > client-cert.pem

```

以下は、MySQL に SSL 証明書をセットアップする方法を示すスクリプト例です。

```

DIR='pwd'/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/cacert.pem \
    -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# ----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# ----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# .....++++++
# writing new private key to '/home/monty/openssl/server-key.pem'

```

```
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key (optional)
#

openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -policy policy_anything -out $DIR/server-cert.pem \
-config $DIR/openssl.cnf -infile $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName       :PRINTABLE:'FI'
# organizationName  :PRINTABLE:'MySQL AB'
# commonName        :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
$DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to /home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
```

```
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove a passphrase from the key (optional)
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca -policy policy_anything -out $DIR/client-cert.pem \
-config $DIR/openssl.cnf -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName      :PRINTABLE:'FI'
# organizationName :PRINTABLE:'MySQL AB'
# commonName       :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " ""
' > $DIR/my.cnf
```

SSL 接続をテストするには、サーバを次のように立ち上げます。[\\$DIR](#) のあるところが、サンプルの [my.cnf](#) オプション ファイルがあるディレクトリのパスです。

```
shell> mysqld --defaults-file=$DIR/my.cnf &
```

同じオプション ファイルを使用して、クライアント プリグラムを呼び出します。

```
shell> mysql --defaults-file=$DIR/my.cnf
```

MySQL ソース配布がある場合、この [my.cnf](#) ファイルを変更して、セットアップしたものをテストすることができま
す。ソース配布の [mysql-test/std_data](#) ディレクトリにあるデモンストレーション用の証明書とキー ファイルを使用し
ます。

SSH で Windows からリモート接続

SSH で 遠隔の MySQL サーバに安全接続を行う方法について説明します。(David Carlson <dcarlson@mplcomm.com> 提供)

1. まず、Windows マシンに SSH クライアントをインストールする。有償版では、<http://www.vandyke.com/> の SecureCRT のもの、または <http://www.f-secure.com/> の f-secure のものが妥当。無償版では、Google のhttp://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/ にあるものが妥当。
2. Windows SSH クライアントを起動後、Host_Name = yourmysqlserver_URL_or_IP とする。そして、サーバへのログインには、userid=your_userid とする。userid という値は、MySQL アカountのユーザ名とは別物。
3. ポート転送をセットアップする。リモート転送は、local_port: 3306, remote_host: yourmysqlservername_or_ip, remote_port: 3306 として、ローカル転送は、port: 3306, host: localhost, remote port: 3306 とする。
4. ここで、すべてを保存する。保存しない場合、次回に同じことを繰り返すことになる。
5. 作り立ての SSH セッションでサーバにログインする。
6. Windows のマシンで、cess などの ODBC アプリケーションのどれかを立ち上げる。
7. Windows で新規ファイルを作成して、通常と同じ方法で ODBC ドライバを使用して、MySQL にリンクする。ただし、MySQL ホスト サーバに localhost (yourmysqlservername) を入力しないこと。

この時点で、SSH を使用した暗号化で、MySQL との ODBC接続が確立する。

バックアップとリカバリ

このセクションでは、全体 (フル)そして増加分 (インクリメント) のデータベースのバックアップを行う方法を説明します。SQL ステートメントのシンタックスについては、[12章SQL ステートメント構文](#) を参照してください。このセクションでは、[MyISAM](#) テーブルに関連した事柄について重点を置いています。[InnoDB](#) テーブルのバックアップ手順については、[InnoDB データベースのバックアップと復旧](#) を参照してください。

データベースのバックアップ

MySQL テーブルはファイルとして保存するため、バックアップを簡単に行えます。整合性のあるバックアップを行うには、関連するテーブルで [LOCK TABLES](#) を行い、そのテーブルを [FLUSH TABLES](#) します。(LOCK TABLES と [UNLOCK TABLES 構文](#) と [FLUSH 構文](#) を参照のこと。) 読み込みロックだけを行うため、データベース ディレクトリのファイル コピーを行う一方で、別のクライアントはテーブル照会を続けることができます。ここで、[FLUSH TABLES](#) ステートメントを必要とする理由は、バックアップを開始する前に、アクティブのインデックス ページすべてのディスクへの書き込みを確実に行うためです。

テーブルを SQL レベルでバックアップするには、[SELECT INTO ... OUTFILE](#) を使用します。ファイルの上書きはセキュリティ リスクに繋がるため、このステートメントでは既存のファイル名を指定する事は出来ません。[SELECT 構文](#) を参照してください。

データベースをバックアップする別のテクニックとして、[mysqldump](#) または [mysqlhotcopy script](#) を使用する方法があります。[mysqldump — データベースバックアッププログラム](#)、[mysqlhotcopy — データベースバックアッププログラム](#) をそれぞれ参照してください。

1. データベースのフル バックアップ方法

```
shell> mysqldump --tab=/path/to/some/dir --opt db_name
```

または

```
shell> mysqlhotcopy db_name /path/to/some/dir
```

サーバが更新作業中でなければ、バイナリ バックアップを *.frm、*.MYD、*.MYI などのテーブル ファイルをコピーする方法もある。そのときは、[mysqlhotcopy](#) スクリプト を使用する。ただし、データベースに [InnoDB](#) テーブルがある場合には、この方法でバックアップすることはできない。[InnoDB](#) にはデータベース ディレクトリにテーブル内容を保存していないため、[mysqlhotcopy](#) は、[MyISAM](#) テーブルの場合にだけ使用する。

2. `mysqld` を実行している場合は、終了して、`--log-bin=[file_name]` オプションでこれを立ち上げる。(バイナリ ログを参照のこと。) バイナリ ログ ファイルには、`mysqldump` を実行したその時点から後のデータ変更を複製するときに必要な情報が入っている。

InnoDB テーブルに対して、オンライン バックアップはできますが、テーブルにはロックがありません。[mysqldump — データベースバックアッププログラム](#) を参照のこと。

MySQL の増加分バックアップ (インクリメント)：バイナリ ロギングができるようにするために、サーバを `--log-bin` オプションで起動します。(バイナリ ログを参照のこと。) インクリメント バックアップを行う時点で、`FLUSH LOGS` を使用して、バイナリ ログをローテートします。(このときのバックアップ内容とは、前回のフルまたはインクリメント バックアップをしてから発生した変更のことです。) ローテートを行った後、バックアップする部分をコピーします。この部分の範囲は、前回のバックアップ (フルまたはインクリメント) を行った時点を経由し、このバックアップ操作を行う時点までを終点とします。つまり、追加部分です。(インクリメント バックアップには、リストアした時点でのバイナリ ログが 2 つあるということで、これは順次説明します。つまり、次回、フル バックアップを行うときも、`FLUSH LOGS`、`mysqldump --flush-logs`、`mysqlhotcopy --flushlog` のいずれかを使用してバイナリ ログのローテートが必要です。詳細は、[mysqldump — データベースバックアッププログラム](#) と [mysqlhotcopy — データベースバックアッププログラム](#) を参照のこと。)

MySQL サーバがレプリケーション スレーブである場合、どのようなバックアップの方法を取るとしても、スレーブのデータをバックアップするときには、`master.info` と `relay-log.info` の両ファイルをバックアップしてください。これらのファイルは、スレーブのデータをリストアするときに、レプリケーションのレジューム (再開) が必要です。スレーブが `LOAD DATA INFILE` コマンドを使用するレプリケーションの対象である場合、ディレクトリの `SQL_LOAD-*` ファイルもバックアップしてください。このファイルは、`--slave-load-tmpdir` オプションを指定すると出てきます。(このファイル位置を指定しない場合、この位置は `tmpdir` 変数の値 (デフォルト) になります。) このファイルは、`LOAD DATA INFILE` 操作が中断した場合などに、スレーブのレプリケーション再開で必要になります。

MyISAM テーブルをリストアする必要がある場合、まず `REPAIR TABLE` または `myisamchk -r` を使用して、リカバリしてください。この方法は、99.9% 確実です。`myisamchk` コマンドで失敗した場合は、次の手順を試行します。ノート：この方法は、MySQL を `--log-bin` オプションで立ち上げ、バイナリ ロギングを行っていた場合にだけ適用します。

1. オリジナルの `mysqldump` バックアップ、またはバイナリ バックアップをリストアする。
2. 次のコマンドを実行し、バイナリ ログの更新を再実行する。

```
shell> mysqlbinlog binlog.[0-9]* | mysql
```

場合によっては、特定の場所からのバイナリ ログだけを再実行する必要がある。通常は、バイナリ ログのすべてを再実行にはリストアしたバックアップの日から行うが、これには適切ではないステートメントが含まれている場合がある。[mysqlbinlog — バイナリログファイルを処理するためのユーティリティ](#) で、`mysqlbinlog` ユーティリティとその使用に関する詳細を参照のこと。

個別ファイルの部分的なバックアップを行う場合：

- テーブルをダンプするには、`SELECT * INTO OUTFILE 'file_name' FROM tbl_name` を使用する。
- テーブルをリロードするには、`LOAD DATA INFILE 'file_name' REPLACE ...` を使用する。テーブルに `PRIMARY KEY` または `UNIQUE` インデックスがあれば、レコードの重複を避けることができる。一意のキー値がある場合、`REPLACE` キーワードは、古いレコードが新しいものと入れ替えるので注意が必要。

バックアップするときにサーバにパフォーマンス上の問題が発生する場合、レプリケーションをセットアップして、マスタではなくスレーブ上でバックアップを実行することによりこの問題を解決できます。[5章レプリケーション](#) を参照のこと。

Veritas ファイルシステムを使用している場合、次の手順でバックアップを行います。

1. クライアント プログラムから、`FLUSH TABLES WITH READ LOCK` を実行する。
2. 別のシェルから、`mount vxfs snapshot` を実行する。
3. 最初のクライアントから、`UNLOCK TABLES` を実行する。
4. スナップショットからファイルをコピーする。

5. スナップショットのマウントを解除する。

バックアップとリカバリ手法の例示

このセクションでは、クラッシュなどで、リカバリが必要にあるデータのバックアップ手順について説明します。クラッシュとは次に示す事柄です。

- オペレーティング システムのクラッシュ
- 停電
- ファイルシステムのクラッシュ
- ハードウェアの問題 (ハード ドライブ、マザーボードなど)

例示のコマンドには、`mysqldump` および `mysql` プログラムなどの `--user` および `--password` オプションなどは入っていません。MySQL サーバで接続が必要な場合などに応じて、そのようなオプションを追加してください。

ここでは、データが `InnoDB` ストレージ エンジンで保存しているものとします。つまり、トランザクションと自動クラッシュ リカバリのサポートがあるという前提です。このときの MySQL サーバには問題になる負荷がかかっているものとします。

オペレーティング システムのクラッシュまたは停電などの場合、MySQL のディスク データは再起動すると利用できるようになると考えられます。クラッシュすると `InnoDB` データ ファイルは、データの整合性を失う可能性があります。ただし、`InnoDB` はログを読み込み、まだデータ ファイルにフラッシュしていないコミット/非コミット トランザクションのリストを検索します。`InnoDB` は自動的にまだコミットしていないトランザクションをロール バックし、コミットしたものはデータ ファイルにフラッシュします。ユーザは、このリカバリ プロセスの情報を MySQL エラー ログから読むことができます。

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

ファイルシステムのクラッシュまたはハードウェアの問題などの場合は、MySQL のディスク データが再起動しても利用可能にならないと考えられます。これは、ディスク データの一部はすでに読めない状態になっているため、MySQL は起動に失敗するという意味です。この場合、ディスクを再フォーマットする、または、新しいものをインストールする必要があります。放置すると、問題は解決しません。そして、バックアップから MySQL データのリカバリを行います。つまり、このときにはすでにバックアップを取っていることが必要です。このような場合を回避するためには、バックアップのポリシーをデザインしなければなりません。(次のセクションを参照のこと。)

バックアップ ポリシー

万が一に備えて、定期的にバックアップを取る必要があります。MySQL では、いくつかのツールを使用して、フル バックアップ (ある時点でのデータのスナップショット) を取ることができます。たとえば、`InnoDB Hot Backup` を使用すると、オンラインでブロックしていない `InnoDB` データ ファイルのフィジカル バックアップ (物理的) が取れ、`mysqldump` を使用すると、オンラインのロジカル バックアップ (理論的) を取ることができます。このセクションでは、`mysqldump` について説明します。

MySQL Enterprise

MySQL Network Monitoring and Advisory Service #####
<http://www.jp.mysql.com/products/enterprise/advisors.html> #####

負荷が少ない日曜の午後 1 時にバックアップを取ると仮定します。次のコマンドで、すべてのデータベースの InnoDB テーブルすべてをフル バックアップします。

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

これは、オンラインのバックアップをブロックしていない場合のため、テーブルの読み書き込みには影響しません。ここでは、バックアップ対象のテーブルが InnoDB であると仮定しているため、`--single-transaction` オプションでは整合性のあるデータ読み込みになり、`mysqldump` で使用したデータは変更しません。(クライアントによる InnoDB テーブルへの変更は、`mysqldump` プロセスで対象になっていません。)異なる種類のテーブルもある場合には、そのテーブルがバックアップ中に変更しないものとします。たとえば、`mysql` データベースの `MyISAM` テーブルでは、バックアップ中に管理系の変更を MySQL のアカウントに対して行っていないものとします。

`mysqldump` による `.sql` ファイルには、後で使用するテーブル ダンプのリロードに使用する SQL `INSERT` ステートメントのセットが入ります。

フル バックアップは不可欠ですが、時間を要します。大きなバックアップ ファイルの生成には時間がかかります。前回のフル バックアップを行ってから全く変更のない部分も含めて、すべてのデータをフル バックアップを何度も行うことは最適化とはいえません。一度、フル バックアップを行ったら、次からは、インクリメント バックアップを行う方が効率的です。この方法であれば、負荷も減り、生成にかかる時間も短縮できます。トレードオフとしては、リカバリを行うときに、フル バックアップをリロードするだけではデータの回復にはならない、増加分にはインクリメント バックアップも使用して回復しなければならないという良し悪しがあります。

インクリメント バックアップを行うには、増加分/変更分 (インクリメント) を保存する必要があります。これには、MySQL サーバを常に `--log-bin` オプションで起動する必要があります。これにより、データ更新中に合わせて変更をファイルに保存できます。このオプションはバイナリ ロギングを可能にするものであるため、サーバは MySQL バイナリ ログというファイルに、データ更新に関するそれぞれの SQL ステートメントを書き込みます。ここで、数日間実行していた MySQL バイナリ ログ ファイルの例を次に示します。これは `--log-bin` オプションで起動した MySQL サーバのデータ ディレクトリのもです。

```
-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem guilhem    4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem guilhem   79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem guilhem  508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem guilhem 2247446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem guilhem 998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem guilhem   361 Nov 14 10:07 gbichot2-bin.index
```

再起動する度に、MySQL サーバは新たなバイナリ ログ ファイルを作成します。これには連続する番号のシーケンスがあります。サーバを実行する一方で、そのサーバに使用中のバイナリ ログ ファイルを閉じて、新しいファイルを開始するように命令することができます。それには手動で、`FLUSH LOGS` SQL ステートメント、または `mysqladmin flush-logs` コマンドを使用します。`mysqldump` にもログをフラッシュするオプションがあります。ディレクトリの MySQL バイナリ ログのすべてのリストを含む `.index` というファイルがデータ ディレクトリにあります。このファイルはレプリケーションに使用します。

MySQL バイナリ ログでは、インクリメント バックアップのセットを形成しているため、リカバリでは重要になります。フル バックアップを行う場合に、ログのフラッシュを確実にすると、それ以後のバイナリ ログ ファイルは変更した部分だけのデータを含むファイルになります。ここで、前述の `mysqldump` コマンドを若干修正すると、フル バックアップを行った時点での MySQL バイナリ ログをフラッシュするようになります。つまり、ダンプ ファイルには新しいバイナリ ログの名前を含むようになります。

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases > backup_sunday_1_PM.sql
```

このコマンドを実行後、データ ディレクトリには `gbichot2-bin.000007` という新しいバイナリ ログ ファイルができます。`.sql` ファイルは次のラインが入ります。

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',
MASTER_LOG_POS=4;
```

`mysqldump` コマンドでフル バックアップを取っているため、これらのラインは 2 つの意味があります。

- `.sql` ファイルは、`gbichot2-bin.000007` というバイナリ ログ ファイル、またはそれ以降の新しいファイルに変更が書き込まれるよりも前の、すべての変更を含む。
- バックアップ後にログしたデータのすべてが、`.sql` には存在しないが、`gbichot2-bin.000007` というバイナリ ログ ファイル、または以降の新しいファイルに存在する。

そして、月曜の午後 1 時に、新しいバイナリ ログ ファイルでログを開始するために、それまでのログをフラッシュして、インクリメント バックアップを取ります。たとえば、`mysqldump flush-logs` コマンドを実行して、`gbichot2-bin.000008` というファイルを作成します。ここで、日曜の午後 1 時のフル バックアップを起点とし、月曜の午後 1 時までを終点とするすべての変更が、`gbichot2-bin.000007` ファイルとなります。この手順で、このファイルには大事な役割があるので、コピーしてテープ、DVD、別のマシンなど安全な場所に保管しておきます。そして、翌日、火曜日の午後 1 時に、改めて、`mysqldump flush-logs` コマンドを実行すると、`gbichot2-bin.000008` ファイルが、月曜の午後 1 時を起点とし、火曜の午後 1 時を終点とするすべての変更を含むことになります。(これも安全な場所にコピーを保管します。)

MySQL バイナリ ログはディスク スペースを消費するので、必要に応じてスペースの解放が必要です。その 1 つの方法としては、たとえば、フル バックアップ後に不要になったバイナリ ログを削除します。

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

注意：サーバがレプリケーションのマスタ サーバである場合は、`mysqldump --delete-master-logs` というコマンドで MySQL バイナリ ログを削除するということには危険が伴います。つまり、スレーブ サーバでまだバイナリ ログの内容を完全に処理し切れていない可能性があります。[PURGE MASTER LOGS ステートメントの詳細 \(PURGE MASTER LOGS 構文\)](#) で、MySQL バイナリ ログを削除する前の確認操作について参照してください。

バックアップ ファイルでリカバリ

(前セクションの続きです。) ここで、水曜日の午前 8 時に、バックアップを使用してリカバリを行う必要があるクラッシュが発生したとします。まず、前回のフル バックアップ (日曜午後 1 時のもの) をリストアします。フル バックアップのファイルは SQL ステートメントのセットであるため、リストアは簡単にできます。

```
shell> mysql < backup_sunday_1_PM.sql
```

ここで、データが日曜午後 1 時の時点での状態になります (リストアした。) 次に、これ以降に発生した変更についてもリストアします。つまり、インクリメント バックアップであるバイナリ ログ ファイル、`gbichot2-bin.000007` と `gbichot2-bin.000008` を使用します。必要に応じて、このファイルをバックアップしたところからフェッチして、その内容を次のように処理します。

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

この時点で、データは火曜午後 1 時点の状態になります (リカバリできました。) しかし、まだクラッシュするまでのデータ (変更) が欠けています。まず、リカバリしたものを失くさないために保存します。このときは、MySQL サーバが、MySQL バイナリ ログで安全な場所に保存しなければなりません。RAID ディスク、SAN など、データ ファイルとは別の場所 (壊れていないディスク) に保管してください。この状態で、`--log-bin` を使用してサーバを起動すると、物理的には異なるデバイスのデータ ディレクトリから MySQL のログを立ち上げることができます。ここまでの作業 (リカバリしたものを保存) が完了したら、`gbichot2-bin.000009` ファイル (クラッシュするまでのログ) で、`mysqlbinlog` と `mysql` を適用して、クラッシュした時点までのデータ変更を失うことなく、最新のデータ変更をリストアできます。

バックアップ ストラテジーの概要

オペレーティング システムのクラッシュまたは停電の場合、InnoDB 自体がデータ リカバリに関するすべての作業を行います。しかし、万が一の対策として、次のガイドラインについて検討してください。

- MySQL サーバは常に `--log-bin` オプションで起動する。ログ ファイル名がデータ ディレクトリがあるドライブではなく、安全用のメディアにある場合は、`--log-bin=log_name` を使用すること。安全対策用のメディアは、ディスクの負荷とのバランスを取るためにも、パフォーマンスの向上になる。
- 定期的にフル バックアップを行う。[バックアップ ポリシー](#) で示すように `mysqldump` コマンドを使用すると、オンラインでブロックなしのバックアップができる。
- `FLUSH LOGS` または `mysqldump flush-logs` などを使用して、ログをフラッシュして、インクリメント バックアップを定期的に取り。

任意時点のリカバリ

MySQL サーバを `--log-bin` オプションで起動して、バイナリ ロギングを可能にしておくと、`mysqlbinlog` ユーティリティを利用して、起点と終点を指定して (例: 前回バックアップをした時点から現在まで、など)、バイナリ ログ ファイルからデータのリカバリができます。`mysqlbinlog` を使用したバイナリ ログの有効化に関する詳細は、[バイナリ ログ](#) および [mysqlbinlog — バイナリログファイル処理するためのユーティリティ](#) を参照してください。

MySQL Enterprise

MySQL Network Monitoring and Advisory Service #####
<http://www.jp.mysql.com/products/enterprise/advisors.html>

バイナリ ログからデータをリストアするには、まず、現行のバイナリ ログ ファイルの場所と名前を知る必要があります。デフォルトでは、サーバがデータ ディレクトリにバイナリ ログを作成していますが、パスは `--log-bin` を使用して、別の場所に指定することができます。通常、システムにもよりますが、オプションは `my.cnf` または `my.ini` などのオプション ファイルで与えます。サーバ起動時にコマンドラインから与えることも可能です。現行のバイナリ ログ ファイルの名前を確認するには、次のステートメントを発行します。

```
mysql> SHOW BINLOG EVENTS\G
```

または、コマンドラインから、次のコマンドを実行します。

```
shell> mysql -u root -p -E -e "SHOW BINLOG EVENTS"
```

`mysql` プロンプトで、サーバの `root` パスワードを入力します。

リカバリの時刻指定

リカバリの開始/終了時刻を指定するには、`DATETIME` 形式で、`mysqlbinlog` に `--start-date` と `--stop-date` を指定します。たとえば、2005 年 4 月 20 日の午前 10 : 00 時に、何らかの SQL ステートメントの実行で大きなテーブルが削除された、とします。このテーブルとデータをリストアするには、前夜のバックアップをリストアして、次のコマンドを実行します。

```
shell> mysqlbinlog --stop-date="2005-04-20 9:59:59" \  
/var/log/mysql/bin.123456 | mysql -u root -p
```

このコマンドは、`--stop-date` オプションで指定した日時までのデータすべてをリカバリします。時間分の SQL ステートメントの大部分を探し当てることができない場合は、その部分のアクティビティをリカバリします。これを元に、`mysqlbinlog` を開始日時で再度実行します。次はその例です。

```
shell> mysqlbinlog --start-date="2005-04-20 10:01:00" \  
/var/log/mysql/bin.123456 | mysql -u root -p
```

このコマンドでは、午前 10:01 時以降にログした SQL ステートメントを再実行します。前夜のダンプ ファイルのリストアと、この 2 つの `mysqlbinlog` コマンドの組み合わせで、午前 10:00 時の一秒前までのすべてと、午前 10 : 01 以降のすべてのリストアします。このコマンドを使用するときは、指定する時間をログ ファイルで十分に確認してください。ログ ファイルを実行せずに、内容だけを表示するには、次のコマンドを使用します。

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

そして、そのファイルをテキスト エディタなどで開けて、確認します。

リカバリの位置指定

リカバリの日時ではなく、ログの位置で指定するには、`--start-position` および `--stop-position` オプションを `mysqlbinlog` で使用します。これは、開始/終了時間のオプションと同様の使い方をします。日時の部分をログ位置の番号とします。ログのどの部分をリカバリするのかを位置で指定すると、より正確なリカバリができます。SQL ステートメントへのダメージが起きたときに、大量のトランザクションが発生していた場合などに有用です。位置番号を確認するには、予期していないトランザクションがあった時間帯で `mysqlbinlog` を実行します。このとき、結果を確認用にテキスト ファイルにリダイレクトします。この操作は次のように行います。

```
shell> mysqlbinlog --start-date="2005-04-20 9:55:00" \  
--stop-date="2005-04-20 10:05:00" \  
/var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

このコマンドは、`/tmp` ディレクトリのテキスト ファイルに小さく作成します。このテキスト ファイルには、有害な SQL ステートメントを実行した時間の SQL ステートメントがあります。このファイルをテキスト ファイルで開き、リピートすると害になるステートメントを探します。停止点と開始点に指定するバイナリ ログの位置を確認します。位置は、番号が後続する `log_pos` とラベルで見分けます。前回のバックアップ ファイルをリストアした後、この位置番号を使用して、バイナリ ログ ファイルを処理します。たとえば、次のようにコマンドを使用します。

```
shell> mysqlbinlog --stop-position="368312" /var/log/mysql/bin.123456 \
| mysql -u root -p

shell> mysqlbinlog --start-position="368315" /var/log/mysql/bin.123456 \
| mysql -u root -p
```

最初のコマンドは停止位置までのトランザクションすべてをリカバリします。2 番目のコマンドは、開始位置からバイナリ ログの終わりまでのトランザクションすべてをリカバリします。`mysqlbinlog` の出力には SQL ステートメントを記録する前の `SET TIMESTAMPS` ステートメントを含むため、リカバリしたデータおよび関連する MySQL ログはトランザクションを実行したオリジナルの時刻を反映します。

テーブル保守とクラッシュ リカバリ

このセクションでは、`MyISAM` テーブルをチェックまたは修復するための `myisamchk` について説明します。このテーブルには、データを保存する `.MYD` ファイル、`.MYI` ファイル、そしてインデックスがあります。`myisamchk` の基本的な背景に関しては、[myisamchk — MyISAM テーブル メンテナンス ユーティリティ](#) を参照してください。

`myisamchk` を使用して、データベース テーブルの情報を取得、またはチェック、修復、最適化を行います。順次に操作の仕方、テーブルの保守スケジュールのセットアップについて説明します。

`myisamchk` でテーブルの修復は安全性に優れていますが、テーブルに対して多くの変更が伴う保守では、作業を始める前に バックアップを取ります。

インデックスに影響を与える `myisamchk` の操作は、`FULLTEXT` のインデックスをフル テキストのパラメータで再構築することになります。このパラメータは、MySQL サーバで使用する値との互換性があります。よって、この問題を回避するには、`myisamchk` [一般的なオプション](#) のガイドラインに従ってください。

場合によっては、`myisamchk` で、SQL ステートメントを使用して `MyISAM` テーブルの保守を行うことは通説より簡単です。

- `MyISAM` テーブルのチェックまたは修復には、`CHECK TABLE` または `REPAIR TABLE` を使用。
- `MyISAM` テーブルの最適化には、`OPTIMIZE TABLE` を使用。
- `MyISAM` テーブルの分析には、`ANALYZE TABLE` を使用。

これらのステートメントは、`mysqlcheck` クライアント プログラムを利用して、直接に使用できます。これには、`myisamchk` に対するこれらのステートメントは、サーバがすべての作業を行うという利点があります。`myisamchk` で作業を行うときは、`myisamchk` とサーバの間でそのテーブルを同時に使用 (不要なやり取り) していないことを確認してください。詳細は、[ANALYZE TABLE 構文](#)、[CHECK TABLE 構文](#)、[OPTIMIZE TABLE 構文](#)、[REPAIR TABLE 構文](#) を参照してください。

`myisamchk` でクラッシュ リカバリ

このセクションでは、MySQL データベースのデータ破損に対するチェックと対処方法について説明します。テーブルが頻繁に破損する場合は、[What to Do If MySQL Keeps Crashing](#) を参照するなどして、その原因究明を行い必要があります。

`MyISAM` テーブルが破損する原因に関する説明は、[MyISAM テーブルの問題点](#) を参照してください。

外部ロックを無効にして `mysqld` を実行する場合 (MySQL 4.0 以降のデフォルト)、`mysqld` と `myisamchk` で同時にテーブルをチェックするときは、注意が必要です。`myisamchk` を実行している間は、`mysqld` を使用したテーブルへのアクセスできるのは自分だけであると確認できる場合は、テーブルのチェックを開始する前に、`mysqladmin flush-tables` を実行するだけで作業が行えます。もし確認できない場合は、テーブルのチェックするときに、`mysqld` を停止します。`myisamchk` でテーブルをチェックすると同時に、`mysqld` で更新を行っていると、テーブルが破損していても警告がでます。

外部ロックを有効にしてサーバを実行する場合は、[myisamchk](#) を使用していつでもテーブルをチェックできます。この場合、サーバが [myisamchk](#) で使用しているテーブルを更新しようとする、サーバが [myisamchk](#) を優先して、待機します。

[myisamchk](#) をテーブルの修復または最適化に使用する場合は、[mysqld](#) サーバが目的のテーブルを使用していないことを常に確認してください。これは外部ロックを無効にしているときにも通用します。[mysqld](#) を停止できない場合は最低限として、[myisamchk](#) を実行する前に [mysqladmin flush-tables](#) を実行してください。サーバと [myisamchk](#) が同時にテーブルにアクセスすると、破損の原因になります。

クラッシュ リカバリを行うときは、データベースにある [MyISAM](#) テーブルの [tbl_name](#) それぞれが、データベースディレクトリ内の 3 つのファイルに対応していることを理解する必要があります。この 3 ファイルは次の通りです。

ファイル	用途
tbl_name.frm	テーブル定義ファイル
tbl_name.MYD	データ ファイル
tbl_name.MYI	インデックス ファイル

この 3 ファイルは、様々な形で破損の原因に関係していますが、最も問題が発生するのは、データ ファイルとインデックス ファイルです。

[myisamchk](#) は、[.MYD](#) (データ) ファイルのコピーをレコードごとに生成します。修復の最終段階で古い [.MYD](#) ファイルを削除して、新規ファイルをオリジナルの名前に変更します。[--quick](#) を使用している場合、[myisamchk](#) ではテンポラリの [.MYD](#) ファイルを生成しません。その代わりに [.MYD](#) ファイルが正常であるとみなし、[.MYD](#) ファイルに手を加えずに新規インデックス ファイルだけを生成します。[.MYD](#) ファイルに問題があった場合は [myisamchk](#) が自動的に検知して修復を中止するため、この方法は安全です。2 つの [--quick](#) オプションを [myisamchk](#) に設定することもできます。この場合、[myisamchk](#) はいくつかのエラー (重複キーなど) でも中止しませんが、[.MYD](#) ファイルを修正して解決しようとします。通常、修復処理にディスクの空き容量では足りない場合にのみ、2 つの [--quick](#) オプション指定を利用します。その場合、少なくとも [myisamchk](#) を実行する前にバックアップを作成してください。

MyISAM テーブルのエラー チェック方法

[MyISAM](#) テーブルをチェックするには、次のコマンドを使用します。

- [myisamchk tbl_name](#)

このコマンドでエラーの 99.99% を発見できる。これで発見できないエラーは、データファイルだけに関連する稀な破損である。テーブルをチェックする場合、通常、[myisamchk](#) でオプションなし、または [-s](#) (silent) オプションを使用する。

- [myisamchk -m tbl_name](#)

このコマンドでエラーの 99.999% を発見できる。このコマンドを実行すると、最初にすべてのインデックス エントリのエラーをチェックして、次にすべてのレコードを読み込む。レコードのすべてのキーのチェック サムを計算し、その結果がインデックス ツリーのキーのチェックサムと一致するかどうかを確認する。

- [myisamchk -e tbl_name](#)

このコマンドを実行すると、すべてのデータを完全にチェックする ([-e](#) は「extended check」のこと)。それぞれのレコードのすべてのキーを読み取り、チェックし、正しいレコードを指しているかどうかを確認する。この操作は、キーが多い、大きなテーブルでは時間がかかる。[myisamchk](#) は通常、最初のエラーが見つかった時点で停止する。(停止しないで) 操作を続けるには、[-v](#)(verbose) オプションを追加すると、[myisamchk](#) は最大 20 のエラーを検出するまで操作を続行する。

- [myisamchk -e -i tbl_name](#)

前述のコマンドと同類。[-i](#) オプションをつけると [myisamchk](#) で統計情報も出力する。

大抵の場合、テーブルのチェックには、テーブル名以外の引数なしのシンプルな [myisamchk](#) コマンドで対応できます。

テーブルの修復方法

このセクションでは、**MyISAM** テーブルへの **myisamchk** の使用方法について説明します。(拡張子: **.MYI**、**.MYD**)

MyISAM テーブルのチェックと修復には、**CHECK TABLE** や **REPAIR TABLE** などのステートメントを使用します (推奨)。詳細は、**CHECK TABLE 構文** を参照してください。

テーブル破損の症状としては、クエリが予期せず中断したり、次のようなエラーが発生します。

- **tbl_name.frm** is locked against change
- Can't find file **tbl_name.MYI** (Errcode: **nnn**)
- Unexpected end of file
- Record file is crashed
- Got error **nnn** from table handler

pererror nnn を実行すると、エラーの詳細情報を取得できます。**nnn** はエラー番号です。次の例は、**pererror** を使用した、テーブルに問題があることを示す一般的なエラーです。

```
shell> pererror 126 127 132 134 135 136 141 144 145
126 = Index file is crashed / Wrong file format
127 = Record-file is crashed
132 = Old database file
134 = Record was already deleted (or record file crashed)
135 = No more room in record file
136 = No more room in index file
141 = Duplicate unique key or constraint on write or update
144 = Table is crashed and last repair failed
145 = Table was marked as crashed and should be repaired
```

ノート: エラー 135 (no more room in record file) とエラー 136 (no more room in index file) は、簡単に直せるエラーではありません。この場合、**ALTER TABLE** を使用して **MAX_ROWS** または **AVG_ROW_LENGTH** テーブル オプションを値を修正してください。

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

現行のテーブル オプション値がわからない場合は、**SHOW CREATE TABLE** を使用します。

この 2 つ以外エラーが出る場合は、テーブルを修復します。**myisamchk** で検出するときに、発生するエラーの原因を正します。

修復プロセスには、4 つの段階があります。Unix では、**mysqld** を実行する Unix ユーザに読み取り権限があることを確認します (この確認を行うユーザにもこれらのファイルへのアクセス権が必要)。ファイルを修正する必要がある場合は、書き込み権限も必要です。

このセクションでは、**MyISAM テーブルのエラー チェック方法** で説明しているテーブル チェックで失敗した場合、または **myisamchk** の拡張機能を使用する場合の修復作業について説明しています。

myisamchk で行うテーブル保守のオプションに関しては、**myisamchk — MyISAM テーブル メンテナンス ユーティリティ** を参照してください。

コマンドラインからテーブルを修復する場合は、まず、**mysqld** サーバを停止します。ノート: リモートサーバで **mysqladmin shutdown** を実行するときは、**mysqladmin** を返しても、すべてのステートメント処理の停止、続いてすべてのインデックスの変更をディスクにフラッシュするまでの間、**mysqld** が活動を続けます。

段階 1: テーブルのチェック

myisamchk *.MYI、または時間に余裕があれば **myisamchk -e *.MYI** を実行します。**-s** (silent) オプションを使用すると、不要な情報出力を抑制します。

mysqld サーバが停止している場合は、**--update-state** オプションを使用して **myisamchk** にテーブルで「checked」マークを付けるよう指定します。

myisamchk がエラーを返すテーブルだけを修復する場合は、段階 2 へ進みます。

チェック時に複雑なエラー (`out of memory` エラーなど) が発生した場合、あるいは `myisamchk` がクラッシュした場合、段階 3 へ進みます。

段階 2: 簡単に安全な修復

まず、`myisamchk -r -q tbl_name` を試行する。(`-r -q` は「クイック リカバリ モード」で実行するという意味。) これで、データ ファイルには触れずにインデックス ファイルの修復だけを行います。データ ファイルに必要なデータがすべて揃っていて、削除リンクがデータ ファイル内の正しい位置を指していれば、テーブルは正常に修復できます。この後は、次のテーブルの修復を開始します。失敗した場合は以下の手順を実行します。

1. データ ファイルのバックアップを作成する。
2. `myisamchk -r tbl_name` を使用する。(`-r` は「リカバリ モード」で実行するという意味。) これは、不正なレコードとデータ ファイルから削除したレコードを取り除いて、インデックス ファイルを再構築する。
3. 前のステップが失敗した場合、`myisamchk --safe-recover` . を使用する。セーフ リカバリ モードでは、対応できるケースが少ない、古いリカバリ形式を使用している。このケースは、通常のリカバリ モードでは行わない方法で、処理には時間がかかる。

ノート:

修復オペを高速化するには、`myisamchk` を実行するときの `sort_buffer_size` および `key_buffer_size` の変数の値を、利用可能システム メモリの 25 パーセント (1/4) で設定します。

修復時に複雑なエラー (`out of memory` エラーなど) が発生した場合、あるいは `myisamchk` がクラッシュした場合、段階 3 へ進みます。

段階 3: 困難な修復

インデックス ファイルの最初の 16KB ブロックが破損している、またはそこに不正な情報がある場合、あるいは、インデックス ファイルがない場合に、この段階に進みます。この場合、新しいインデックス ファイルを作成する必要があります。以下を実行してください。

1. データ ファイルを安全な場所に移動する。
2. テーブル記述ファイルを使用して、新しい (空白の) データとインデックスファイルを作成する。

```
shell> mysql db_name
mysql> SET AUTOCOMMIT=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. 古いデータファイルを、新しく作成したデータファイルにコピーする。単に古いファイルを新しいファイルに移動するのではなく、万が一に備えて元の場所にも残しておくこと。

そして、段階 2 に戻ります。これで、`myisamchk -r -q` が機能するはずです (無限ループにならないはずです)。

`REPAIR TABLE tbl_name USE_FRM` の SQL ステートメントを使用して、この手順を自動的に行うこともできます。`REPAIR TABLE` を使用すると、サーバがすべての作業を行うため、ユーティリティとサーバの間に不要なやり取りが起こる可能性はありません。[REPAIR TABLE 構文](#) を参照してください。

段階 4: 非常に困難な修復

`.frm` という記述ファイルもクラッシュしている場合に、この手順に従います。ただし、テーブル作成後に記述ファイルが変更になることはないため、通常は発生しない状況です。

1. バックアップから記述ファイルをリストアし、段階 3 に戻る。インデックス ファイルをリストアして段階 2 に戻ることも可能。後者の場合、`myisamchk -r` で起動する。
2. バックアップがない場合でも、そのテーブルがどのように作成したかが正確にわかるときは、テーブルのコピーを別のデータベースに作成する。新しいデータ ファイルを削除してから、`.frm` 記述ファイルと `.MYI` インデックス ファイルを、その別のデータベースからクラッシュしたデータベースへ移動する。これで新しい記述ファイルとインデックス ファイルができ、`.MYD` データ ファイルは前のものがそのまま残る。段階 2 に戻り、インデックス ファイルを再構築する。

テーブルの最適化

断片化したレコードを結合したり、レコードの削除または更新によって発生した無駄なスペースを除去するには、[myisamchk](#) をリカバリモードで実行します。

```
shell> myisamchk -r tbl_name
```

同様に、SQL の [OPTIMIZE TABLE](#) ステートメントを使用して、テーブルを最適化することもできます。[OPTIMIZE TABLE](#) はテーブルの修復とキー分析を行い、さらにインデックス ツリーをソートして、キー走査の処理速度を上げます。また、[OPTIMIZE TABLE](#) を使用した場合、サーバ側ですべての処理を行うため、ユーティリティとサーバ間で不要なやり取りが発生しません。[OPTIMIZE TABLE 構文](#) を参照してください。

[myisamchk](#) には、テーブルのパフォーマンスを向上させるオプションが数多くあります。

- [--analyze, -a](#)
- [--sort-index, -S](#)
- [--sort-records=index_num, -R index_num](#)

利用可能なオプションの詳細に関しては、[myisamchk — MyISAM テーブル メンテナンス ユーティリティ](#) を参照してください。

テーブル情報の取得

テーブルに関する情報またはその統計を取得するには、次に示すコマンドを実行します。これらの情報については後で詳しく説明します。

- [myisamchk -m tbl_name](#)

[myisamchk](#) を「describe モード」で実行し、テーブル情報を生成する。外部ロックが無効になっている MySQL サーバを起動した場合には、[myisamchk](#) は、実行中に更新があったテーブルに対してエラーを報告することがあるが、データ破壊の危険性はない。describe モードで[myisamchk](#) がテーブルを変更することはない。

- [myisamchk -d -v tbl_name](#)

実行中の処理に関する詳細な情報を取得するには、[-v](#) 追加して、[myisamchk](#) を冗長モードで実行する。

- [myisamchk -eis tbl_name](#)

テーブルの最重要情報のみを表示する。このオペではテーブル全体を読み取るため、時間を要する。

- [myisamchk -eiv tbl_name](#)

[-eis](#) とほぼ同じ。このオプションを使用すると、進行中の処理 (それまでに何が行われたか) も表示する。

ここで、この 3 つのコマンドを使用した出力例を示します。テーブル内の任意データとインデックス ファイルのサイズに基づいています。

```
-rw-rw-r-- 1 monty tcx 317235748 Jan 12 17:30 company.MYD
-rw-rw-r-- 1 davida tcx 96482304 Jan 12 18:35 company.MYI
```

[myisamchk -d](#) の出力例

```
MyISAM file:  company.MYI
Record format: Fixed length
Data records: 1403698 Deleted blocks: 0
Recordlength: 226
```

```
table description:
Key Start Len Index Type
1 2 8 unique double
2 15 10 multip. text packed stripped
3 219 8 multip. double
4 63 10 multip. text packed stripped
5 167 2 multip. unsigned short
6 177 4 multip. unsigned long
7 155 4 multip. text
8 138 4 multip. unsigned long
9 177 4 multip. unsigned long
```

193 1 text

myisamchk -d -v の出力例

```
MyISAM file:      company
Record format:    Fixed length
File-version:     1
Creation time:    1999-10-30 12:12:51
Recover time:    1999-10-31 19:13:01
Status:          checked
Data records:     1403698 Deleted blocks:      0
Datafile parts:   1403698 Deleted data:         0
Datafile pointer (bytes): 3 Keyfile pointer (bytes): 3
Max datafile length: 3791650815 Max keyfile length: 4294967294
Recordlength:     226
```

```
table description:
Key Start Len Index Type      Rec/key Root Blocksize
1  2   8  unique double        1 15845376  1024
2 15  10  multip. text packed stripped 2 25062400  1024
3 219  8  multip. double       73 40907776  1024
4  63  10  multip. text packed stripped 5 48097280  1024
5 167  2  multip. unsigned short 4840 55200768  1024
6 177  4  multip. unsigned long   1346 65145856  1024
7 155  4  multip. text          4995 75090944  1024
8 138  4  multip. unsigned long    87 85036032  1024
9 177  4  multip. unsigned long   178 96481280  1024
193 1   text
```

myisamchk -eis の出力例

```
Checking MyISAM file: company
Key: 1: Keyblocks used: 97% Packed: 0% Max levels: 4
Key: 2: Keyblocks used: 98% Packed: 50% Max levels: 4
Key: 3: Keyblocks used: 97% Packed: 0% Max levels: 4
Key: 4: Keyblocks used: 99% Packed: 60% Max levels: 3
Key: 5: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 6: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 7: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 8: Keyblocks used: 99% Packed: 0% Max levels: 3
Key: 9: Keyblocks used: 98% Packed: 0% Max levels: 4
Total: Keyblocks used: 98% Packed: 17%

Records:      1403698 M.recordlength: 226
Packed:        0%
Recordspace used: 100% Empty space: 0%
Blocks/Record: 1.00
Record blocks: 1403698 Delete blocks: 0
Recorddata:   317235748 Deleted data: 0
Lost space:    0 Linkdata: 0

User time 1626.51, System time 232.36
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 627, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 639, Involuntary context switches 28966
```

myisamchk -eiv の出力例

```
Checking MyISAM file: company
Data records: 1403698 Deleted blocks: 0
- check file-size
- check delete-chain
block_size 1024:
index 1:
index 2:
index 3:
index 4:
index 5:
index 6:
index 7:
index 8:
index 9:
No recordlinks
```

```
- check index reference
- check data record references index: 1
Key: 1: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 2
Key: 2: Keyblocks used: 98% Packed: 50% Max levels: 4
- check data record references index: 3
Key: 3: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 4
Key: 4: Keyblocks used: 99% Packed: 60% Max levels: 3
- check data record references index: 5
Key: 5: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 6
Key: 6: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 7
Key: 7: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 8
Key: 8: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 9
Key: 9: Keyblocks used: 98% Packed: 0% Max levels: 4
Total: Keyblocks used: 9% Packed: 17%

- check records and index references*** LOTS OF ROW NUMBERS DELETED ***

Records:      1403698 M.recordlength: 226 Packed:      0%
Recordspace used: 100% Empty space: 0% Blocks/Record: 1.00
Record blocks: 1403698 Delete blocks: 0
Recorddata: 317235748 Deleted data: 0
Lost space:      0 Linkdata: 0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, »
Involuntary context switches 122798
```

次の説明において、[myisamchk](#) が生成する情報のタイプについて説明します。「Keyfile」とはインデックス ファイルのことです。「Record」と「row」はシノニムです。

- [MyISAM file](#)

[MyISAM](#) (インデックス) ファイルの名前

- [File-version](#)

[MyISAM](#) フォーマットのバージョン。現在は常に 2。

- [Creation time](#)

データ ファイルの作成日時。

- [Recover time](#)

インデックスまたはデータ ファイルを前回再構築した日時。

- [Data records](#)

テーブル内のレコード数。

- [Deleted blocks](#)

予約済み領域 (リザーブ) を占有している削除済み (デリート) のブロック数。テーブルの最適化には、この領域を最小にする。[テーブルの最適化](#) を参照のこと。

- [Datafile parts](#)

動的フォーマットに対して、存在するデータ ブロック数。断片化レコードがない最適化テーブルに対する [Data records](#) と同じ。

- [Deleted data](#)

領域を解放していない削除済みデータのバイト数。テーブルの最適化には、この領域を最小にする。[テーブルの最適化](#)を参照のこと。

- [Datafile pointer](#)

データ ファイル ポインタのサイズ (バイト単位)。2、3、4、5 バイトのどれか。通常は 2 バイトで足りるが、現在のところ、MySQL で制御することはできない。固定テーブルでは、レコード アドレスのこと。動的テーブルでは、バイト アドレスのこと。

- [Keyfile pointer](#)

インデックス ファイル ポインタのサイズ (バイト単位)。1、2、3 バイトのどれか。通常のテーブルは 2 バイトで足りるが、MySQL で自動的に計算する。常にブロック アドレスのこと。

- [Max datafile length](#)

テーブルのデータ ファイル (.MYD ファイル) の最大長 (バイト単位)。

- [Max keyfile length](#)

テーブルのインデックス ファイル (.MYI ファイル) の最大長 (バイト単位)。

- [Recordlength](#)

それぞれのレコードで使用する領域サイズ (バイト単位)。

- [Record format](#)

テーブル レコードの格納形式。上記の例では [Fixed length](#) を使用。他に、[Compressed](#) および [Packed](#) がある。

- [table description](#)

テーブル内のすべてのキーの一覧。[myisamchk](#) コマンドで、それぞれのキーの低レベル情報を表示する。内容は次の通り。

- [Key](#)

キー番号。

- [Start](#)

インデックス部が始まるレコード内の位置。

- [Len](#)

インデックス部の長さ。パック数値の場合、これは常にそのカラムの全長となる。文字列の場合、文字列カラムのプリフィックスをインデックスにできるため、インデックス化したカラムの全長よりも短くなることがある。

- [Index](#)

[unique](#) または [multip.](#) (複数)。このインデックスで値の重複が認められているか (在るか) どうかを示す。

- [Type](#)

インデックス部のデータ型。[packed](#)、[stripped](#)、または [empty](#) のいずれかの [ISAM](#) データ型。

- [Root](#)

ルート インデックス ブロックのアドレス。

- [Blocksize](#)

それぞれのインデックス ブロックのサイズ。デフォルトでは 1024 であるが、MySQL をソースから組む場合、コンパイル時に変更可能。

- [Rec/key](#)

オブティマイザで使用する統計値。このインデックスのキー値ごとのレコード数を示す。ユニーク キーの値は常に 1。これは、[myisamchk -a](#) で、テーブルをロード (または大きく変更) すると更新する。更新しない場合は、デフォルト値の 30 のまま。

(1 番目と 2 番目の) 出力例示のテーブルに、[table description](#) の 9 番目のキーが 2 つある。これは、2 パートを持つマルチ パート キーであることを示す。

- [Keyblocks used](#)

使用しているキー ブロックのパーセント。例で使用しているテーブルは [myisamchk](#) で再構成したばかりであるため、値が非常に高い (理論的 maximum に非常に近い)。

- [Packed](#)

MySQL がキー間で共通するサフィックスの部分をパックした割合。これは、[CHAR](#) と [VARCHAR](#) のカラム キーにのみ使用可能。名前のような長い文字列では、MySQL がパックして使用領域を大きく減らす。たとえば、3 番目の出力例示の、4 番目のキーは、10 文字長 (100%) であるのに対して、領域を 60 % パックした (6割減) という意味。

- [Max levels](#)

このキーの B-tree の深さ。長いキーがある大きなテーブルでは、値が高くなる。

- [Records](#)

テーブル内のレコード数。

- [M.recordlength](#)

レコードの平均の長さ。固定長レコードのテーブルでは、これは実際のレコード長となる。

- [Packed](#)

MySQL が節約したパーセント。[Packed](#) 値 (%) は、MySQL が文字列の最後 (suffix) を削除してできたスペース。

- [Recordspace used](#)

使用してデータ ファイルのパーセント。

- [Empty space](#)

使用していないデータ ファイルのパーセント。

- [Blocks/Record](#)

レコードごとの平均ブロック数 (断片化レコードを構成するリンク数)。これは、固定形式テーブルでは常に 1.0。この値は可能な限り、1.0 に近くしておく。大きくなりすぎた場合は、[myisamchk](#) で再編成する。[テーブルの最適化](#) を参照のこと。

- [Recordblocks](#)

使用しているブロック (リンク) 数。固定形式では、レコード数と同じになる。

- [Deleteblocks](#)

削除したブロック (リンク) 数。

- [Recorddata](#)

使用しているデータ ファイルのバイト数。

- [Deleted data](#)

削除した (使用していない) データ ファイルのバイト数。

- [Lost space](#)

失った領域の合計バイト数。レコード長さを短くして更新した場合の、短くなった領域分。

- [Linkdata](#)

ポインタが使用しているストレージ量の合計 (これを [Linkdata](#) 値 と呼ぶ)。動的テーブルの場合、レコード断片をポインタでリンクしている (それぞれ 4 から 7 バイト)。

テーブルを [myisampack](#) で圧縮している場合は、[myisamchk -d](#) を実行すると、それぞれのテーブル カラムに関する追加情報を出力します。この情報の詳細は [myisampack — 圧縮された、読み取り専用MyISAM テーブルを作成する](#) を参照してください。

テーブル保守計画

問題が発生する前に、テーブル チェックを定期的に行います (推奨)。[MyISAM](#) テーブルをチェックまたは修復するには、[CHECK TABLE](#) および [REPAIR TABLE](#) ステートメントを使用します。詳細は、[CHECK TABLE 構文](#) と [REPAIR TABLE 構文](#) を参照してください。

別の方法としては、[myisamchk](#) を使用してテーブル チェックを行います。保守を目的とする場合は、[myisamchk -s](#) を使用します。[-s \(--silent\)](#) の短縮形) を使用すると、サイレントモードで [myisamchk](#) を実行でき、エラー発生部分のメッセージを出力します。

[MyISAM](#) のテーブル チェックには自動化をお勧めします。自動チェックを有効にしておくと、「予期していないテーブル クラッシュ」などでマシンが更新中であるにもかかわらず、再起動した場合などに、それぞれのテーブルをチェックする手間を省いて、影響があったテーブルを探すことができます。[MyISAM](#) テーブルの自動化設定を行うには、サーバを [--myisam-recover](#) オプションで起動してください。詳細は、[コマンド オプション](#) を参照してください。

テーブル チェックは日常的に行うことをお勧めします。MySQL AB では、一週間に一度、重要なテーブルに対して、[cron](#) コマンドを使用して確認作業を行っています。たとえば、[crontab](#) ファイルでは、次のようにしています。

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

この出力には、クラッシュしたテーブルの情報が出力するため、必要に応じて、確認や修復の作業ができます。

実際に、MySQL AB ではここ数年間、ハードウェアの故障以外のテーブル クラッシュは発生していないため、一週間に一度という頻度がその有効性を示しています。

MySQL AB 自体では、この方法でテーブル チェックを行っています。実際、ユーザがこの作業を行うときは、MySQL を完全に信用できると確認できるまでは、[myisamchk -s](#) を 1 日 1 度、行うことをお勧めします。

[VARCHAR](#)、[BLOB](#)、[TEXT](#) などのテーブルで、[MyISAM](#) テーブルを動的レコードで更新していたり、テーブルに削除できるレコードが沢山あっても、時々そのテーブルでデフラグやリクレーン (返還) を行なっていれば、MySQL テーブルの保守は簡単に行なえます。これができるようにするには、対象となるテーブルに [OPTIMIZE TABLE](#) を使用します。または、しばらくの間、[mysqld](#) サーバを停止することが可能な場合は、そのサーバを止めて、場所をデータディレクトリに置き換えて、次のコマンドを使用します。

```
shell> myisamchk -r -s --sort-index --sort_buffer_size=16M /*/*.MYI
```

MySQL のローカライズと国際的使用

このセクションでは、様々なキャラクタ セットを使用したサーバの設定方法について説明します。サーバのタイムゾーンと接続ごとのタイムゾーン対応の設定方法についても説明します。

データおよびソート用キャラクタ セット

MySQL のデフォルトでは、米国と西ヨーロッパに適したキャラクタセットの [latin1](#) (ISO-8859-1)、そして、照合順序(コアレーション)はスウェーデン語およびフィンランド語に準拠している [latin1_swedish_ci](#) で、ソートを行います。

MySQL 標準バイナリは、[--with-extra-charsets=complex](#) でコンパイルしてます。そのため、標準プログラムで、[latin1](#) とすべてのマルチ バイト キャラクタ セットを処理できます。必要なキャラクタ セットは、キャラクタセットの定義ファイルからロードします。

キャラクタセットは、識別子、つまり名前に使用できる文字を決定します。そして、[SELECT](#) ステートメントの [ORDER BY](#) 節および [GROUP BY](#) 節で行うの文字列の照合順序 (コアレーション) も決定します。

キャラクタ セットの変更は、サーバ起動時に、[--character-set-server](#) オプションで行い、照合順序には、[--collation-server](#) オプションを使用します。照合順序は設定するキャラクタ セットと対応するように使用します。(SHOW [COLLATION](#) ステートメントでキャラクタ セットに対応する照合順序を確認します。[コマンド オプション](#) を参照してください。

利用可能なキャラクタセットは、[--with-charset=charset_name](#) オプションと [--with-extra-charsets=list-of-charsets | complex | all | none](#) オプションを [configure](#) しているかどうか、そして [SHAREDIR/charsets/Index](#) のキャラクタ セット設定ファイルに依存します。[典型的な configure オプション](#) を参照してください。

注意: MySQL の実行中にキャラクタ セットを変更すると、ソート順序が変わる可能性もあります。つまり、インデックスが正しい順序ではなくなる可能性があります。結果として、すべての [MyISAM](#) テーブルで [myisamchk -r -q --set-collation=collation_name](#) の実行が必要になります。

クライアントを MySQL サーバに接続すると、サーバがデフォルトのキャラクタ セットをクライアントに送ります。このクライアントは、サーバと接続するためにそのキャラクタ セットに切り替えます。

SQL クエリ of 文字列をエスケープする場合は、[mysql_real_escape_string\(\)](#) を使用します。[mysql_real_escape_string\(\)](#) は旧 [mysql_escape_string\(\)](#) 関数と同じですが、最初のパラメータとして [MYSQL](#) 接続を扱うときに、適切なキャラクタ セットをアカウントにエスケープする場合に使用します。

サーバをインストールしたときのパス以外のパスで、クライアントをコンパイルしていて、MySQL をコンフィギュしたユーザがすべてのキャラクタ セットを MySQL のバイナリに組み込んでいない場合があります。そのときは、サーバがクライアントとは異なるキャラクタセットで実行していることを、クライアントに知らせ、必要なキャラクタセットがある場所を、クライアントに指定する必要があります。それには、[--character-sets-dir](#) オプションを使用して、MySQL の動的キャラクタ セットを保存しているディレクトリのパスを示します。たとえば、次のようにオプション ファイルに示します。

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

または、クライアントに特定のキャラクタ セットの使用を強制することも可能です。

```
[client]
default-character-set=charset_name
```

しかし、このように指定することは、あまりありません。

ドイツ語のキャラクタ セット

MySQL 5.1 では、キャラクタ セットと照合順序は別々に指定します。つまり、ドイツ語照合でソートする場合などに、[latin1](#) というキャラクタ セットを選択して、照合順序には、[latin1_german1_ci](#) または [latin1_german2_ci](#) を使用してソートする、ということです。よって、ドイツ語でソートするには、[latin1_german1_ci](#) を照合順序としてサーバを起動し、[--character-set-server=latin1](#) および [--collation-server=latin1_german1_ci](#) のオプションを使用した設定になります。

照合順序の相違に関する情報は、[西ヨーロッパのキャラクタセット](#) を参照してください。

英語以外のエラーメッセージ

[mysqld](#) では、次の言語でエラーメッセージを出力できます。チヨコ語、デンマーク語、オランダ語、英語 (デフォルト)、エストニア語、フランス語、ドイツ語、ギリシャ語、ハンガリー語、イタリア語、日本語、韓国語、ノルウェー語、新ノルウェー語、ポーランド語、ポルトガル語、ルーマニア語、ロシア語、スロバキア語、スペイン語、スウェーデン語。

特定の言語でエラー メッセージを表示する [mysqld](#) を起動するには、[--language](#) または [-L](#) オプションを使用します。オプション値に使用する言語名を指定するか、またはエラー メッセージ ファイルで指定します。次の通りです。

```
shell> mysqld --language=swedish
```

または

```
shell> mysqld --language=/usr/local/share/swedish
```

注意: 言語名はすべて小文字で指定します。

言語ファイルは、MySQL ベース ディレクトリの `share/LANGUAGE` (デフォルト)にあります。

サーバで生成するエラー メッセージの内容を変更も可能です。詳細は MySQL Internals のマニュアル (<http://dev.mysql.com/doc/>) を参照してください。MySQL を新しいバージョンにアップグレードするときには、それまで使用していたエラー メッセージに関わる変更も合わせて行ってください。

新しいキャラクタ セットの追加

このセクションでは、MySQL へ新しいキャラクタ セットを追加する方法について説明します。この作業には、MySQL ソース配布が必要になります。キャラクタ セットがシンプルな場合と、コンプレックスな場合で、選択する手順が異なります。

- キャラクタ セットが、ソートのために特殊文字照合ルーチンを必要とせず、マルチ バイト文字のサポートも必要なければ、シンプルと判断します。
- どちらかを必要とする場合は、コンプレックスと判断します。

たとえば、`latin1` と `danish` はシンプルですが、`big5` や `czech` はコンプレックスです。

次の手順では、キャラクタ セットの名前を `MYSET` と前提とします。

シンプルなキャラクタセットの場合

1. `MYSET` を `sql/share/charsets/Index` ファイルの最後に追加し、これに一意の番号を割り当てる。
2. `sql/share/charsets/MYSET.conf` という ファイルを作成する (`sql/share/charsets/latin1.conf` のコピーをベースとして使用可能)。

ファイルの構文は、非常にシンプル

- コメントは `#` 文字で始まり、行の最後まで続く。
- 単語は任意の数のスペースで区切る。
- キャラクタ セットを定義するときは、すべての単語を 16 進数値とする。
- `ctype` 配列は、最初の 257 語を占める。`to_lower[]`、`to_upper[]`、`sort_order[]` などの配列はそれぞれ、その後の 256 語を占める。

配列に関する詳細は [キャラクタ定義配列](#) を参照してください。

3. キャラクタ セット名を、`configure.in` の `CHARSETS_AVAILABLE` リストおよび `COMPILED_CHARSETS` リストに追加する。
4. 再設定してから、再コンパイルして、テストする。

コンプレックスなキャラクタ セットの場合

1. MySQL ソース配布に `strings/ctype-MYSET.c` というファイルを作成する。
2. `MYSET` を `sql/share/charsets/Index` ファイルの最後に追加し、これに一意の番号を割り当てる。
3. `strings/ctype-big5.c` など、既存の `ctype-*.c` ファイルの 1 つを見て、何の定義が必要か調べる。注意: ファイル内の配列名には、`ctype_MYSET`、`to_lower_MYSET` などが必要。これは、シンプルなキャラクタ セットの配列に対応する。キャラクタ定義配列に関する詳細は [キャラクタ定義配列](#) を参照してください。
4. ファイルの先頭付近に、次のようなコメントを置く。

```
/*
 * This comment is parsed by configure to create ctype.c,
 * so don't change it unless you know what you are doing.
 *
 * .configure. number_MYSET=MYNUMBER
 * .configure. strxfrm_multiply_MYSET=N
 * .configure. mbmaxlen_MYSET=N
```

```
*/
```

`configure` プログラムはこのコメントを使用して、自動的にキャラクタ セットを MySQL ライブラリに組み込む。

必要に応じて、`strxfrm_multiply` のライン (文字列照合関数)、および `mbmaxlen` のライン (マルチバイト文字セット関数) を組み込む。これについては、後続のセクションで説明。

5. そして、次の関数を作成する。

- `my_strncoll_MYSET()`
- `my_strcoll_MYSET()`
- `my_strxfrm_MYSET()`
- `my_like_range_MYSET()`

配列照合に関する詳細は [文字列照合サポート](#) を参照してください。

6. キャラクタ セット名を、`configure.in` の `CHARSETS_AVAILABLE` リストおよび `COMPILED_CHARSETS` リストに追加する。

7. 再設定してから、再コンパイルして、テストする。

より詳細な手順は、`sql/share/charsets/README` ファイルにあります。

MySQL 配布バージョンへのキャラクタ セットの組み込みを希望する場合には、MySQL [internals](#) の メーリングリストにパッチをメールしてください。メーリング リストに関する詳細は [MySQL メーリング リスト](#) を参照してください。

キャラクタ定義配列

`to_lower[]` と `to_upper[]` は、キャラクタセットの各メンバに対応する、大文字と小文字を格納するシンプルな配列です。次に例を示します。

```
to_lower['A'] should contain 'a'
to_upper['a'] should contain 'A'
```

`sort_order[]` は、文字をどのような照合順序でソートするかを示すマップです。多くの場合 (すべてのキャラクタセットについてではありませんが)、これは `to_upper[]` と同じです (ソートで大文字と小文字は区別しません)。MySQL は `sort_order[]` の値に基づいて文字をソートします。複雑なソート ルールに関しては、[文字列照合サポート](#) の文字列照合に関する記述を参照してください。

`ctype[]` はビット値の配列で、1 文字が 1 要素です。注意: `to_lower[]`、`to_upper[]`、`sort_order[]` などは文字値でインデックス化しますが、`ctype[]` の場合は文字値 + 1 でインデックス化します。これは、EOF を処理することが必要だったときの古いレガシー (古い技術) です。

`m_ctype.h` に、次に示すビットマスク定義があります。

```
#define _U 01 /* Uppercase */
#define _L 02 /* Lowercase */
#define _N 04 /* Numeral (digit) */
#define _S 010 /* Spacing character */
#define _P 020 /* Punctuation */
#define _C 040 /* Control character */
#define _B 0100 /* Blank */
#define _X 0200 /* hexadecimal digit */
```

それぞれの文字の `ctype[]` エントリは、文字を指定するビットマスク値の組み合わせになる必要があります。たとえば、`'A'` は大文字 (`_U`) であると同時に 16 進数 (`_X`) でもあるため、`ctype['A'+1]` には次の値が含まれます。

```
U + _X = 01 + 0200 = 0201
```

文字列照合サポート

使用する言語のソート ルールが、シンプルな `sort_order[]` テーブルで処理するには複雑すぎる場合、文字列照合を行う必要があります。

現在のところ、これに関する最良のドキュメントは、すでに実装しているキャラクタ セットです。たとえば、[big5](#)、[czech](#)、[gbk](#)、[sjis](#)、[tis160](#) などのキャラクタ セットを参照してください。

ファイル先頭の特典コメントで、[strxfrm_multiply_MYSET=N](#) 値を指定する必要があります。[N](#) には、[my_strxfrm_MYSET](#) で文字列が大きくなる最大比率 (正の整数) を設定してください。

マルチ バイト文字サポート

マルチ バイト文字を含む新しいキャラクタ セットのサポートを追加する場合、マルチ バイト文字関数を使用する必要があります。

現在のところ、これに関する最良のドキュメントは、すでに実装しているキャラクタ セットです。たとえば、[euc_kr](#)、[gb2312](#)、[gbk](#)、[sjis](#)、[ujis](#) などのキャラクタセットを参照してください。これらは、[strings](#) ディレクトリ内の [ctype-charset_name.c](#) ファイルに実装しています。

ソース ファイル先頭の特典コメントで [mbmaxlen_MYSET=N](#) 値を指定する必要があります。[N](#) には、キャラクタ セット内の最大文字のバイト数を設定してください。

キャラクタ セットに関する問題

使用しているバイナリにコンパイルしていないキャラクタ セットを使用すると、いくつかの問題が発生する可能性があります。

- プログラムが認識しているパスが、キャラクタセットを保存しているものとは異なるパスになる。(デフォルトは [/usr/local/mysql/share/mysql/charsets](#))。これは、該当プログラムで [--character-sets-dir](#) オプションを使用すると解決する。
- キャラクタ セットが動的にロードできないマルチ バイト キャラクタ セットである場合、そのキャラクタ セットをサポートするようにプログラムを再コンパイルする必要がある。
- キャラクタ セットは動的なキャラクタ セットであるが、その設定ファイルがない。この場合、新しい MySQL 配布から、そのキャラクタ セットの設定ファイルをインストールする必要がある。
- [Index](#) ファイルにキャラクタセットの名前がない場合に、プログラムが次のようなエラー メッセージを表示する。

```
ERROR 1105: File '/usr/local/share/mysql/charsets/?_conf'
not found (Errcode: 2)
```

この場合、新しい [Index](#) ファイルを入手するか、または手動でキャラクタ セット名を追加する。

[MyISAM](#) テーブルに、[myisamchk -dvv tbl_name](#) を使用すると、テーブルのキャラクタ セットの名前と番号を確認できます。

MySQL サーバのタイム ゾーン サポート

MySQL サーバにはいくつかのタイム ゾーンがあります。

- システムのタイム ゾーン。サーバ起動時に、ホスト マシンのタイム ゾーンを使用して、[system_time_zone](#) システム変数で設定する。それ以降にこの値が変更することはない。

起動時の MySQL Server のシステム タイム ゾーンは、[mysqld_safe](#) で、[--timezone=timezone_name](#) を使用する。または、[mysqld](#) を起動する前に、[TZ](#) 環境変数で設定する。[--timezone](#) と [TZ](#) の許容値は、システムに依存するため、OS のドキュメントを参照のこと。

- サーバのカレント タイム ゾーン。[time_zone](#) のグローバル システム変数はサーバ稼働中のタイム ゾーンを示す。[time_zone](#) の初期値は、['SYSTEM'](#) で、サーバとシステムのタイム ゾーンが同じであることを示す。

サーバのタイム ゾーンの初期グローバル値は、コマンドラインで [--default-time-zone=timezone](#) オプションで起動するとき明示的に指定する。または、オプション ファイルに次のラインを使用する。

```
default-time-zone='timezone'
```

[SUPER](#) 権限がある場合は、ラインタイムで次のステートメントを使用して、サーバのタイム ゾーンにグローバル値を設定する。

```
mysql> SET GLOBAL time_zone = timezone;
```

- 接続毎のタイムゾーン。接続するそれぞれのクライアントには、`time_zone` 変数で与えられた、それぞれのタイムゾーン セットアップがある。最初は、セッション変数の値が `time_zone` 変数の値になるが、クライアントは次のステートメントを使用して、それぞれのタイムゾーンを指定できる。

```
mysql> SET time_zone = timezone;
```

カレント セッションのタイムゾーン セットアップは、時間との関わりが深いディスプレイやストレージのタイム値に影響します。これには、`NOW()` または `CURTIME()` などの関数で表示する値や、`TIMESTAMP` カラムに保存し、そこから読み出す値も含まれます。`TIMESTAMP` カラムの値は、ストレージでは現在のタイムゾーンから UTC へ、読み出しでは UTC からカレントのタイムゾーンに変換します。カレントのタイムゾーン セットアップは、`DATE`, `TIME`, or `DATETIME` などのカラム値、または `UTC_TIMESTAMP()` 関数などによって表示する値には影響しません。

グローバルおよびクライアント指定のタイムゾーンのカレント値は、次のように読み出します。

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

`timezone` 値には、いくつかの形式があります。次に示す形式では文字区別はありません。

- `'SYSTEM'` 値は、システム タイムゾーンと同じ値を示す。
- UTC オフセットを示す文字列の値。たとえば、`'+10:00'` または `'-6:00'` など。
- 指定したタイムゾーンの値。たとえば、`'Europe/Helsinki'`、`'US/Eastern'`、`'MET'` など。指定したタイムゾーンは、タイムゾーンの情報テーブルが `mysql` データベースにあるときにだけ使用できる。

MySQL のインストール手順でタイムゾーン テーブルを `mysql` データベースに作成しますが、そのときは、ロードまではしません。その前に、テーブルを作成する必要があります。MySQL 4.1.3 以降にアップグレードする場合は、`mysql` データベースを更新すると、そのテーブルを作成できます。アップグレードに関する手順は、[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照してください。次に、テーブル作成してから、タイムゾーン テーブルをロードするまでの手順 (手動) を示します。

注記

タイムゾーンの情報をロードは、情報が変更する場合に応じて、変更します。たとえば、夏時間を導入している米国、メキシコ、カナダでは 2007 年に導入ルールが変更しました。そのような変更がある場合、古いルールを使用しているアプリケーションとの誤差がでるため、MySQL サーバの時間で使用している情報を維持するために、タイムゾーン テーブルをリロードする必要があります。このセクションで後述のノートを参照してください。

システムに独自の `zoneinfo` データベース (タイムゾーンに関するファイル セット) がある場合、[mysql_tzinfo_to_sql](#) プログラムを使用して、タイムゾーン テーブルの充てんを行います。Linux、FreeBSD、Sun Solaris、Mac OS X などのシステムでは、通常、タイムゾーンに関するファイルは、`/usr/share/zoneinfo` ディレクトリにあります。システムに `zoneinfo` データベースがない場合、このセクションで後述しているパッケージをダウンロードします。

`mysql_tzinfo_to_sql` プログラムをタイムゾーン テーブルのリロードに使用します。コマンドラインで、`zoneinfo` ディレクトリへのパスを `mysql_tzinfo_to_sql` へ渡し、`mysql` プログラムに出力します。たとえば次のようにします。

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` はシステム内のタイムゾーン ファイルを読み、そこから SQL ステートメントを生成します。`mysql` でそのステートメントを処理して、タイムゾーン テーブルにロードします。

`mysql_tzinfo_to_sql` コマンドを使用して、単一のタイムゾーン ファイルをロードしたり、うるう秒 (リープ セカンド) の情報を生成することも可能です。

- `tz_name` というタイムゾーンに対応する単一のタイムゾーン ファイル、`tz_file` をロードするには、次のように `mysql_tzinfo_to_sql` を呼び出す。

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

注意：この方法で、サーバが必要とする、それぞれの指定ゾーンのファイルを別々のコマンドでロードしてください。

- うるう秒のカウントが必要な場合は、次のように、うるう秒の情報を初期化します。ここで `tz_file` をタイムゾーンファイルの名前とします。

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

Windows または HP-UX など、zoneinfo データベースを使用しないシステムの場合は、パッケージのタイムゾーンテーブルを使用します。これは、MySQL Developer Zone からダウンロードできます。

<http://dev.mysql.com/downloads/timezones.html>

このタイムゾーンパッケージには、`MyISAM` のタイムゾーンテーブル用に、`.frm`、`.MYD`、`.MYI` などのファイルが入っています。このテーブルを `mysql` データベースの一部にする必要があるため、そのファイルをMySQLサーバのデータディレクトリの `mysql` サブディレクトリに入れてください。そのときは、サーバを停止してから作業を行い、再起動してください。

警告：システムに zoneinfo データベースがある場合は、ダウンロードパッケージを使用してはいけません。MySQL とシステムアプリケーション間での時間に関わる処理に誤差が生じる原因になります。代わりに、`mysql_tzinfo_to_sql` ユーティリティを使用してください。

レプリケーションを行うときのタイムゾーンセッティングに関する情報は、[レプリケーション機能と既知問題](#) を参照してください。

タイムゾーン変更に対応

タイムゾーンのルールに変更があるときは、古いルールを使用しているアプリケーションでも調節が必要になります。タイムゾーンのずれを回避するために、システムのタイムゾーン情報がカレント(最新の時間)であることを確認してください。MySQL では、タイムゾーンをカレントにする必要がある要素が 2 つあります。

- MySQL サーバでタイムゾーンを `SYSTEM` で設定している場合は、オペレーティングシステムの時間が、MySQL で使用する時間の値に影響するため、オペレーティングシステムが最新(アップデート)のタイムゾーンを使用していることを確認する必要があります。大抵のオペレーティングシステムでは、アップデートやサービスバックで時間変更に対応している。時間変更に関するアップデートなどオペレーティングシステムのベンダーウェブサイトを確認する。インターネットのパブリックタイムでサーバを稼働させている場合、システムが自動的に調整を行う。
- MySQL で特定のタイムゾーンを使用する場合、`mysql` データベースのタイムゾーンテーブルのアップデートを行う。システムに独自の zoneinfo データベースがある場合は、zoneinfo データベースをアップデートする度に、このセクションで前述した手順に従って、MySQL のタイムゾーンテーブルをリロードする。zoneinfo データベースがない場合は、MySQL Developer Zone のアップデート情報に従う。アップデートがある場合は、ダウンロードして、カレントのタイムゾーンテーブルと置き換える。

MySQL サーバのローケル サポート

MySQL 5.1.12 から、`lc_time_names` 変数で示すローケルで、日時や略記の表示に使用する言語を制御します。この変数は、`DATE_FORMAT()`、`DAYNAME()`、`MONTHNAME()` 関数での出力に影響します。

ローケル名は、`'ja_JP'` や `'pt_BR'` などの POSIX 規格の値です。システムのローケルセッティングに関わらず、`'en_US'` をデフォルト値としますが、クライアントで `lc_time_names` 値を調べたり、その値をセットすることは可能です。次にその例を示します。

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| en_US          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| Friday                | January                  |
+-----+-----+
```

```

1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| Friday Fri January Jan                  |
+-----+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| es_MX          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| viernes              | enero                  |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| viernes vie enero ene                  |
+-----+
1 row in set (0.00 sec)

```

関数で影響を受ける月日の名前は、`utf8` から `character_set_connection` システム変数で指すキャラクタ セットに変換します。

`lc_time_names` に設定できるローケル値は次の通りです。

<code>ar_AE</code> : アラビア語 - アラブ首長国連邦	<code>ar_BH</code> : アラビア語 - バーレーン
<code>ar_DZ</code> : アラビア語 - アルジェリア	<code>ar_EG</code> : アラビア語 - エジプト
<code>ar_IN</code> : アラビア語 - イラン	<code>ar_IQ</code> : アラビア語 - イラク
<code>ar_JO</code> : アラビア語 - ヨルダン	<code>ar_KW</code> : アラビア語 - クウェート
<code>ar_LB</code> : アラビア語 - レバノン	<code>ar_LY</code> : アラビア語 - リビア
<code>ar_MA</code> : アラビア語 - モロッコ	<code>ar_OM</code> : アラビア語 - オマーン
<code>ar_QA</code> : アラビア語 - カタール	<code>ar_SA</code> : アラビア語 - サウジ アラビア
<code>ar_SD</code> : アラビア語 - スーダン	<code>ar_SY</code> : アラビア語 - シリア
<code>ar_TN</code> : アラビア語 - チュニジア	<code>ar_YE</code> : アラビア語 - イエメン
<code>be_BY</code> : ベラルーシ語 - ベラルーシ	<code>bg_BG</code> : ブルガリア語 - ブルガリア
<code>ca_ES</code> : カタロニア語 - カタロニア (スペイン、アンドラ)	<code>cs_CZ</code> : チェコ語 - チェコ共和国
<code>da_DK</code> : デンマーク語 - デンマーク	<code>de_AT</code> : ドイツ語 - オーストリア
<code>de_BE</code> : ドイツ語 - ベルギー	<code>de_CH</code> : ドイツ語 - スイス
<code>de_DE</code> : ドイツ語 - ドイツ	<code>de_BE</code> : ドイツ語 - ルクセンブルグ
<code>EE</code> : エストニア語 - エストニア	<code>en_AU</code> : 英語 - オーストラリア
<code>en_CA</code> : 英語 - カナダ	<code>en_GB</code> : 英語 - 英国 (UK)
<code>en_IN</code> : 英語 - インド	<code>en_NZ</code> : 英語 - ニュージーランド
<code>en_PH</code> : 英語 - フィリピン	<code>en_US</code> : 英語 - アメリカ

en_ZA : 英語 - 南アフリカ	en_ZW : 英語 - ジンバブエ
es_AR : スペイン語 - アルゼンチン	es_BO : スペイン語 - ボリビア
es_CL : スペイン語 - チリ	es_CO : スペイン語 - コロンビア
es_CR : スペイン語 - コスタリカ	es_DO : スペイン語 - ドミニカ共和国
es_EC : スペイン語 - エクアドル	es_ES : スペイン語 - スペイン
es_GT : スペイン語 - グアテマラ	es_HN : スペイン語 - ホンジュラス
es_MX : スペイン語 - メキシコ	es_NI : スペイン語 - ニカラグア
es_PA : スペイン語 - パナマ	es_PE : スペイン語 - ペルー
es_PR : スペイン語 - プエルトリコ	es_PY : スペイン語 - パラグアイ
es_SV : スペイン語 - エルサルバドル	en_US : スペイン語 - アメリカ
es_UY : スペイン語 - ウルグアイ	es_VE : スペイン語 - ベネズエラ
eu_ES : バスク語 - バスク (スペイン)	fi_FI : フィンランド語 - フィンランド
fo_FO : フェロー語 - フェロー諸島	fr_BE : フランス語 - ベルギー
fr_CA : フランス語 - カナダ	fr_CH : フランス語 - スイス
fr_FR : フランス語 - フランス	fr_LU : フランス語 - ルクセンブルグ
gl_ES : ガリシア語 - ガリシア (スペイン)	gu_IN : グジャラート語 - インド
he_IL : ヘブライ語 - イスラエル	hi_IN : ヒンディー語 - インド
hr_HR : クロアチア語 - クロアチア	hu_HU : ハンガリー語 - ハンガリー
id_ID : インドネシア語 - インドネシア	is_IS : アイスランド語 - アイスランド
it_CH : イタリア語 - スイス	it_CH : イタリア語 - イタリア
ja_JP : 日本語 - 日本	ko_KR : 韓国語 - 韓国
lt_LT : リトアニア語 - リトアニア	lv_LV : ラトビア語 - ラトビア
mk_MK : マケドニア語 - マケドニア・旧ユーゴスラビア (FYROM)	mn_MN : モンゴル語 - モンゴル
ms_MY : マレー語 - マレーシア	nb_NO : ノルウェー語 - ノルウェー
nl_BE : オランダ語 - ベルギー	nl_NL : オランダ語 - オランダ
no_NO : ノルウェー語 - ノルウェー	pl_PL : ポーランド語 - ポーランド
pt_BR : ポルトガル語 - ブラジル	pt_PT : ポルトガル語 - ポルトガル
ro_RO : ルーマニア語 - ルーマニア	ru_RU : ロシア語 - ロシア
ru_UA : ロシア語 - ウクライナ	sk_SK : スロバキア語 - スロバキア
sl_SI : スロベニア語 - スロベニア	sq_AL : アルバニア語 - アルバニア
sr_YU : セルビア語 - ユーゴスラビア	sv_FI : スウェーデン語 - フィンランド
sv_SE : スウェーデン語 - スウェーデン	ta_IN : タミル語 - インド
te_IN : テルグ語 - インド	th_TH : タイ語 - タイ
tr_TR : トルコ語 - トルコ	uk_UA : ウクライナ語 - ウクライナ
ur_PK : ウルドゥー語 - パキスタン	vi_VN : ベトナム語 - ベトナム
zh_CN : 中国語 - 中国	zh_HK : 中国語 - 香港 SAR
zh_TW : 中国語 - 台湾	

現在のところ、[lc_time_names](#) には [STR_TO_DATE\(\)](#) または [GET_FORMAT\(\)](#) 関数との関係はありません。

MySQL サーバ ログ

MySQL には様々なログ ファイルがあり、[mysqld](#) 内での出来事を調べることができます。

ログ ファイル	説明
エラー ログ	<code>mysqld</code> の起動、実行、および停止で発生した問題。
一般クエリログ	クライアントとの接続と実行したクエリ。
バイナリ ログ	データ変更のステートメント。レプリケーションにも使用。
スロー クエリ ログ	<code>long_query_time</code> 秒より時間を要したクエリ、またはインデックスを使用しなかったクエリ。

デフォルトでは、`mysqld` データ ディレクトリにすべてのログ ファイルを作成します。`mysqld` を行使して、ログ ファイルの開閉、置換を行います。`FLUSH LOGS` ステートメント、または `mysqladmin flush-logs` あるいは `mysqladmin refresh` などのコマンドで、ログをフラッシュします。詳細は [FLUSH 構文](#) および [mysqladmin — MySQL サーバの管理を行うクライアント](#) を参照してください。

MySQL のレプリケーション機能を活用している場合、スレーブ レプリケーション サーバで、リレー ログというログ ファイルを保管しています。リレー ログおよびコンフィギュレーションに関しては、[5章レプリケーション](#) を参照してください。

MySQL 5.1.6 から、サーバで一般クエリとスロー クエリのエントリをログ テーブル、ログ ファイル (またはこの両方) に書き込むようになりました。詳細は [一般クエリとスロー クエリのログ出力先の選択](#) を参照してください。

MySQL 5.1.12 からは、一般クエリとスロー クエリ ログのランタイム制御に追加機能があります。ロギングを有効化、無効化、そしてログ ファイルの変更などが行えます。詳細は [一般クエリ ログ](#) および [スロー クエリ ログ](#) を参照してください。

一般クエリとスロー クエリのログ出力先の選択

MySQL 5.1.6 前は、サーバではログ ファイルが一般クエリとスロー ログ クエリのエントリとして機能しています (有効の場合)。MySQL 5.1.6 からは、サーバで、柔軟性があるログ出力先の制御ができます。従来通り、ログ エントリをログ ファイルに書き込みますが、`mysql` データベースの `general_log` そして `slow_log` のテーブルにもエントリを書き込むことができます。ロギングを有効にすると、テーブルの出力先を選択することができます。両方選択することも可能です。

ロギングを有効にすると、`--log-output` オプションでログ出力先を指定できます。このオプションは、次のように `--log-output=[value,...]` というシンタックスの使い方をします。

- 値を `--log-output` で指定すると、この値はカンマ区切りのリストになります。`TABLE` はログからテーブルへの出力、`FILE` はログからファイルへの出力。`NONE` はテーブルやファイルにログしないときに使用し、これはどのような指定子よりも優先になる。
- `--log-output` を値なしで使用する、または省略すると、その効果は `--log-output=TABLE` に同じ。つまり、テーブルの出力先の選択が行われる、ということ。これは、MySQL 5.1.6 前のファイルを使用したデフォルトの出力先とは異なる。
- `--log-output` オプションは、`log_output` グローバル システム変数の値を指す。これはランタイムで変更ができ、実行中のサーバのログ先を変更できる。

注記

MySQL 5.1.6 以上のインストールでは、ログ テーブルをシステム テーブルなどと一緒に作成します。MySQL を 5.1.6 より古いリリース バージョンから MySQL 5.1.6 以上にアップグレードするときは、アップグレードを行なった後に、ログ テーブルがあるかどうかを確認するために、システム テーブルのアップグレードも行ってください。

MySQL 5.1.6 より、デフォルトのログ先がファイルからテーブルに変更します。ロギングをログ ファイルで行う設定になっている場合は、その設定を保存するために、5.1.6 以上にアップグレードを行なった後、`--log-output=FILE` を使用します。

`--log=[file_name]` を使用すると、一般クエリ ログのログ先を選択式で変更して、ロギングができます。同様に、`--log-slow-queries=[file_name]` を使用すると、スロー クエリ ログも選択式でログ先を変更できます。どちらのオプションでも、`FILE` の出力先を指定しない限り、ファイル名は無視になります。

`--log` または `--log-slow-queries` を指定する場合は、サーバが関連ログ ファイルを開き、スタートアップ メッセージを書き込みますが、ファイルへのクエリのロギングは、[FILE](#) ログ先を選択するまで始まりません。

例：

- 一般クエリ エントリをログ テーブルとログ ファイルに書き込むには、`--log-output=TABLE,FILE` を使用して、両方の出力先を選択して、`--log` オプションで、一般クエリ ログを返す。
- 一般クエリとスロー クエリのログをログ テーブルにだけ書き込むには、`--log-output=TABLE` でログ先のテーブルを選択し、`--log` と `--log-slow-queries` で両方のログを有効にする。この場合、デフォルトのログ出力先が [TABLE](#) になっているため、`--log-output` オプションを省略することも可能。

テーブルでのログ出力には、次のような利点があります。

- ログ エントリが標準形式になる。ログ テーブルの現行のストラクチャを表示するには、次のステートメントを使用する。

```
SHOW CREATE TABLE mysql.general_log;
SHOW CREATE TABLE mysql.slow_log;
```

- ログ内容は、SQL ステートメントでアクセス可能。これは、特定のクライテリアを充たすエントリを選択するという、クエリの使い方をすることができる。たとえば、特定のクライアントに関連するログ内容を選択することが簡単になるということ。これは、クライアントからの問題があるクエリを識別するときに役立つ。
- サーバに接続し、クエリを出すクライアントからログにリモート アクセスできる。(これには、クライアントに適切なテーブル権限が必要。) サーバ ホストへのログイン、またはファイルシステムへの直接アクセスが不要になる。
- [TRUNCATE TABLE](#) を使用して、ログ エントリに有効期限をつけることができる。

デフォルトでは、ログ テーブルへのデータ書き込みは [CSV](#) ストレージ エンジンへ、カンマ区切り形式で行います。ログ テーブルにデータがある [CSV](#) ファイルへのアクセスができるユーザは、CSV を処理する表計算など別のプログラムへファイルを簡単にインポートできます。

MySQL 5.1.12 から、ログ テーブルは [MyISAM](#) ストレージ エンジン (メモリ) での使用に変更できます。使用中のログ テーブルを変更するために、[ALTER TABLE](#) を使用することはできません。その場合は、まずログを停止してください。ログ テーブルのエンジン (メモリ) には、[CSV](#) または [MyISAM](#) だけを使用してください。

ログ テーブルに [DROP TABLE](#) を使用することも同様に禁止です。使用中のログ テーブルをドロップすることはできません。まず、ログを停止してください。

ログを停止してから、ログ テーブルを変更するには、次の方法を取ります。

```
SET @old_log_state = @@global.slow_query_log;
SET GLOBAL slow_query_log = 'OFF';
ALTER TABLE mysql.slow_log ENGINE = MyISAM;
SET GLOBAL slow_query_log = @old_log_state;
```

MySQL 5.1.12 から、[FLUSH TABLES](#) ではログ テーブルを出力しません。ログ テーブルをフラッシュするには、[FLUSH LOGS](#) を使用してください。

MySQL 5.1.13 から、ログのローテーションを行うときなどに、ログ テーブルをアトミックに改名できます。それには、次の方法を取ります。

```
USE mysql;
CREATE TABLE IF NOT EXISTS general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

エラー ログ

エラーログファイルでは、[mysqld](#) の起動時刻と停止時刻、および実行中に発生したエラーに関する情報を記録しています。自動でチェックまたは修復が必要なテーブルを [mysqld](#) で見つけた場合には、エラー ログに警告メッセージが書き込まれます。

オペレーティングシステムによっては、エラー ログに [mysqld](#) が異常終了した場合のスタック トレースを記録しています。そのため、このトレースを [mysqld](#) が動かなくなった場合の確認に使用できます。スタック トレースに関しては [Using a Stack Trace](#) を参照してください。

`mysqld_safe` を使用して、`mysqld` を起動したときに、`mysqld` が異常終了する場合は、`mysqld_safe` がそれを認識し、`mysqld` を再起動する必要があることを、`restarted mysqld` メッセージとして、エラー ログに書き込みます。

`mysqld` を保存するエラー ログ ファイルは、`--log-error=[file_name]` オプションで指定できます。`file_name` を指定しない場合は、`mysqld` で、`host_name.err` という名前を使用して、データ ディレクトリに書き込みます。そして、`FLUSH LOGS` を実行するときに、エラー ログは `-old` というサフィックスでの改名になり、`mysqld` が新たな空ログ ファイルを作成します。(`--log-error` で指定しないと、名前の変更は起こりません。)

`--log-error` を指定しない場合、または Windows 環境である場合に、`--console` オプションを使用すると、エラーは `stderr` という標準のエラー出力で端末書き込みになります。

Windows では、`--console` の指定がない限り、エラー出力は `.err` ファイルへの書き込みになります。

`--log-warnings` オプション、または `log_warnings` システム変数を使用すると、エラー ログの警告ロギングを制御できます。値を 1 (デフォルト) にすると、有効化し、0 にすると無効化します。値を 1 より大きな値にすると、中断した接続についてもエラー ログを記録します。詳細は、[Communication Errors and Aborted Connections](#) を参照してください。

一般クエリ ログ

`mysqld` での出来事を記録しているのが、一般クエリ ログです。サーバはこのログに、クライアント接続や切断の情報を書き込み、そのときのクライアントからの SQL ステートメントも記録します。一般クエリ ログはクライアント側でのエラーを検討するときに、クライアントが `mysqld` に何を送ったことによって問題が発生したかを正確に知ることができます。

`mysqld` ではステートメントを到着順にクエリ ログに書き込みますが、実行した順番とは異なることがあります。このロギングの順番は、ステートメントを実行した後のクエリがロックがリリースされる前である場合には、バイナリログとは対照的です。また、クエリ ログにはすべてのステートメントが入る一方で、バイナリ ログにはデータだけを選択するステートメントは入りません。

MySQL 5.1.6 から、一般クエリ ログを有効化するには、`mysqld` を `--log=[file_name]` または `-l [file_name]` で起動するか、あるいは `--log-output` を使用してログ先を指定します ([一般クエリとスロー クエリのログ出力先の選択](#) を参照)。MySQL 5.1.6 前は、一般クエリ ログの出力先は常にファイルです。一般クエリ ログ ファイルを有効化するには、`--log=[file_name]` または `-l [file_name]` のオプションを使用します。

`--log` または `-l` に `file_name` 値を指定しない場合、デフォルト名は、`host_name.log` というデータ ディレクトリのファイル名になります。絶対パスでファイル名を指定しない場合、このファイルはデータ ディレクトリに置かれます。

MySQL 5.1.12 以降、`--log` または `-l` を指定する場合、`--general-log` オプションで最初の一般クエリ ログ状態を指定することも可能です。このオプションで、引数なし、または 値を 0 にすると、ログが無効化します。省略する、または 値を 1 とすると、ログが有効化します。`--log` または `-l` を指定しない場合、`--general-log` には何の影響もありません。

`general_log` および `general_log_file` のグローバル システム変数で、一般クエリ ログのランタイム制御ができます。`general_log` を 0 (または OFF) にすると、ログが無効化し、1 (または ON) で有効化します。`general_log_file` を指定して、ログ ファイルの名前を指定することもできます。ログ ファイルがすでに開いている場合は、それを閉じて、新規ファイルを開けます。

一般クエリ ログを有効化した場合、出力の書き込み先は `--log-output` オプション、または `log_output` 環境変数で指定します。ノート：出力先が `NONE` である場合、一般ログを有効化していても、出力書き込みはできません。同様に、ログ出力先の値に `FILE` がない場合は、ログ効果はありません。

サーバの再起動やログのフラッシュでは、一般クエリ ログの新規ファイルを生成しません。フラッシュはファイルを閉じて、それを再び開けるだけです。Unix では、ファイル名を変更して、次のコマンドを使用して、新規ファイルを作成できます。

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> cp host_name-old.log backup-directory
shell> rm host_name-old.log
```

Windows では、サーバがログ ファイルを使用している間は、ログ ファイルの名前変更はできません。まずサーバを停止してから、ファイルの名前を変更し、そして、サーバを再起動してから新規のログ ファイルを作成します。

MySQL 5.1.12 から、ランタイムで一般クエリ ログを無効化できるようになりました。

```
SET GLOBAL general_log = 'OFF';
```

ログを無効化した状態で、コマンドラインなどを使用して、ログ ファイルの名前を外部的に変更します。そして、ログを再び有効化します。

```
SET GLOBAL general_log = 'ON';
```

このやり方は、どのプラットフォームでも使用でき、サーバの再起動は不要です。

バイナリ ログ

バイナリ ログの内容には更新データのステートメントがあり、レコードに一致しない場合の **DELETE** などで更新済みのステートメントがあります。ステートメントは、変更の内容を説明する「イベント」として保存の対象になります。さらに、クエリの更新に要した時間に関する情報も、バイナリ ログの内容です。

注意：バイナリ ログは、更新ログの代わりになるものです。更新ログは MySQL 5.0 以降では利用できません。バイナリ ログには、更新ログで利用可能だった情報がすべて、より効率的かつトランザクション セーフな内容になっています。トランザクションを行うときは、MySQL バイナリ ログをバックアップに使用します。更新ログを使用しません。

バイナリ ログの内容は、データ変更に関数ステートメントを含みません。そのため、問題があるクエリの識別などで、すべてのステートメントが必要な場合は、一般クエリ ログを使用します。詳細は [一般クエリ ログ](#) を参照してください。

バイナリ ログの主要目的は、リストア作業中にできるだけデータベースの更新を行う、ということです。バイナリ ログにはバックアップ後のすべての更新情報が入ることになるため、スレーブ サーバに送るステートメントのマスタ記録として、レプリケーションで使用します。詳細は [5章レプリケーション](#) を参照してください。

MySQL Enterprise

```
##### DDL ##### MySQL Network Monitoring and Advisory
Service ##### http://www-jp.mysql.com/products/enterprise/advisors.html #
#####
```

バイナリ ログを有効化したサーバを実行すると、パフォーマンスが 1% ほど遅れます。しかし、リストア作業が必要になった場合のバイナリ ログの有用性、レプリケーション セットアップでの利便性を考慮すると、1% の遅れは許容できるものとします。

`--log-bin=[base_name]` オプションで起動すると、`mysqld` で、データ更新に関わる SQL ステートメントのすべてをログ ファイルに書き込みます。`base_name` の値を指定しない場合、デフォルト名は、`-bin` を元にするホスト マシンの名前になります。ベース名を指定するときに、絶対パスを使用しない場合は、サーバでデータ ディレクトリにファイルを作成します。できるだけ、ベース名は指定することをお勧めします。その理由に関しては [Open Issues in MySQL](#) を参照してください。

`--log-bin=base_name.extension` など、ログ名に拡張子を付ける場合は、その拡張子はサイレントで削除、無視の対象になります。

`mysqld` ではバイナリ ログのベース名で数値の拡張子を付加します。この数値はサーバで新規ログ ファイルを作成する度に増加し、ファイルの番号が連続する形になります。サーバは起動、そしてログ フラッシュ毎に、新規のバイナリ ログ ファイルを作成します。さらにサーバは現行のログ サイズが `max_binlog_size` に到達すると、自動的に新規のバイナリ ログ ファイルを作成します。大きなトランザクションがあると、バイナリ ログ ファイルの `max_binlog_size` を超えることになるため、1 つのトランザクションで最大値を使い切らないようにする必要があります。

使用しているバイナリ ログ ファイルの追跡を行うために、`mysqld` で、使用があったバイナリ ログ ファイルのすべての名前を含めた、バイナリ ログ インデックスを作成します。デフォルトではバイナリ ログ ファイルと同じベース名で、`'index'` という拡張子がつきます。バイナリ ログ インデックス ファイルの名前は、`--log-bin-index=[file_name]` オプションを使用して変更できます。`mysqld` が作業中の場合は、このファイルを手動で変更することはしないでください。`mysqld` が混乱する原因になります。

バイナリ ログ ファイルとインデックス ファイルへの書き込みは、**MyISAM** テーブルへの書き込みと同じ方法で処理します。[How MySQL Handles a Full Disk](#) を参照してください。

RESET MASTER ステートメントでバイナリ ログ ファイルをすべて削除する場合、または **PURGE MASTER LOGS** でそれらをサブセットする場合は、**RESET** 構文 および **マスタ サーバをコントロールする SQL ステートメント** を参照してください。

バイナリ ログは、バックアップでリカバリ作業を行うときに影響があることから、いくつかの制限があります。詳細は **レプリケーション機能と既知問題** を参照してください。

トリガや記憶したルーチンに関するバイナリ ログgingは、**ストアドルーチンとトリガのバイナリログ** を参照してください。

次に示すオプションを **mysqld** で使用して、何をどうバイナリ ログに記録するかを指定します。後続の説明も参考にしてください。

レプリケーションでは、ここで説明するオプションは、マスタからスレーブに送るステートメントに影響します。また、マスタからのステートメントを受けたスレーブが、そのうちの何を実行して、何を無視するかを制御するオプションもあります。その詳細は **レプリケーションのオプションと変数** を参照してください。

- **--binlog-do-db=db_name**

db_name などデフォルトのデータベースに関わる更新のバイナリ ログgingを制限する。(データベースは **USE** で選択可能。) 明示的な指定がない限り、デフォルト以外のデータベースは無視の対象。このオプションを使用するときは、デフォルトのデータベースだけが更新の対象であることを確認すること。

これには、例外があり、**CREATE DATABASE**、**ALTER DATABASE**、**DROP DATABASE** などのステートメントを使用する場合は、この限りではない。サーバがそのステートメントで指定するデータベース (デフォルトのデータベース以外) によって、ステートメントをログするかどうかを判断する。

期待通りにならない可能性がある例として、サーバを **binlog-do-db=sales** で起動した場合に、**USE prices; UPDATE sales.january SET amount=amount+1000;** を実行すると、このステートメントはバイナリ ログへの書き込み対象ではない。

複数のデータベースをログするには、複数のオプションを使用のこと。それぞれのデータベースにそれぞれのオプションを使用する。

- **--binlog-ignore-db=db_name**

db_name などデフォルトのデータベースに関わる更新のバイナリ ログgingを優先する。(データベースは **USE** で選択可能。) このオプションを使用するときは、デフォルトのデータベースだけが更新の対象であることを確認すること。

--binlog-do-db 同様に、これにも例外があり、**CREATE DATABASE**、**ALTER DATABASE**、**DROP DATABASE** などのステートメントを使用する場合は、この限りではない。サーバがそのステートメントで指定するデータベース (デフォルトのデータベース以外) によって、ステートメントをログするかどうかを判断する。

期待通りにならない可能性がある例として、サーバを **binlog-ignore-db=sales** で起動した場合に、**USE prices; UPDATE sales.january SET amount=amount+1000;** を実行すると、このステートメントはバイナリ ログへの書き込み対象になる。

複数のデータベースを無視するには、複数のオプションを使用のこと。それぞれのデータベースにそれぞれのオプションを使用する。

サーバはバイナリ ログへの更新の処理の仕方 (ログが無視か) に関するオプションを、次に示すルールに従って評価します。前述の通り、**CREATE DATABASE**、**ALTER DATABASE**、**DROP DATABASE** のステートメントは例外とします。データベースに対する 作成、変更、ドロップ は、次のルールに従って辿り着きます。

1. **--binlog-do-db** または **--binlog-ignore-db** ルールがあるかどうか。

- No: バイナリ ログにステートメントを書き込み、終了。
- Yes: 次のステップへ進む。

2. デフォルトのデータベース、**USE** で選択しているデータベースがあるかどうか。(**--binlog-do-db**、**--binlog-ignore-db**、またはその両方があるため。)

- No: ステートメントを書き込まず、終了。
 - Yes: 次のステップへ進む。
3. (デフォルトのデータベースがあるので) `--binlog-do-db` ルールがあるかどうか。
- Yes: デフォルトのデータベースは、`--binlog-do-db` ルールの適用を受けるかどうか。
 - Yes: ステートメントを書き込み、終了。
 - No: ステートメントを書き込まず、終了。
 - No: 次のステップへ進む。
4. (`--binlog-ignore-db` ルールがあるため、) `--binlog-ignore-db` ルールの適用を受けるかどうか。
- Yes: ステートメントを書き込まず、終了。
 - No: ステートメントを書き込み、終了。

たとえば、`--binlog-do-db=sales` だけのオプションで実行していると、スレーブは `sales` とは異なるデータベースのステートメントはバイナリ ログに書き込みません。つまり、`--binlog-do-db` オプションは、「他のデータベースを無視する」という働きがあります。

レプリケーションでは、バイナリ ログ ファイルをスレーブで必要としないことを確認できるまでは削除しないでください。たとえば、スレーブが 3 日以上遅れることは有り得ない場合、一日に一度、`mysqladmin flush-logs` をマスタで実行し、3 日以上経過したログを削除します。ファイルは手動で削除することもできますが、`PURGE MASTER LOGS` の使用をお勧めします。これは、バイナリ ログ インデックス ファイルの更新を安全に行えます。(日の引数使用。) 詳細は、[マスタ サーバをコントロールする SQL ステートメント](#) を参照してください。

`SUPER` 権限があるクライアントは、`SET SQL_LOG_BIN=0` ステートメントを使用して独自にステートメントのバイナリ ログを無効化できます。詳細は [SET 構文](#) を参照してください。

`mysqlbinlog` ユーティリティを使用して、バイナリ ログ ファイルの内容を表示できます。これは、ログ内のステートメントを再処理するときに活用できます。たとえば、次のように、バイナリ ログから MySQL サーバを更新できます。

```
shell> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` ユーティリティと、その使用方法については [mysqlbinlog — バイナリログファイルを処理するためのユーティリティ](#) を参照してください。バイナリ ログ ファイルとリレーログ ファイルは同じ形式で書き込みを行うため、`mysqlbinlog` をリレー ログ ファイルでも使用できます。

バイナリ ロギングはステートメントの完了(クエリの完了)とともに実行となりますが、その前にロックのリリース、またはコミットを行います。そのため、ログの記録は実行順になります。

非トランザクションの更新は、実行とともに、バイナリ ログへの保存となります。コミットなしのトランザクション内では、`InnoDB` テーブルなど、トランザクションのテーブルに変更を与える `UPDATE`、`DELETE`、`INSERT` などの更新は、サーバからの `COMMIT` ステートメントを受けるまでは、キャッシュでの保管になります。その時点で、`COMMIT` 実行前に `mysqld` でトランザクションの内容をバイナリ ログに書き込みます。トランザクションを扱うスレッドが開始すると、`binlog_cache_size` というバッファから、ステートメントのバッファにアロケートします。ステートメントがそのバッファより大きい場合は、スレッドがそのトランザクションを保管する一時テーブルを開きます。スレッドが終了すると、テンポラリ テーブルは削除になります。

非トランザクションのテーブルへの変更は、ロール バックが利きません。非トランザクション テーブルへの変更を含むロール バックがあるトランザクションの場合は、トランザクション全体が `ROLLBACK` ステートメントでのログになり、そのテーブルへの変更の複製を確実にに行い(請け負い)ます。

`Binlog_cache_use` ステータス変数は、(テンポラリファイルとともに) このバッファを使用したトランザクションの数を表示します。`Binlog_cache_disk_use` ステータス変数はそのトランザクション内で一時テーブルを使用する必要があった回数を表示します。この 2 つの変数は、`binlog_cache_size` の調整に使用して、一時ファイルの使用を避けるために十分な値を設定します。

`max_binlog_cache_size` システム変数は、デフォルトで 4 GB (最大値) としていますが、複数のステートメントのトランザクションのキャッシュを行うために、合計サイズを制限することができます。トランザクションがこのバイトより大きい場合はロール バックします。最大値は 4096 です。

バイナリ ログでは、並列のインサートを、`CREATE ... SELECT` または `INSERT ... SELECT` ステートメントの通常インサートと置き換えます。これは、たとえば、バックアップ作業中にログを適用して、テーブルの完全コピーを再生できるようにする作用です。

ノート：古いバージョンの MySQL と MySQL 5.1 とでは、バイナリ ログの形式が異なります。これはレプリケーション効果を改善した結果です。詳細は [MySQL バージョン間のレプリケーション互換性](#) を参照してください。

デフォルトのバイナリ ログでは、ディスクへのそれぞれの書き込みは非同期です。そのため、MySQL サーバに限らず、オペレーティングシステムまたはマシンがクラッシュすると、バイナリ ログの最後のステートメントを失う可能性があります。これを防ぐには、バイナリ ログへのそれぞれの書き込み `N` をディスクで同期します。それには、`sync_binlog` システム変数を使用します。(システム変数を参照のこと。) `sync_binlog` での最も安全な値は 1 ですが、これは最も遅くなる値です。`sync_binlog` を 1 で設定しても、クラッシュの場合によってテーブルとバイナリ ログの内容が異なる可能性があります。たとえば、`InnoDB` テーブルに対して、`COMMIT` ステートメントを MySQL サーバで処理した場合、トランザクション全体をバイナリ ログに書き込むことになり、そのトランザクションを `InnoDB` にコミットすることになります。この 2 つの動作を処理している最中にサーバがクラッシュすると、このトランザクションは起動時の `InnoDB` にロール バックしますが、バイナリ ログでは存在している、ということになります。この問題は `--innodb-safe-binlog` オプションで解決できます。このオプションは、`InnoDB` テーブルとバイナリ ログの内容に整合性を持たせます。(ノート：MySQL 5.0 以降、`--innodb-safe-binlog` は、XA トランザクション サポートの導入とともに、廃止になりました。)

このオプションでさらに安全性を高めるには、MySQL サーバを、トランザクション毎にバイナリ ログと `InnoDB` ログを同期してディスクに送るように設定する必要があります。`InnoDB` ログはデフォルトで同期化しているため、バイナリ ログには `sync_binlog=1` を使用して同期化します。このオプションの効果としては、クラッシュ後に再起動したときに、ロール バックしたトランザクションに対して、MySQL がバイナリ ログから `InnoDB` トランザクションのロール バック返しを行います。これにより、バイナリ ログが `InnoDB` テーブルのデータと完全に一致します。そして、スレーブはマスタと同期化したままであるため、ロール バックしたステートメントを受けることはありません。

(ノートとして、`--innodb-safe-binlog` オプションは、MySQL サーバが `InnoDB` 以外のストレージ エンジンを更新するときにも使用できます。) `InnoDB` のクラッシュ リカバリでは、`InnoDB` テーブルに影響があるステートメントとトランザクションだけを、バイナリ ログから削除します。MySQL サーバがクラッシュ リカバリ中に、バイナリ ログが通常より短いと判断する場合には、`InnoDB` のトランザクションの中から、成功していないコミットが 1 つ以上あることを示します。つまり、サーバは、`The binary log <name> is shorter than its expected size` というエラー メッセージを出力します。これは、`sync_binlog=1` と指定し、ディスク/ファイル システムが実際に同期していれば、発生しません。もし、このエラー メッセージがでる場合には、このバイナリ ログは未修正のままであるため、レプリケーションを行うときには、マスタ データのスナップショットを最初からやり直す必要が出てきます。

スロー クエリ ログ

スロー クエリ ログの内容は、`long_query_time` 秒より実行に時間がかかる SQL ステートメントすべてが入ります。最初のテーブル ロックを取得するまでの時間は、実行時間としてはカウントしていません。すべてのステートメントを実行し、すべてのロックをリリースした後に、`mysqld` で、ステートメントをスロー クエリ ログとして書き込むため、ログ順は実行順とは異なることがあります。最低限値は 1 で、`long_query_time` のデフォルト値 (最大値) は 10 です。

MySQL 5.1.6 以降、スロー クエリ ログを有効化するには、`mysqld` を `--log-slow-queries[=file_name]` オプションで起動します。必要に応じて、`--log-output` オプションを使用して、ログの出力先を指定します。(一般クエリとスロー クエリのログ出力先の選択を参照のこと。) MySQL 5.1.6 より前は、スロー クエリ ログの出力先はファイルです。スロー クエリ ログ ファイルを有効化するには、`--log-slow-queries[=file_name]` オプションを使用します。

`--log-slow-queries` に `file_name` 値を指定しない場合、デフォルト名は、`host_name-slow.log` というデータ ディレクトリのファイル名になります。絶対パスでファイル名を指定しない場合、このファイルはデータ ディレクトリに置かれます。

MySQL 5.1.12 以降、`--log-slow-queries` を指定する場合、`--slow-query-log` オプションで最初のスロー クエリ ログ状態を指定することも可能です。このオプションで、引数なし、または 値を 0 にすると、ログが無効化します。省略する、または 値を 1 とすると、ログが有効化します。`--log-slow-queries` を指定しない場合、`--slow-query-log` には何の影響もありません。

`slow_query_log` および `slow_query_log_file` のグローバル システム変数で、スロー クエリ ログのランタイム制御ができます。`slow_query_log` を 0 (または OFF) にすると、ログが無効化し、1 (または ON) で有効化します。`general_log_file` を指定して、ログ ファイルの名前を指定することもできます。ログ ファイルがすでに開いている場合は、それを閉じて、新規ファイルを開けます。

スロー クエリ ログを有効化した場合、出力の書き込み先は `--log-output` オプション、または `log_output` 環境変数で指定します。ノート：出力先が `NONE` である場合、スロー ログを有効化していても、出力書き込みはできません。同様に、ログ出力先の値に `FILE` がない場合は、ログ効果はありません。

スロー クエリ ログには、実行に時間がかかるクエリが入るため、最適化の対象になります。しかし、時間がかかるスロー クエリ ログの検査は手間がかかります。ここで、`mysqldumpslow` コマンドを使用してスロー クエリ ログを処理することで、そのクエリをログに概括表示します。`mysqldumpslow --help` を使用して、このコマンドに関するサポートを探してください。

MySQL 5.1 では、インデックスを使用しないクエリは、`--log-queries-not-using-indexes` オプションで指定すると、スロー クエリ ログで記録されるようになります。詳細は [コマンド オプション](#) を参照してください。

MySQL Enterprise

#####MySQL Network Monitoring and Advisory Service ##### <http://www.jp.mysql.com/products/enterprise/advisors.html> #####

MySQL 5.1 では、スロー クエリ ログに対して、`--log-slow-admin-statements` というサーバ オプションで、`OPTIMIZE TABLE`、`ANALYZE TABLE`、`ALTER TABLE` など、時間がかかる管理ステートメントのロギング要求を有効化します。

クエリ キャッシュで扱うクエリは、スロー クエリ ログには付加しません。テーブルのレコードがない、または 1 つだけであるときは、インデックスで管理する必要がないため、これもスロー クエリ ログには入りません。

ログ ファイルの保守

MySQL Server は、様々なやりとりを把握できるように、様々なログ ファイルを数多く作成しています。(MySQL [サーバ ログ](#) を参照のこと。) しかし、ディスク スペースを空けるために、これらのファイルは定期的にクリーンアップすることが必要です。

MySQL でロギングを有効化しているときは、定期的にファイルのバックアップを取り、古いファイルは削除するなどして、時には MySQL ロギング を新しいファイルで行うことも検討してください。詳細は [データベースのバックアップ](#) を参照してください。

Linux (Red Hat) では、`mysql-log-rotate` スクリプトを使用して、古いログ ファイルの処理を自動的に行うことができます。RPM 配布から MySQL をインストールすると、このスクリプトを自動的にインストールしています。注意: レプリケーションにバイナリ ログを使用している場合は、このスクリプトの扱いには注意が必要です。

他のシステムでは、ログ ファイルを処理するための短いスクリプトを自分でインストールする必要があります。このスクリプトは、`cron` などで行います。

MySQL に新しいログ ファイルの使用を強制するには、`mysqladmin flush-logs` または `mysqladmin refresh` を使用するか、SQL コマンド `FLUSH LOGS` を使用します。詳細は、[FLUSH 構文](#) および [mysqladmin — MySQL サーバの管理を行うクライアント](#) を参照してください。

ログのフラッシュ操作は次のことを行います。

- ログ ファイルへの一般クエリ ロギング (`--log`)、またはスロー クエリ ロギング (`--log-slow-queries`) を有効化した場合、サーバが一般クエリ ログ ファイルまたはスロー クエリ ログ ファイルを閉じ、再開する。
- バイナリ ロギング (`--log-bin`) を使用している場合、サーバは現行のログ ファイルを閉じ、シーケンス番号で新しいログ ファイルを開く。
- `--log-error` オプションでエラー ログ ファイル名を指定している場合、`-old` をサフィックスとするエラー ログ ファイルの名前になり、新しいエラー ログ ファイルを作成する。

ログをフラッシュすると、サーバで新しいバイナリ ログ ファイルを作成します。しかし、これは一般ログとスロー ログを閉じて、また再開するだけに過ぎません。Unix 環境で新しいファイルを作成するには、フラッシュ

する前に現行ログの名前を変更します。そうすると、フラッシュするときに、サーバがオリジナルの名前で新しいログを開きます。たとえば、一般クエリ ログを `mysql.log` とし、スロークエリのログを `mysql-slow.log` とした場合には、次のようにコマンドをつなげます。

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

この時点で、`mysql.old` と `mysql-slow.log` のバックアップができるので、ディスクから削除することができます。

Windows では、サーバがログ ファイルを使用している間は、ログ ファイルの名前変更はできません。まずサーバを停止してから、ファイルの名前を変更し、そして、サーバを再起動してから新しいログ ファイルを作成します。

MySQL 5.1.12 から、ランタイムで一般クエリ ログとスロー クエリ ログを無効化できるようになりました。

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

ログを無効化した状態で、コマンドラインなどを使用して、ログ ファイルの名前を外部的に変更します。そして、ログを再び有効化します。

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

このやり方は、どのプラットフォームでも使用でき、サーバの再起動は不要です。

同じマシン上での複数 MySQL サーバの実行

場合によっては、複数の `mysqld` サーバを同一のマシンで起動することがあります。たとえば、既存の稼働環境のままにして、新しい MySQL リリースをテストしたい場合が考えられます。また、ユーザごとに異なる `mysqld` サーバへのアクセス権を与える場合などもあります (顧客ごとに独立した MySQL インストールを提供するインターネット サービス プロバイダなど)。

単一のマシン上で複数のサーバを実行するには、いくつかのパラメータでサーバ固有の値を設定する必要があります。これらはコマンドラインまたはオプション設定ファイルで設定します。プログラム オプションに関しては、[プログラム・オプションの指定](#) を参照してください。

少なくとも、次のオプションをサーバごとに変えます。

- `--port=port_num`

`--port` は、TCP/IP 接続のポート番号を制御する。

- `--socket=path`

`--socket` で、Unix のソケット ファイル パス、または Windows の名前付きパイプを制御する。Windows では、名前付きパイプ接続をサポートしているサーバに対して、独特のパイプ名を指定する必要がある。

- `--shared-memory-base-name=name`

Windows のみで使用可能。Windows で使用する共有メモリ名を指し、共有メモリ経由でクライアントの接続を許可する。共有メモリ接続をサポートするサーバに対して、共有メモリに独特な名前を指定する必要がある。

- `--pid-file=file_name`

Unix のみで使用可能。サーバがプロセス ID を書き込むファイルのパス名を指定する。

次のログ ファイル オプションを使用する場合は、それぞれのサーバに対して異なる値を設定する。

- `--log=file_name`
- `--log-bin=file_name`
- `--log-update=file_name`
- `--log-error=file_name`

[ログ ファイルの保守](#)で、ログ ファイルのオプションについて参照してください。

パフォーマンスを高めるには、次のオプションをサーバに別々に指定して、物理ディスク間の負荷を分けます。

- `--tmpdir=path`

複数のテンポラリ ディレクトリを置くと、どの MySQL サーバにどの一時ファイルが属するのかわかりやすくなるため、お勧めです。

データ ディレクトリについても、それぞれのサーバで異なるディレクトリを使用するようにします。これは `--datadir=path` オプションで指定します。

警告: 2 つのサーバから同じデータベースのデータを更新しないでください。使用しているオペレーティング システムで障害からの保護ができるシステム ロックをサポートしていない場合、予期しない事態が発生する可能性があります。また、複数のサーバが同じデータ ディレクトリを使用し、ログを有効化している場合、適切なオプションを使用して、それぞれのサーバに異なるログ ファイル名を指定する必要があります。そうしないと、サーバ同士で同じファイルにログします。このセットアップは、[MyISAM](#) または [MERGE](#) テーブルでのみ機能します。他のストレージ エンジンでは使用できません。

サーバ間でのデータ ディレクトリ共有に関するこの警告は、NFS 環境にも当てはまります。NFS 環境で複数の MySQL サーバに同じデータ ディレクトリへのアクセスを認めることはしないでください。

- 重要な問題として、NFS は速度のボトルネック。NFS にはそのような使用目的はない。
- 2 つ以上のサーバが互いに干渉しないようにすることが困難。通常、NFS ファイルロックは `lockd` デモンで処理するが、現在のところ、どのような状況でも 100% の信頼性でロックを実行できるプラットフォームは存在しない。

NFS で複数のサーバにデータ ディレクトリを共有することは賢明ではありません。また、複数の CPU を持つ 1 台のコンピュータを用意し、スレッドを効率的に処理するオペレーティング システムを使用してください。

複数の MySQL インストールを異なる場所にする場合、`--basedir=path` オプションを使用して、それぞれのサーバに対してベース ディレクトリを指定し、それぞれのサーバがお互いに別のデータ ディレクトリ、ログ ファイル、および PID ファイルを使用するようにします (これらの値のデフォルトは、ベース ディレクトリと相対的に決定します)。そして、他にも指定する必要があるオプションとして、`--socket` と `--port` があります。たとえば、バイナリ配布の `.tar` ファイルを使用して MySQL の複数のバージョンをインストールするとします。これらを別々の場所にインストールすれば、対応するベース ディレクトリ下で `./bin/mysqld_safe` コマンドを使用して、インストールしたサーバを別々に起動できます。`mysqld_safe` が、`mysqld` に渡す適切な `--basedir` オプションを特定するため、`--socket` オプションと `--port` オプションを `mysqld_safe` に設定するだけで済みます。

次のセクションで説明するように、環境変数の設定または適切なコマンドライン オプションの指定で、追加サーバを起動することが可能です。ただし、より永続的に複数のサーバを実行する必要がある場合には、オプション設定ファイルを使用して、それぞれサーバ固有のオプション値を指定する方法が便利です。これには、`--defaults-file` オプションを活用します。

Windows で複数サーバの実行

適切なパラメータで、コマンドラインからサーバを手動で起動すると、Windows 上で複数のサーバを実行できます。Windows NT ベースのシステムでは、複数のサーバを Windows サービスとしてインストールし、Windows サービスとして実行する方法もあります。コマンドラインから、またはサービスとして MySQL サーバを実行する方法については、[Windows に MySQL をインストールする](#) を参照してください。このセクションでは、データ ディレクトリなど、スタートアップ オプション値をサーバごとに固有設定してサーバを起動する方法について説明します。オプションについては、[同じマシン上での複数 MySQL サーバの実行](#) を参照してください。

コマンドラインで複数の Windows サーバの起動

コマンドラインから複数のサーバを手動で起動するには、コマンドラインまたはオプション ファイルで必要なオプションを指定します。オプション ファイルでオプションを設定する方は簡単ですが、それぞれのサーバに固有のオプション セットを確実に設定するには注意が必要です。これを行うには、それぞれのサーバにオプション ファイルを作成し、サーバ起動時に `--defaults-file` を使用して、サーバにファイル名を指定します。

たとえば、`mysqld` を、`C:\mydata1` というデータディレクトリ、ポート 3307 で実行し、一方で、`mysqld-debug` を、`C:\mydata2` というデータディレクトリ、ポート 3308 で実行するとします。実行する前に、それぞれのディレク

トリがあることを確認し、それぞれに権限テーブルを含む `mysql` データベースのコピーがあることも確認します。それから、2つのオプション ファイルを作成します。たとえば、次のような `C:\my-opts1.cnf` という名前のファイルを作成します。

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

そして、`C:\my-opts2.cnf` という名前の 2 番目のファイルを、次のように作成します。

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

それぞれのオプション ファイルで、それぞれのサーバを起動します。

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-debug --defaults-file=C:\my-opts2.cnf
```

Windows NT では、サーバがフォアグラウンドで起動するため、2つのコマンドを別々のコンソール ウィンドウで実行します。

サーバをシャットダウンするには、該当するポート番号に接続する必要があります。

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

この例示のように設定したサーバは、クライアントに対して TCP/IP 接続を許可します。Windows の名前付きパイプ接続も可能にするには、`mysqld-nt` サーバを使用し、名前付きパイプを有効にするオプションでその名前を指定します（名前付きパイプ接続をサポートするサーバは、それぞれ固有のパイプ名を使用します）。たとえば、`C:\my-opts1.cnf` ファイルを次のように修正します。

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

そして、サーバを次のように起動します。

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

2 つ目のサーバで使用する `C:\my-opts2.cnf` も同様に修正します。

共有メモリ接続をサポートするときにも、同様の手順で設定します。`--shared-memory` オプションで接続を有効化し、`--shared-memory-base-name` オプションで固有の共有メモリ名をそれぞれのサーバに指定します。

複数の Windows サーバをサービスとして起動

Windows NT ベースのシステムでは、MySQL サーバを Windows サービスとして実行します。単一の MySQL サービスのインストール、制御、および削除の手順については、[Windows のサービスとして MySQL を起動する](#) を参照してください。

複数のサーバをサービスとしてインストールことも可能です。その場合、それぞれのサーバで異なるサービス名を使用します。また、サーバごとに一意である必要があるため、その他のパラメータにも注意が必要です。

次の手順は、`CC:\mysql-5.0.19` および `C:\mysql-5.1.15-beta` と、2つの異なる MySQL バージョンをインストールした `mysqld-nt` サーバを実行する場合を想定しています。たとえば、本稼働サーバとして 5.0.19 を実行しているときに、アップグレード前に 5.1.15-beta をテストする場合などが該当します。

`--install` または `--install-manual` オプションで MySQL サービスをインストールする場合には、次の原則があります。

- サービス名を指定しない場合、サーバは `MySQL` のデフォルト サービス名を使用し、標準オプション ファイルの `[mysqld]` グループからオプションを読み取る。
- `--install` オプションの後でサービス名を指定すると、サーバは `[mysqld]` オプション グループを無視し、サービスと同じ名前のグループからオプションを読み取る。サーバは、標準オプション ファイルからオプションを読み取る。

- サービス名の後で `--defaults-file` オプションを指定すると、サーバは標準オプション設定ファイルを無視し、指定ファイルの `[mysqld]` グループからのみオプションを読み取る。

注意：MySQL 4.0.17 より前では、デフォルトのサービス名 (`(MySQL)`) でインストールしたサーバ、または `mysqld` というサービス名で明示的にインストールしたサーバでだけ、標準オプション設定ファイルの `[mysqld]` グループを読み込みが可能でした。4.0.17 以降は、インストールしたサービス名を問わず、標準オプション ファイルを読み込めるすべてのサーバで `[mysqld]` グループを読みます。これにより、すべての MySQL サービスで使用する必要があるオプションに `[mysqld]` グループを使用でき、それぞれのサービスでインストールしたサーバで、そのサービス名を元に名付けたオプション グループを使用できるようになりました。

この情報を元に、複数のサービスを様々な方法でセットアップできます。次の手順では、その例を示します。これらを試行するときは、シャットダウンし、既存の MySQL サービスを別の場所などに移動/削除してください。

- アプローチ 1: オプションをすべてのサービスに対して 1 つの標準オプション ファイルで指定し、それぞれのサーバに対して別々のサービス名を与えます。MySQL 5.0.19 には、`mysqld1` というサービス名を `mysqld-nt` で、MySQL 5.1.15-beta では `mysqld2` というサービス名を `mysqld-nt` で、それぞれ実行するとした場合、MySQL 5.0.19 には `[mysqld1]` グループを、MySQL 5.1.15-beta には `[mysqld2]` グループを使用します。たとえば、`C:\my.cnf` を次のようにセットアップします。

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-5.0.19
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.1.15-beta
port = 3308
enable-named-pipe
socket = mypipe2
```

それぞれのサービスを次のようにインストールします。フル パスを使用して、Windows で正確な実行プログラムをそれぞれのサービスで登録することを確認してください。

```
C:\> C:\mysql-5.0.19\bin\mysqld-nt --install mysqld1
C:\> C:\mysql-5.1.15-beta\bin\mysqld-nt --install mysqld2
```

サービスを起動するには、サービス マネージャを使用するか、または該当するサービス名で `NET START` を使用します。

```
C:\> NET START mysqld1
C:\> NET START mysqld2
```

サービスを停止するには、サービス マネージャを使用するか、または該当するサービス名で `NET STOP` を使用します。

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- アプローチ 2: 別々のファイルでそれぞれのサーバに対して、オプションを指定します。サービスをインストールするときは、`--defaults-file` を使用して、それぞれのサーバでどのファイルを使用するかを指します。この場合、それぞれのファイルで `[mysqld]` を使用して、オプションをリストします。

このアプローチで、MySQL 5.0.19 の `mysqld-nt` に対するオプションを指定し、`C:\my-opts1.cnf` というファイルを次のように作成します。

```
[mysqld]
basedir = C:/mysql-5.0.19
port = 3307
enable-named-pipe
socket = mypipe1
```

MySQL 5.1.15-beta の `mysqld-nt` に対しては、`C:\my-opts2.cnf` というファイルを次のように作成します。

```
[mysqld]
basedir = C:/mysql-5.1.15-beta
```



```
port = 3308
enable-named-pipe
socket = mypipe2
```

サービスを次のようにインストールします。(それぞれのコマンドを一行で入力してください。)

```
C:\> C:\mysql-5.0.19\bin\mysqld-nt --install mysqld1
--defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-5.1.15-beta\bin\mysqld-nt --install mysqld2
--defaults-file=C:\my-opts2.cnf
```

MySQL サーバをサービスとしてインストールするには、`--defaults-file` オプションを使用します。サービス名でそのオプションを優先化します。

サービスをインストール後、それぞれを前述の例示と同じ方法で、起動と停止を行います。

複数のサービスを削除するには、`mysqld --remove` をそれぞれに使用し、後続の `--remove` オプションでサービス名を指定します。デフォルトのサービス名 (MySQL) を使用している場合には、この指定は省略可能です。

Unix で複数サーバの実行

Unix で複数のサーバを実行する最も簡単な方法は、異なる TCP/IP ポートと Unix ソケット ファイルでサーバをコンパイルすることです。これは、それぞれが異なるネットワークのインターフェースとしてリストします。それぞれのインストールに対して、異なるベース ディレクトリにコンパイルすることも、コンパイルしたデータ ディレクトリ、ログ ファイル、PID ファイル場所を、サーバごとに自動的に切り離す結果を生みます。

ここでは、既存の MySQL 5.0.19 サーバをデフォルトの TCP/IP ポート番号 (3306) と Unix ソケットファイル (`/tmp/mysql.sock`) で設定していると想定します。MySQL 5.1.15-beta サーバの設定で、異なる操作パラメータを持たせるには、`configure` コマンドを次のように使用します。

```
shell> ./configure --with-tcp-port=port_number \
--with-unix-socket-path=file_name \
--prefix=/usr/local/mysql-5.1.15-beta
```

ここで、`port_number` と `file_name` は、デフォルトの TCP/IP ポート番号と Unix ソケット ファイルのパスとは異なる必要があります。`--prefix` 値は、既存の MySQL をインストールしたディレクトリとは異なるディレクトリに指定します。

MySQL サーバに特定のポート番号をリストしている場合は、次のコマンドを使用して、重要な設定可能変数において、ベース ディレクトリおよび Unix ソケット ファイル名を含め、どの操作パラメータをそこで使用しているかを確認します。

```
shell> mysqladmin --host=host_name --port=port_number variables
```

このコマンドで表示する情報を元に、追加サーバを設定するときに 使用できない オプション値がどれであることを確認します。

ノート: `localhost` をホスト名として指定する場合、`mysqladmin` は TCP/IP ではなく Unix ソケット ファイルでの接続に初期化します。MySQL 4.1 以降では、使用する接続プロトコルを、`--protocol={TCP|SOCKET|PIPE|MEMORY}` オプションを使用して指定することができます。

新しい MySQL サーバを、単に異なる Unix ソケット ファイルと TCP/IP ポート番号で起動するためだけにコンパイルする必要はありません。同一のサーバのバイナリを使用して、ランタイムに別々のパラメータ値で、それぞれの起動 (呼び出し) を行うことができます。これを行う 1 つの方法としては、次のように、コマンドライン オプションを使用します。

```
shell> mysqld_safe --socket=file_name --port=port_number
```

2 つ目のサーバを起動するには、異なる値を `--socket` と `--port` のオプションで使用して、`--datadir=path` オプションを `mysqld_safe` に渡します。これにより、このサーバは異なるデータ ディレクトリを使用ようになります。

同様の効果を成す別の方法としては、環境変数を使用して、Unix ソケット ファイル名と TCP/IP ポート番号をセットします。

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
```



```
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

テスト用で 2 番目のサーバを起動するには、この方法が一番早くできます。この手順の利点は、環境変数の設定で、同じシェルから呼び出すクライアント プログラムにも適用できるため、クライアント接続は自動的に 2 番目のサーバになります。

環境変数 では、`mysqld` に反映する環境変数のリストについて説明しています。

自動的なサーバ実行では、ブートしたときに実行する起動スクリプトが次のコマンドが、それぞれのコマンドに該当するオプション ファイルのパスで、それぞれのサーバの対して一度実行します。

```
shell> mysqld_safe --defaults-file=file_name
```

それぞれのオプション ファイルには、特定のサーバ固有のオプション値があります。

Unix では、この `mysqld_multi` スクリプトで複数のサーバを起動することが 1 つの方法といえます。`mysqld_multi — 複数のMySQL サーバ管理` を参照してください。

複数サーバ環境でのクライアントプログラムの使用

クライアントにコンパイルしたネットワーク インタフェースとは異なる (以外の) ネットワーク インタフェースを使用している MySQL サーバに、そのクライアントから接続するには、次のいずれかの方法を使用します。

- クライアントを以下のいずれかで開始する。`--host=host_name --port=port_number` を使用して TCP/IP 経由でリモートホストに接続する。`--host=127.0.0.1 --port=port_number` を使用して TCP/IP 経由でローカルサーバに接続する。`--host=localhost --socket=file_name` を使用して Unix ソケットファイルまたは Windows 名前付きパイプ経由で接続する。
- MySQL 4.1 から、TCP/IP 経由で接続する場合は `--protocol=tcp` を、Unix ソケットを使用する場合は `--protocol=socket` を、名前付きパイプを使用する場合は `--protocol=pipe` を、共有メモリを使用する場合には `--protocol=memory` を指定して、クライアントを開始する。TCP/IP 接続では、`--host` オプションと `--port` オプションを指定することが必要な場合もある。他の接続タイプでは、場合によっては、`--socket` オプションで、Unix ソケットファイルまたは Windows 名前付きパイプ名を指定したり、`--shared-memory-base-name` オプションで共有メモリ名を指定することが必要になる。共有ファイルを使用した接続は Windows だけで可能。
- Unix では、クライアントを起動する前に、`MYSQL_UNIX_PORT` 環境変数と `MYSQL_TCP_PORT` 環境変数を設定して、Unix ソケットおよび TCP/IP ポート番号を指定する。特定のソケットファイルまたはポートを永続的に使用する場合、これらの環境変数を設定するコマンドを `.login` ファイルに置いて、ログインごとにこれらの環境変数を適用するようにできる。詳細は **環境変数** を参照のこと。
- オプション ファイルの `[client]` グループに、デフォルトソケットファイルと TCP/IP ポートを指定する。たとえば、Windows では `C:\my.cnf` を、Unix ではホームディレクトリの `.my.cnf` ファイルを使用できる。詳細は **オプションファイルの使用** を参照のこと。
- C プログラムでは、ポートまたはソケット引数を `mysql_real_connect()` の呼び出しで指定できる。また、`mysql_options()` を呼び出して、プログラムにオプション ファイルを読み取らせることもできる。詳細は、**C API機能の説明** を参照のこと。
- Perl の `DBD::mysql` モジュールを使用している場合、MySQL オプション ファイルからオプションを読み取ることができる。次に例を示す。

```
$dsn = "DBI:mysql:test:mysql_read_default_group=client;"
. "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
$dbh = DBI->connect($dsn, $user, $password);
```

配列照合に関する詳細は **MySQL Perl API** を参照してください。

他のプログラミング インターフェースでも、オプション ファイル読み込みに関する同様の機能を提供していることがあります。

MySQL クエリ キャッシュ

クエリ キャッシュには、[SELECT](#) ステートメントのテキストを、クライアント送信の関連結果を合わせて格納します。後でまったく同じクエリを受け取ると、サーバはそのクエリの解析と実行を繰り返す代わりに、クエリ キャッシュから結果を取り出します。

テーブルの一部を頻繁には変更することがなく、同じクエリを何度も実行するような環境では、クエリ キャッシュが非常に役立ちます。動的コンテンツを大量に持つ多くの Web サーバでは、このような状況が一般的です。

注意：クエリキャッシュから古いデータが返ることはありません。データ変更があると、クエリ キャッシュに関連するエントリをすべてフラッシュします。

注意：複数の [mysqld](#) サーバで 同一の [MyISAM](#) テーブルを更新している環境では、クエリ キャッシュは機能しません。

注意：クエリ キャッシュは、サーバサイドの準備されたステートメント (準備文) には使用できません。サーバサイドの準備文がある場合には、クエリ キャッシュの条件を満たしません。詳細は [準備されたC APIステートメント](#)。を参照してください。

クエリ キャッシュのパフォーマンスに関するデータの一部を次に示します。これらの結果は、2 GB の RAM、64 MB のクエリキャッシュを搭載する Linux Alpha 2 × 500 MHz での MySQL ベンチマークスイートの実行により生成したものです。

- レコードが 1 つしかないテーブルからレコードを 1 つ選択する場合など、実行しているクエリが単純であるに関わらず、互いに異なることが原因で、クエリをキャッシュすることができないときは、クエリ キャッシュをアクティブにしておくことによるオーバーヘッドは 13% になる。これは最悪の場合のシナリオとみなすことができる。現実には、クエリはこの例よりもはるかに複雑なため、通常オーバーヘッドはかなり低くなる。
- レコードが 1 つだけのテーブルでの 1 つのレコードの検索では、238% 迅速化する。これは、キャッシュに格納しているクエリで想定している迅速化の最小値に近い数値である。

クエリ キャッシュのコードを無効にするには、[query_cache_size=0](#) と、設定する。クエリ キャッシュ コードを無効にすると、目立ったオーバーヘッドはなくなる。MySQL をソースから建てている場合、[--without-query-cache](#) オプションで [configure](#) を呼び出し、クエリ キャッシュをコードから除外できる。

クエリ キャッシュの動作

このセクションは、クエリ キャッシュのメカニズムを説明します。設定方法は、[クエリ キャッシュの設定](#) を参照してください。

解析前のクエリには、解釈が始まる前にクエリ キャッシュにあるクエリとの照合を行います。そのため、次の 2 つのクエリは、クエリ キャッシュで異なるものである、とみなします。

```
SELECT * FROM tbl_name
Select * from tbl_name
```

クエリは、バイト同士など、完全に一致する しない限り、同一とは判断しません。たとえば、クライアントで新しい形式の通信プロトコルを使用している場合や、別のクライアントが使用しているものとは異なるキャラクタセットを使用している場合も、同じものであるはずのクエリが異なるものとして、認識することがあります。

クエリ キャッシュは、解釈が始まる前にクエリ同士を照合することから、次のような種類のクエリはキャッシュの対象になりません。

- 準備されたステートメント (準備文)
- クエリが外部クエリのサブクエリである場合
- Stored プロシージャ、Stored 関数、トリガ、イベントなどのボディ内で実行したクエリ

クエリ結果をキャッシュからフェッチする前に、MySQL で、そのユーザがすべてのデータベースと関連するテーブルにおいて、[SELECT](#) 権限があるかどうかを調べます。権限がない場合は、キャッシュ結果は使用しません。

クエリ結果がキャッシュから返る度に、サーバは [Qcache_hits](#) システム変数の値を増加します。[Com_select](#) ではありません。詳細は [クエリ キャッシュのステータスと保守](#) を参照してください。

テーブルに変更があった場合、そのテーブルからキャッシュしたクエリのすべてが無効になるため、キャッシュからは削除します。変更があったクエリのマップである **MERGE** テーブルを使用するクエリも削除の対象になります。**INSERT**、**UPDATE**、**DELETE**、**TRUNCATE**、**ALTER TABLE**、**DROP TABLE**、**DROP DATABASE** など、様々なステートメントでテーブルは変化します。

InnoDB テーブルを使用するトランザクションでもクエリ キャッシュを使用します。

MySQL 5.1 では、ビューで生成したクエリもキャッシュします。

クエリ キャッシュは、**SELECT SQL_CALC_FOUND_ROWS ...** のクエリで動作し、後続する **SELECT FOUND_ROWS()** クエリで返る値を格納します。**FOUND_ROWS()** は前のクエリがキャッシュからフェッチしていても、正確な値を返します。これは、検索したレコードの数をキャッシュで保管しているためです。**SELECT FOUND_ROWS()** クエリ自体はキャッシュの対象ではありません。

次のテーブルに示す関数を含む場合は、どのようなクエリでもキャッシュの対象にはなりません。

BENCHMARK()	CONNECTION_ID()	CURDATE()
CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
CURTIME()	DATABASE()	ENCRYPT() (パラメータなし)
FOUND_ROWS()	GET_LOCK()	LAST_INSERT_ID()
LOAD_FILE()	MASTER_POS_WAIT()	NOW()
RAND()	RELEASE_LOCK()	SYSDATE()
UNIX_TIMESTAMP() (パラメータなし)	USER()	

次の条件がある場合のクエリもキャッシュの対象にはなりません。

- ユーザ定義関数 (UDF) または格納された関数 (stored functions) を指す場合
- ユーザ変数を指す場合
- **mysql** システム データベースのテーブルを指す場合
- 次に示す形のものである場合

```
SELECT ... IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

最後の例がキャッシュにならない理由は、それが ODBC のワークアラウンド (特別の手段) として、最後のインサート ID 値が存在するため。[24章MySQL コネクタ](#) の MyODBC セクションを参照のこと。

- 準備されたステートメントとして発行した場合。プレースホルダを採用していない場合も同様。例として、次のようなクエリはキャッシュにならない。

```
char *my_sql_stmt = "SELECT a, b FROM table_c";
/* ... */
mysql_stmt_prepare(stmt, my_sql_stmt, strlen(my_sql_stmt));
```

詳細は [準備されたC APIステートメント](#)。を参照してください。

- **TEMPORARY** テーブルを使用している場合
- テーブルを全く使用しない場合。
- 関連テーブルのすべてに対して、ユーザがカラム レベルの権限を持つ場合。

クエリ キャッシュでの **SELECT** オプション

SELECT ステートメントで、クエリ キャッシュ関連の 2 つのパラメータを指定することができます。

- [SQL_CACHE](#)

[query_cache_type](#) 環境変数の値が、[DEMAND](#) または [ON](#) のときは、クエリ結果をキャッシュします。

- [SQL_NO_CACHE](#)

クエリ結果をキャッシュしません。

例：

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

クエリ キャッシュの設定

[have_query_cache](#) というサーバのシステム変数は、クエリ キャッシュの利用可能にします。

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+
```

MySQL 標準バイナリを使用しているときは、この値は常に、[YES](#) です。クエリ キャッシュを無効化している場合も同様です。

システム変数によっては、クエリ キャッシュのオペレーション制御を行います。[mysqld](#) 起動時に、オプション ファイルやコマンドラインで指定できます。クエリ キャッシュのシステム変数名はすべて、[query_cache_](#) という文字で始まります。これに関する簡単な説明を [システム変数](#) で参照してください。このセクションでは、設定に関する情報を提供します。

クエリ キャッシュのサイズを設定するには、[query_cache_size](#) システム変数を使用します。値を 0 にすると、クエリ キャッシュは無効化します。デフォルトは 0 で設定しています。

注記

Windows Configuration Wizard を使用して、MySQL をインストールまたは設定するときは、利用可能なコンフィギュレーションの種類に合わせて、自動的に [query_cache_size](#) の値をデフォルト設定します。Windows Configuration Wizard を使用するときは、場合によって、クエリ キャッシュが有効化します。つまり、ゼロではない値になることがあります。クエリ キャッシュの制御は、[query_cache_type](#) 変数で行います。コンフィギュレーションを始めるときには、[my.ini](#) ファイルでこの変数の値を確認してください。

[query_cache_size](#) の値がゼロではない場合は、ストラクチャのアロケートにおよそ 40 KB を必要とするため、クエリ キャッシュのサイズを最低 40 KB でセットしてください。正確なサイズは、システムのアーキテクチャによります。サイズが小さすぎると、警告がでます。

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1282
Message: Query cache failed to set size 39936; »
        new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 41984 |
+-----+-----+
```

クエリ キャッシュでクエリ結果を維持するには、サイズを大きく設定してください。

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 999424 |
+-----+-----+
1 row in set (0.00 sec)
```

`query_cache_size` は、1024 バイトに近い値のブロックで位置合わせしています。そのため、報告の値は、設定したものとは異なる可能性があります。

クエリ キャッシュのサイズが 0 より大きい場合、`query_cache_type` 変数とその機能に影響します。この変数は次のような値に設定できます。

- 0 または **OFF** という値で、キャッシュを行わない、または キャッシュした結果の読み出しを行わない、という効果になります。
- 1 または **ON** という値で、`SELECT SQL_NO_CACHE` で始まるステートメント以外のキャッシュになります。
- 2 または **DEMAND** という値で、`SELECT SQL_CACHE` で始まるステートメントだけのキャッシュになります。

`GLOBAL query_cache_type` の値を設定すると、変更後に接続するクライアントに対するクエリ キャッシュの動作を指定できます。`SESSION query_cache_type` の値を設定すると、それぞれのクライアントで接続時のキャッシュ動作を制御できます。たとえば、クライアントは自分のクエリに対するクエリ キャッシュの使用を無効化します。そのクエリは次のようなものです。

```
mysql> SET SESSION query_cache_type = OFF;
```

キャッシュしたそれぞれのクエリ結果の最大サイズの制御は、`query_cache_limit` 変数で行います。デフォルトは 1 MB です。

クエリをキャッシュする設定である場合、その結果 (クライアントに送信したデータ) を、結果の読み出し中に、クエリ キャッシュに格納します。そのため、データの扱いは、ひとまとめでありません。つまり、クエリ キャッシュで、データをブロックに分割するため、1 つのブロックが埋まれば、次のブロックを埋めることになります。これは、メモリの割り当てに時間がかかるため、クエリ キャッシュではブロックにします。そのときのブロックのサイズを決定するのが、`query_cache_min_res_unit` 変数です。そして、クエリ実行毎に、ブロックはサイズでトリムすることになるので、メモリを節約できます。サーバで実行するクエリの種類によっては、`query_cache_min_res_unit` の値を調整すると効果が高まります。

- `query_cache_min_res_unit` のデフォルト値は、4 KB です。大抵の場合、これで十分です。
- 結果が小さいクエリが多い場合は、フリーのブロックが多く存在することになり、デフォルトのブロックサイズはメモリのフラグメントになります。フラグメントは、メモリ不足を解消するために、キャッシュからクエリを取り除く (削除) 動作を強制的に行います。そのため、`query_cache_min_res_unit` の値を減らす必要が出てきます。フリー ブロックの数は `Qcache_free_blocks` を、そして、この動作で強制的に削除の対象になったクエリは `Qcache_lowmem_prunes` を、それぞれのステータス変数で確認してください。
- クエリの大部分が大きな結果である場合は、`query_cache_min_res_unit` で値を増やして、パフォーマンスを改善できます。(`Qcache_total_blocks` および `Qcache_queries_in_cache` で、ステータス変数を確認してください。) しかし、値を増やすと、前述のようにメモリ不足の状態になるので、注意が必要です。

MySQL Enterprise

```
#### #####MySQL Network Monitoring and Advisory Service #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

クエリ キャッシュのステータスと保守

次のステートメントを使用して、MySQL サーバでクエリ キャッシュを行っているかどうかを確認できます。

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
```

```

+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+

```

メモリ不足対策で、クエリ キャッシュのフラグには、[FLUSH QUERY CACHE](#) ステートメントを使用します。このステートメントでは、キャッシュからクエリが消えることはありません。

[RESET QUERY CACHE](#)ステートメントは、クエリ キャッシュからクエリ結果を削除します。[FLUSH TABLES](#) ステートメントでも同様のことができます。

クエリ キャッシュのパフォーマンスを監視するには、[SHOW STATUS](#) を使用して、キャッシュのステータス変数を見ます。

```

mysql> SHOW STATUS LIKE 'Qcache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_free_blocks | 36 |
| Qcache_free_memory | 138488 |
| Qcache_hits | 79570 |
| Qcache_inserts | 27087 |
| Qcache_lowmem_prunes | 3114 |
| Qcache_not_cached | 22989 |
| Qcache_queries_in_cache | 415 |
| Qcache_total_blocks | 912 |
+-----+-----+

```

それぞれの変数に関する詳細は、[ステータス変数](#) を参照してください。

[SELECT](#) クエリの合計数は、次の計算式で求めます。

```

Com_select
+ Qcache_hits
+ queries with errors found by parser

```

[Com_select](#) 値は次の計算式で求めます。・

```

Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check

```

クエリ キャッシュでは、変数長さのブロックを使用するため、クエリ キャッシュのメモリ フラグメンテーションは、[Qcache_total_blocks](#) および [Qcache_free_blocks](#) で確認できます。[FLUSH QUERY CACHE](#) 後には、フリーのブロックが 1 つになります。

キャッシュするクエリには、少なくとも 2 つのブロックを必要とします。1 つはクエリ テキスト用で、もう 1 つはクエリ結果用です。さらに、もう 1 つ、テーブルのクエリ要求用にもブロックを必要とします。ただし、複数のクエリで同じテーブルを使用している場合は、1 ブロックの割り当てで済みます。

[Qcache_lowmem_prunes](#) システム変数の情報は、クエリのキャッシュ サイズを調節するときに役立ちます。この変数は、新しいクエリのキャッシュを入れるために取り除かれたクエリ数をカウントしています。クエリ キャッシュでは、古い順番にクエリをキャッシュから削除 (LRU) します。サイズの調節方法については、[クエリ キャッシュの設定](#) を参照してください。

第5章 レプリケーション

目次

レプリケーション設定	318
レプリケーションのセットアップ方法	319
レプリケーション フォーマット	326
レプリケーションのオプションと変数	330
レプリケーションでの管理タスク	336
Replication Topologies	338
Replication with a Single Slave	338
Replication with Multiple Slaves	338
Replication with Two Masters	339
Replication with Circular Masters	340
Replication Chains	340
Replicating Multiple Masters to One Slave	341
レプリケーション ソリューション	341
バックアップのレプリケーション	341
ストレージ エンジンが異なるマスタとスレーブのレプリケーション	343
スケールアウトのレプリケーション	344
異なるデータベースから異なるスレーブへのレプリケーション	346
レプリケーション パフォーマンスの改善	346
フェイルオーバーでのマスタ切り替え	348
SSLを使用するレプリケーションの設定	351
レプリケーション ノートとヒント	352
レプリケーション機能と既知問題	352
MySQL バージョン間のレプリケーション互換性	357
レプリケーション セットアップのアップグレード	357
レプリケーション FAQ	358
レプリケーションのトラブルシューティング	361
レプリケーション バグまたは問題を報告する方法	362
レプリケーションの実装	363
レプリケーション実装の詳細	363
マスタレプリケーションのスレッド状態	365
スレーブレプリケーションの I/O スレッド状態	365
スレーブレプリケーションの SQL スレッド状態	366
レプリケーション リレーとステータス ファイル	366
サーバのレプリケーション ルール評価	368

レプリケーションは1つのMySQLサーバ(マスタ)にあるデータベースを別のサーバ(スレーブ)に複製できるレプリケーション機能があります。レプリケーションは非同期、つまり複製スレーブをマスタから更新するときに常時接続である必要がなく、接続距離が離れていても更新することができ、ダイヤルアップなどの一時的なソリューションとしても利用することができます。コンフィギュレーションによっては、すべてのデータベースまたは選択したデータベースを複製でき、さらにデータベース内で選択したテーブルを取り込むこともできます。

以下は、MySQL のレプリケーションの使用例です。

- スケールアウト ソリューション - パフォーマンス向上のために複数のスレーブにロードを分散します。この環境では、すべての書き込みと更新をマスタサーバで実行する必要があります。読み込みの際には、一つ以上のスレーブでの実行が必要になる場合があります。このモデルは、スレーブ数が増加しても読み込みスピードが劇的に向上し、さらにマスタが更新専用になるために書き込み性能も向上します。
- データ セキュリティ - スレーブでデータを複製するため、スレーブは複製プロセスを一時停止することで、後続のマスタデータを破壊することなくスレーブにバックアップ サービスを実行できます。
- 分析 - マスタでライブ データの作成を行い、スレーブで情報分析を行うため、マスタのパフォーマンスに支障をきたしません。

- ・長距離データ配布 - マスタへの常時接続を不要とするため、別の場所でメイン データを利用したいときなどに、複製をデータのローカル コピーとして使用できます。

MySQL のレプリケーションは、一方向性の非同期複製のサポートを特徴とし、一つのサーバがマスタとして機能し、別のサーバがスレーブとして機能します。これは、MySQL クラスタ の特徴である同期複製とは対照的です。(参照 [14章MySQL Cluster](#))

二つのサーバ間でレプリケーションを設定するには、いくつかのソリューションがあり、データ形状や使用しているエンジン タイプにあわせて最適設定することが可能です。利用可能なオプションの詳細については、[レプリケーションのセットアップ方法](#) を参照してください。

レプリケーション形式には、SQL すべてのステートメントを複製する Statement Based Replication (SBR: クエリベース レプリケーション)、変更があった row (行列) だけを複製する Row Based Replication (RBR: 行ベース レプリケーション) の 2 種類があります。また、第 3 の形式、Mixed Based Replication (MBR) を使用することも可能です。これは、MySQL 5.1.14 以上のデフォルト モードです。レプリケーション形式の詳細については、[レプリケーションフォーマット](#) を参照してください。

レプリケーションは、様々なオプションと変数によってコントロールされます。これらは、レプリケーション、タイムアウト、データベース、フィルタなど、データベースやテーブルに適用する操作の中核をコントロールします。利用可能なオプション詳細については、[レプリケーションのオプションと変数](#) を参照してください。

パフォーマンス向上に関わる課題、異なるデータベースのバックアップ サポート、またはシステム不良を回避するための保全ソリューションの一部としてなど、様々な問題を解決するためにレプリケーションを活用できます。ソリューションに関する詳細については、[レプリケーション ソリューション](#) を参照してください。

レプリケーション機能の詳細、バージョン間の互換性、アップグレード、既知問題およびソリューション、FAQ など、レプリケーション作業中に異なるデータ タイプおよびクエリがどのように処理されるかに関する情報は、[レプリケーション ノートとヒント](#) を参照してください。

レプリケーションの実行、レプリケーションの手順、バイナリ ログ内容とその処理、バックグラウンドスレッド、そしてステートメント (クエリ) をどのように記録するかを決めるオプションに関する詳細は、[レプリケーションの実装](#) を参照してください。

MySQL Enterprise

MySQL Network Monitoring and Advisory Service #####
#####<http://www-jp.mysql.com/products/enterprise/advisors.html>#####

レプリケーション設定

MySQL のサーバ間におけるレプリケーションは、バイナリのロギング メカニズムを使用して行います。マスタ (データベース変更元) として機能している MySQL インスタンスは、データベースへの更新および変更をバイナリ ログに書き込みます。バイナリ ログの情報はデータベースでの変更記録に従い、異なるロギング フォーマットに格納されます。スレーブを設定して、マスタからのバイナリ ログを読み込み、そして、スレーブのローカル データベースにあるバイナリ ログでのイベント実行を行います。

このシナリオでのマスタには、データ処理能力がありません。バイナリ ロギングを実行可能にした後、すべてのステートメントはバイナリ ログへの記録になります。それぞれのスレーブがバイナリ ログの全内容のコピーを受信します。スレーブは、バイナリ ログのどのステートメントを実行するべきかを決定する役割を担い、マスタに特定のイベントだけをログするようには設定できません。指定がない限り、マスタのバイナリ ログのすべてのイベントがスレーブでの実行対象になります。必要に応じて、スレーブが特定のデータベースまたはテーブルに該当するイベントだけを処理するように設定できます。

スレーブは、バイナリ ログ ファイルと位置と読み込みおよび処理したログ ファイル内での記録を保持します。これは、複数のスレーブがマスタに接続することができ、同一のバイナリ ログで異なる部分を実行できます。スレーブはこのプロセスをコントロールするため、それぞれのスレーブがマスタと接続または未接続の状態でも、マスタのオペレーションに影響することはありません。そして、それぞれのスレーブがバイナリ ログ内の位置を記憶しているため、スレーブが未接続の場合でも、再接続して切断前の記録位置から継続してキャッチアップすることができます。

マスタとそれぞれのスレーブの両方をユニーク ID で設定 ([server-id](#) オプション) する必要があります。さらに、スレーブはファイル内のマスタ ホスト名、ログ ファイル名、位置などの情報で設定します。これらの詳細は、[CHANGE MASTER](#) を使用して、MySQL セッション内からコントロールできます。詳細は、[master.info](#) ファイル内にあります。

この章では、レプリケーション環境に必要なセットアップとコンフィギュレーションを示し、新たなレプリケーション環境を作成するためのステップ バイ ステップの手順が記されています。この章の主な内容は次の通りです。

- 2 つ以上のサーバでレプリケーションを行うためのガイドは、[レプリケーションのセットアップ方法](#) を参照してください。この章はシステムのセットアップについて説明し、スレーブとマスタ間でのデータ コピー方法を提供します。
- バイナリ ログのイベントはいくつかのフォーマットで記録します。このフォーマットをステートメント ベース レプリケーション (SBR) あるいは 行ベース レプリケーション (RBR) と呼びます。第三のフォーマットは、ミックス レプリケーション (MIXED) で、SBR と RBR レプリケーションを自動的に使い分け、適切に SBR と RBR の両方のフォーマットの利点を活用します。フォーマットに関しては、[レプリケーション フォーマット](#) を参照してください。
- レプリケーションにおける様々なコンフィギュレーションのオプションと変数に関する詳細は [レプリケーションのオプションと変数](#) を参照してください。
- レプリケーションを開始すると、そのプロセスで管理権限と監視が必要になります。実行するときの共通タスクに関するアドバイスは [レプリケーションでの管理タスク](#) を参照してください。

レプリケーションのセットアップ方法

この章は、MySQL サーバのレプリケーションを完全に行うためのセットアップについて説明します。レプリケーションのセットアップには様々な方法があり、レプリケーションをどのようにセットアップするか、そしてマスタのデータベースにデータがすでに存在するかどうかにより、その方法が変わります。

すべてのレプリケーション セットアップに必要なとされる一般的なタスクは次の通りです。

- 認証用に独立したユーザを作成する。これは、スレーブがマスタのバイナリ ログを複製するために読み込むときに使用する。[レプリケーション ユーザの作成](#) を参照。
- バイナリ ログをサポートするためにマスタを設定し、ユニーク ID を設定する。[レプリケーション マスタのコンフィギュレーション設定](#) 参照。
- マスタと接続するそれぞれのスレーブでユニーク ID を設定する。[レプリケーション スレーブのコンフィギュレーション設定](#) を参照。
- データ スナップショットまたはレプリケーションを開始する前に、マスタのバイナリ ログの位置を記録する。この情報はスレーブを設定するときに必要になり、これによりスレーブはバイナリ ログ内のどこからイベントを実行するかを認識する。[マスタ レプリケーション情報の取得](#) を参照。
- すでにマスタにデータがある場合、スレーブと元データを同期化し、データベースのデータ スナップショットを作成する。スナップショットには、[mysqldump](#) を使用する ([mysqldump を使用したデータ スナップショットの制作](#)) か、またはデータ ファイルを直接コピーする ([生データ ファイルでデータ スナップショットの作成](#))。
- Master のセッティングでスレーブを設定する。例：ホスト名、ログイン認証、バイナリ ログ名、位置など。[マスタ コンフィギュレーションのスレーブでの設定](#) を参照。

ここまでの基本的なオプションを設定後、次に示すレプリケーション セットアップの手順に従います。これにはいくつかの方法があります。

- 新たな MySQL マスタおよび複数のスレーブをセットアップする場合、交換するデータがまだ存在しないということから、コンフィギュレーションをセットアップする必要がある。この状況でのレプリケーション セットアップのガイドは [新たなマスタとスレーブのレプリケーション セットアップ](#) を参照。
- MySQL サーバがすでに稼働している場合、つまりレプリケーションを開始する前にスレーブにデータを転送する必要がある、バイナリ ログ構成をしていない状況下で、処理中に短時間 MySQL サーバをシャットダウンできるときは、[既存データでのレプリケーション セットアップ](#) を参照。
- 既存のレプリケーション環境で、追加サーバをセットアップ、そしてマスタに影響することなくスレーブをセットアップできる場合は、[既存のレプリケーション環境へのスレーブ追加](#) を参照。

MySQL レプリケーション セットアップの管理者は、この節を十分に読み、[マスタ サーバをコントロールする SQL ステートメント](#) および [スレーブ サーバをコントロールする SQL ステートメント](#) のすべてのステートメントを試行してください。[レプリケーションのオプションと変数](#) に示すレプリケーションのスタートアップ オプションに関しても十分な理解が必要です。

注記

ノートとして、セットアッププロセス中の特定のステップで、[SUPER](#) 権限を必要とします。この権限がない場合は、レプリケーションはできません。

レプリケーション ユーザの作成

それぞれの Slave は、通常のユーザ名とパスワードで Master と接続する必要があります。このオペレーションに使用するこのユーザは、[REPLICATION SLAVE](#) 権限を持つユーザのことです。

レプリケーションのために特定のユーザを作成する必要はありませんが、ユーザ名とパスワードは [master.info](#) ファイル内のテキスト ファイルに保存されるため、レプリケーション プロセスにだけ権限があるユーザを作成することです。

レプリケーションに必要な権限をユーザまたは既存のユーザに与えるには、[GRANT](#) ステートメントを使用します。レプリケーションのためだけにユーザを作成する場合は、そのユーザは [REPLICATION SLAVE](#) 権限だけを必要とします。たとえば、ユーザ作成時に、ドメイン [mydomain.com](#) 内のすべてのホストにレプリケーション接続を許可するには、[repl](#) を使用します。

```
mysql> GRANT REPLICATION SLAVE ON *.*
-> TO 'repl'@'%mydomain.com' IDENTIFIED BY 'slavepass';
```

[GRANT](#) ステートメントに関しては、[GRANT 構文](#) を参照してください。

スレーブ毎のユーザを作成する場合、または接続時にそれぞれのスレーブに同一のユーザを使う場合に、レプリケーション プロセスに使用するそれぞれのユーザに [REPLICATION SLAVE](#) 権限があるという前提であれば、必要に応じてユーザ作成ができます。

レプリケーション マスタのコンフィギュレーション設定

レプリケーションを成功させるには、必ず マスタのバイナリ ログを実行可能にしてください。バイナリ ログの実行できないということは、マスタとスレーブの間でデータ交換に使うバイナリ ログがないということになり、レプリケーションは不可能です。

レプリケーション グループのそれぞれのサーバには、ユニークな [server-id](#) が必要です。この [server-id](#) はそれぞれのサーバを識別するために使うため、1 から $(2^{32}-1)$ 間の正整数を使用します。採番方法は自由です。

これらのオプションを設定するには、MySQL サーバをシャットダウンし、[my.cnf](#) あるいは [my.ini](#) ファイルのコンフィギュレーションを編集します。

[\[mysqld\]](#) セクション内のコンフィギュレーション ファイルに次のルールを付加します。これらのルールがすでに存在し、コメントアウトしている場合は、そのルールを非コメント化し、必要に応じて書き換えます。

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

注記

ノート： トランザクションで [InnoDB](#) を使用したレプリケーションに、できる限りの耐用性と一貫性を期待する場合は、マスタの [my.cnf](#) ファイルの [sync_binlog=1](#) と [innodb_flush_log_at_trx_commit=1](#) を使用します。

注記

[skip-networking](#) ルールがレプリケーション マスタで無効であることを確認してください。ネットワークが使用不可の場合は、スレーブはマスタとの通信ができないため、レプリケーションは成功しません。

レプリケーション スレーブのコンフィギュレーション設定

スレーブで唯一設定しなければならないオプションは、ユニーク サーバ ID の設定です。このオプションを設定していない場合、またはマスタ サーバに指定した値と実行値が干渉する場合、スレーブ サーバをシャットダウンし、サーバ ID を指定するためにコンフィギュレーションを編集します。たとえば次のようにします。

```
[mysql]
server-id=2
```

複数のスレーブをセットアップする場合、それぞれにユニークな `server-id` 値を与えます。この値はマスタおよびその他のスレーブとは異なる必要があります。`server-id` 値は、IP アドレスのようなものと考えます。これらの ID はコミュニティ内のレプリケーション パートナー間で、それぞれのサーバ インスタンスを一意的に識別します。

`server-id` 値を指定しない場合、`master-host` を定義していなければ、この値は 1 です。それ以外は、2 で設定します。`server-id` の省略 する場合は、マスタがすべてのスレーブからの接続を拒否し、スレーブはマスタへの接続を拒否します。そのため、`server-id` を省略することは、バイナリ ログでのバックアップにのみ有効といえます。

レプリケーション用にスレーブのバイナリ ログイングを可能にする必要はありません。しかし、スレーブのバイナリ ログイングを可能にすると、データ バックアップとクラッシュ リカバリにバイナリ ログを使用でき、スレーブを接続形態が複雑なレプリケーションに使用できます。

マスタ レプリケーション情報の取得

スレーブのレプリケーションを設定するには、マスタのバイナリ ログ内でマスタの現在位置を特定する必要があります。この情報は、スレーブがレプリケーション プロセスを開始するときに必要とします。それにより、正確な位置でバイナリ ログからのイベントを開始できます。

マスタに既存のデータがあり、それをレプリケーション プロセスを開始する前にスレーブと同期化するには、マスタのステートメント処理を停止し、現在位置を取得して、マスタにステートメント実行の継続を許可する前に、そのデータをダンプします。もし、ステートメントの実行を停止しないでデータのダンプを行うと、マスタのステータス情報に不一致が生じ、スレーブのデータベースが破損します。

マスタのステータス情報は、次のステップに従い取得します。

1. コマンドライン クライアントを開始し、すべてのデータをフラッシュし、`FLUSH TABLES WITH READ LOCK` ステートメントを実行して書き込みステートメントをブロックする。

```
mysql> FLUSH TABLES WITH READ LOCK;
```

このとき大切なこととして、`InnoDB` テーブルでは、`FLUSH TABLES WITH READ LOCK` が、`COMMIT` オペレーションもブロックすることに留意してください。

警告

注意: 実行中の `FLUSH TABLES` コマンドからクライアントを切り離します。読み込みブロックはそのまま有効です。クライアントを終了すると、このブロックはリリースされます。

2. `SHOW MASTER STATUS` ステートメントを使用して、現在のバイナリ ログ名を指定し、マスタとオフセットする。

```
mysql> SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003 | 73       | test         | manual,mysql      |
+-----+-----+-----+-----+
```

`File` カラムはログの名前を示し、`Position` はファイル内のオフセットを示します。この例では、バイナリ ログ ファイルは `mysql-bin.003` で、オフセットは 73 です。これらの値は、後でスレーブをセットアップするときに必要なので、書き控えめします。マスタからの新たなアップデートを処理するスレーブのレプリケーション座標です。

バイナリ ログイングを行わない状態で、マスタが稼働していた場合、ログ名と位置の値は `SHOW MASTER STATUS` に示されるか、または `mysqldump --master-data` は空の状態です。この場合、後でスレーブのログ ファイルと位置を指定するときの値は 空文字列 ("") および 4 です。

ここで、バイナリ ログからの読み込みを開始し、正確な位置からレプリケーションを行う準備がスレーブにできます。

レプリケーションを開始する前に、スレーブと同期化する必要がある既存データがある場合、クライアントをそのまま稼働させます。これによりロックは正しい位置に留まり、[mysqldump を使用したデータ スナップショットの制作](#)または [生データ ファイルでデータ スナップショットの作成](#)へ進みます。

新たにマスタとスレーブのレプリケーション グループをセットアップする場合には、クライアントを終了し、ロックをリリースします。

mysqldump を使用したデータ スナップショットの制作

既存のマスタ データベースでデータのスナップショットを作成する方法の一つに、[mysqldump](#) ツールを使うことがあります。データのダンプが完了したら、レプリケーション プロセスを開始する前に、そのデータをスレーブにインポートします。

[mysqldump](#) を使用してデータのスナップショットを取得する方法

- データを更新するクエリの実行を防ぐために、サーバのテーブルをまだロックしていない場合

コマンドライン クライアントを開始し、すべてのデータをフラッシュし、[FLUSH TABLES WITH READ LOCK](#) ステートメントを実行して書き込みステートメントをブロックする。

```
mysql> FLUSH TABLES WITH READ LOCK;
```

注意：[SHOW MASTER STATUS](#) を使用し、スレーブをスタートアップするときに使うバイナリ ログの詳細を記録してください。このときのスナップショットとバイナリ ログの位置は一致する必要があります。詳細は、[マスタ レプリケーション情報の取得](#)を参照。

- 別のセッションでは、[mysqldump](#) を使用して、データベースすべて、または複製する分、あるいは個別に特定のデータベースを選択して、ダンプを作成する。

```
shell> mysqldump --all-databases --lock-all-tables >dbdump.db
```

- ベア ダンプの別方法としては、[--master-data](#) オプションを使用し、自動的に スレーブでのレプリケーション プロセスに必要な [CHANGE MASTER](#) ステートメントを付加する。

```
shell> mysqldump --all-databases --master-data >dbdump.db
```

ダンプを含むデータベースを選択する場合は、レプリケーション プロセスには不要なスレーブのデータベースにフィルタをかける必要があります。

データをインポートするときに、スレーブへ遠隔的に接続する場合は、ダンプ ファイルをスレーブにコピーするか、またはマスタからのファイルを使用します。

生データ ファイルでデータ スナップショットの作成

データベースが特段に大きい場合は、[mysqldump](#) を使用してそれぞれのスレーブにファイルをインポートするよりも、生データ ファイルをコピーする方が効率的な場合があります。

しかし、複雑なキャッシュやロギング アルゴリズムを使用しているストレージ エンジンのテーブルにこの方法を使うことは、完全にイン タイムのスナップショットにならない可能性があり、キャッシュ情報とロギング アップデートは、グローバル 読み込みロックを使っていたとしても、適用されない場合があります。これにストレージ エンジンがどのように反応するかは、クラッシュ リカバリ能力に依存します。

たとえば、グローバル 読み込みロックを使用していた場合、[InnoDB](#) テーブルのファイルシステム スナップショットを開始できます。内部的 ([InnoDB](#) ストレージ エンジンの中) には、[InnoDB](#) キャッシュをフラッシュしていないなどの理由で、スナップショットが乱れます。しかし、これはスタートアップ時に [InnoDB](#) によって解決され、一貫した結果が運ばれるため、問題になることはありません。つまり、[InnoDB](#) は破損を伴わずに、クラッシュ リカバリを行うことができる、ということですが、しかし、これは、[InnoDB](#) テーブルの一貫したスナップショットを確保する一方で、MySQLサーバをストップすることができないということです。

生データのスナップショットを作成するには、[cp](#) または [copy](#) などの標準のコピー ツール、[scp](#) or [rsync](#) などのリモート コピー ツール、[zip](#) or [tar](#) などのアーカイブ ツール、[dump](#) などのファイル システム スナップショット ツールなどを使用し、MySQL データ ファイルが単一のファイルシステムに存在すると定めます。特定のデータベースを複製するだけである場合は、テーブルに関係のあるファイルのコピーを取るだけであることを確認します。[InnoDB](#)

で、`innodb_file_per_table` オプションを利用しない場合は、すべてのデータベースにあるすべてのテーブルを一つのファイルに格納します。

アーカイブから次のファイルを指定して取り除く場合

- `mysql` データベースに関連するファイル
- `master.info` ファイル
- マスタのバイナリ ログ ファイル
- リレー ログ ファイル

生データのスナップショットで最も一貫した結果を得るには、次の通りにプロセス中にサーバをシャットダウンします。

1. 読み込みロック、マスタ ステータスを取得する。[マスタ レプリケーション情報の取得](#) 参照。
2. 別のセッションで、MySQL サーバをシャットダウンする。

```
shell> mysqladmin shutdown
```

3. MySQL データ ファイルのコピーを取る。一般的なソリューションは次の例示の通り。この中から一つだけを選択する。

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. マスタの MySQL インスタンスを立ち上げる。

データベースをシャットダウンしないで、マスタからスナップショットを得る。

1. 読み込みロック、マスタ ステータスを取得する。[マスタ レプリケーション情報の取得](#) 参照。
2. MySQL データ ファイルのコピーを取る。一般的なソリューションは次の例示の通り。この中から一つだけを選択する。

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

`InnoDB` テーブルを使用している場合、`InnoDB Hot Backup` ツールの使用をお勧めします。これは、マスタサーバのロックを取らずに一貫したスナップショットを取り、後にスレーブで使用するスナップショットに関連するログ名とオフセットを記録します。`Hot Backup` は業務用ツールであるため、標準の MySQL には含まれていません。詳細は、<http://www.innodb.com/manual.php> で、`InnoDB Hot Backup`を参照してください。

3. 読み込みロックを取得したクライアントでは、ロックを解除する。

```
mysql> UNLOCK TABLES;
```

データベースのアーカイブまたはコピーを作成した後は、スレーブでレプリケーション プロセスを開始する前に、それぞれのスレーブにファイルをコピーします。

新たなマスタとスレーブのレプリケーション セットアップ

既存データがない場合などで、レプリケーションをセットアップする最も簡単な正攻法は、新たな Master と Slaves でセットアップすることです。

この方法は、新しいサーバをセットアップする場合に、レプリケーションのコンフィギュレーションにロードしたいデータベースの既存ダンプがあるときにも有効です。新しいマスタにデータをロードすると、データは自動的にサーバへ複製されます。

新しいマスタとスレーブでレプリケーションをセットアップする方法

1. 必要なコンフィギュレーション属性で MySQL マスタを設定する。[レプリケーション マスタのコンフィギュレーション設定](#) を参照。

2. MySQL マスタを起動する。
3. ユーザをセットアップする。 [レプリケーション ユーザの作成](#) を参照。
4. マスタのステータス情報を取得する。 [マスタ レプリケーション情報の取得](#) を参照。
5. 読み込みロックを解除する。

```
mysql> UNLOCK TABLES;
```

6. スレーブで、MySQL コンフィギュレーションを編集する。 [レプリケーション スレーブのコンフィギュレーション設定](#) を参照。
7. MySQL スレーブを起動する。
8. [CHANGE MASTER](#) コマンドを実行し、マスタ レプリケーション サーバのコンフィギュレーションを設定する。

ロードまたは交換するデータが新しいサーバのコンフィギュレーションにないため、情報をコピーまたはインポートする必要はありません。

既存のデータベース サーバからのデータを使用して、新たなレプリケーション環境をセットアップする場合は、ここでマスタでダンプ ファイルを実行します。データベースの更新は自動的にスレーブへ伝播されます。

```
shell> mysql -h master < fulldb.dump
```

既存データでのレプリケーション セットアップ

既存データでレプリケーションをセットアップするとき、レプリケーションを開始する前に、マスタからスレーブへの最も適当なデータ取得方法を検討します。

既存データでの基本的なレプリケーション セットアップ プロセスは次の通り。

1. [server-id](#) とバイナリ ロギングの設定をまだ行っていない場合、これらのオプションを設定するためにマスタをシャットダウンする。 [レプリケーション マスタのコンフィギュレーション設定](#) を参照。

マスタのデータベースをシャットダウンする必要があるということは、データベースのスナップショットを取る良い機会です。データベースの取り出し、コンフィギュレーションの更新、スナップショット取得などの前に、まず、マスタ ステータスを取得します。([マスタ レプリケーション情報の取得](#) 参照。) 生データ ファイルを使用したスナップショットの作成に関しては [生データ ファイルでデータ スナップショットの作成](#) を参照してください。

2. サーバがすでに正確に設定されている場合は、マスタ ステータスを取得し ([マスタ レプリケーション情報の取得](#) 参照)、 [mysqldump](#) を使用してスナップショットを取る ([mysqldump を使用したデータ スナップショットの制作](#) 参照)、または [生データ ファイルでデータ スナップショットの作成](#) のガイドを使用してライブ データベースの生スナップショットを取る。
3. MySQL マスタが稼動している状態で、レプリケーション中にスレーブがマスタへ接続するときに使うユーザを作成する。 [レプリケーション ユーザの作成](#) 参照。
4. スレーブのコンフィギュレーションを更新する。 [レプリケーション スレーブのコンフィギュレーション設定](#) 参照。
5. マスタにあるデータのスナップショットをどのように取るかによって、次のステップは異なります。

[mysqldump](#) を使用した場合

- a. [--skip-slave](#) オプションを使用してレプリケーションをスキップし、スレーブを立ち上げる。
- b. ダンプ ファイルをインポートする。

```
shell> mysql < fulldb.dump
```

生データ ファイルでスナップショットを作成した場合

- a. スレーブのデータ ディレクトリにデータ ファイルを展開する。

```
shell> tar xvf dbdump.tar
```

ノート：スレーブのコンフィギュレーションと一致するようにファイルの権限と所有権の設定が必要な場合があります。

b. `--skip-slave` オプションを使用してレプリケーションをスキップし、スレーブを立ち上げる。

6. マスタ ステータス情報でスレーブを設定します。これにより、レプリケーションを開始するために必要なバイナリ ログ ファイルと位置 (ファイル内) を伝え、ログイン認証とマスタのホスト名を設定します。ここで必要なステートメントに関する詳細は、[マスタ コンフィギュレーションのスレーブでの設定](#) を参照してください。

7. スレーブ スレッドを立ち上げる。

```
mysql> START SLAVE;
```

この手順を行った後、スレーブはマスタに接続し、最後にスナップショットからのアップデートにキャッチアップします。

マスタに `server-id` オプションの設定をしていなかった場合、スレーブは接続できません。

スレーブに `server-id` オプションを設定していなかった場合、スレーブのエラー ログに次のようなエラーが示されます。

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

何らかの理由で、複製ができない場合は、スレーブのエラー ログにエラー メッセージがあります。

スレーブでの複製開始後、そのデータ ディレクトリに `master.info` と `relay-log.info` という名前のファイルをそれぞれ見つけることができます。スレーブはこれら 2 つのファイルからマスタのバイナリ ログでどれくらい処理されたかを読み取ります。そのため、動作への影響に関して完全に理解している場合を除いて、これらのファイルは決して削除または編集しないでください。その必要がある場合は、[CHANGE MASTER TO](#) ステートメントを使用して、レプリケーションのパラメータを変更することをお勧めします。スレーブは、ステートメントで指定した値に従い自動的にステータス ファイルを更新します。

注記

`master.info` の内容は、コマンドラインまたは `my.cnf` などで指定したサーバ オプションの一部を優先します。詳細は [レプリケーションのオプションと変数](#) を参照してください。

マスタのスナップショットの準備が整ったら、上記に示したスレーブ部分の手順に従って、別のスレーブのセットアップにそれを使用します。マスタから新たに別のスナップショットを作成する必要はありません。それぞれのスレーブに同一のスナップショットを使用できます。

既存のレプリケーション環境へのスレーブ追加

既存のレプリケーション コンフィギュレーションにスレーブを追加する場合には、マスタを止める必要はありません。スレーブ (複数) のセッティングを複製します。

スレーブを複製する方法

1. 既存スレーブをシャットダウンする (Slave A)

```
shell> mysqladmin shutdown
```

2. 既存スレーブから新スレーブにデータ ディレクトリをコピーする。これは、`tar` または `WinZip` などの使用したアーカイブを作成するか、もしくは `cp` または `rsync` などのツールを使用して直コピーを実行するかのどちらかで行う。さらに、ログ ファイルやリレー ログ ファイルをコピーしておく。
3. `master.info` または `relay.info` ファイルを既存スレーブからコピーする。これらのファイルはその段階でのログ位置を保持している。
4. 既存スレーブを起動する。
5. 新スレーブでは、コンフィギュレーションを編集し、新スレーブに新ユニーク `server-id` を与える。

6. 新スレーブを起動する。[master.info](#) ファイルのルールでレプリケーション プロセスが開始される。

マスタ コンフィギュレーションのスレーブでの設定

レプリケーションで、スレーブをマスタと通信するようセットアップするには、スレーブに必要な接続情報を伝える必要があります。それには、次のステートメントをスレーブで実行して、オプション値をシステムに合わせた実際の値と置き換えます。

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='master_host_name',
-> MASTER_USER='replication_user_name',
-> MASTER_PASSWORD='replication_password',
-> MASTER_LOG_FILE='recorded_log_file_name',
-> MASTER_LOG_POS=recorded_log_position;
```

次のテーブルは文字列値オプションの最大許容長さを示します。

MASTER_HOST	60
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255

レプリケーション フォーマット

バイナリ ログに書き込まれたイベントをマスタが読み込み、スレーブで処理することでレプリケーションが成り立ちます。このイベントは記録されるイベントに従って様々なフォーマットで記録されます。このフォーマットは次の通りです。

- MySQL のレプリケーション能力は、マスタからスレーブへの SQL ステートメントの伝播に基づいています。これをステートメント ベース レプリケーション (SBR) と呼びます。
- 行ベースのレプリケーション (RBR) では、マスタがイベントをバイナリ ログに書き込み、ログは個々のテーブル行がどのように影響を受けたかを示します。MySQL 5.1.5 で追加された RBR に関するサポートは [を参照してください](#)。
- MySQL 5.1.8 より、第 3 のオプションが利用可能になりました。ミックス ベース レプリケーション (MBR) です。MBR では、デフォルトでステートメント ベース レプリケーションが行われますが、自動的に行ベース レプリケーションに切り替わります。次のケースがそれに該当します。[ミックス レプリケーション フォーマット](#) も参照してください。

MySQL 5.1.12 から、ミックス ベース レプリケーション (MBR) がデフォルト フォーマットで、指定のない限り、すべてのレプリケーション環境に対応します。

ステートメント ベースと行ベースのレプリケーション比較で、それぞれのメリット、デメリットを確認できます。詳細は [ステートメント ベースと行ベースのレプリケーション比較](#) を参照してください。

MySQL クラスタ レプリケーション (MySQL Cluster Replication) は行ベース レプリケーションに最適です。詳細は [MySQL Cluster レプリケーション](#) を参照してください。

MySQL の典型的なステートメント ベース レプリケーションには、格納ルーチンやトリガを複製するときに問題が生じる可能性があります。これらの問題は、MySQL の行ベース レプリケーションを代用して回避できます。問題に関する詳細は [ストアドルーチンとトリガのバイナリログ](#) を参照してください。

MySQL をソースから構築した場合、`--without-row-based-replication` オプションで `configure` を呼び出さない限り、行ベース レプリケーションはデフォルトで使用できます。

レプリケーション フォーマットのセッティング

デフォルトのレプリケーション フォーマットは、使用している MySQL のバージョンによって異なります。

- MySQL 5.1.11 以前の場合は、ステートメント ベース レプリケーションがデフォルトです。
- MySQL 5.1.12 以降の場合は、ミックス ベース レプリケーションがデフォルトです。

`--binlog-format=type` オプションにフォーマットを指定すると、デフォルトのレプリケーション フォーマットを強制できます。その場合、サーバに接続しているすべてのレプリケーション スレーブは、そのセッティングに従ってイベントを読み込みます。サポートされているオプションは次の通りです。

- **ROW** — は行ベース レプリケーションをデフォルトに設定。
- **STATEMENT** — はステートメント ベース レプリケーションをデフォルトに設定。MySQL 5.1.11 以前のフォーマット。
- **MIXED** — はミックス ベース レプリケーションをデフォルトに設定。MySQL 5.1.12 以降のフォーマット。

ロギング フォーマットはランタイムでも変更できます。すべてのクライアントに対して、グローバル フォーマットを指定するには、`binlog_format` システム変数のグローバル値を設定します。グローバル変数を変更するには、**SUPER** 権限が必要です。

ステートメント ベース フォーマットに切り替えるには、次のステートメントのどれかを使用します。

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 1;
```

行ベース フォーマットに切り替えるには、次のステートメントのどれかを使用します。

```
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 2;
```

ミックス ベース フォーマットに切り替えるには、次のステートメントのどれかを使用します。

```
mysql> SET GLOBAL binlog_format = 'MIXED';
mysql> SET GLOBAL binlog_format = 3;
```

それぞれのクライアントは、それぞれのステートメントのロギング フォーマットをコントロールできます。それには、`binlog_format` のセッション値を設定します。次はその例です。

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

ロギング フォーマットの手動による切り替えのほかに、スレーブ サーバが自動的に そのフォーマットを変更する場合もあります。これは、サーバが **STATEMENT** または **MIXED** のフォーマットのどちらかで実行しているときに生じ、**ROW** ロギング フォーマットでバイナリ ログの書き込みをしている行と衝突します。この場合、スレーブは一時的にそのイベントに合わせて行ベース レプリケーションに移行し、その後は元のフォーマットに戻ります。

接続毎でのレプリケーション ロギングの設定には、2 つの検討事項があります。

- データベースに多少の変更を加えるスレッドには、行ベース ロギングが妥当である。**WHERE** 節に一致するアップデートを行うスレッドには、数の多い行をログするよりもステートメントの方が効率的であるためステートメントベース ロギングが妥当である。
- マスタにおいて多くの実行を必要とするステートメントがあり、それによる修正行が少ない場合、それらを行ベース ロギングで複製する方が有益である。

ランタイムでレプリケーション フォーマットを切り替えることができない場合があります。

- 格納関数またはトリガ内からの場合
- **NDB** が有効な場合
- セッションが行ベース レプリケーション モードであり、一時テーブルを開いている場合

これらのケースでフォーマットを変更すると、エラーになります。

一時テーブルが存在する場合に、ランタイムでのレプリケーション フォーマットの切り替えることはしないでください。ステートメント ベース レプリケーションを使用しているときにだけ一時テーブルはログでき、行ベース レプリケーションの場合にはログできません。ミックス レプリケーションの場合は、一時テーブルはログできますが、ユーザ定義関数 (UDF) および **UUID()** 関数を使用している場合はこの限りではありません。

ROW にセットした `binlog` フォーマットでは、行ベースのフォーマットを使用してバイナリ ログに多くの変更が書き込まれます。しかし、変更の一部はステートメント ベース フォーマットである場合があります。たとえば、**CREATE**

TABLE、ALTER TABLE、DROP TABLE など DDL (データ定義言語) ステートメントを含む場合がこれに該当します。

--binlog-row-event-max-size オプションは行ベース レプリケーションができるサーバで使用できます。行はオプション値を越えないバイト サイズを一塊としてバイナリ ログに格納されます。この値は 256 の倍数です。デフォルト値は 1024 です。

警告

データ修正が non-deterministic であるようにステートメントがデザインされていた場合、行ベース レプリケーション を使用するとき、マスタとスレーブにあるデータをそれぞれ異なるものにすることができます。つまり、クエリ オプティマイザの意向次第ということです。レプリケーション以外の目的で、これを行うことは一般的ではありません。詳細は、[Open Issues in MySQL](#) を参照してください。

ミックス レプリケーション フォーマット

MIXED モードで実行している場合、次の条件下にあるレプリケーションはステートメント ベースから行ベースに自動的に切り替わります。

- DML ステートメントが NDB テーブルを更新するとき
- 関数に UUID() が含まれているとき
- AUTO_INCREMENT カラムを伴う 2 つ以上のテーブルを更新するとき
- INSERT DELAYED を実行するとき
- ビュー ボディが行ベース レプリケーションを要求し、そのビューを作成しているステートメントがそれを使用するとき — たとえば、ビューを作成しているステートメントが UUID() 関数を使用するとき
- UDF の呼び出しに関わる時

ステートメント ベースと行ベースのレプリケーション比較

バイナリ ロギングのフォーマットにはそれぞれ、メリットとデメリットがあります。大抵の場合、ミックス ベース レプリケーションのフォーマットで対応でき、データの整合性とパフォーマンスにおいては最適なコンビネーションです。しかし、特定のアップデートや大量データの挿入などを行うときに、レプリケーション フォーマットの違いをメリットとして扱う場合などがあります。そのため、この章では、行ベースとステートメント ベースのそれぞれのフォーマットにおけるメリットとデメリットを概説します。

ステートメント ベース レプリケーションのメリット

- バージョン 3.23 以来、MySQLに存在する実証済みテクノロジー
- ログ ファイルが小さい。更新や削除が数多くの行に影響する場合は、ログ ファイルが一段と 小さくなる。少量ログ ファイルはストレージ スペースを節約でき、バックアップも早くできる。
- ログ ファイルには変更があったすべてのステートメントが含まれるため、データベースの監査に使える。
- ログ ファイルはポイント イン タイムのリカバリなど、レプリケーション目的以外にも使える。[任意時点のリカバリ](#) 参照。
- テーブルの行ストラクチャが異なる場合でも、マスタで使っているものよりも新しいバージョンを使用しているスレーブを使用できる。これはマスタのアップグレードはできないが、スレーブの最新バージョンに備わっている機能を活用できるなどの有用性がある。これは、テストや評価などの目的としても有効である。

ステートメント ベース レプリケーションのデメリット

- UPDATE ステートメントのすべてを複製することができない。非決定性の動作 (例: SQL ステートメントのランダム関数使用時など) は、ステートメント ベース レプリケーションを使用している場合は複製が困難である。非決定性のユーザ定義関数 (UDF) を使用したステートメントの場合、行ベース レプリケーションでは UDF の戻り値を複製するだけであることに対して、ステートメント ベース レプリケーションでの結果は複製することができない。
- 非決定性の UDF を使用している場合に、ステートメントが適切に複製されない。(値が与えられたパラメータよりも別のファクタに依存する。)

- 次の関数を使用するステートメントは正確な複製にならない。
 - `LOAD_FILE()`
 - `UUID()`
 - `USER()`
 - `FOUND_ROWS()`
 - `SYSDATE()` (`--sysdate-is-now` オプションでサーバを起動した場合を除く)
 これ以外の関数での複製は正確である。(`RAND()`、`NOW()`、`LOAD DATA INFILE` など)
- `INSERT ... SELECT` は、行ベース レプリケーションのときよりも、行レベルのロック数をより必要とする。
- (`WHERE` 節でインデックスを使用していないなどの理由で、テーブル スキャンを必要とする `UPDATE` ステートメントは、行ベース レプリケーションのときよりも行数をより多くロックしなければならない。
- InnoDB の場合、`AUTO_INCREMENT` を使用する `INSERT` ステートメントは、干渉しない `INSERT` ステートメントなどもブロックする。
- 複雑なクエリの場合、ステートメントの評価および行の更新または挿入を行う前にスレーブで実行する必要がある。行ベース レプリケーションでは、スレーブはクエリ全体ではなく、部分的に違いを適用するためだけに、ステートメントを実行する。
- 格納機能 (格納プロシージャではない) を呼び出しステートメントと同一の `NOW()` 値で実行する。(これには良し悪しがある)
- 決定性のある UDF をスレーブに適用しなければならない。
- スレーブでの評価にエラーがあった場合、特に複雑なクエリを実行しているときには、ステートメント ベース レプリケーションでは、行への影響があるエラーのマージンが時間をかけてゆっくり増加することがある。
- マスタとスレーブは殆ど同一でなければならない。

行 ベース レプリケーションのメリット

- すべて複製できる。最も安全なレプリケーションの形式である。
- 5.1.14 以前のバージョンの MySQL では、`CREATE TABLE` のような DDL (データ定義言語) ステートメントはステートメント ベース レプリケーションを使用して複製します。一方の DML (Data Manipulation Language) ステートメントの場合は、`GRANT` や `REVOKE` ステートメントと同様にレプリケーションを使用した複製です。
- MySQL 5.1.14 以降では、`mysql` データベースは複製されません。`mysql` データベースはノード指定型データベースとして考えます。行ベース レプリケーションはこのテーブルをサポートしません。その代わりに、`GRANT` や `REVOKE` といった通常、情報を更新するステートメント、操作トリガ、格納ルーチン/プロシージャ、そしてビューなどすべてをステートメント ベース レプリケーションでスレーブへ複製します。
- `CREATE ... SELECT` のようなステートメントの場合は、`CREATE` ステートメントがテーブル定義から生成され、ステートメント ベースの複製、一方、行挿入は行ベースです。
- このテクノロジーは他のデータベース管理システムとほぼ同じで、別システムに関する知識は MySQL でも使える。
 - 多くの場合、主キーを保持しているテーブルにはスレーブにデータを適用する方が速い。
 - 次のタイプのステートメントでは、マスタのロック数が少ない (高い同時並行性) 。
 - `INSERT ... SELECT`
 - `AUTO_INCREMENT` で `INSERT` ステートメント
 - キーを使用しない、またはチェック済み行の殆どを変更しない。 `WHERE` 節で `UPDATE` または `DELETE` ステートメント。

- `INSERT`、`UPDATE`、`DELETE` ステートメント へのスレーブのロック数が少ない。
 - 将来的に、データをスレーブに適用する複数のスレッドを加えることができる。(SMP マシンとの相性が良い)
- 行 ベース レプリケーションのデメリット
- ログ ファイルが大きい (ケースによってはかなり大きい)
 - バイナリ ログにはロールバックした大きなステートメントが含まれる。
 - ステートメントを複製するために、行ベース レプリケーションを使用するときに (例: `UPDATE` または `DELETE` など)、変更された行のそれぞれがバイナリ ログに書き込まなければならない。一方では、ステートメント ベース レプリケーションを使用する場合は、そのステートメントだけがバイナリ ログに書き込まれる、ステートメント が多くの行を変更する場合、行ベース レプリケーションはバイナリ ログのより多くのデータを書き込む可能性がある。これらのケースでは、バイナリ ログはデータを書き込むために長時間ロックされ、これは、同時並行性の問題を偶発する。
 - 大きな `BLOB` 値を生成する決定性 UDF は複製速度を著しく低下させる。
 - どのステートメントが実行できたかを調べるためにログをチェックすることができない。
 - スレーブがマスタからどのステートメントを受信し、実行したかを知ることができない。
 - 非トランザクション ストレージ エンジンを含め、バルク オペレーションを行う場合、変更はステートメントが実行するものとして適用される。これは、行ベース レプリケーション ロギングの場合、バイナリ ログがステートメント実行中に書き込まれることを示す。一方、マスタでは、テーブルはバルク オペレーションが済むまでロックされているため、これが同時並行性に影響を与えることはない。しかし、スレーブ サーバでは、これらの変更はバルク オペレーションの一部ということを認識しないため、スレーブが変更を適用している間にテーブルがロックされない。

このシナリオでは、`SELECT * FROM table_name` などで、マスタのテーブルからデータを取り戻す場合に、サーバは `SELECT` ステートメントを実行する前に、バルク オペレーションの完了を待機します。これは読み込むテーブルがロックされているためです。スレーブでは、ロックされていないため、サーバは待機しません。これは、スレーブでの「バルク オペレーション」が完了するまで、同一の `SELECT` クエリから異なる結果がマスタとスレーブに生じるということです。

この動作は、最終的に変更しますが、それが実現するまでは、このようなシナリオに至る可能性がある場合は、ステートメント ベース レプリケーションを行うことをお勧めします。

レプリケーションのオプションと変数

ここでは、スレーブ レプリケーション サーバに使用するオプションを説明します。これらのオプションはコマンドラインまたはオプション ファイルで指定します。

マスタとスレーブ (複数) では、`server-id` オプションを使用して、ユニークなレプリケーション ID を設置します。それぞれのサーバには、1 から $2^{32}-1$ までの範囲のユニークな正整数を使用します。それぞれの ID は別の ID と重複しないようにしてください。例: `server-id=3`

バイナリ ロギングをコントロールするためにマスタ サーバに使用できるオプションは [バイナリ ログ](#) を参照してください。

スレーブ サーバレプリケーション オプションのいくつかは、特別の方法で扱います。それぞれが無視される、という意味においては、スレーブの起動時に `master.info` ファイルが存在し、オプションの値を含む場合です。次のオプションはこのように扱います。

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`

- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

MySQL 5.1 の `master.info` ファイル フォーマットには、SSL オプションに対応する値を含みます。さらに、ファイルフォーマットは、その最初のラインとしてラインの数をファイルに含みます。(レプリケーション リレーとステータス ファイル 参照) 古いバージョン (MySQL 4.1.1以前) から新しいバージョンにアップグレードする場合は、新しいサーバは `master.info` ファイルを新しいフォーマットへ起動とともに自動的にアップグレードします。しかし、新しいサーバを古いバージョンにダウングレード (格下げ) する場合には、古いサーバを起動させる前に、まず手動で最初のラインを取り除く必要があります。

`master.info` ファイルが存在しない状態でスレーブ サーバを起動する場合には、オプション ファイルまたはコマンドラインで指定されたルールの値が適用されます。これは、一番最初にレプリケーション スレーブとしてサーバを起動するとき、または `RESET SLAVE` の実行後にスレーブをシャットダウンし再起動した場合などに起こります。

`master.info` ファイルがスレーブ起動時に存在する場合は、サーバはそのファイル内にあるものを使用し、ファイルにリストされた値に呼応するオプションを無視します。このため、`master.info` の値に呼応するスタートアップ オプションとは異なる値でスレーブ サーバを起動する場合には、その異なる値が影響を与えることはありません。つまりサーバは `master.info` ファイルを使用し続けるということです。異なる値を使用するには、`master.info` ファイルを取り除いてから再起動するか、または `CHANGE MASTER TO` ステートメントを使用して、スレーブ実行中に値をリセットすることをお勧めします。

`my.cnf` ファイルで次のルールを指定したとします。

```
[mysqld]
master-host=some_host
```

レプリケーション スレーブとして初めてサーバを起動するとき、そのサーバは `my.cnf` ファイルからのオプションを読み込み、使用します。そして、`master.info` ファイルの値を記録します。そして、この次にサーバを起動するときには、`master.info` ファイルからの値をマスタ ホスト値として読み込み、オプション ファイルの値は無視されます。`my.cnf` ファイルを修正する場合に、`some_other_host` で別のサーバ ホストを指定するときには、この変更は反映されません。よって、`CHANGE MASTER TO` を使用します。

サーバが、記述したばかりのスタートアップ オプションよりも、既存 `master.info` ファイルを優先するため、これらの値をスタートアップ オプションに使用するよりも、`CHANGE MASTER TO` ステートメントを使用して値を指定する方が賢明です。詳細は、`CHANGE MASTER TO` 構文 を参照してください。

以下は、スレーブ サーバを設定するときのスタートアップ オプションを拡大的にしようした場合の例です。

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

次のリストは、レプリケーションをコントロールするオプションと変数について説明します。これらのオプションの多くは、`CHANGE MASTER TO` ステートメントを使用してサーバ実行中にリセットできます。しかし、`--replicate-*` のようなオプションは、スレーブ サーバが起動するときにだけセットできます。

- `--log-slave-updates`

スレーブは通常、マスタ サーバから受けるアップデートを自身のバイナリ ログに記録しない。つまり、スレーブの SQL スレッドで実行された更新を、スレーブのバイナリログに記録するようにスレーブに指示する。このオプションを有効にするには、バイナリログを有効にする `--log-bin` オプションを使用して、スレーブを起動する必要がある。

ある。レプリケーション サーバをチェーン状に構成するには、`--log-slave-updates` を使用する。たとえば、次のようにレプリケーション サーバをセットアップできる。

A → B → C

A はスレーブ B のマスタとして機能し、B はスレーブ C のマスタとして機能する。この構成では、B がマスタでもあり、スレーブでもある。そのため、A と B は両方とも、`--log-bin` オプションでバイナリログを有効にして起動し、`--log-slave-updates` オプションで B を起動する必要がある。これにより、A から受けたアップデートは B のバイナリ ログに記録される。

- `--log-warnings[=level]`

このオプションは、スレーブにより詳細なメッセージを出力させる。たとえば、ネットワークまたは接続が切断された後で再接続に成功したというメッセージや、それぞれのスレーブ スレッドがどのように開始したかについての情報メッセージを出力することができる。このオプションはデフォルト設定。これを無効にするには、`--skip-log-warnings` オプションを使用する。中断された接続は、その値が 1 を越えない限り、エラー ログには記録されない。

- `--master-connect-retry=seconds`

マスタがダウンするか接続不可の場合にマスタへ再接続を試行する前に、スレーブ スレッドがスリープ状態になる秒数。`master.info` ファイルの値が読み込める場合、その値が優先される。設定しなければ、デフォルトで 60 秒。`--slave-net-timeout` の値に基づくマスタからのデータ読み込みに対してタイムアウトするまで、スレーブによる再接続の自動呼び出しは行われない。再接続の試行の回数は、`--master-retry-count` で制限する。

- `--master-host=host_name`

マスタ レプリケーション サーバのホスト名または IP アドレスを指定する。`master.info` の値を読み取れる場合は、ファイルで指定する値が優先になる。マスタ ホストを指定しない場合、スレーブ スレッドは開始されない。

- `--master-info-file=file_name`

マスタの情報をスレーブが記録するファイルに使う名前。デフォルトの名前は `master.info` で、データ ディレクトリにある。

- `--master-password=password`

マスタへの接続時に、スレーブスレッドが認証に使用するアカウントのパスワード。`master.info` の値を読み取れる場合は、この値が優先される。設定しなければ、空白パスワードと見なされる。

- `--master-port=port_number`

マスタがリスニングする TCP/IP ポート番号。`master.info` の値を読み取れる場合、その値が優先される。設定しなければ、コンパイルされた設定 (3306) が採用される。

- `--master-retry-count=count`

スレーブがギブアップするまで、マスタへの接続を試行する回数。再接続のインターバルは `--master-connect-retry` で設定する。再接続のトリガは、`--slave-net-timeout` オプションのスレーブのデータ読み込みのタイムアウトに基づく。デフォルト値は 86400。

- `--master-ssl, --master-ssl-ca=file_name, --master-ssl-capath=directory_name, --master-ssl-cert=file_name, --master-ssl-cipher=cipher_list, --master-ssl-key=file_name`

SSL を使用してマスタ サーバに接続する安全なレプリケーション接続のセットアップに使用するオプション。`--ssl, --ssl-ca, --ssl-capath, --ssl-cert, --ssl-cipher, --ssl-key` などの意義は、[SSL コマンド オプション](#) を参照する。`master.info` ファイルの値を読み取れる場合は、それが優先される。

- `--master-user=user_name`

マスタへの接続時に、スレーブ スレッドが認証に使用するアカウントのユーザ名。アカウントには `REPLICATION SLAVE` 権限が必要。`master.info` の値を読み取れる場合は、その値が優先される。マスタ ユーザ名が設定されていない場合、名前は `test` と想定する。

- `--max-relay-log-size=size`

サーバがリレー ログを自動的にローテートするサイズ。詳細は [レプリケーション リレーとステータス ファイル](#) を参照。

- `--read-only`

スレーブ スレッドまたは `SUPER` 権限を持つユーザ以外からはアップデートを受けないようにスレーブに設定。これで、スレーブ サーバがクライアントからのアップデートを受けないように設定できる。このオプションは `TEMPORARY` には適用されない。

- `--relay-log=file_name`

リレー ログの名前。デフォルトでは `host_name-relay-bin.nnnnnn` である。`host_name` はスレーブ サーバ ホストの名前、`nnnnnn` はシーケンス番号でのリレー ログ。このオプションを指定して、ホスト名とは独立したリレー ログ名を作成できる。あるいは、リレー ログが大きくなり、`max_relay_log_size` を下げない場合は、データ ディレクトリとは別の場所に置く必要がある。またはディスク間の負荷バランスにあわせてスピードを上げる場合にも使用できる。

- `--relay-log-index=file_name`

リレー ログ インデックス ファイルに使用する名前。デフォルトでは `host_name-relay-bin.index` で、データ ディレクトリにある。`host_name` はスレーブ サーバの名前。

- `--relay-log-info-file=file_name`

スレーブがリレー ログの情報を記録するファイルに使う名前。デフォルトの名前は `relay-log.info` で、データ ディレクトリにある。

- `--relay-log-purge={0|1}`

リレー ログ ファイルが不要になったときの自動パージを有効または無効にする。デフォルト値は 1 (=有効)。これは、`SET GLOBAL relay_log_purge = N` で動的に変更できるグローバル変数である。

- `--relay-log-space-limit=size`

このオプションは、スレーブのすべてのリレー ログの合計サイズ条件 (上限) を設定する。値 0 は「無制限」という意味。スレーブ サーバ ホストのハードディスクに限りがある場合に便利である。上限に達すると、SQL スレッドが、キャッチアップしてクエリを実行し終えて、不要になったリレー ログを削除するまで、I/O スレッドがマスタ サーバからのバイナリ ログ イベントの読み込みを一時的に停止する。注意：この上限は絶対的なものではない。SQL スレッドがリレー ログを削除するためにさらにイベントを必要とする場合があり、その場合は削除が可能になるまで、I/O スレッドは制限を超えて続行する。続行しなければデッドロックが発生する。`--relay-log-space-limit` は、`--max-relay-log-size` 値の 2 倍より小さく設定してはいけない。また、`--max-relay-log-size` が 0 の場合は `--max-binlog-size` 値の 2 倍より小さく設定してはいけない。小さく設定した場合、`--relay-log-space-limit` が超過しているために I/O スレッドが待機している間、SQL スレッドにはパージできるリレーログがない。そして、I/O スレッドは一時的に `--relay-log-space-limit` を無視することになる。

- `--replicate-do-db=db_name`

デフォルトのデータベース `db_name` のステートメントにレプリケーションを制限するようスレーブに指示する。つまり、`USE` で選択したもの。一つ以上のデータベースを指定するには、このコマンドを数回使用する。これは、クロス データベース ステートメントの複製には使用しない。これは、別のデータベースを選択、あるいはデータベースを全く選択しない、`UPDATE some_db.some_table SET foo='bar'` のようなものである。

警告

複数のデータベースを指定するには、このオプションに複数のインスタンスを使う必要がある。データベース名にはカンマが含まれているため、カンマ区切りのリストの場合に、リストが単一のデータベースの名前として扱われます。

以下は、期待とは沿わない可能性がある事柄の一例です。`--replicate-do-db=sales` オプションでスレーブを起動し、マスタに次のステートメントを発行するが、`UPDATE` ステートメントが複製されない。

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```


「デフォルト データベースをチェックするだけ」という動作の主な理由は、ステートメントだけでは複製をするべきかどうかの判断が難しいということである。たとえば、複数テーブルの `DELETE` ステートメント、または複製テーブルの `UPDATE` ステートメントの場合は、複数のデータベースに作用することということである。さらに必要がない限り、すべてのデータベースよりも府デフォルト データベースだけをチェックする方が速いということである。

クロス データベース アップデートを行うには、`--replicate-wild-do-table=db_name.%` を使用の方が好ましい。詳細は [サーバのレプリケーション ルール評価](#) を参照。

- `--replicate-do-table=db_name.tbl_name`

指定テーブルへのレプリケーションを限定するようスレーブスレッドに指示。一つ以上のテーブルを指定するには、これを数回使用する。これは、`--replicate-do-db` とは対照的に、クロス データベース アップデートに利用できる。詳細は [サーバのレプリケーション ルール評価](#) を参照。

- `--replicate-ignore-db=db_name`

`db_name` のデフォルト データベースである場合のステートメントを複製しないようスレーブに指示。このデータベースは、`USE` で選択したものである。無視するテーブルを一つ以上指定するには、このオプションを複数回使用する。クロス データベース アップデートを使用して、そのアップデートを複製しない場合は、このオプションを使用しない。詳細は [サーバのレプリケーション ルール評価](#) を確認。

以下は、期待とは沿わない可能性がある事柄の一例です。`--replicate-ignore-db=sales` オプションでスレーブを起動し、マスタに次のステートメントを発行するが、`UPDATE` ステートメントが複製される。

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

注記

上記の例では、`--replicate-ignore-db` はデフォルト データベースだけに適用されるため、ステートメントの複製が行えます。(`USE` ステートメントで設定。) `sales` データベースはステートメントで明確に指定があったために、ステートメントはフィルタされない。

クロス データベース アップデートを行うには、`--replicate-wild-ignore-table=db_name.%` を使用の方が好ましい。詳細は [サーバのレプリケーション ルール評価](#) を参照。

- `--replicate-ignore-table=db_name.tbl_name`

指定テーブルを更新するステートメントを複製しないようスレーブ スレッドに指示。同一のステートメントで別のテーブルが更新されている可能性がある場合でも。一つ以上のテーブルを指定するには、これを複数回、それぞれのテーブルに使用する。これは、`--replicate-ignore-db` とは対照的に、クロス データベース アップデートに利用できる。詳細は [サーバのレプリケーション ルール評価](#) を参照。

- `--replicate-rewrite-db=from_name->to_name`

デフォルト データベースを `to_name` にトランスレートするようスレーブに指示。このデータベースは、`USE` で選択したものであり、`from_name` がマスタのものである場合である。テーブルに関わるステートメントだけに影響し (`CREATE DATABASE`、`DROP DATABASE`、`ALTER DATABASE` などのステートメントは例外)、`from_name` がマスタのデフォルト データベースである場合だけである。これはクロス データベース アップデートには利用できない。データベース名のトランスレーションは、`--replicate-*` ルールでテストする前に行う。

このオプションをコマンドラインに使用し、`'>'` キャラクタがコマンドのインタープリタとして特別である場合は、オプション値を指定する。次はその例である。

```
shell> mysqlid --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id`

スレーブ サーバで使用する。循環レプリケーションによる無限ループを防ぐために、デフォルトでは 0 に設定されている。1 で設定する場合、スレーブは自身のサーバ ID を持つイベントをスキップしない。通常、これは特別なコンフィギュレーションでのみ有効である。`--log-slave-updates` を使用している場合は、1 を設定することはできない。デフォルトで、スレーブ I/O スレッドにスレーブ サーバの ID がある場合は、バイナリ ログにバイナリ ログイ

イベントを書き込まない。(スレーブ デスク使用量の最適化。) そのため、`--replicate-same-server-id` を使用する場合は、スレーブがスレーブ SQL スレッドで実行するイベントを自分のものとして読み取るようにする前に、このオプションでスレーブを起動する。

- `--replicate-wild-do-table=db_name.tbl_name`

スレーブ スレッドにステートメントへのレプリケーションを制限するよう指示。このステートメントは指定のデータベースとテーブル名パターンと一致するテーブルのアップデートのことである。パターンには '`%`' そして '`_`' などのワイルドカード文字が含まれる。これは、`LIKE` パターン マッチング オペレータとして同一の意義を持つ。一つ以上のテーブルを指定するには、このオプションを複数回、それぞれのテーブルに使用する。クロス テーブル アップデートにも使用できる。詳細は [サーバのレプリケーション ルール評価](#) を参照。

例:`--replicate-wild-do-table=foo%.bar%` はデータベース名が `foo` で、テーブル名が `bar` で、それぞれ始まるテーブルを使用しているアップデートだけを複製する。

テーブル名のパターンが `%` の場合、テーブル名と一致し、オプションはデータベース レベル ステートメントに適用する。(CREATE DATABASE、DROP DATABASE、ALTER DATABASE) たとえば、`--replicate-wild-do-table=foo%.%` オプションを使用する場合、データベース レベル ステートメントは、データベース名が `foo%` のパターンと一致する場合に複製する。

リテラルのワイルドカード文字をデータベースまたはテーブル名パターンに含むには、バックスラッシュでそれらをエスケープする。たとえば、`my_own%db` という名前のデータベースのすべてのテーブルを複製するが、`my1ownAABCdb` データベースからはテーブルを複製しないという場合、'`_`' と '`%`' の文字を エスケープする。例示すると、`--replicate-wild-do-table=my\_own%db` になる。コマンドラインでオプションを使用している場合に、コマンドのインタープリターによっては、ダブル バックスラッシュまたはオプション値を指定する必要がある可能性がある。たとえば、`bash` シェルの場合、`--replicate-wild-do-table=my\\_own%db` と入力する。

- `--replicate-wild-ignore-table=db_name.tbl_name`

スレーブ スレッドにステートメントを複製しないように指示する。このステートメントは、任意のワイルドカードパターンと一致するテーブルもの。無視するテーブルを一つ以上指定するには、このオプションをそれぞれのテーブル毎に使用する。じれはクロス データベース アップデートでも利用できる。詳細は [サーバのレプリケーション ルール評価](#) を参照。

例:`--replicate-wild-ignore-table=foo%.bar%` はデータベース名が `foo` で、テーブル名が `bar` で、それぞれ始まるテーブルを使用しているアップデートを複製しない。

このマッチング (一致) がどのように行われるかは、`--replicate-wild-do-table` の詳細を参照のこと。オプション値にリテラルのワイルドカード文字を含むルールはに関しては、`--replicate-wild-ignore-table` と同様である。

- `--report-host=slave_name`

スレーブ レジストレーションでマスタにレポートするスレーブのホスト名または IP アドレス。この値はマスタサーバの `SHOW SLAVE HOSTS` に出力される。スレーブ自体をマスタとして登録しない場合は、この値をそのままにしておく。スレーブ接続時に、マスタが単にTCP/IP ソケットからスレーブの IP アドレスを読み込む、ということは十分とは言えない。NAT およびルーティングの問題で、この IP はマスタまたは別のホストからスレーブへの接続に対して有効でない可能性がある。

- `--report-port=slave_port_num`

スレーブに接続するTCP/IP ポート番号。スレーブ登録でマスタにレポートする。スレーブが非デフォルトのポートでリスニングしている場合、または、マスタもしくは別のクライアントからスレーブへ特別なトンネルがある場合のみに設定する。これについて定かでない場合は、このオプションは使用しない。

- `--skip-slave-start`

スレーブ サーバにサーバ起動時にスレーブ スレッドを開始しないように指示。このスレッドを後で開始するには、`START SLAVE` ステートメントを使用する。

- `--slave_compressed_protocol={0|1}`

このオプションが 1 で設定する場合に、スレーブとマスタの両方がこれをサポートするときには、スレーブ/マスタ間に圧縮プロトコルを使用する。デフォルトは 0 (非圧縮)。

- `--slave-load-tmpdir=file_name`

スレーブが一時ファイルを作成するディレクトリの名前。このオプションは、デフォルトで `tmpdir` システム変数の値と同等である。スレーブ SQL スレッドは `LOAD DATA INFILE` ステートメントを複製するとき、リレー ログから一時ファイルにロードするファイルを抽出し、それをテーブルにロードする。マスタへのロード ファイルが大きい場合は、スレーブの一時ファイルも大きくなる。そのため、スレーブに一時ファイルを余裕があるファイルシステム内のディレクトリに置くよう指示することを検討するとよい。その場合、リレー ログもまた大きくなるため、そのファイルシステムのリレー ログを置くために `--relay-log` オプションを使用することを検討する。

このオプションで指定するディレクトリはディスク ベースのファイルシステムを使用する。メモリ ベースのファイルシステムは不可。`LOAD DATA INFILE` での複製に使用する一時ファイルは、マシンの再起動に耐える必要があるためである。このディレクトリはまた、システム スタートアップのプロセスでオペレーティング システムによってクリアされたものとは別のものである必要がある。

- `--slave-net-timeout=seconds`

スレーブが読み取りを中止する前に、マスタからのデータを待つ秒数。スレーブが接続切断と判断して再接続を試行するときのもの。最初の接続試行はタイムアウト直後に行われる。再試行のインターバルは、`--master-connect-retry` オプションでコントロールできる。再接続の試行回数は `--master-retry-count` オプションで設定する。デフォルトでは、3600 秒 (1時間)。

- `--slave-skip-errors=[err_code1,err_code2,...|all]`

通常、スレーブでエラーが起けるとレプリケーションは停止する。これは、データの不一致を手動で解決する機会でもある。このオプションはスレーブ SQL スレッドにオプション値にリスト化したエラーをステートメントが返す場合でも、レプリケーションを続けるよう指示する。

エラーの原因が明確ではない場合は、このオプションを使用しない。レプリケーション セットアップやクライアント プログラムにバグがなく、MySQL 自体にもバグがない場合は、レプリケーションを抑制するエラーは起こり得ない。このオプションの無差別的な使用は、スレーブがマスタとの同期化を妨げることに繋がる。そのため、十分な理解が必要である。

エラー コードに関しては、スレーブのエラー ログおよび `SHOW SLAVE STATUS` 出力のエラー メッセージによって提供される数字を使用する。サーバ エラー コードに関しては [Errors, Error Codes, and Common Problems](#) を参照のこと。

`all` を使用してスレーブにすべてのエラー メッセージを無視し、何が起きようとも複製を続けるよう指示ことも可能ではあるが、できるだけ、この値の使用は避ける。`all` を使用するということは、データの整合性を確認できない。これを行ったがために、スレーブとマスタのデータに相違が発生した場合に、バグ報告などでクレームしてはいけない。そのため、十分な注意が必要である。

例：

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

レプリケーションでの管理タスク

レプリケーション開始後は、管理者側のタスクをあまり必要としない実行になります。レプリケーション環境にもよりますが、定期的、日常的、またはできる限り、それぞれのスレーブのレプリケーション ステータスをチェックすることをお勧めします。

レプリケーション ステータスのチェック

レプリケーション プロセスを管理するときの一般的なタスクとして、レプリケーションが正確に行われ、スレーブとマスタの間でエラーが発生していないかどうかを確認することがあります。

これに対するプライマリのコマンドは、`SHOW SLAVE STATUS` であり、それぞれのスレーブで実行します。

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: master1
```

```

Master_User: root
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000004
Read_Master_Log_Pos: 931
Relay_Log_File: slave1-relay-bin.000056
Relay_Log_Pos: 950
Relay_Master_Log_File: mysql-bin.000004
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 931
Relay_Log_Space: 1365
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
1 row in set (0.01 sec)

```

検査するステータス レポートからのキー フィールド

- **Slave_IO_State** — スレーブのカレント ステータスを示す。スレーブ レプリケーションの I/O スレッド状態 および スレーブ レプリケーションの SQL スレッド状態 を参照。
- **Slave_IO_Running** — マスタのバイナリ ログを読む IO スレッドが実行されているかどうかを示す。
- **Slave_SQL_Running** — バイナリ ログのイベントを実行する SQL スレッドが作動しているかどうかを示す。
- **Last_Error** — リレー ログを処理したときに最後に登録されたエラーを示す。これがブランクの場合は、エラーがないことを示す。
- **Seconds_Behind_Master** — スレーブ SQL スレッドがマスタのバイナリ ログに遅れた時間を示す。この数字が大き
い、または上昇している場合は、スレーブがマスタからの大量クエリに対応できないことを示す。

マスタでは、実行プロセスのリストをチェックしてスレーブのステータスを調べることができます。スレーブは **Binlog Dump** コマンドを実行します。

```

mysql> SHOW PROCESSLIST \G;
***** 4. row *****
Id: 10
User: root
Host: slave1:58371
db: NULL
Command: Binlog Dump
Time: 777
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL

```

スレーブがレプリケーション プロセスの主体として働くため、このレポートの情報には限りがあります。

--report-host オプションを使用している場合は、**SHOW SLAVE HOSTS** ステートメントが接続しているスレーブに関する基本的な情報を示します。

```

mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+
| Server_id | Host | Port | Rpl_recovery_rank | Master_id |
+-----+-----+-----+-----+

```

```
| 10 | slave1 | 3306 | 0 | 1 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

この出力は、スレーブ サーバの ID、`--report-host` オプションの値、接続レポート、マスタ ID、バイナリ ログ アップデート受信に関するスレーブの優先順位などを含まれます。

スレーブでレプリケーションを一時停止

`STOP SLAVE` や `START SLAVE` などのコマンドを使用してスレーブにステートメントのレプリケーションの開始、停止を促すことができます。

マスタからのバイナリ ログの実行を停止するには、`STOP SLAVE` を使用します。

```
mysql> STOP SLAVE;
```

実行を停止すると、スレーブはマスタからのバイナリ ログの読み込みを止め (`IO_THREAD`)、未処理のリレー ログのイベント処理を停止します (`SQL_THREAD`)。スレッド タイプを指定して、IO スレッド、または SQL スレッドのどちらかを別々に一時停止できます。次はその例です。

```
mysql> STOP SLAVE IO_THREAD;
```

SQL スレッドを停止することは、マスタからのイベントのみを処理するスレーブでバックアップなどのタスクを実行するときに便利です。IO スレッドはマスタからの読み込みを続けますが、実行はしません。このため、スレーブのオペレーションを再開するときにスレーブが簡単にキャッチアップできます。

IO スレッドを停止すると、リレー ログが新たなイベントの受信を停止した時点までのリレー ログのステートメントを実行します。このオプションの使用は、実行を一時停止し、スレーブがマスタからのイベントにキャッチアップさせるとき、そして、スレーブでのアドミニストレーション タスクを行うとき、あるいは、特定ポイントまでの最新アップデートを確かめることなどに役立ちます。この方法は、マスタでのアドミニストレーション タスクを行うためにスレーブでの実行を停止するとき、レプリケーションを再開するときに大量にバックログのイベントがあるかどうかを調べるときになどに活用できます。

再開するには、`START SLAVE` ステートメントを使用します。

```
mysql> START SLAVE;
```

必要に応じて、`IO_THREAD` または `SQL_THREAD` のどちらかのスレッドを別々に開始できます。

Replication Topologies

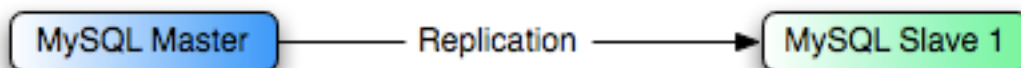
MySQL supports many different topologies for replication. Which topology you use will depend on your requirements and what you want to use replication to achieve.

- Single slave

Replication with a Single Slave

Replication with a single slave

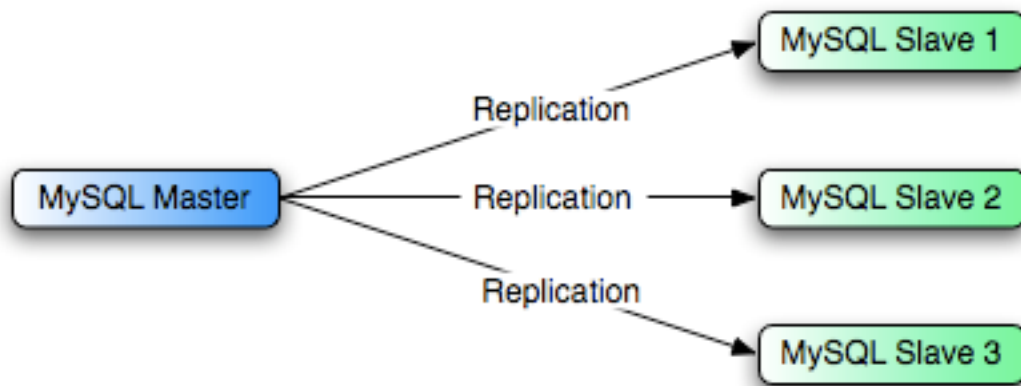
図5.1 Replication with a single slave



Replication with Multiple Slaves

Replication with multiple slaves

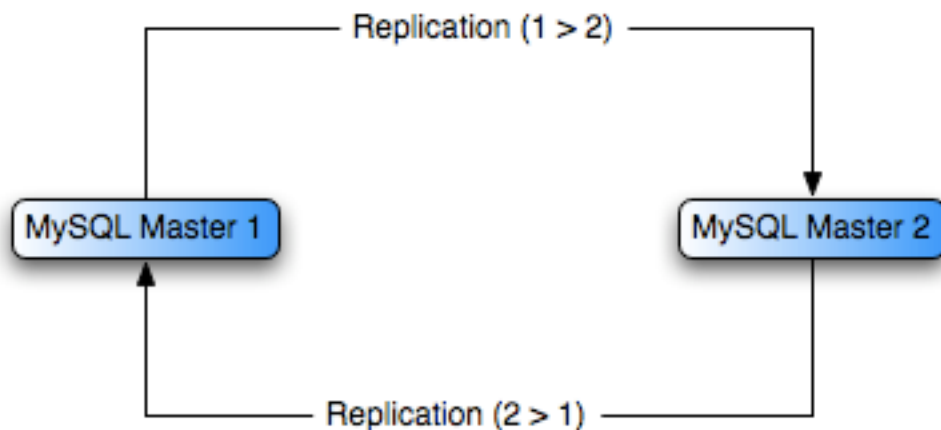
图5.2 Replication with multiple slaves



Replication with Two Masters

Replication with multiple masters

图5.3 Replication with twin masters



Auto-Increment in Multiple-Master Replication

When multiple servers are configured as replication masters, special steps must be taken to prevent key collisions when using `AUTO_INCREMENT` columns, otherwise multiple masters may attempt to use the same `AUTO_INCREMENT` value when inserting rows.

The `auto_increment_increment` and `auto_increment_offset` system variables help to accommodate multiple-master replication with `AUTO_INCREMENT` columns. Each of these variables has a default and minimum value of 1, and a maximum value of 65,535.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` controls the increment between successive `AUTO_INCREMENT` values.
- `auto_increment_offset` determines the starting point for `AUTO_INCREMENT` column values.

By choosing non-conflicting values for these variables on different masters, servers in a multiple-master configuration will not use conflicting `AUTO_INCREMENT` values when inserting new rows into the same table. To set up *N* master servers, set the variables like this:

- Set `auto_increment_increment` to `N` on each master.
- Set each of the `N` masters to have a different `auto_increment_offset`, using the values 1, 2, ..., `N`.

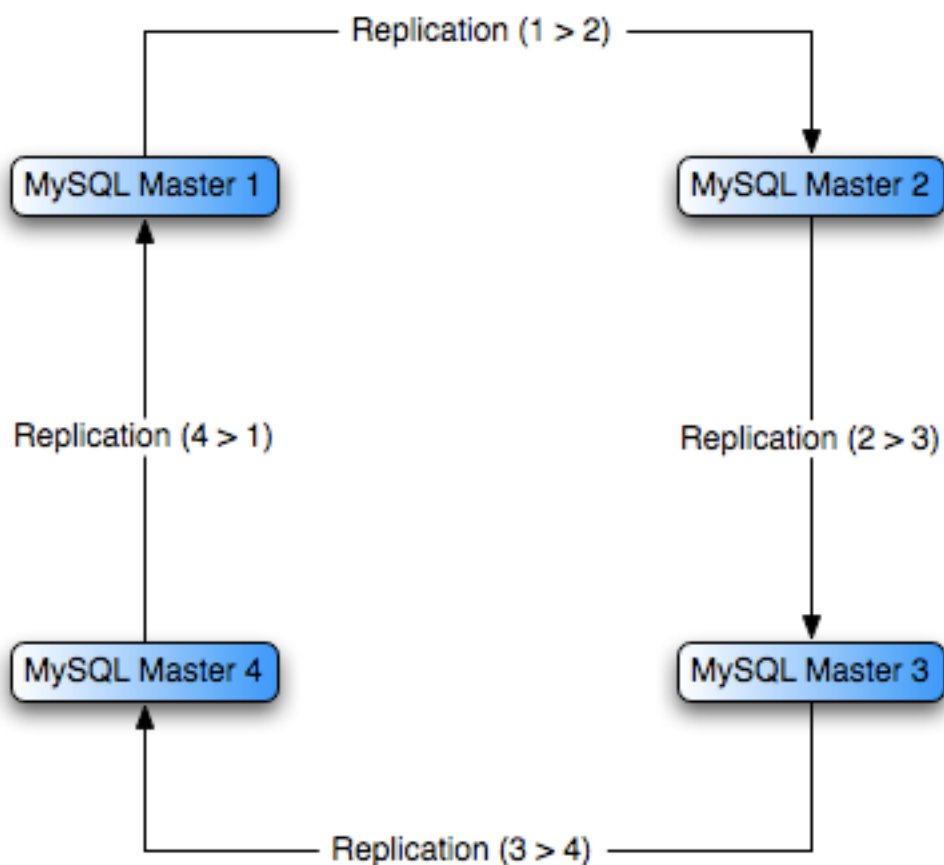
For additional information about `auto_increment_increment` and `auto_increment_offset`, see システム変数.

Replication with Circular Masters

Multiple masters

It is safe to connect servers in a circular master/slave relationship if you use the `--log-slave-updates` option. That means that you can create a setup as shown in 図5.5 「Replication with multiple masters in a chain topology」.

図5.4 Replication with multiple masters in a circular topology



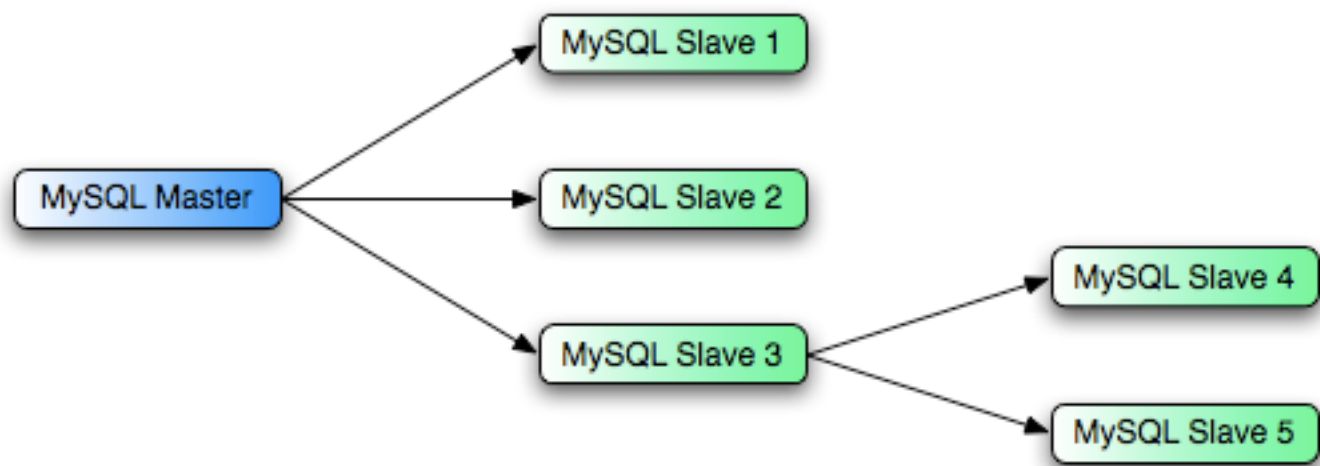
However, many statements do not work correctly in this kind of setup unless your client code is written to take care of the potential problems that can occur from updates that occur in different sequence on different servers.

Server IDs are encoded in binary log events, so server A knows when an event that it reads was originally created by itself and does not execute the event (unless server A was started with the `--replicate-same-server-id` option, which is meaningful only in rare cases). Thus, there are no infinite loops. This type of circular setup works only if you perform no conflicting updates between the tables. In other words, if you insert data in both A and C, you should never insert a row in A that may have a key that conflicts with a row inserted in C. You should also not update the same rows on two servers if the order in which the updates are applied is significant.

Replication Chains

TODO

図5.5 Replication with multiple masters in a chain topology



Replicating Multiple Masters to One Slave

レプリケーション ソリューション

レプリケーションは広範囲かつ異なる環境で使用できます。この章では、固有のソリューション タイプに対応したレプリケーションの手順に関して、一般的なメモとアドバイスを提供します。

バックアップ環境でのレプリケーションに関する詳細、セットアップ、バックアップ手順、バックアップするファイルに関するノートに関しては、[バックアップのレプリケーション](#) を参照してください。

マスタとスレーブで異なるストレージ エンジンを使用している場合のアドバイスやヒントは、[ストレージ エンジンが異なるマスタとスレーブのレプリケーション](#) を参照してください。

スケールアウト ソリューションとしてレプリケーションを使用するには、対象アプリケーションのロジックとオペレーションでの若干の変更が必要になります。詳細は [スケールアウトのレプリケーション](#) を参照してください。

パフォーマンスまたはデータ分散などでは、異なるデータベースを異なるレプリケーション スレーブに複製することをお勧めします。詳細は [異なるデータベースから異なるスレーブへのレプリケーション](#) を参照してください。

レプリケーション スレーブの数が増えると、それぞれのスレーブにバイナリ ログを複製する必要があるため、マスタでの負荷が増加し、マスタのパフォーマンスが低下することに繋がります。レプリケーション パフォーマンスを改善するヒント、単一のセカンダリ サーバをレプリケーション マスタとして使用する方法に関しては、[レプリケーション パフォーマンスの改善](#) を参照してください。

非常時のフェイルオーバー ソリューションとして、マスタへの切り替えやスレーブをマスタにするためのガイドは、[フェイルオーバーでのマスタ切り替え](#) を参照してください。

レプリケーションのコミュニケーションを安全に行うには、SSL をデータ交換に使用して通信チャネルを暗号化します。段階的な指示説明は、[SSLを使用するレプリケーションの設定](#) を参照してください。

バックアップのレプリケーション

レプリケーションは、バックアップ ソリューションとして、マスタからスレーブへデータを複製してデータ スレーブをバックアップできます。スレーブはマスタで稼働しているオペレーションに影響を与えることなく、一時停止やシステム終了ができるため、通常はマスタ データベースのシャットとダウンしなければならない、ライブデータのスナップショットを効率的に生成できます。

データベースをどのようにバックアップするかは、データベースのサイズに依存します。また、データだけをバックアップするのか、予期していないイベントが発生したときにスレーブを立て直すためにデータとレプリケーション スレーブの状態をバックアップするのかなどによって異なります。これには、2つの選択肢があります。

マスタのデータをバックアップするようにするソリューションとしてレプリケーションを利用する場合には、データベースのサイズが超過している場合は、[mysqldump](#) ツールを使うことをお勧めします。詳細は [mysqldump を使用したバックアップ](#) を参照してください。

[mysqldump](#) が実用的ではない大型のデータベースには、生データのファイルをバックアップできます。生データ ファイルのオプションを使用するということは、スレーブ障害のイベントでスレーブを再生できるバイナリ ログとリレー ログをバックアップすることです。詳細は [生データのバックアップ](#) を参照してください。

mysqldump を使用したバックアップ

データベース コピーの作成に [mysqldump](#) を使用すると、MySQL の別インスタンスに情報をインポートできる形式に、データベースのデータすべてを取り込むことができます。情報の形式は、SQL ステートメントであるため、緊急にデータへアクセスを必要とするイベントなどで、ファイルを簡単に分散でき、稼動しているサーバで利用できます。しかし、データ サイズが大きい場合は、[mysqldump](#) は実用的ではないことがあります。

[mysqldump](#) を使用するときには、ダンプ処理を開始する前にスレーブを停止して、ダンプ (出力) に整合データ セットが含まれていることを確認してください。

1. マスタの処理要求を停止する。または [mysqladmin](#) を使用して完全にスレーブを停止する。

```
shell> mysqladmin stop-slave
```

別の方法としては、レプリケーション SQL スレッドを停止してリレー ログ ファイルの処理を停止します。この方法は、バイナリ ログのデータの転送を許可します。この方法を活発なレプリケーション環境で使用すると、スレーブ処理を再開をしたときにキャッチ アップ プロセスをスピードアップする可能性があります。

```
shell> mysql -e 'STOP SLAVE SQL_THREAD;'
```

2. データベースをダンプするために、[mysqldump](#) を実行する。ダンプするデータベース選択するか、データベースすべてをダンプするかを決める。詳細は [mysqldump — データベースバックアッププログラム](#) を参照してください。データベースすべてをダンプするには、

```
shell> mysqldump --all-databases >fulldb.dump
```

3. ダンプが完了したら、スレーブのオペレーションを再開する。

```
shell> mysqladmin start-slave
```

上記の例示では、ログイン資格情報 (ユーザ名、パスワード) をコマンドに加え、日常、自動的に実行するスクリプトにこのプロセスをバンドルすることができます。

この方法でアプローチするときには、このバックアップの所要時間が、マスタからのイベントに対応しているスレーブの能力への影響を避けるために、スレーブのレプリケーション プロセスを監視してください。詳細は [レプリケーション ステータスのチェック](#) を参照してください。スレーブが遅れる場合には、別のサーバを追加して、バックアップ プロセスを分散することをお勧めします。このシナリオの構成例は、[異なるデータベースから異なるスレーブへのレプリケーション](#) を参照してください。

生データのバックアップ

MySQL に生データ ファイルをバックアップするときは、コピー ファイルの整合性を確認するために、スレーブ サーバがシステム終了した状態で、レプリケーション スレーブを行います。MySQL サーバが稼動している場合は、バックグラウンド タスクに、特に InnoDB などのバックグラウンド プロセスを伴うトレージ エンジンなどのときには、データベース ファイルを依然として更新している可能性があります。InnoDB に関しては、これらの問題はクラッシュ リカバリ中に解決するものですが、マスタ側での実行に影響を与えないこと、およびバックアップ プロセス中にスレーブ サーバのシステム終了が可能であることを基に、この利点を生かすことをお勧めします。

サーバのシステム終了とファイルのバックアップ方法

1. MySQL サーバをシャットダウンする。

```
shell> mysqladmin shutdown
```

2. データ ファイルをコピーする。[cp](#)、[tar](#)、[WinZip](#) などのユーティリティを使用してアーカイブする。

```
tar cf /tmp/dbbackup.tar ./data
```

3. [mysqld](#) プロセスを再度立ち上げる。

```
shell> mysqld_safe &
```

Windows の場合

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld"
```

通常はデータ フォルダ全体をスレーブ MySQL サーバにバックアップします。スレーブ障害のイベントで、データをリストアしてスレーブとして使うには、スレーブのデータをバックアップするときに、[master.info](#) および [relay.info](#) のサーバステータス ファイルをリレー ログ ファイルとともにバックアップします。これらのファイルは、スレーブのデータをリストアした後、レプリケーションをレジューム (再開) するときに必要になります。

リレー ログは紛失したが [relay-log.info](#) ファイルはまだ健在であるという場合には、そのファイルで、マスタのバイナリ ログで SQL スレッドがどれくらい実行されたかを調べます。そして、スレーブに起点からのバイナリ ログを再度読み込むよう指示するために、[MASTER_LOG_FILE](#) および [MASTER_LOG_POS](#) オプションと [CHANGE MASTER TO](#) を使用します。この方法はバイナリ ログがまだマスタ サーバに存在している場合だけ有効です。

スレーブが [LOAD DATA INFILE](#) ステートメントの複製に関連している場合は、スレーブが使用しているディレクトリの [SQL_LOAD_*](#) ファイルもバックアップしてください。そのファイルは、中断した [LOAD DATA INFILE](#) オペレーションのレプリケーションをレジュームするときに、スレーブが必要とします。ディレクトリの保存場所は、[--slave-load-tmpdir](#) オプションで指定します。指定できない場合は、ディレクトリの保存場所は [tmpdir](#) システム変数の値です。

ストレージ エンジンが異なるマスタとスレーブのレプリケーション

レプリケーション プロセスは、マスタのソース テーブルとマスタの複製テーブルが異なるエンジンを使用しているかどうかを重視しません。実際には、システム変数 [storage_engine](#) と [table_type](#) は複製されません。

このレプリケーション プロセスでの優位性を異なるエンジン タイプでのレプリケーション シナリオに役立てることができます。たとえば、スケール アウトのシナリオ ([スケールアウトのレプリケーション](#) 参照) では、通常、トランザクション機能をマスタの [InnoDB](#) テーブルに使用しますが、データがリード オンリーということから、トランザクション サポートを必要としないスレーブの [MyISAM](#) を使用できます。データ ロギングの環境でレプリケーションの場合には、スレーブの [Archive](#) ストレージ エンジンを使うことも可能です。

イニシャル レプリケーション プロセスをどのように設定するかによって、マスタとスレーブでエンジンが異なる場合の設定は異なります。

- マスタのデータベース スナップショットを作成する場合は、[mysqldump](#) を使用して、ダンプ テキストを操作し、それぞれのテーブルで使用しているエンジン タイプを変更します。

[mysqldump](#) の別の利点としては、スレーブで使いたくないエンジン タイプを無効にすることができ、これは、スレーブでデータを起こす前にダンプします。たとえば、[InnoDB](#) エンジンが無効にするには、[--skip-innodb](#) オプションをスレーブに追加します。特定のエンジンがない場合、MySQL では通常、[MyISAM](#) などのデフォルトのエンジン タイプを使います。この方法でそのほかのエンジンを無効にする場合には、そのエンジンをサポートする特別のバイナリをスレーブを使うように構成してください。

- 生データ ファイルをスレーブ集団で使用している場合は、イニシャルのテーブル型を変更することはできません。その場合は、スレーブが稼働してから、テーブル型の変更に [ALTER TABLE](#) を使います。
- マスタにテーブルがない時点でのマスタ・スレーブ レプリケーション設定には、新たなテーブルを作成するときのエンジン タイプの指定を避けてください。

レプリケーション ソリューションをすでに実行している場合に既存のテーブルを別のエンジン タイプに変更するには、次のステップに従います。

1. レプリケーション アップデートの実行からスレーブを停止する。

```
mysql> STOP SLAVE;
```

これにより、中断することなく、エンジン タイプの変更が可能になります。

2. エンジン タイプを変更するテーブルのそれぞれで、`ALTER TABLE ... Engine='enginetype'` を実行する。
3. スレーブのレプリケーションを再開する。

```
mysql> START SLAVE;
```

`storage_engine` と `table_type` 変数は複製されませんが、エンジンの仕様を含む `CREATE TABLE` および `ALTER TABLE` ステートメントはスレーブに正確に複製されます。CSV テーブルがある場合には次を実行します。

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

例示のステートメントはスレーブに複製され、そのスレーブのエンジン タイプは `MyISAM` になります。CSV のほかに、スレーブのテーブル型をエンジンにすでに変更していた場合も同様です。マスタとスレーブでエンジンに違いを付ける場合に、新たなテーブルを作成するときは、マスタの `storage_engine` 変数を扱うときには十分に注意してください。

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

次のフォーマットを使用します。

```
mysql> SET storage_engine=MyISAM;  
mysql> CREATE TABLE tablea (columna int);
```

複製後、`storage_engine` 変数は無視され、`CREATE TABLE` ステートメントはスレーブのデフォルト エンジン タイプで実行になります。

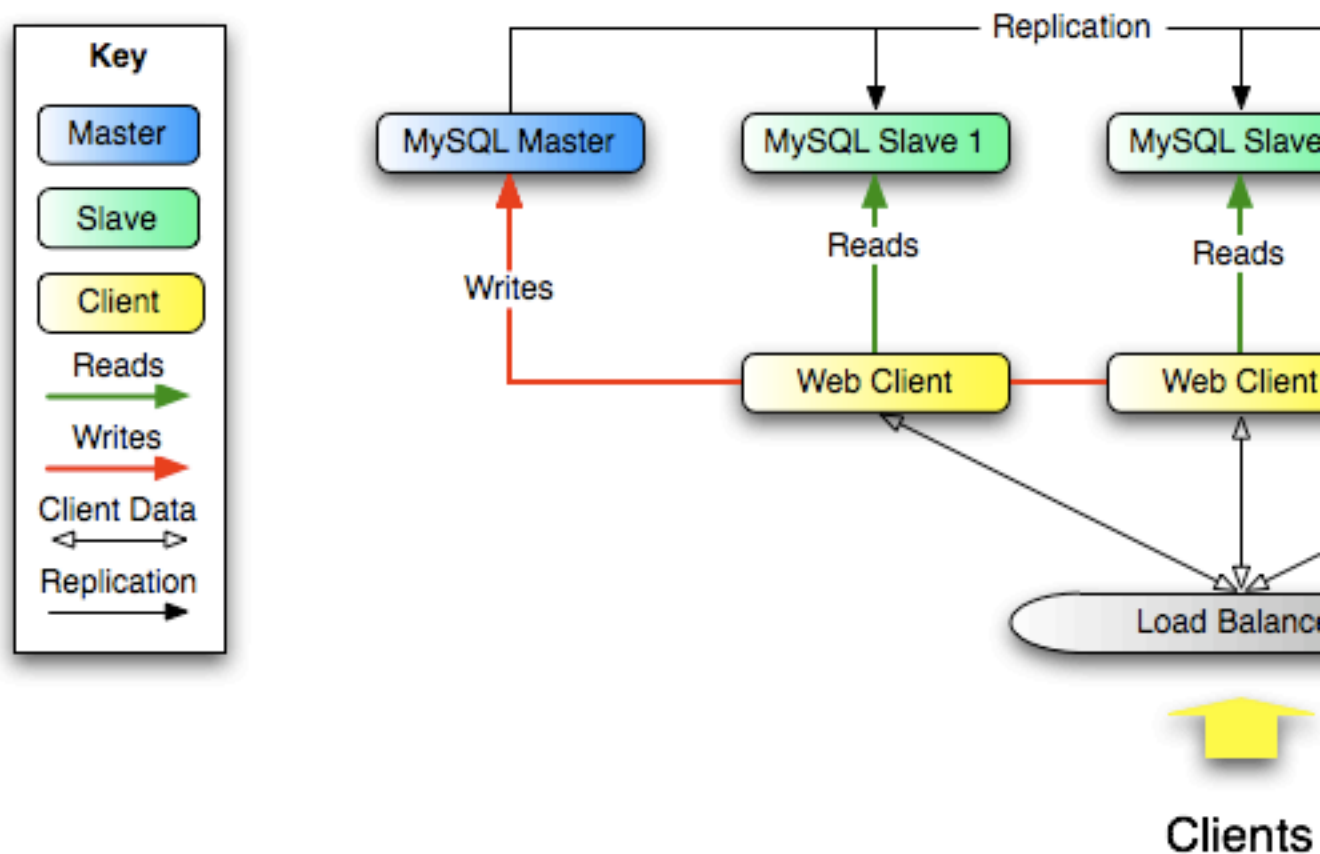
スケールアウトのレプリケーション

スケールアウト ソリューションとしてレプリケーションを使用できるため、データベース クエリ負荷を複数のデータベース サーバに分けることができます。ただし、これには一定の制約があります。

レプリケーションは、ひとつのマスタから複数のスレーブに分散できるため、読み込み頻度が高く、書き込みと更新の頻度が低い場合でのスケールアウトに最適です。ウェブサイトなどはこのカテゴリに該当し、ユーザによるウェブサイトの閲覧、情報取得または入力、製品の検索などに対応します。

書き込みが必要な場合に、ウェブ サーバがレプリケーション マスタと通信する一方で、レプリケーション スレーブが読み込み分を担当します。このシナリオでのレプリケーション レイアウトのサンプルは [図5.6「スケールアウト レプリケーションのパフォーマンス向上の概略図」](#) を参照してください。

図5.6 スケールアウト レプリケーションのパフォーマンス向上の概略図



データベース アクセスを担うコードの一部を適当にモジュール化するとき、それを複製したセットアップで実行するよう変換すると、スムーズかつ簡単です。マスタにすべての書き込み分を、そしてマスタまたはスレーブに読み込み分を送るには、データベース アクセスの実装を変更します。コードでこのレベルの抽出を確保できない場合は、クリーンアップなどのモチベーション向上にもなるので、複製システムをセットアップすることをお勧めします。次の関数を実装してラッパー ライブラリまたはモジュールを作成することから始めます。

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

関数名 `safe_` の `safe_` は、関数がすべてのエラー条件を処理することを意味します。ここでは、読み取りのための接続、書き込みのための接続、読み取り実行、書き込み実行で統合インターフェースを持つことが重要です。

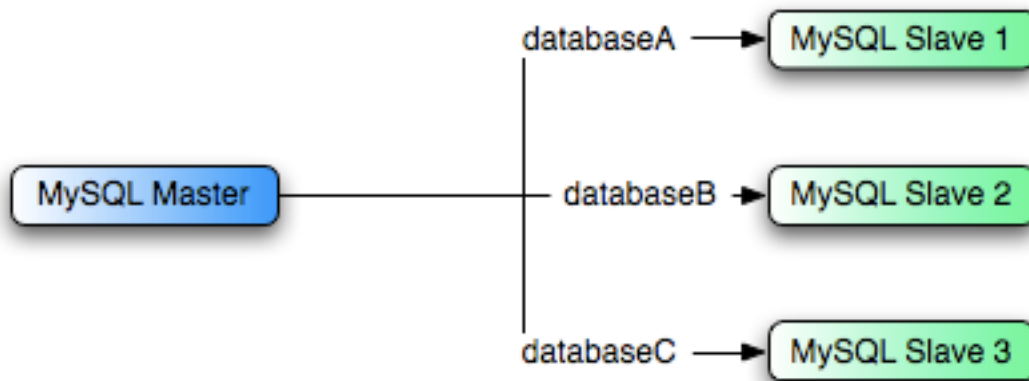
次に、ラッパー ライブラリを使用するようにクライアントコードを変換します。このプロセスは困難ですが、長期的に見ると有意義です。ここで説明したアプローチを使用するアプリケーションはすべて、マスタ・スレーブ構成の利点を活用できます。このコードは保守が非常に簡単で、トラブルシューティング オプションの追加にも手間がかかりません。つまり、1つか2つの関数を修正するだけで、各クエリに所要時間をログしたり、どのクエリがエラーの原因になったかを特定できます。

コード作成の経験が豊かであれば、MySQL の標準ディストリビューションに含まれている `replace` ユーティリティを使用して変換タスクを自動化することも可能です。または独自の変換スクリプトを作成することもできますが、その際にはプログラミング コードが一貫して認識できるスタイルが理想的です。そうでない場合は、書き換えることをお勧めしますが、少なくとも一貫したスタイルに整えてください。

異なるデータベースから異なるスレーブへのレプリケーション

単一のマスタで異なるデータベースを異なるスレーブに複製する場合には、たとえば、異なる販売データを別の部署へ配布するときにはデータ分析の負荷を低減することになります。このレイアウトは [図5.7「別々の DB を複数のホストに複製するレプリケーション概略図」](#) を参照してください。

図5.7 別々の DB を複数のホストに複製するレプリケーション概略図



マスタとスレーブを普通に構成し、セパレーションを行い、`replicate-wild-do-table` コンフィギュレーションをそれぞれのスレーブに使用して、それぞれのスレーブが処理するバイナリ ログ ステートメントを制限します。

たとえば、[図5.7「別々の DB を複数のホストに複製するレプリケーション概略図」](#) で示すようにセパレーションをサポートするには、`START SLAVE` を使用して複製を可能にする前に、次のようにそれぞれのスレーブを構成します。

- MySQL Slave 1 には、次のコンフィギュレーション オプションが必要。

```
replicate-wild-do-table=sales.%
replicate-wild-do-table=finance.%
```

- MySQL Slave 2 には、次のコンフィギュレーション オプションが必要。

```
replicate-wild-do-table=support.%
```

- MySQL Slave 3 には、次のコンフィギュレーション オプションが必要。

```
replicate-wild-do-table=service.%
```

レプリケーションを開始する前にスレーブと同期しなければならないデータがある場合は、いくつかのオプションがあります。

- それぞれのスレーブとすべてのデータを同期化し、不要なデータベースまたはテーブル、あるいはその両方を削除する。
- それぞれのデータベース用に別々のダンプ ファイルを作成するために、`mysqldump` を使用し、それぞれのスレーブに適切なダンプ ファイルをロードする。
- 生データ ファイル ダンプを使用し、それぞれのスレーブで必要とする指定ファイルとデータベースを入れる。これは、`innodb_file_per_table` オプションを使用すると、InnoDB でも機能します。

このコンフィギュレーションのスレーブは、マスタからのバイナリ ログ全体へ転送しますが、構成したデータベースとテーブルに適用する範囲のバイナリ ログのイベントだけを実行します。

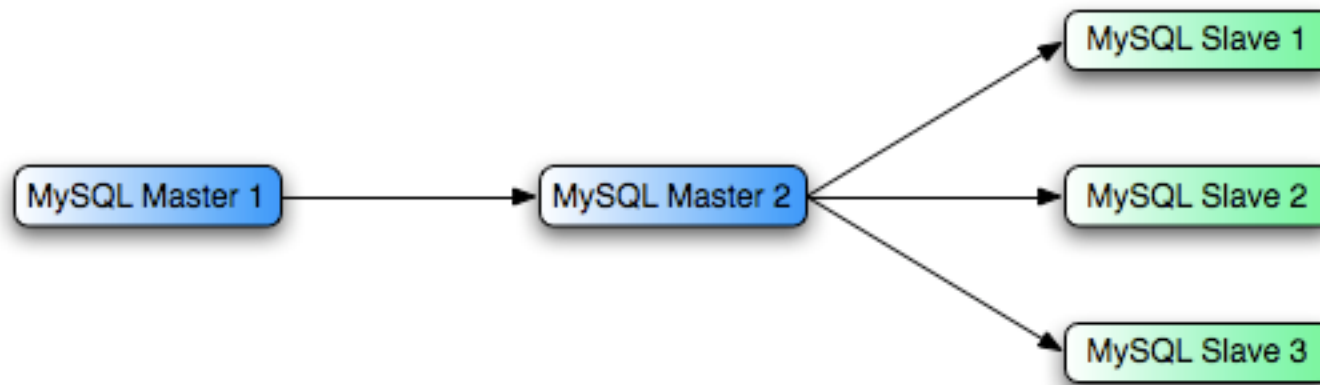
レプリケーション パフォーマンスの改善

マスタに接続したスレーブの数が増えると、若干の負荷も同様に増え、それぞれのスレーブがマスタへのクライアント コネクションを使い切ります。さらに、それぞれのスレーブはマスタのバイナリ ログの完全なコピーを受け取る必要があるため、マスタのネットワーク負荷も同様に増え、ボトルネックを生成しシステム全体の性能が低下します。

スケールアウトソリューションなどで、マスタに接続しているスレーブの数が多いときは、それに対応してマスタでの処理量は膨大になるため、レプリケーションプロセスのパフォーマンスを改善することをお勧めします。

レプリケーションプロセスのパフォーマンスを改善する方法の一つには、よりディープなレプリケーションストラクチャを構築することがあります。これは、マスタが一つのスレーブにだけ複製を行い、ほかのスレーブは個別のレプリケーション要求に対応するプライマリスレーブに接続するという方法です。このストラクチャのサンプルは [図 5.8「追加のレプリケーション ホストでパフォーマンス改善」](#) を参照してください。

図5.8 追加のレプリケーション ホストでパフォーマンス改善



これを実現するには、MySQL インスタンスを次のように設定します。

- Master 1 はプライマリ マスタで、このデータベースにすべての変更とアップデートが書き込まれます。バイナリ ログはこのマシンで実行可能にします。
- Master 2 は Master 1 へのスレーブです。Master 1 はレプリケーション ストラクチャにおいて、レプリケーションの機能性をスレーブの残留分に提供します。ここで Master 2 は Master 1 に唯一接続しているマシンです。Master 2 はバイナリ ログが可能な状態です。[--log-slave-updates](#) オプション で Master 1 からの複製指示が Master 2 のバイナリ ログに書き込まれ、これにより、両者が正当なスレーブに複製するようになります。
- Slave 1、Slave 2、Slave 3 は Master 2 のスレーブとして稼働し、Master 2 からの情報を複製しますが、実際には Master 1 でログしたデータです。

このソリューションは、プライマリ マスタのクライアント負荷だけでなくネットワーク インターフェイス負荷を減らすことができ、プライマリ マスタのパフォーマンス全体を改善するダイレクト データベース ソリューションとして活用できます。

マスタのレプリケーション プロセスに追いつくことに支障をきたしているスレーブがある場合には、次のオプションで対応します。

- リレー ログとデータ ファイルをできるだけ物理的に独立したドライブに割り振ります。そのためには、[--relay-log \[333\]](#) オプションを使用して、リレー ログの保管場所を指定します。
- スレーブがマスタよりも特段に遅い場合は、データベースの種類にあわせて複製の役割を別のスレーブに分けることをお勧めします。詳細は [異なるデータベースから異なるスレーブへのレプリケーション](#) を参照してください。
- マスタのトランザクションを活用し、スレーブがそのトランザクション サポートをしているかどうかを確認するには、[MyISAM](#) またはその他の非トランザクション エンジンを使用します。詳細は [ストレージ エンジンが異なるマスタとスレーブのレプリケーション](#) を参照してください。
- スレーブがマスタとして稼働していない状況で、なにかしら対処できる方法があり、障害イベント中のマスタを持ち込むことができる場合には、[--log-slave-updates](#) オプションをオフにします。これは、“処理能力のない”スレーブがまた、それぞれのバイナリのログに実行したイベントを記録することを防ぎます。

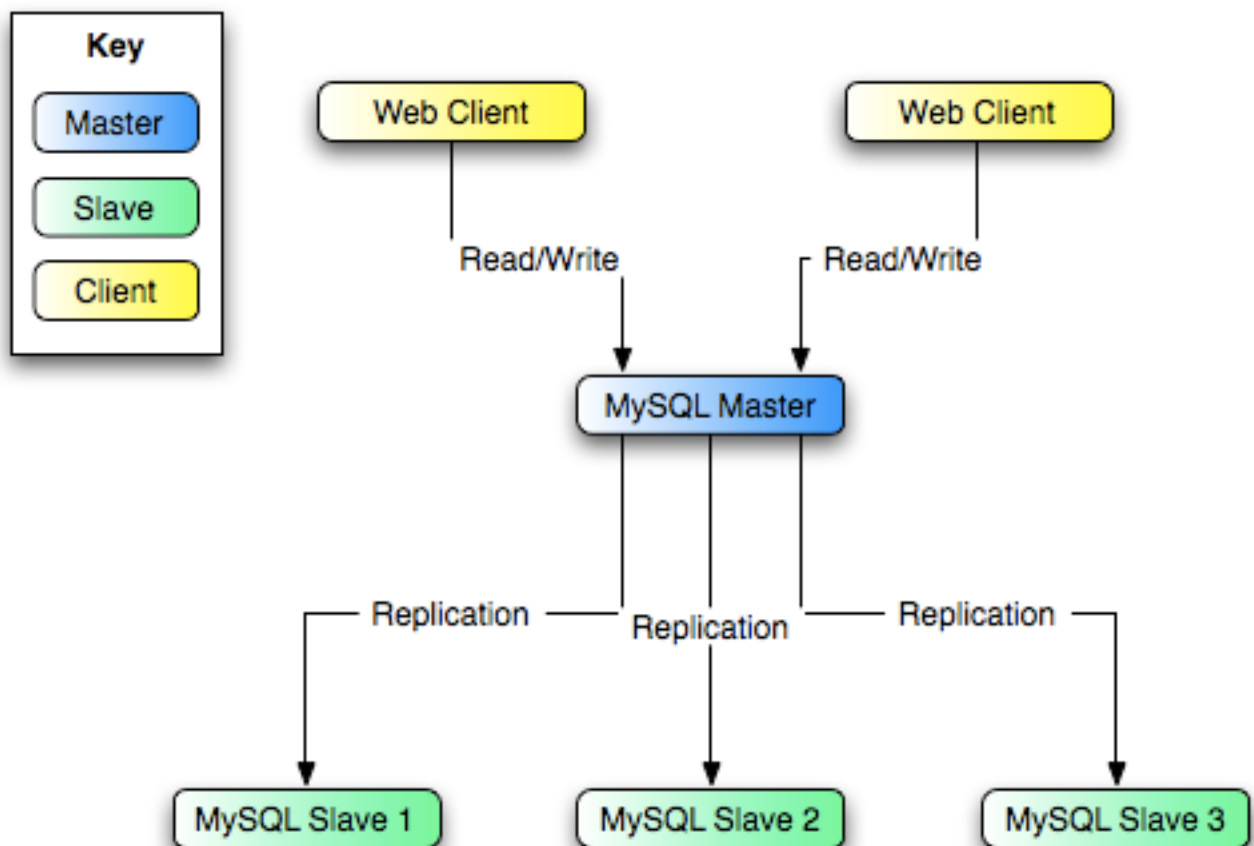
フェイルオーバーでのマスタ切り替え

障害が発生した場合にマスタとスレーブ間でのフェイルオーバーに対応する正式なソリューションは現在の段階ではありません。現在利用可能な機能内では、マスタとスレーブ (または複数のスレーブ) をセットアップし、状況を把握するためにマスタを監視するスクリプトを作成することが挙げられます。そのときには、アプリケーションとスレーブに障害を認識した場合にマスタを変更するよう指示します。

CHANGE MASTER TO ステートメントを使用して、いつでもスレーブにマスタを変えるように指示することが重要です。スレーブはマスタのデータベースがスレーブとの互換性を保持しているかどうかを確認することができないため、新しいマスタが指定するログと場所からイベントを実行し始めます。フェイルオーバーの状況で、グループ内すべてのサーバが同一のバイナリ ログからの同一のイベントを実行しています。そのため、イベント元の変更がデータベースストラクチャまたは整合性に影響することのないように慎重に扱う必要があります。

スレーブを `--log-bin` はあるけれども `--log-slave-updates` はないという組み合わせで実行します。この方法では、スレーブで **RESET MASTER** と **CHANGE MASTER TO** を実行し、別のスレーブで **STOP SLAVE** を実行すると同時に、スレーブがマスタになる準備ができます。図5.9「レプリケーションを活用した冗長性、初期ストラクチャ」のストラクチャで例示を参照してください。

図5.9 レプリケーションを活用した冗長性、初期ストラクチャ



この図では、**MySQL Master** にはマスタ データベースがあり、**MySQL Slave** のコンピュータはレプリケーション スレーブ、そして **Web Client** マシンは読み書き込みをするデータベースという関係になります。通常スレーブに接続していて読み込みだけを行うウェブ クライアントは、障害イベントで新しいサーバへは切り替わらないため、図には含まれていません。読み書き込みのスケールアウト ソリューション ストラクチャに関しては、[スケールアウトのレプリケーション](#) を参照してください。

MySQL Slave のそれぞれ (Slave 1、Slave 2、Slave 3) は、`--log-slave-updates` を伴わずに、`--log-bin` だけで実行しているスレーブです。`--log-slave-updates` の指定がなければマスタからスレーブが受信したアップデートがバイナリ ログに記録されないため、それぞれのスレーブのバイナリ ログは最初から空です。何らかの原因により MySQL Master が利用不可になる場合、複数あるスレーブの中から新しいマスタを選ぶことができます。たとえば、Slave 1 を選択する場合、すべての Web Clients を Slave 1 にリダイレクトして、そのバイナリ ログ にアップデートを記録します。つまり、Slave 2 と Slave 3 は Slave 1 から複製することになります。

`--log-slave-updates` なしで実行するという理由には、複数のスレーブの中から一つが新たなマスタに切り替わる時に、アップデートを 2 度受信しないようにするためです。たとえば、Slave 1 は `--log-slave-updates` するようになると、Master から受信するアップデートを自分のバイナリ ログに書き込みます。Slave 2 が Master から Slave 1 に切り替わった場合は、Master からアップデートをすでに受信しているに関わらず、Slave 1 から再度、アップデートを受信するという結果になります。

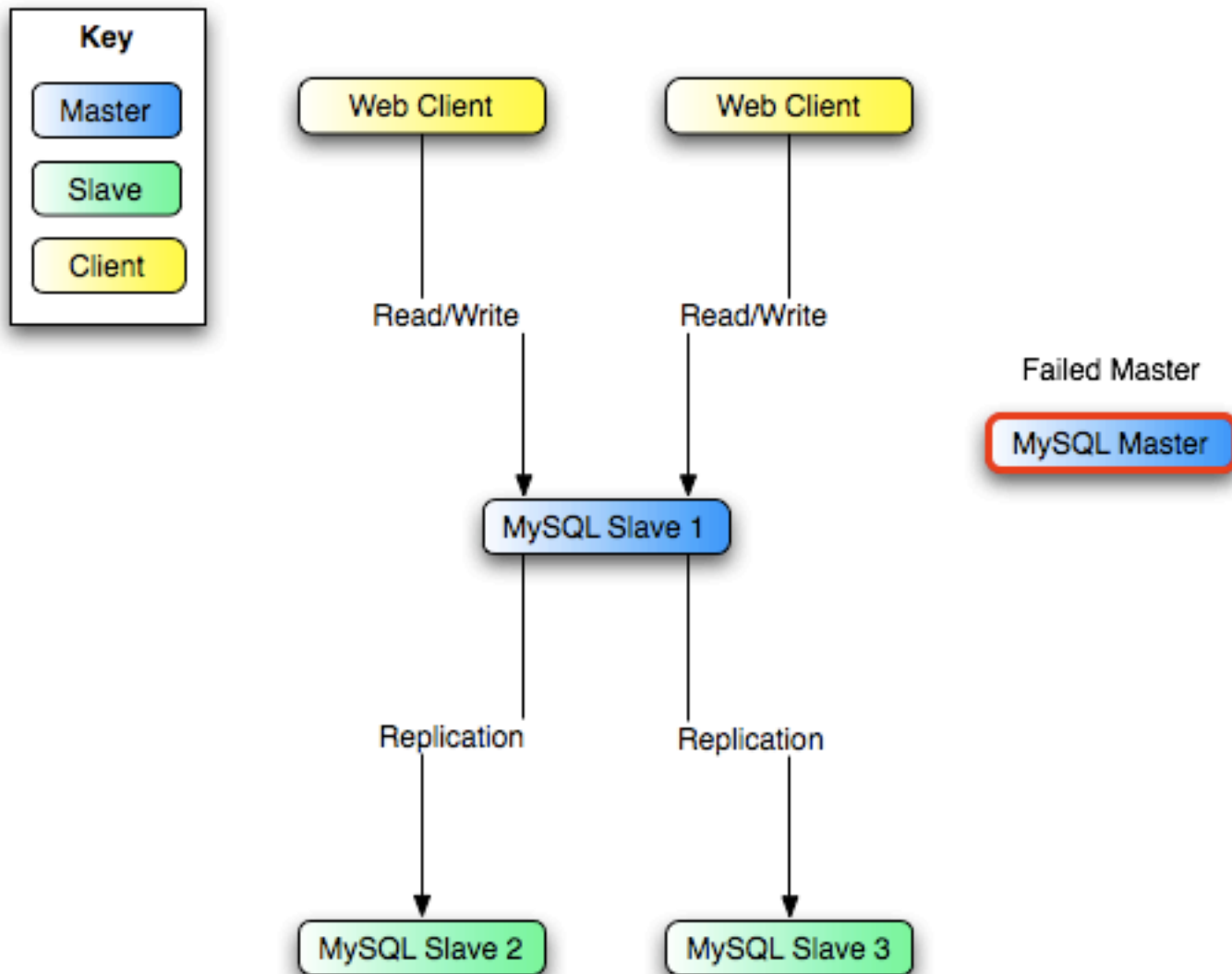
すべてのスレーブがリレー ログ内のクエリを処理したかどうかを確認してください。それぞれのスレーブでは、`STOP SLAVE IO_THREAD` を発行して、`Has read all relay log` を確認できるまで、`SHOW PROCESSLIST` の出力をチェックします。すべてのスレーブでこれが確認できたら、これらを新たな設定として構成できます。マスタに昇格した Slave 1 のスレーブでは、`STOP SLAVE` と `RESET MASTER` を発行します。

別のスレーブ Slave 2 と Slave 3 では、`STOP SLAVE` と `CHANGE MASTER TO MASTER_HOST='Slave1'` を使います。('Slave 1' が、Slave1 の実際のホスト名を表示する場合)。CHANGE MASTER には、Slave 2 または Slave 3 から Slave 1 へのどのようにするかに関するすべての情報を付加します。(user、password、port など) CHANGE MASTER では、Slave 1 のバイナリ ログ名や読み込み先のバイナリ ログ位置を指定する必要はありません。CHANGE MASTER のデフォルトでは、一番初めのバイナリ ログ、そして位置は 4 です。そして最後に、Slave 2 と Slave 3 で `START SLAVE` を使用します。

新たなレプリケーションを整えた後に、Web Client が Slave 1 へクエリを送信するよう指示します。この時点から、Web Client が Slave 1 へ送信したすべてのアップデート クエリが Slave 1 のバイナリ ログに書き込まれ、すでに Master は機能していないため、Slave 1 へ送信したすべてのアップデート クエリが含まれることになります。

サーバストラクチャの結果は 図5.10 「レプリケーションを活用した冗長性、マスタ障害後」 に示す通りです。

図5.10 レプリケーションを活用した冗長性、マスタ障害後



Master マスタの再稼働するときは、Slave 2 と Slave 3 に発行した通りに、同様の `CHANGE MASTER` を発行してください。これにより、Master はスレーブの S1 になり、ダウン後に実行された Web Client の書き込みをすべてピックアップします。

たとえば、マスタが最もパワフルなマシンであるなど物理的な理由で、Master をマスタへ戻すには、Slave 1 を利用不可の状態、Master が新たなマスタになる、というようにそれぞれの立場を逆にしてこの手順を繰り返します。この手順では、Slave 1、Slave 2 そして Slave 3 を Master のスレーブにする前に、Master に `RESET MASTER` を実行することを忘れないでください。これをしないと、Master が動かなくなる前の古い Web Client 書き込みをピックアップすることになります。

複数スレーブとマスタは同期していません。スレーブのいくつかは他のスレーブよりも前に出ていることがあります。つまり、前述の例で示したコンセプトが機能しない可能性があるということです。しかし、実情では、複数スレーブのリレーログがマスタからそれほど遅れをとらないので、この方法が適用できるとの考えに基づいています。そのため、この方法が確証できるものではないことに留意してください。

動的な DNS エントリをマスタに持たせると、マスタの位置に応じてアプリケーションを調節することをお勧めします。bind で、DNS を動的に更新する `nsupdate` を使います。

SSLを使用するレプリケーションの設定

クライアントとサーバの SSL と同様に、SSL 接続を使用してレプリケーションをセットアップします。マスタに使用する適切なセキュリティ証明書、ならびに同類の証明書を同一の認証機関から、それぞれのサーバ用に取得します。

レプリケーションに必要なバイナリ ログの暗号化トランスファに SSL を使用するには、まず最初に SSL ネットワークをサポートするようにマスタをセットアップします。SSL 用にコンパイルしていない、あるいは構成していないなどの理由で、マスタが SSL 接続をサポートしない場合は、SSL 接続を介してレプリケーションを行うことはできません。

サーバとクライアントの SSL 接続 をセットアップに関する詳細は、[SSL接続](#) を参照してください。

SSL をマスタで利用可能にするには、適切な証明書を用意して、次に示すコンフィギュレーション オプションを `mysqld` セクション内にあるマスタのコンフィギュレーションに付加します。

```
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

オプションは次の通りです。

- `ssl-ca` が CA 証明書 を認識する。
- `ssl-cert` がサーバのパブリック キーを認識する。これをクライアントに送信すると、そこにある CA 証明書を認証する。
- `ssl-key` がサーバ プライベート キーを認識する。

スレーブでは、SSL 情報の設定に 2 つのオプションがあります。スレーブの証明書をスレーブのコンフィギュレーション ファイルの `client` セクションに付加するか、もしくは `CHANGE MASTER` ステートメント を使って SSL 情報を明確に指定します。

前述のオプションを使用して、次の文字列をスレーブのコンフィギュレーション ファイルの `client` セクションに付加します。

```
[client]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

スレーブ サーバ を再稼動し、`--skip-slave` をスレーブがマスタへ接続しないようにします。`CHANGE MASTER` を使用して、マスタのコンフィギュレーションを指定し、`master_ssl` オプションを使用して、SSL 接続ができるようにします。

```
mysql> CHANGE MASTER TO \
  MASTER_HOST='master_hostname', \
  MASTER_USER='replicate', \
  MASTER_PASSWORD='password', \
  MASTER_SSL=1;
```

`CHANGE MASTER` コマンドで、SSL 証明ルールを指定するには、SSL ルールを付加します。

```
CHANGE MASTER TO \
  MASTER_HOST='master_hostname', \
  MASTER_USER='replicate', \
  MASTER_PASSWORD='password', \
  MASTER_SSL=1, \
  MASTER_SSL_CA = 'ca_file_name', \
  MASTER_SSL_CAPATH = 'ca_directory_name', \
  MASTER_SSL_CERT = 'cert_file_name', \
  MASTER_SSL_KEY = 'key_file_name';
```

マスタの情報が更新されたら、スレーブのレプリケーション プロセスを開始します。

```
mysql> START SLAVE;
```

`SHOW SLAVE STATUS` を使用して、SSL 接続が完了したかどうかを確認します。

`CHANGE MASTER TO` 構文に関する詳細は[CHANGE MASTER TO 構文](#) を参照してください。

レプリケーション中に使用する SSL 接続 を強化する場合は、[REPLICATION SLAVE](#) 権限でユーザを作成し、そのユーザに [REQUIRE_SSL](#) ルールを使用します。

```
mysql> GRANT REPLICATION SLAVE ON *.*
-> TO 'repl'@'%'.mydomain.com IDENTIFIED BY 'slavepass' REQUIRE SSL;
```

レプリケーション ノートとヒント

レプリケーション機能と既知問題

SQL レベルでのレプリケーション互換性では通常、マスタおよびスレーブの両方のサーバでサポートする機能を必要とします。マスタ サーバの機能がMySQLのバージョン内で使用可能である場合は、古いバージョンのスレーブに複製することはできません。このような非互換はシリーズ間で起こる可能性が高いため、MySQL 5.1 から 5.0 への複製はできません。しかし、非互換はシリーズ内でのレプリケーションでも発生する可能性があります。例: [SLEEP\(\)](#) 機能は、MySQL 5.0.12 以降のバージョンでのみ使用できます。そのため、マスタ サーバでこの機能を使用する場合に、サーバが MySQL 5.0.12 以前のバージョンの場合には複製できません。

5.1 とこれより古いバージョンの MySQL でレプリケーションを行う場合は、その古いバージョンの MySQL レファレンス マニュアルで複製機能に関する情報を確認してから、実行してください。

以下のセクションでは、なにがサポートされているか、そしてなにがサポートされていないかを説明するとともに、複製中に起こる可能性のある問題と状況について説明します。レプリケーションの[InnoDB](#)に特化した情報に関しては、[InnoDB](#) と [MySQL 複製](#) を参照してください。

既存の MySQL でのステートメント ベースの複製に関しては、保存したルーチンとトリガの複製で問題が発生する可能性があります。この問題は、MySQL の行ベースの複製を行うことで回避できます。この問題に関する詳細は、[ストアドルーチンとトリガのバイナリログ](#) を参照してください。行ベース レプリケーションに関する詳細は、[レプリケーション フォーマット](#) を参照してください。

レプリケーションと [AUTO_INCREMENT](#)

レプリケーションは、[AUTO_INCREMENT](#)、[LAST_INSERT_ID\(\)](#) そして [TIMESTAMP](#) の値で正しく実行されますが、次のことに留意してください。

[LAST_INSERT_ID\(\)](#) を使用した格納手順は、ステートメント ベースのバイナリ ログを使用した場合に正しく複製できません。この制約は、MySQL 5.1.12 で修正の予定です。

[AUTO_INCREMENT](#) カラムを [ALTER TABLE](#) でテーブルに加える場合には、行の順序がスレーブとサーバで同じにならない可能性がある。これは、行の順序が、テーブルに使う特定の記憶エンジンあるいは行が挿入された順番に依存する場合に発生します。マスタとスレーブで同じ順序に並べる必要がある場合は、[AUTO_INCREMENT](#) 番号を割り当てる前に行を整理してください。テーブル [t1](#) に [AUTO_INCREMENT](#) を加えると仮定した場合、[t1](#) と同一の新たなテーブル [t2](#) を、[AUTO_INCREMENT](#) のカラムで、次のステートメントがこれを可能にします。

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

これは、テーブル [t1](#) にカラム [col1](#) および [col2](#) があると仮定しています。

重要マスタとスレーブの両方で同一の順番を確保するには、[t1](#) のすべてのカラムが [ORDER BY](#) 節でレファレンスしている必要があります。

ここで示された手順は、[CREATE TABLE ... LIKE](#) 制限を対象としています。[DATA DIRECTORY](#) および [INDEX DIRECTORY](#) テーブル オプションと同様に、外部キーの定義を無視します。テーブル定義がこれらの特性を含む場合、[t1](#) に使用したものと同一の [CREATE TABLE](#) 構文を使用して [t2](#) を作成しますが、そのときは [AUTO_INCREMENT](#) カラムの追加として行います。

[AUTO_INCREMENT](#) 属性をカラムに持つコピーを作成し、投入することに使用した方法に関係なく、このファイナルステップは、オリジナルのテーブルをドロップしてから、そのコピーの名前を変更します。

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

[Problems with ALTER TABLE](#) も参照してください。

レプリケーションとキャラクタ セット

以下は、異なるキャラクタ セットを使用している MySQL サーバ間でのレプリケーションに該当します。

1. マスタで MySQL 4.1 が作動している場合、スレーブで作動している MySQL バージョンに関係なく、常に同一のグローバルキャラクタ セットとコレーション (照合順序) をマスタとスレーブに使用します。(これらは、`--character-set-server` および `--collation-server` オプションがコントロールします。) これ以外の場合は、マスタのキャラクタ セットのユニーク キーが、スレーブのキャラクタ セットではユニークではないと認識する可能性があるため、スレーブで重複キー エラーが発生することがあります。ノート: マスタとスレーブの両方が MySQL 5.0 以降のバージョンである場合は、この心配は不要です。
2. MySQL 4.1.3 より古いバージョンのマスタでは、クライアントのキャラクタ セットのグローバル値が異なると、スレーブではそのキャラクタ セットの変更を認識しません。そのため、`SET NAMES` や `SET CHARACTER SET` などをクライアントに使用しないでください。マスタとスレーブの両方が 4.1.3 以降である場合、クライアントはキャラクタ セット変数のセッション値を自由に設定できます。これは、これらの設定がバイナリ ログに書き込まれるため、スレーブが認識することに基づきます。つまり、クライアントは `SET NAMES` あるいは `SET CHARACTER SET` を使用でき、`collation_client` あるいは `collation_server` などに変数設定ができます。しかし、前述のように、クライアントではこれら変数のグローバル値を変更できないため、マスタとスレーブでは常に同一のグローバル キャラクタ セット値を使用する必要があります。
3. `character_set_server` のグローバル値とは異なるキャラクタ セットでマスタのデータベースを使用している場合は、`CREATE TABLE` コマンドを設計する必要があります。その設計で、データベースのテーブルはデータベースのデフォルトのキャラクタ セットに定義的に依存することがなくなります。推奨の回避方法としては、`CREATE TABLE` 構文でキャラクタ セットとコレーションを定義することをお勧めします。

レプリケーションと DIRECTORY 構文

`DATA DIRECTORY` または `INDEX DIRECTORY` のテーブル オプションをマスタ サーバの `CREATE TABLE` 構文で使っている場合、スレーブでもそのテーブル オプションを使用します。これは、スレーブ ホストのファイル システムに対応するディレクトリが存在しない場合、またはディレクトリは存在するがスレーブ サーバにアクセスできない場合に、問題を引き起こす可能性があります。MySQL は `NO_DIR_IN_CREATE` と呼ばれる `sql_mode` オプションをサポートしています。スレーブ サーバが実行可能な SQL モードで作動している場合は、`CREATE TABLE` ステートメントを複製する際に `DATA DIRECTORY` と `INDEX DIRECTORY` のテーブル オプションを無視します。その結果、テーブルのデータベース ディレクトリには、`MyISAM` データとインデックス ファイルが作成されます。

浮動小数点でのレプリケーション

浮動小数点は近似値であるため、これを利用した比較には誤差が生じます。明示的に浮動小数点を使用する操作、または暗黙的に浮動小数点に変換した値を使用する操作では正当です。コンピュータのアーキテクチャや MySQL のビルドに使用したコンパイラの違いなどにより、浮動小数点での比較はマスタとスレーブで異なる結果を引き起こす可能性があります。[式評価でのタイプ変換](#) および [Problems with Floating-Point Comparisons](#) を参照してください。

レプリケーションと FLUSH

`FLUSH` コマンドのいくつかのフォームは万が一スレーブに複製がされた場合に問題を引き起こす可能性があるため、ログの対象ではありません。(例: `FLUSH LOGS`、`FLUSH MASTER`、`FLUSH SLAVE` および `FLUSH TABLES WITH READ LOCK`) 構文例は、[FLUSH 構文](#) を参照してください。`FLUSH TABLES`、`ANALYZE TABLE`、`OPTIMIZE TABLE` そして `REPAIR TABLE` のコマンドはバイナリ ログに書き込まれ、スレーブに複製します。これらのコマンドはテーブル データを修正しないため、通常は問題にはなりません。しかし場合によっては、何かしらの問題を伴う可能性があります。`mysql` データベースの権限テーブルを複製し、`GRANT` を使用しないでテーブルを直接更新する場合は、新たな権限を有効にするために、`FLUSH PRIVILEGES` をスレーブで実行してください。また、`MERGE` テーブルにある `MyISAM` テーブルの名前を変更する場合に `FLUSH TABLES` を使用するときは、`FLUSH TABLES` をスレーブで手動で行います。`NO_WRITE_TO_BINLOG` もしくはエイリアスの `LOCAL` を指定しない場合は、このコマンドはバイナリ ログに書き込まれます。

レプリケーションとシステム機能

一定の条件下では、特定の機能が複製できないことがあります。

- `USER()`、`CURRENT_USER()`、`UUID()` そして `LOAD_FILE()` 関数は、変更のない複製を行うため、スレーブでは確実には機能しません。行ベースのレプリケーションが使用可能な場合はこの限りではありません。詳細は、[レプリケーション フォーマット](#) を参照してください。

ロギングに関してはコマンドベースから行ベースに形式移行しないため、混在形式ロギングの初期実行段階の格納関数、トリガそしてこれらを主体とした関数を使用するビューは、混在形式のロギングモードでは確実に複製されません。たとえば、`INSERT INTO t SELECT FROM v` の `v` が `UUID()` を選択するビューの場合、問題を引き起こす可能性があります。この制約は、MySQL 5.1.12 で修正の予定です。

- `NOW()`ではなく、`SYSDATE()` 関数の場合は、バイナリ ログの `SET TIMESTAMP` コマンドに影響を受けないこと、そしてコマンドベースのロギングが使用された場合に非決定性であることから、レプリケーション セーフではありません。行ベースのロギングを使用している場合には、この点の心配はありません。もう一つのオプションには、`SYSDATE()` が `NOW()` のエイリアスになることをもたらす `--sysdate-is-now` オプションでサーバを起動する方法もあります。
- 以下の制約はステートメントベースのレプリケーションを行うときに該当します。行ベースのレプリケーションには該当しません。`GET_LOCK()`、`RELEASE_LOCK()`、`IS_FREE_LOCK()`、`IS_USED_LOCK()` などユーザレベルでロックを処理する関数の場合には、スレーブはマスタの並列化の前後関係を認識しないまま複製を行います。これにより、スレーブの内容がマスタのものとは異なることになり、これらの関数をマスタのテーブルに挿入するためには使用しないでください。(例: `INSERT INTO mytable VALUES(GET_LOCK(...))` のようなコマンドは使わない)

ステートメントベースのレプリケーションが有効である場合は、前述の制約への代替方法として、ユーザ変数に問題を起こす関数を使い、後続のコマンドなどでユーザ変数を参照する方法があります。以下は、`INSERT` 一行挿入が `UUID()` 関数に対して問題があるとした場合の例示です。

```
INSERT INTO t VALUES(UUID());
```

この問題を回避するには、次のことを行います。

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

このシーケンスのコマンドは `@my_uuid` という値がバイナリ ログに `INSERT` コマンドより前のユーザ変数イベントとして格納され、`INSERT` で使用可能になるため、複製できます。

この考え方は、複数行の挿入に適用できますが、扱いがより複雑になります。そのため、2行挿入の場合は、次のように行うことができます。

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

この方法は行数が大きいまたは未知である場合には実用的ではありません。そのため、それぞれの行に関連するユーザ変数が与えられている場合は、次の例で示すようにコマンドを変更することはできません。

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

マスタがクラッシュ中のレプリケーション

マスタ側でクラッシュした場合は、マスタのバイナリ ログ ファイルをフラッシュしていないときには、スレーブで読み込みをした位置より古い位置をマスタのバイナリ ログに送る原因となります。これは、マスタが戻ってきた場合に、スレーブで複製できなくなる可能性があります。`sync_binlog=1` をマスタの `my.cnf` ファイルに設定し、マスタがバイナリ ログを頻繁にフラッシュするように設定すると、この問題は最小化できます。

マスタのシャットダウン中のレプリケーション

マスタサーバをシャットダウンして、後で起動することは安全です。スレーブ - マスタの接続が切断した場合、スレーブは直ちに再接続を試行し、それに失敗すると、定期的に再試行を続けます。デフォルト設定では、60 秒に一度の再試行します。これは、`--master-connect-retry` オプションで変更できる可能性があります。スレーブはネットワーク接続の出力停止に対応できる能力があります。しかし、スレーブはネットワーク出力停止から `slave_net_timeout` 秒間が経過し、マスタからデータを受信できないことを確認した時点で、ネットワーク出力の停止を認識します。停止時間が短い場合は、`slave_net_timeout` を減らすことをお勧めします。詳細は [システム変数](#) を参照してください。

MEMORY テーブル型でレプリケーション

サーバがシャットダウンし、再起動するとき、サーバの `MEMORY` テーブルは空になります。このときマスタはこのエフェクトを次のようにして、スレーブに複製します。起動後マスタがそれぞれの `MEMORY` を最初に使用する場合、マスタはイベントをログし、スレーブにテーブルを空にする必要があることを、`DELETE` コマンドをバイナリ

ログへのテーブルに書き込むこと通知します。[MEMORY](#) テーブルに関する詳細は、[MEMORY \(HEAP\) ストレージエンジン](#) を参照してください。

システム [mysql](#) データベース レプリケーション

MySQL 5.1.14以降のバージョンでは、[mysql](#) データベースは [binlog_format](#) の内容に従ってレプリケーションが実行されます。[binlog_format=MIXED](#) の場合には、行ベースフォーマットを利用してレプリケーションが行われます。通常は、これらのテーブルは [GRANT](#)、[REVOKE](#) を実行したり、トリガやストアドルーチン、またはビューに対する変更を行うことによって、間接的に更新されます。その場合、ステートメントベースレプリケーションを用いていると、各ステートメントがスレーブへ伝搬されます。

MySQL 5.1.13以前のバージョンでは、ユーザ権限は[mysql](#) データベースがスレーブへレプリケーションされたときだけ伝搬されます。即ち、[GRANT/REVOKE/SET PASSWORD/CREATE USER/DROP USER](#)などのコマンドは、[mysql](#) データベースが複製されるように設定されている場合のみスレーブへ伝搬されます。

すべてのデータベースを複製する場合に、ユーザ権限に影響するステートメントを不要とするときは、[--replicate-wild-ignore-table=mysql.%](#) オプションを使用して、スレーブが [mysql](#) データベースを複製しないように設定します。このスレーブは、権限に関連する SQL コマンドを実行することは有効ではない、と認識するため、そのクエリは実行されません。

レプリケーション中のスレーブ エラー

スレーブのステートメントによりエラーが発生する場合は、スレーブの SQL スレッドは処理を終了し、スレーブはそのメッセージをエラーログに書き込みます。その場合は、手動でスレーブ接続を行い、問題の原因を特定します。(そのときには、[SHOW SLAVE STATUS](#) を使用することをお勧めします。そして、仮想テーブルを作成する必要があるなど、問題に解決策を投じてから、[START SLAVE](#) を実行します。

スレーブのシャットダウン中のレプリケーション

スレーブはどこでレプリケーションが終了したかを記録しているため、シャットダウンがクリーンに行われた場合は安全です。クリーンではない(不明確な)シャットダウン、特に、システム終了前にディスク キャッシュがフラッシュされていない場合には、問題を引き起こす可能性があります。高性能の無停電電源装置を使用すると、システムのフォールト トレランス(耐障害性)は向上します。マスタでの不明確なシステム終了は、マスタ内のテーブル内容とバイナリ ログ間での不一致を引き起こしますが、これは、[InnoDB](#) テーブルと [--innodb-safe-binlog](#) オプションをマスタで使用することで避けることができます。詳細は [バイナリ ログ](#) を参照してください。

レプリケーションとテンポラリ テーブル

この項目は、行ベースのレプリケーションを使用している場合にはテンポラリ テーブルを複製できないため、該当しません。(レプリケーション フォーマット を参照してください。)

テンポラリ テーブルの複製は可能ですが、スレーブ スレッドだけでなく、スレーブ サーバをシャットダウンした場合や、テンポラリ テーブルを既に複製していて、そのテーブルを更新などに使用し、それがまだスレーブで実行されていないという場合にはできません。スレーブ サーバをシャットダウンする場合、更新対象のテンポラリ テーブルはスレーブを再起動するときには、すでに利用できない状態になります。この問題を回避するには、テンポラリ テーブルが開いている状態でスレーブをシャットダウンしないよう注意が必要です。代替方法として、次の手順を使用します。

1. [STOP SLAVE](#) ステートメントを実行する。
2. [SHOW STATUS](#) を使用して、[Slave_open_temp_tables](#) 変数の値を調べる。
3. 値が 0 であれば、[mysqladmin shutdown](#) コマンドを実行して、スレーブを停止する。
4. 値が 0 でなければ、[START SLAVE](#) でスレーブ スレッドを再開する。
5. この手順を繰り返し、[Slave_open_temp_tables](#) 変数が 0 になったら、スレーブを停止する。

レプリケーション リトライとタイムアウト

グローバル システム変数 [slave_transaction_retries](#) はレプリケーションに影響します。[InnoDB](#) デッドロック、[InnoDB innodb_lock_wait_timeout](#) の値を超過、[NDBCluster TransactionDeadlockDetectionTimeout](#) もしくは

`TransactionInactivityTimeout` の値を超過したときなど、スレーブ SQL スレッドのレプリケーションでトランザクションに失敗した場合、そのトランザクションは、エラー停止する前に、自動的に `slave_transaction_retries` 回の再試行の対象になります。デフォルト値は 10 です。合計再試行の回数は `SHOW STATUS` の出力で確認できます。(スレーブ変数を参照)

レプリケーションとタイムゾーン

マスタが MySQL 4.1 を使用している場合、マスタとスレーブの両方を同じシステムのタイムゾーンで設定する必要があります。両者のタイムゾーンが異なる場合、ステートメントの正確な複製はできません。`NOW()` または `FROM_UNIXTIME()` 関数を使用しているステートメントがこれに該当します。`mysqld_safe` スクリプトの `--timezone=timezone_name` オプションを使用するか、`TZ` 環境変数を設定すると、作動している MySQL サーバのタイムゾーンで設定できます。マスタとスレーブは同一のデフォルト接続のタイムゾーンである必要があります、つまり `--default-time-zone` のパラメータが同一の値である必要があります。これは、マスタが MySQL 5.0 以降である場合には不要です。

マスタとスレーブの両方が MySQL 5.0.4 以降である場合は、`CONVERT_TZ(..., @session.time_zone)` は正確に複製できます。

レプリケーションとトランザクション

スレーブの非トランザクションのテーブルを使用して、マスタにトランザクションテーブルを複製することが可能です。たとえば、`InnoDB` マスタテーブルを `MyISAM` スレーブテーブルとして複製できます。しかし、これを行う場合、スレーブが `BEGIN` あるいは `COMMIT` セグメントの最中に停止した場合は問題が生じます。これは、スレーブが `BEGIN` セグメントの開始時点で再開することに起因します。

トランザクションと非トランザクションのテーブルへ混在した更新が行われる場合には、バイナリログのクエリ順序が正確になり、必要とされるすべてのクエリがバイナリログに書き込まれます。`ROLLBACK` の場合でも同様です。しかし、最初のトランザクションが完了する前に 2 番目の接続が非トランザクションテーブルの更新をすると、ステートメントは順番が乱れた状態での記録になります。これは、最初の接続でのトランザクションの状況に関わらず、2 番目の接続の更新実行と書き込みが同時に発生することに起因します。

`MyISAM` テーブルの非トランザクション性により、部分的にテーブルを更新しエラーコードを返すコマンドを保持することがあります。これは、挿入する複数行にキー制約に違反して、または長文の更新クエリで数行を更新したあとに落とされた場合に起こります。これがマスタで発生した場合、スレーブスレッドは終了し、データベースアドミニストレータの指示を待機します。エラーコードが正当で、クエリを実行したときに同一のエラーコードがスレーブに返る場合はこの限りではありません。このエラーコード検証の動作が望まれるものではない場合、`--slave-skip-errors` オプションで固有化またはすべてのエラーをマークアウト (無視) できます。

注意

トランザクションと非トランザクションの両テーブルを更新するレプリケーション環境では、このトランザクションは行わないでください。

スレーブの複数カラムでレプリケーション

スレーブテーブルのカラム数が、それに対応しているマスタテーブルのものより多い場合、マスタからスレーブへ複製できます。

Table T1 をマスタから、スレーブの T2 へ複製する場合、次の条件に従う必要があります。

- 行ベースのレプリケーションを使用する。
- T1 と T2 は同一のデータベース名とテーブル名を持つ。
- T2 には T1 よりもカラム数があっても、T1 のカラムに対応後、連続して表示されなければならない。
- T1 と T2 で一致するすべてのカラムは同一のタイプである。
- T1 ではなく、T2 のカラムすべてがデフォルト値である。

この機能は MySQL 5.1.12 に追加されました。

レプリケーションと変数

`FOREIGN_KEY_CHECKS`、`SQL_MODE`、`UNIQUE_CHECKS`、`SQL_AUTO_IS_NULL` の変数はすべて複製可能です。これは MySQL 5.0 以来、実現しました。`storage_engine` システム変数 (`table_type`) は、MySQL 5.1 ではまだ複製できませんが、異なるストレージ エンジン間での複製には適しています。

更新対象のテーブルのステートメントで使用する場合には、セッション変数を正確に複製することはできません。たとえば、`INSERT INTO mytable VALUES(@@MAX_JOIN_SIZE)` に続く `SET MAX_JOIN_SIZE=1000` の場合は、マスタとスレーブに同一のデータを挿入できません。これは、`INSERT INTO mytable VALUES(CONVERT_TZ(...,...,@@time_zone))` に続く `SET TIME_ZONE=...` という一般的なシーケンスの場合には、この限りではありません。

セッション変数のレプリケーションは、行ベースのレプリケーションを使用している場合には干渉しません。詳細は [レプリケーション フォーマット](#) を参照してください。

レプリケーションとビュー

ビューは常にスレーブに複製できます。ビューはそれぞれの名前でフィルタされます。対象テーブル毎ではありません。これは、ビューをスレーブに複製できることを示し、ビューが通常は `replication-ignore-table` オプションでフィルタされるテーブルを含んでいる場合でも複製が可能です。ここでは、通常、セキュリティ上の理由でフィルタが行われるテーブル データをもビューが複製しないように確かめる必要があります。

MySQL バージョン間のレプリケーション互換性

MySQL 5.1 に実装したバイナリ ログ形式は、以前のバージョンで使用していたものとは大きく異なり、キャラクタセットの処理、`LOAD DATA INFILE`、タイムゾーンに関しては特に異なります。

一般的なルールとしては、マスタとスレーブを同じバージョンで実行しているときにレプリケーションを設定してください。(MySQL 5.1, 5.0 または 4.1 など)異なるバージョン間でレプリケーションを実行する必要がある場合は、クライアントにマスタと同等またはそれ以上のバージョンを使用していることを確認してください。(例: マスタで 4.1.23、スレーブで 5.0.24)

レプリケーションはMySQL 5.0 および 5.1 のマスタとサーバであれば、コンビネーションに関係なく、正確にできます。マスタとスレーブで異なるグローバル キャラクタ セットの変数を使用している場合や、マスタとスレーブで異なるグローバル タイムゾーンの変数を使用している場合でも同様です。これはマスタとスレーブのどちらか、もしくはその両方がMySQL 4.1 以前を使用している場合には該当しません。

レプリケーションの互換性は継続的に向上するため、最新の MySQL バージョンを使用することをお勧めします。マスタとスレーブで同一のバージョンを使用することも効果的です。アルファまたはベータ バージョンをマスタとスレーブで使用している場合には、プロダクション バージョンにアップグレードすることをお勧めします。マスタで使用中のものが新しく、スレーブで使用中のものが古い場合にレプリケーションを行うと失敗するケースが多分にあります。一般的には、MySQL 5.1.x を実行しているスレーブは、マスタで使用しているバージョンが古い (MySQL 3.23、4.0 または 4.1) でも使用できますが、その逆の場合はできません。

注記

新しいバイナリ ログ形式のマスタから、古い形式を使用するスレーブへ複製することはできません。(例: MySQL 5.0 から MySQL 4.1 へ) [レプリケーション セットアップのアップグレード](#) に記載の通り、これは、レプリケーション サーバのアップグレードに大きな影響を与えます。

前項の情報はプロトコル レベルでのレプリケーション互換に関連します。しかし、SQL レベルでの互換性問題など、他にも制限があります。たとえば、複製したステートメントが5.0ではなく 5.1 で利用可能なSQLの機能を使用している場合は、5.1 のマスタは、5.0 のスレーブに複製できません。その問題に関しては、[レプリケーション機能と既知問題](#) を参照してください。

レプリケーション セットアップのアップグレード

レプリケーション セットアップに関連するサーバをアップグレードする場合、アップグレード手順は使用中のサーバのバージョンおよびアップグレードしようとしているバージョンによって異なります。

この項は、MySQL 3.23、4.0 または 4.1 から MySQL 5.1 へのレプリケーションをアップグレードするときに該当します。4.0 のサーバは 4.0.3 以降である必要があります。

マスタを初期の MySQL リリース シリーズから 5.1 にアップグレードする場合、そのマスタのすべてのスレーブで同一の 5.1.x リリースを使用していることを確認してください。そうでない場合は、まずスレーブをアップグレードしてください。それぞれのスレーブをアップグレードするには、まずスレーブらをシャットダウンし、それぞれを適切な 5.1.x にアップグレードし、再起動させ、複製を再開します。5.1 のスレーブは、アップグレード前に書き込まれたリレー ログを読み込むことができ、そこに含まれているステートメントを実行することができます。アップグレードが行われた後にスレーブによって作成されたリレーログは、5.1 形式です。

スレーブをアップグレードした後は、マスタをシャットダウンし、スレーブと同様にマスタを同一の 5.1.x リリースでアップグレードし、再起動します。5.1 のマスタは、アップグレード前に書き込まれたリレー ログを読み込むことができ、それらを 5.1 のスレーブに送信することができます。スレーブは、古い形式を認識し、それに応じて正確に処理します。アップグレード後にマスタによって作成されたバイナリ ログは、5.1 形式です。これも同様に、5.1 スレーブが認識します。

言い換えれば、MySQL 5.1 へアップグレードする際には特に注意することはありません。ただし、マスタを 5.1 にアップグレード可能な状態にする前に、スレーブが MySQL 5.1 である必要があります。ノート：5.1 から古いバージョンへダウングレード (格下げ) することはできません。5.1 のバイナリ ログまたはリレー ログでの処理が完了したことを確認してから、ダウングレードで先に進む前に、それらを削除してください。

レプリケーション セットアップを以前のバージョンにダウングレードすることは、ステートメント ベースから行ベースのレプリケーションに変更した後、あるいは最初に行ベースのステートメントが binlog に書き込まれた後にはできません。詳細は [レプリケーション フォーマット](#) を参照してください。

レプリケーション FAQ

Questions

- [5.4.4.1: \[359\]](#) スレーブは常にマスタに接続している必要がありますか。
- [5.4.4.2: \[359\]](#) マスタをネットワーク化しなければ、レプリケーションを実行できませんか。
- [5.4.4.3: \[359\]](#) スレーブがマスタと比較してどれだけ遅れているかを知る方法がありますか。または、スレーブによって複製が行われた最後のクエリ日を知る方法がありますか。
- [5.4.4.4: \[359\]](#) スレーブが追いつくまでマスタの更新をブロックする方法がありますか。
- [5.4.4.5: \[359\]](#) 二方向レプリケーションをセットアップするときに注意する点がありますか。
- [5.4.4.6: \[360\]](#) レプリケーションを使用してシステムのパフォーマンスを改善する方法がありますか。
- [5.4.4.7: \[360\]](#) パフォーマンス改善レプリケーションを用いたアプリケーションを作成するときに、クライアントコードはどのように準備しますか。
- [5.4.4.8: \[360\]](#) MySQL のレプリケーションで、どれくらいのシステム パフォーマンスの向上が期待できますか、そしてそれはいつ行うものですか。
- [5.4.4.9: \[361\]](#) 冗長性と高可用性を実現するために、レプリケーションをどのように使用すればよいですか。
- [5.4.4.10: \[361\]](#) マスタ サーバのロギング形式が、ステートメント ベースまたは行ベースのどちらであるかを確かめる方法がありますか。
- [5.4.4.11: \[361\]](#) スレーブが行ベースのレプリケーションを使用するように設定する方法がありますか。
- [5.4.4.12: \[361\]](#) スレーブのマシンへの複製で、GRANT および REVOKE のステートメントを抑制する方法がありますか。
- [5.4.4.13: \[361\]](#) 混在のオペレーティング システムでレプリケーションを実行できますか。(例：マスタが Linux、スレーブが Mac OS X と Windows の場合)
- [5.4.4.14: \[361\]](#) 混在のハードウェア アーキテクチャでレプリケーションを実行できますか。(マスタが 64-bit、スレーブが 32-bit のマシンの場合など)

Questions and Answers

5.4.4.1: スレーブは常にマスタに接続している必要がありますか。

いいえ、その必要はありません。スレーブは何時間でも、あるいは何日間でもシャットダウンしておいたり、非接続にしておいても、再接続してマスタの更新に追いつくことができます。たとえば、マスタとスレーブの関係をダイヤルアップリンクでセットアップし、リンクアップを散発的かつ短時間に設定できます。この場合、任意の時点でスレーブがマスタと同期している保証がないため、これに対応する特別な対策を投じてください。

この対策としては、スレーブに情報が複製されていない場合は、マスタからバイナリ ログを削除しないでください。非同期のレプリケーションは、スレーブが最後に読み込んだレプリケーション ステートメントのバイナリ ログを読み込みできる場合に限りです。

5.4.4.2: マスタをネットワーク化しなければ、レプリケーションを実行できませんか。

はい。マスタのネットワーク化してください。ネットワーク化できない場合は、スレーブがマスタに接続してバイナリ ログを転送することができません。skip-networking ルールがコンフィギュレーション ファイルで実行可能になっているかどうかを確認してください。

5.4.4.3: スレーブがマスタと比較してどれだけ遅れているかを知る方法がありますか。または、スレーブによって複製が行われた最後のクエリ日を知る方法がありますか。

SHOW SLAVE STATUS の Seconds_Behind_Master カラムを参考にしてください。詳細は [レプリケーション実装の詳細](#) を参照してください。

スレーブの SQL スレッドでマスタから読み込んだイベントを実行する場合、このスレッドは自らのタイムをイベントのタイムスタンプに修正します。ノート: [TIMESTAMP](#) が十分に複製される理由はここにあります。SHOW PROCESSLIST 出力の Time カラムに表示される数字は、スレーブ SQL スレッドに対する秒数であり、最後に複製したイベントのタイムスタンプとスレーブ マシンの実際の時刻の差です。これを使用して、最後に複製したイベントの日付を特定できます。注意: スレーブがマスタから切断してから 1 時間後に再接続した場合、SHOW PROCESSLIST の Time カラムでスレーブ SQL スレッドが 3600 の値を表示する場合があります。これは、スレーブは 1 時間遅れたクエリを実行しているためです。(タイムスタンプから 1 時間経過している)

5.4.4.4: スレーブが追いつくまでマスタの更新をブロックする方法がありますか。

次の手順を使用します。

1. マスタで、次のコマンドを実行する。

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

SHOW ステートメントの出力からログファイル名とオフセットを記録する。

2. スレーブで次のコマンドを実行する。ここで、MASTER_POS_WAIT() 関数の引数であるレプリケーション座標は、前のステップで記録した値。

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
```

指定のログ ファイルとオフセットにスレーブが到達するまで、SELECT ステートメントがブロックする。到達した時点でスレーブはマスタと同期して、ここでステートメントが戻る。

3. マスタで次のステートメントを発行し、マスタによる更新処理の再開を許可する。

```
mysql> UNLOCK TABLES;
```

5.4.4.5: 二方向レプリケーションをセットアップするときに注意する点がありますか。

MySQL レプリケーションでは、現在のところ、分散（サーバ間の）更新の原子性を保証するマスタとスレーブ間のロッキング プロトコルをサポートしていません。つまり、クライアント A が co-master 1 に更新を行い、それを co-master 2 に伝播する前に、クライアント B が co-master 2 に更新を行う場合、クライアント A の更新が co-master 1 で更新したものとは異なる更新になる可能性があります。この場合、co-master 2 からの更新すべてが伝播した後であっても、クライアント A の更新が co-master 2 に伝わるとき、co-master 1 とは異なるテーブルが生成されます。このため、どのような順序でも更新が安全に行われるという確証がある場合またはクライアントコードで更新順序の不正に対処できる場合以外では、二方向レプリケーションで 2 つのサーバをチェーン状に設定しないでください。

更新については、二方向レプリケーションは、それほどあるいはまったく、パフォーマンス向上には役立ちません。1 つのサーバで更新を行うときと同様に、どのサーバでも同等量の更新を行う必要があります。別のサーバから発生

した更新が 1 つのスレーブ スレッドでシリアル化されるため、ロックの競合が少なくなる、という違いがあります。この利点も、ネットワーク遅延によっては相殺されてしまう可能性があります。

5.4.4.6: レプリケーションを使用してシステムのパフォーマンスを改善する方法はありますか。

1 つのサーバをマスタとしてセットアップし、書き込みのすべてをそこで直接行います。そして割当量とスペースが許容する限り多くのスレーブで構成し、マスタと複数のスレーブで読み取りを分散します。スレーブ側での速度を向上するには、`--skip-innodb`、`--low-priority-updates` および `--delay-key-write=ALL` でスレーブを起動することもできます。この場合、スレーブは InnoDB テーブルの代わりに非トランザクションの MyISAM テーブルを使用して、トランザクション オーバーヘッドを取り除きながら、速度を上げます。

5.4.4.7: パフォーマンス改善レプリケーションを用いたアプリケーションを作成するときに、クライアントコードはどのように準備しますか。

スケールアウトソリューションとしてレプリケーションを使用するためのガイドは、[スケールアウトのレプリケーション](#) を参照してください。

5.4.4.8: MySQL のレプリケーションで、どれくらいのシステム パフォーマンスの向上が期待できますか、そしてそれはいつ行うものですか。

MySQL レプリケーションは、読み取りが頻繁に行われ、書き込みはそれほどでもないシステムに最も適しています。理論的には、単一のマスタや複数のスレーブのセットアップを使用することで、システムを拡張することができます。つまりネットワーク帯域幅を超えるか、更新負荷がマスタで処理しきれないポイントに到達するまでは、スレーブの数をシステムに追加できます。

追加する利点がなくなるまでいくつのスレーブを追加できるか、あるいはサイトのパフォーマンスをどれだけ向上できるかを判断するには、クエリ パターンを知る必要があります。そして読み取りには毎秒ごとの読み取り、または `reads`、書き取りには `writes` で、マスタとスレーブの通常のスループットの関係をベンチマークして、実証的に決定します。ここでは、例として、仮想システムのレプリケーションでの単純化した計算を示します。

システム負荷が 10% の書き込みと 90% の読み取りで構成され、`reads` が $1200 - 2 \times \text{writes}$ であると仮定します。つまり、書き込みがなければシステムは毎秒 1,200 の読み取りを実行します。書き込みの平均は読み取り平均の 2 倍がかかります。この関係はリニア (直線的) です。マスタとそれぞれスレーブの能力は同じで、1 マスタと N スレーブがあると想定します。それぞれのサーバ (マスタまたはスレーブ) は次のようになります。

$$\text{reads} = 1200 - 2 \times \text{writes}$$

$$\text{reads} = 9 \times \text{writes} / (N + 1) \text{ (読み取りは分散、書き込みはすべてのサーバへ)}$$

$$9 \times \text{writes} / (N + 1) + 2 \times \text{writes} = 1200$$

$$\text{writes} = 1200 / (2 + 9/(N+1))$$

最後の数式は、 N スレーブ毎の最大数を示し、ここでは、毎分 1,200 の可能な最大読み込み割合と書き込みあたり 9 の読み込み割合と仮定しています。

この分析は、次の結論を導き出します。

- $N = 0$ (レプリケーションがないことを意味する) の場合、システムは $1200/11$ 、つまり毎秒 109 書き込みを処理できる (アプリケーションの性質上、読み取りが 9 倍になる)
- $N = 1$ の場合、毎秒 184 の書き込みまで処理可能
- $N = 8$ の場合、毎秒 400 の書き込みまで処理可能
- $N = 17$ の場合、毎秒 480 の書き込みまで処理可能
- N が無限に近づくと (割当量も無限大に膨らみ)、毎秒 600 の書き込みに近くなり、システム スループットは 5.5 倍になる。しかし、8 サーバだけで 4 倍近くなる。

注意: この計算ではシステム帯域を無限として想定しています。そのため、実際のシステムでは重要であるファクタを無視している可能性があります。多くの場合、 N アプリケーション スレーブを追加した場合の結果を正確に予測するために、上記と同様の計算を行うことは適していません。このため、レプリケーションによってシステムのパフォーマンスが改善するかどうか、またどの程度改善するか、以下の質問の答えを参照して判断してください。

- システムの読み取りと書き込みの比率
- 読み取りを減らした場合、1 つのサーバで処理できる書き込み負荷を増やせる程度
- ネットワークの帯域幅を使用できるスレーブ数の上限

5.4.4.9: 冗長性と高可用性を実現するために、レプリケーションをどのように使用すればよいですか。

どのように冗長性を向上するかは、使用しているアプリケーションと状況により異なります。(自動フェールオーバーを伴う) 高可用性ソリューションは、アクティブなモニタリングに加え、オリジナルのMySQL サーバからそのスレーブへのフェールオーバーを行うためのカスタム スクリプトまたはサードパーティのツールのいずれかを必要とします。

この処理を手動で行うには、失敗したスレーブから事前構成のスレーブに移行できることを確認してください。(失敗時にアプリケーションとスレーブにマスタの変更を指示するスクリプトを作成する)。これを行うには、新たなサーバと通信するアプリケーションで代替するか、または、失敗したサーバから新たなサーバに MySQL サーバの DNS を調節します。

詳細とソリューション例に関しては [フェイルオーバーでのマスタ切り替え](#) を参照してください。

5.4.4.10: マスタ サーバのロギング形式が、ステートメント ベースまたは行ベースのどちらであるかを確認する方法はありますか。

`binlog_format` システム変数の値を確認します。

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

その値は `STATEMENT` または `ROW` のどちらかを示します。

5.4.4.11: スレーブが行ベースのレプリケーションを使用するように設定する方法はありますか。

スレーブは自動的にどちらの形式を使用するかを認識します。

5.4.4.12: スレーブのマシンへの複製で、GRANT および REVOKE のステートメントを抑制する方法はありますか。

サーバを `--replicate-wild-ignore-table=mysql.%` オプションで起動します。.

5.4.4.13: 混在のオペレーティング システムでレプリケーションを実行できますか。(例：マスタが Linux、スレーブが Mac OS X と Windows の場合)

はい、できます。

5.4.4.14: 混在のハードウェア アーキテクチャでレプリケーションを実行できますか。(マスタが 64-bit、スレーブが 32-bit のマシンの場合など)

はい、できます。

レプリケーションのトラブルシューティング

指示に従って設定して、レプリケーションを設定しても機能しない場合は、エラー ログのメッセージをまず調べてください。多くの問題は、ログを調べることでより解決します。

エラー ログから問題の発生源が特定できない場合は、次の事柄を確かめます。

- マスタのバイナリ ログで記録しているかどうかを、`SHOW MASTER STATUS` で確認する。記録していれば、`Position` はゼロ以外の値。記録していない場合、マスタの `--log-bin` および `--server-id` で設定を確認する。
- スレーブが稼働しているかどうかを確認する。`SHOW SLAVE STATUS` を実行し、`Slave_IO_Running` および `Slave_SQL_Running` の値が両方とも `Yes` であることを確認する。Yes ではない場合、スレーブ サーバを起動するときのオプションを確認する。ヒント：`--skip-slave-start` は、`START SLAVE` クエリが出るまで、スレーブ スレッドの起動を抑制します。
- スレーブが稼働している場合は、マスタとの接続を確認する。`SHOW PROCESSLIST` を実行して、I/O スレッドと SQL スレッドを見つけ、その `State` カラムでの表示を確認する。詳細は [レプリケーション実装の詳細](#) を参照してください。I/O スレッドが `Connecting to master` である場合は、次のことを確認する。
 - レプリケーション ユーザのマスタでの権限を確認する。

- マスタ ホスト名が正当であるかどうかを調べ、適切なポートでマスタと接続しているかどうかを確認する。レプリケーションに使用するポートはクライアント ネットワーク通信のものと同一。(デフォルトは3306)ホスト名に関しては、適切な IP アドレスと対応しているかどうかを確認する。
- マスタ - スレーブ間のネットワーク接続が無効化しているかどうかを確認する。コンフィギュレーション ファイルで `skip-networking` オプションを調べる。ここでコメントがある、ない場合は完全に削除された可能性がある。
- マスタにファイアウォールもしくはIP フィルタのコンフィギュレーションなどを設定している場合は、MySQL で使用しているネットワーク ポートがフィルタされているかどうかを確認する。
- `ping` もしくは `traceroute/tracert` を使用して、マスタがホストに達しているかどうかを確認する。
- スレーブが以前は稼動していた、そして停止した場合は、マスタで実行されたステートメントがスレーブで失敗している可能性がある。ノート：これは、マスタの適切なスナップショットが行われている場合には発生しません。または、スレーブ スレッドの外側のスレーブでデータを修正しなければ発生することはありません。スレーブが不意に停止した場合、バグである可能性、または既存のレプリケーション障害である可能性がある。詳細は [レプリケーション機能と既知問題](#) を参照。ノート： 原因がバグによるものである場合は、[レプリケーション バグまたは問題を報告する方法](#) を参照し、どのように報告するか指示に従ってください。
- マスタで実行したステートメントがスレーブで実行できない場合は、次の手順に従う。それができない場合は、スレーブのデータベースを削除し、新たなスナップショットをマスタからコピーする方法で、データベースの全体的な再同期化を行う。
 1. 影響する可能性のあるスレーブのテーブルがマスタのテーブルとは異なるかどうかを判断する。これを処理することにより、なにが起こるか、なにが必要になるかを検討する。そして、マスタのテーブルと同一になるようスレーブのテーブルを作成し、`START SLAVE` を実行する。
 2. 前述のステップが機能しない、または適用されない場合は、必要に応じて、マニュアルでの更新を安全に実行できるかどうかを検討し、マスタから次に出されるステートメントを無視する。
 3. マスタからの次のステートメントをスキップすると決めた場合は、以下のクエリを実行する。


```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N;
mysql> START SLAVE;
```

マスタから出される次のステートメントが `AUTO_INCREMENT` または `LAST_INSERT_ID()` を使用しない場合は、`N` の値は 1。そうでない場合は、その値は 2。ノート： `AUTO_INCREMENT` または `LAST_INSERT_ID()` を使用するステートメントの値が 2 である必要がある理由は、マスタのバイナリ ログで 2 イベントを処理するためです。
 4. スレーブがマスタと最初から完全に同期していた、そしてスレーブのスレッドの外側のテーブルを更新していないと確認できる場合は、バグによる不具合である可能性がある。ノート： 最新バージョンの MySQL を使用している場合は、その問題を報告してください。古いバージョンを使用している場合は、最新のプロダクションリリースにアップグレードし、それでもその問題が続くかどうかを確認する。

レプリケーション バグまたは問題を報告する方法

ユーザによるエラーではないことが確認でき、レプリケーションが全く機能しないまたは不安定である場合は、バグとして報告してください。バグ問題を解決するには、より多くの情報を必要とします。そのため、バグ報告を行う際には、時間をかけて、できるだけ詳細な情報を送信してください。

そのバグを実証する再現可能なテスト ケースがある場合には、[質問またはバグの報告](#) を参照して、それをバグ用のデータベースに入れてください。「phantom」問題(説明できない問題など)の場合は、次の手順に従ってください。

1. ユーザによるエラーではないことを確認します。例：スレーブ スレッド外側のスレーブを更新する場合は、非同期のデータになり、更新時にユニーク キー制約である可能性があります。この場合、スレーブ スレッドは停止している状態で、それらを同期で持ち込むために、手動によるテーブルのクリーン アップを待機しています。これは、レプリケーションに関する問題ではありません。これは外部からの障害がレプリケーションの失敗に起因しているということです。
2. `--log-slave-updates` および `--log-bin` オプションでスレーブを実行します。これらのオプションでスレーブがマスタから受信する更新をスレーブのバイナリ ログに記録するようにします。

3. レプリケーション状態をリセットする前にすべての証拠を保存します。ノート：この問題を解決するには、できるだけ多くの問題を控えておいてください。収集する必要がある証拠

- マスタからのすべてのバイナリ ログ
- スレーブからのすべてのバイナリ ログ
- 問題を認識した時点のマスタからの `SHOW MASTER STATUS` 出力
- 問題を認識した時点のスレーブからの `SHOW SLAVE STATUS` 出力
- マスタおよびスレーブのエラー ログ

4. バイナリ ログを調べるには、`mysqlbinlog` を使用します。次に示すものは、問題があるステートメントを探し出すために役立ちます。`log_pos` および `log_file` は、`SHOW SLAVE STATUS` からの `Master_Log_File` および `Read_Master_Log_Pos` 値です。

```
shell> mysqlbinlog -j log_pos log_file | head
```

問題からの証拠を収集した後は、まず全く別のテスト ケースとして、それを隔離してください。そして、より多くの情報とともに、バグ用のデータベースに入れてください。詳細は [質問またはバグの報告](#) で参照してください。

レプリケーションの実装

レプリケーションのメカニズムは、マスタサーバが使用中のデータベースへの変更(更新、削除など)を追跡し続け、マスタサーバのバイナリ ログからクエリを読み込むというに基づきます。バイナリ ログは、データベース化が開始された時点からの記録として、その役割を果たします。バイナリ ログは、データベース ストラクチャまたはストラクチャの保持データを編集または修正するクエリすべての記録を含みます。`SELECT` ステートメントは、記録されません。同様にデータベースのデータまたはストラクチャも修正されません。

マスタに接続しているそれぞれのスレーブは、バイナリ ログのコピーを受信し、そのバイナリ ログ内のイベントを実行します。これには、元の クエリ (ステートメント) をリピートし、マスタで作成されたように変更する、ということです。マスタ元で実行したクエリに従って、テーブル作成またはストラクチャ修正、データの挿入、削除、更新が行われます。

それぞれのスレーブが独立しているため、マスタのバイナリ ログにあるクエリのリプレイを、マスタに接続しているそれぞれのスレーブで行います。さらに、それぞれのスレーブは、バイナリ ログをマスタに要求することによってコピーを受け取る、つまり、マスタがスレーブにデータを与える、というよりは、マスタからデータを取り込むため、スレーブはデータベースのコピーの読み取りおよび書き込みをスレーブのベースおよびレートで行います。そのため、データベースの最新情報を更新する際には、マスタまたはスレーブの処理能力を損なうことなく、複製プロセスの開始および停止ができます。

レプリケーションの実装に関する詳細は、[レプリケーション実装の詳細](#)を参照してください。

スレーブとマスタは、レプリケーション プロセスに関するステータスを定期的にレポートするため、処理状況の監視が可能です。スレーブの状態に関する詳細は、[スレーブ レプリケーションの I/O スレッド状態](#) もしくは [スレーブ レプリケーションの SQL スレッド状態](#) を参照してください。マスタの状態に関する詳細は、[マスタレプリケーションのスレッド状態](#) を参照してください。

マスタのバイナリ ログは、プロセスされる前に、スレーブのローカル リレー ログが記憶します。スレーブもまた、マスタのバイナリ ログおよびローカル リレーのログとの位置関係を記録します。詳細は、[レプリケーション リレーとステータス ファイル](#) を参照してください。

クエリ評価をコントロールする変数およびコンフィギュレーションのオプションに従って適用されたルールの組み合わせを基に、データベースとテーブルの更新はスレーブで行います。これらのルールがどのように適用されるかに関しては、[サーバのレプリケーション ルール評価](#) を参照してください。

レプリケーション実装の詳細

MySQL レプリケーションには 3 つのスレッドが関連しています。マスタには 1 スレッド、スレーブでは2つのスレッドがあります。`START SLAVE` ステートメントが発行されると、I/O スレッドがスレーブに作成されます。このスレッドはマスタに接続し、マスタのバイナリ ログに記録されているクエリの送信を要求します。そして、マスタが、そ

のバイナリ ログを送信するために、スレッドを作成します。このスレッドは、マスタの `SHOW PROCESSLIST` 出力で `Binlog Dump` として確認できます。スレーブの I/O スレッドは、マスタの `Binlog Dump` スレッドが送信するアップデートを読み取り、それをスレーブのデータ ディレクトリにある relay logs と呼ばれるローカル ファイルにコピーします。そして、スレーブがリレー ログを読み取りそれに含まれるクエリを実行するために、3 つ目の スレッドである SQL スレッドを作成します。

マスタとスレーブ間の 1 接続につき、3 つのスレッドがあります。つまり、マスタには複数のスレーブがあり、マスタは接続したスレーブごとにスレッドを作成します。そして、このスレーブには I/O スレッドおよび SQL スレッドの両方が存在します。

スレーブは2つのスレッドを使い、マスタからのアップデートを読み取りと実行を2つの独立したタスクに区切りまします。これにより、クエリの実行が遅い場合でも、クエリの読み取りというタスクが遅くなることはありません。たとえば、スレーブ サーバが一時的に稼働していない場合に、SQL スレッドが遅れていたとしても、スレーブが起動する際には、I/O スレッドがマスタのすべてのバイナリ ログ内容をすばやく取り出します。つまり、SQL スレッドが取り出したクエリのすべてを実行し終わる前にスレーブが停止する場合でも、I/O スレッドの方ですべてを取り出しているため、クエリのコピーがスレーブのリレー ログで安全に格納されていることになり、スレーブが起動するときには、そのクエリを実行する準備ができています。このため、マスタはスレーブの読み出しを待つ必要がなくなり、マスタサーバがマスタ元のバイナリ ログをより早く削除することを可能にします。

`SHOW PROCESSLIST` ステートメントを使用すると、レプリケーションに関するマスタ側およびスレーブ側での処理内容を確認できます。以下で、`SHOW PROCESSLIST` の出力で、この3つのスレッドがどのように表示されるかを例示します。

マスタサーバでは、`SHOW PROCESSLIST` の出力は以下のようになります。

```
mysql> SHOW PROCESSLIST
***** 1. row *****
  Id: 2
  User: root
  Host: localhost:32931
  db: NULL
  Command: Binlog Dump
  Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
        be updated
  Info: NULL
```

ここで、スレッド Id 2 は 接続したスレーブの `Binlog Dump` レプリケーション スレッドです。`State` の情報は、すべてのアップデート (更新情報) がスレーブに送信されたことを示し、マスタは次の更新情報を待機している状態です。マスタサーバの `Binlog Dump` スレッドが表示されない場合は、このレプリケーションは実行されていないことを意味します。つまり、この時点でスレーブが接続されていないということです。

スレーブ サーバでは、`SHOW PROCESSLIST` の出力は以下のようになります。

```
mysql> SHOW PROCESSLIST
***** 1. row *****
  Id: 10
  User: system user
  Host:
  db: NULL
  Command: Connect
  Time: 11
  State: Waiting for master to send event
  Info: NULL
***** 2. row *****
  Id: 11
  User: system user
  Host:
  db: NULL
  Command: Connect
  Time: 11
  State: Has read all relay log; waiting for the slave I/O
        thread to update it
  Info: NULL
```

この情報は、スレッド Id 10 が I/O スレッドであり、マスタサーバと通信していることを示し、スレッド Id 11 は SQL スレッドであり、リレー ログに格納されたアップデートを処理中であることを示します。`SHOW PROCESSLIST` が実行された場合、スレッドの両方がアイドル状態であり、次のアップデートを待機しています。

Time カラムの値は、マスタと比較した場合に、スレーブがどれくらい遅れているかを示します。詳しくは [レプリケーション FAQ](#) を参照してください。

マスタレプリケーションのスレッド状態

以下の一覧は、マスタの **Binlog Dump** スレッドの **State** カラムに表示される一般的な状態です。**Binlog Dump** スレッドがマスタサーバに表示されない場合、このレプリケーションは実行されていないことを意味します。つまり、この時点でスレーブは接続していないということです。

- **Sending binlog event to slave**

バイナリ ログはクエリなどの情報を組み合わせたイベントを含む。バイナリ ログからのイベントを読み取り、その内容をスレーブに送信中であることを示す状態。

- **Finished reading one binlog; switching to next binlog**

バイナリ ログ ファイルの読み取りを完了し、スレーブに送信する次のファイルを開いている状態。

- **Has sent all binlog to slave; waiting for binlog to be updated Info:**

すべてのバイナリ ログ ファイルからのアップデートの読み取りを完了し、スレーブそれを送信した状態。マスタで発生中のアップデートの結果から新たなイベントがバイナリ ログに出ることを待機しているアイドル状態。

- **Waiting to finalize termination**

スレッド停止中に発生する一時的な状態。

スレーブ レプリケーションの I/O スレッド状態

以下の一覧は、スレーブの I/O スレッドの **State** カラムに表示される一般的な状態です。これは、**SHOW SLAVE STATUS** 出力の **Slave_IO_State** カラムにも表示され、このステートメントを使用して現状を把握できます。

- **Connecting to master**

マスタへの接続を試行中の状態。

- **Checking master version**

マスタへの接続が確立した直後に発生する瞬間的な状態。

- **Registering slave on master**

マスタへの接続が確立した直後に発生する瞬間的な状態。

- **Requesting binlog dump**

マスタへの接続が確立した直後に発生する瞬間的な状態。マスタにバイナリ ログの内容を送信するよう要求している状態。要求したバイナリ ログのファイル名と位置を最初に要求する。

- **Waiting to reconnect after a failed binlog dump request**

接続切断などにより、バイナリ ログ ダンプ要求に失敗した場合の状態。スレッドはスリープ中この状態になり、定期的に再接続を試みる。ノート： **--master-connect-retry** オプションを使用して、再接続のインターバルを設定できます。

- **Reconnecting after a failed binlog dump request**

マスタへの再接続を試行中の状態。

- **Waiting for master to send event Master_Host:**

マスタへの接続を完了し、バイナリ ログ イベントの到達を待機している状態。ノート： マスタがアイドル状態の場合には、この状態の待機時間は長くなり、**slave_net_timeout** 秒継続した場合、タイムアウトになります。そのときには接続切断とみなされ、スレッドは再接続を試行します。

- [Queueing master event to the relay log](#)

イベントの読み取りが完了し、SQL スレッドで処理できるように、読み取ったイベントをリレー ログにコピー中の状態。

- [Waiting to reconnect after a failed master event read](#)

接続切断などが原因で、読み取り中にエラーが発生した状態。再接続が試行される前に、[master-connect-retry](#) 秒間、スリープ状態になる。

- [Reconnecting after a failed master event read](#)

スレッドがマスタへの再接続を試行中の状態。再び接続が確立されると、[Waiting for master to send event](#) の状態になる。

- [Waiting for the slave SQL thread to free enough relay log space](#)

0 以外の [relay_log_space_limit](#) 値を使用しているため、合計サイズがその値を超えるまでに、リレー ログが過大した状態。I/O スレッドはリレー ログ ファイルの一部を削除するために、リレー ログ内容を処理しながら、SQL スレッドに領域余裕ができるのを待機している状態。

- [Waiting for slave mutex on exit](#)

スレッド停止中に発生する一時的な状態。

スレーブ レプリケーションの SQL スレッド状態

以下の一覧は、スレーブの SQL スレッドの [State](#) カラムに表示される一般的な状態です。

- [Reading event from the relay log](#)

イベントを処理するために、そのイベントをリレー ログから読み取っている状態。

- [Has read all relay log; waiting for the slave I/O thread to update it](#)

リレー ログ ファイルのイベントをすべて処理し、I/O スレッドが新たなイベントがリレー ログに書き込むことを待機している状態。

- [Waiting for slave mutex on exit](#)

スレッド停止中に発生する一時的な状態。

I/O スレッドの [State](#) カラムはクエリ文字列を表示する場合があります。これは、スレッドがリレー ログからイベントを読み取り、クエリを抽出して、そのクエリを実行中であることを示します。

レプリケーション リレーとステータス ファイル

レプリケーション中にMySQL サーバは、マスタからリレーされたバイナリ ログの保持に使うファイルをいくつか作成し、リレーされたログ内からステータスと位置に関する情報を記録します。このプロセスでは 3 種類のファイル タイプを使用します。

- relay log は、マスタのバイナリ ログから読み込まれたイベントから成る。バイナリ ログのイベントは、レプリケーション スレッドの一環としてスレーブで実行する。

- master.info ファイルはステータスやスレーブがマスタへ接続する際のコンフィギュレーション情報を含む。このファイルは、マスタのホスト名、ログイン資格情報、そしてマスタのバイナリ ログ内の現状位置に関する情報を保持する。

- relay.info ファイルは、スレーブのリレー ログ ファイル内の実行ポイントに関するステータス情報を保持する。

以下は、この 3 種類のファイルとレプリケーション プロセスの関係です。[master.info](#) ファイルは、マスタ バイナリ ログ内のポイントを保持し、このログはマスタからの読み込みです。そのリレー ログにこの読み出されたイベントを記憶します。[relay.info](#) ファイルは、リレー ログ内にあるクエリの位置を記録し、このクエリは実行済みです。

スレーブ リレー ログ

リレー ログのファイル名はデフォルトで `host_name-relay-bin.nnnnnn` という形式です。`host_name` はスレーブ サーバ ホスト名、`nnnnnn` の部分はシーケンス番号です。連続するリレー ログ ファイルは、連続するシーケンス番号になり、この番号は `000001` で始まります。スレーブはインデックス ファイルを使用して、使用中のリレー ログ ファイルを追跡します。リレー ログのインデックス ファイル名はデフォルトで、`host_name-relay-bin.index` です。

スレーブ サーバは、リレー ログ ファイルをデータ ディレクトリにデフォルトで作成します。デフォルトのファイル名は、`--relay-log` および `--relay-log-index` の各サーバ オプションで上書きできます。詳しくは [レプリケーションのオプションと変数](#) を参照してください。

リレー ログはバイナリ ログと同様のフォーマットであるため、`mysqlbinlog` を使用して読み取れます。SQL スレッドは、ファイル内のイベントをすべて実行した後、不要になったリレー ログ ファイルを自動的に削除します。SQL スレッドがこの作業を行うため、リレー ログ削除のコマンドは不要です。ただし、`FLUSH LOGS` がリレー ログをローテートする場合は、SQL スレッドがそれらを削除するタイミングに影響します。

以下の条件で、スレーブ サーバは新たなリレー ログ ファイルを作成します。

- (スレーブサーバの起動後、最初に) I/O スレッドが開始された場合
- ログをフラッシュ (一括書き出し) する場合。例: `FLUSH LOGS` または `mysqladmin flush-logs`
- その時点でのリレー ログ ファイルのサイズが過大化した場合以下は、「過大化」の定義。
 - `max_relay_log_size` > 0 の場合 (この値はファイル サイズの最大値)
 - `max_relay_log_size` = 0 の場合 (`max_binlog_size` はファイル サイズの最大値を決定)

スレーブ ステータス ファイル

スレーブ レプリケーション サーバは、データ ディレクトリに ファイルを2 つ作成します。これらのファイルは status files (ステータス ファイル) と呼ばれ、デフォルトで `master.info` および `relay-log.info` というファイル名です。このデフォルトのファイル名は `--master-info-file` および `--relay-log-info-file` オプションを使用して変更できます。詳しくは [レプリケーションのオプションと変数](#) を参照してください。

この2つのステータス ファイルは、`SHOW SLAVE STATUS` ステートメントの出力に表示される情報を含みます。(コマンドの説明については [スレーブ サーバをコントロールする SQL ステートメント](#) を参照)ステータス ファイルはディスクに保存されるため、スレーブ サーバがシャットダウンした場合でも保持されます。そのため、次回にスレーブが起動する際には、スレーブがこの2つのファイル (マスタのバイナリ ログおよびスレーブ自体のリレー ログ) から前回の進み具合を読み取ります。

I/O スレッドは `master.info` ファイルを更新します。以下のテーブルは、ファイル内のライン (行) と `SHOW SLAVE STATUS` で表示されるカラムの対応表です。

行	カラム ステータス	説明
1		ファイル内のライン番号
2	<code>Master_Log_File</code>	マスタのバイナリ ログ名。このログはその時点でマスタ側から読み込み中。
3	<code>Read_Master_Log_Pos</code>	マスタのバイナリ ログ内の現在位置。その時点分のマスタの読み込みは完了している。
4	<code>Master_Host</code>	マスタのホスト名
5	<code>Master_User</code>	マスタに接続するためのユーザ名
6	パスワード (<code>SHOW SLAVE STATUS</code> では表示されない)	マスタに接続するためのパスワード
7	<code>Master_Port</code>	マスタに接続するためのネットワーク ポート
8	<code>Connect_Retry</code>	インターバル時間(秒)。スレーブがマスタに再接続を試行する際に待機する時間
9	<code>Master_SSL_Allowed</code>	サーバが SSL 接続をサポートするかどうかを示す

10	Master_SSL_CA_File	証明機関 (Certificate Authority) に使用したファイル
11	Master_SSL_CA_Path	証明機関 (CA) へのパス
12	Master_SSL_Cert	SSL 証明のファイル名
13	Master_SSL_Cipher	SSL 接続に使用している暗号法 (サイファ) 名
14	Master_SSL_Key	SSL キー ファイル名

SQL スレッドは [relay-log.info](#) ファイルを更新します。以下のテーブルは、ファイル内のライン (行) と [SHOW SLAVE STATUS](#) で表示されるカラムの対応表です。

行	スタートス カラム	説明
1	Relay_Log_File	現行のリレー ログ ファイル名
2	Relay_Log_Pos	リレー ログ ファイル内の現在位置。この位置までのイベントはスレーブのデータベースで実行済み。
3	Relay_Master_Log_File	マスタの (リレー ログ ファイルで読み取られたイベントの) バイナリ ログ ファイル名
4	Exec_Master_Log_Pos	(既に実行されたイベントの) マスタのバイナリ ログ ファイル内の対応位置

注意 : [relay-log.info](#) ファイルをディスクにフラッシュしていない場合、[SHOW SLAVE STATES](#) コマンド表示のステート (状態) および [relay-log.info](#) ファイルの内容が一致しない可能性があります。スレーブの [relay-log.info](#) はオフラインのときにだけ閲覧することをお勧めします。(mysql が稼動していないときなど)システムが稼動している場合は、[SHOW SLAVE STATUS](#) を使用してください。

サーバのレプリケーション ルール評価

マスタサーバがバイナリ ログにステートメント (クエリ) を記憶しない場合、ステートメントは複製されません。サーバがステートメント (クエリ) をログする場合、そのステートメントはすべてのスレーブに送信されます。そして、それぞれのスレーブ (受信側) が受信したステートメントを実行するかどうかを決めます。

ノート : バイナリへのログをコントロールするには、[--binlog-do-db](#) および [--binlog-ignore-db](#) のオプションを使用して、どのデータベースがバイナリ ログにイベントを書き込むかをマスタでコントロールできます。これらのオプションを評価する際にサーバが使用するルールの詳細は、[バイナリ ログ](#) を参照してください。注意 : スレーブで実行しているイベントをコントロールするには、スレーブにフィルターを使います。複製されたデータベースおよびテーブルをコントロールする目的で、これらのオプションを使用しないでください。

スレーブ側では、マスタから受信したクエリを実行するかどうかの決定は、スレーブ起点の [--replicate-*](#) オプション (ルール) に従って行われます。([レプリケーションのオプションと変数](#) を参照してください。) 以下の手順に従い、スレーブはこれらのオプションを評価します。最初に、データベース レベルのオプションを比較し、続いてテーブルレベルのオプションを比較します。

[--replicate-*](#) オプションがないなど、ルールが単純な場合は、この手順により、スレーブがマスタから受信するすべてのクエリを実行するという結果を生成します。それ以外には、この結果は設定オプションに依存します。ノート : 「do」 および 「ignore」 のオプション、またはワイルド カードなどのオプションの混在を避けることで、ルール セットの設定効果を決定する作業が簡素化します。

ステージ 1. データベース オプションの評価

このステップでは、データベース設定を特定する [--replicate-do-db](#) または [--replicate-ignore-db](#) ルールがあるかどうかを、スレーブが確認します。

- No: クエリを許可して、テーブル比較のステップへ進む。
- Yes: クエリを許可もしくは無視する決定を行うために、[--binlog-do-db](#) および [--binlog-ignore-db](#) のオプションと同様のルールを使用して、オプションをテストする。(以下の選択をするために) テスト結果を確認する。
 - Permit: すぐには実行しない。決定を保留して、テーブル比較ステップに進む。
 - Ignore: クエリを無視して終了する。

このステージでは、後続のオプション確認するという目的で、クエリを許可または無視するようになります。そのため、クエリは許可されますが、実際にはまだ実行されません。後続のステージで、テーブル オプションを比較するステップへ進みます。

ステージ 2. テーブル オプションの評価

最初に、前段階として、スレーブ側でクエリ ベースのレプリケーションが可能かどうかをテストします。それが可能であり、格納された機能内でクエリが発生する場合、そのクエリを実行して、終了します。メモ：行ベースのレプリケーションを可能にした場合、スレーブ側はマスタ内の格納機能でクエリが発生したことを知らないため、その条件は有効になりません。

次に、スレーブ側でテーブル オプションをテストし、それらを評価します。サーバがこのポイントに達する際に、テーブルにオプションがない場合、サーバはすべてのクエリを実行します。テーブルに「do」ルールが複数ある場合には、実行するクエリはそのテーブルのルールのうち 1 つと一致する必要があります。一致しない場合は、無視されます。「ignore」ルールが複数ある場合、「ignore」ルールに一致するものを除き、すべてのクエリが実行されます。以下のステップで、この評価の手順を説明します。

1. `--replicate-*table` ルールはあるか

- No: テーブルに制限がない。すべてのクエリが一致する。クエリを実行して、終了する。
- Yes: テーブルに制限がある。更新対象のテーブルと既存ルールを比較する。複数のテーブルを更新する可能性がある。次のステップで、それぞれのテーブルに一致するルールを探す。ノート：この場合、クエリ ベースまたは行ベースのどちらでレプリケーションが可能になるかによって、その後の動作は左右されます。
- クエリ ベースのレプリケーション: 次のステップに進み、表示された順序でテーブル ルールの評価を開始する。(注意：最初にノン ワイルド、続いて、ワイルド)更新対象のテーブルだけがルールと比較される。(例：クエリが `INSERT INTO sales SELECT * FROM prices` の場合、`sales` だけがルールとの比較対象)複数のテーブル (マルチ テーブルのクエリ) が更新対象の場合、「do」または「ignore」で一致する最初のテーブルが比較される。つまり、サーバ側は、最初のテーブルをルールと比較する。その比較で、決定できない場合は、2番目のテーブルがルールとの照合になる。
- 行ベースのレプリケーション: テーブル行すべての変更は別々にフィルタにかかる。対象テーブルが複数の場合、それぞれのテーブルが別々に、ルールに従ってフィルタにかかる。ルールと変更内容をよっては、更新が実行される部分と実行しない部分がある。行ベースのレプリケーションでは、クエリ ベースのレプリケーションで正確に複製されないケースを正確に処理する。以下は、`foo` データベースのテーブルを複製する必要があると仮定した場合の例。

```
mysql> USE bar;
mysql> INSERT INTO foo.sometable VALUES (1);
```

2. `--replicate-do-table` ルールはあるか

- No: 次のステップへ進む。
- Yes: テーブルはどれかと一致するか。
 - No: 次のステップへ進む。
 - Yes: クエリを実行して、終了する。

3. `--replicate-ignore-table` ルールはあるか。

- No: 次のステップへ進む。
- Yes: テーブルはどれかと一致するか。
 - No: 次のステップへ進む。
 - Yes: クエリを無視して、終了する。

4. `--replicate-wild-do-table` ルールはあるか。

- No: 次のステップへ進む。

- Yes:テーブルはどれかと一致するか。
 - No:次のステップへ進む。
 - Yes:クエリを実行して、終了する。
5. `--replicate-wild-ignore-table` ルールはあるか。
- No:次のステップへ進む。
 - Yes:テーブルはどれかと一致するか。
 - No:次のステップへ進む。
 - Yes:クエリを無視して、終了する。
6. `--replicate-*-table` ルールはどれにも一致しなかった。これらのルールでテストするテーブルはほかにあるか。
- No:すべての対象テーブルをテストしたが、どのルールにも一致しない。`--replicate-do-table` または `--replicate-wild-do-table` ルールはあるか
 - No:テーブルの「do」ルールがない。ゆえに「do」との明確な一致は不要。クエリを実行して、終了する。
 - Yes:テーブルに「do」ルールがある。それと明確に一致したものだけでクエリを実行。クエリを無視して、終了する。
 - Yes:ループする。

例：

- `--replicate-*` ルールがまったくない。

スレーブは、マスタから受信するすべてのクエリを実行する。

- `--replicate-*-db` ルールはあるが、テーブル ルールがない。

スレーブはデータベースのルールを使用して、クエリを許可または無視する。そしてテーブルでの制限がないので、スレーブはデータベースのルールによって許可したすべてのクエリを実行する。

- `--replicate-*-table` ルールはあるが、データベース ルールがない

データベースに条件がないため、すべてのクエリがデータベース比較の段階で許可される。スレーブは、テーブルのルールを基にすべてのクエリを実行または無視する。

- データベースとテーブルのルールが混在する場合。

スレーブはデータベースのルールを使用して、クエリを許可または無視する。そして、テーブルのルールに従って、スレーブはデータベースのルールで許可したすべてのクエリを実行する。場合によって、このプロセスは直感に反する結果のようなものを生成する可能性があります。その場合、以下のルール セットを検討する。

```
[mysqld]
replicate-do-db = db1
replicate-do-table = db2.mytbl2
```

db1 がデフォルトのデータベースであり、スレーブがクエリを受信する場合。

```
INSERT INTO mytbl1 VALUES(1,2,3);
```

このデータベース **db1**は、データベース比較の段階で、`--replicate-do-db` ルールに一致します。よって、アルゴリズムはテーブル比較の段階でプロセスされます。テーブルのルールがない場合、クエリは実行されます。ルールには「do」のテーブル ルールが含まれるため、クエリを実行する場合は、そのクエリが一致するものである必要があります。この場合のクエリは、一致しないため、無視されます。**db1** のどのテーブルでも同様の場合があります。

第6章 最適化

目次

最適化の概要	372
MySQL の設計上の制約とトレードオフ	372
移植性のためのアプリケーション設計	372
MySQL 使用実績	373
MySQL ベンチマークスイート	374
独自のベンチマークの使用	374
SELECTステートメントおよびその他のクエリの最適化	375
EXPLAINを使用して、クエリを最適化する	375
クエリパフォーマンスの推定	384
SELECTクエリ速度	384
WHERE 節最適化	384
Range 最適化	386
インデックス結合最適化	389
IS NULL最適化	391
DISTINCT最適化	391
LEFT JOINとRIGHT JOIN最適化	392
入れ子結合最適化	393
外側Join 単純化	397
ORDER BY最適化	399
GROUP BY最適化	401
LIMITの最適化	402
テーブルスキャンを避ける方法	403
INSERTステートメントの速度	403
UPDATEステートメントの速度	405
DELETEステートメントの速度	405
その他の最適化のヒント	405
ロック関連の問題	408
MySQL のテーブルロック方法	408
テーブルロック関連の問題	410
同時挿入	411
データベース構造の最適化	411
設計上の選択	411
データの小型化	412
カラムインデックス	413
複合インデックス	413
MySQLにおけるインデックスの使用	414
MyISAMキーキャッシュ	416
MyISAMインデックス統計コレクション	420
MySQL でのテーブルのオープンとクローズの方法	422
1 つのデータベースに大量のテーブルを作成した場合の欠点	423
MySQL サーバの最適化	423
システム、コンパイル時間およびスタートアップパラメータのチューニング	423
サーバパラメータのチューニング	424
クエリオプティマイザパフォーマンスの管理	428
MySQL の速度に対するコンパイルとリンクの影響	429
MySQL でのメモリの使用	430
MySQLの DNS の使用	431
ディスク関連の問題	431
シンボリックリンクの使用	432

最終的に全てのシステムの理解が必要なため、最適化は複雑な作業です。ユーザのシステムやアプリケーションの知識でローカル最適化は可能ですが、システムをさらに最適化するには、システムに関する深い知識が必要になります。

この章ではMySQLを最適化するためのさまざまな手法を、例を交えて説明します。取得には更なる努力が必要ですが、必ずシステムの速度を上げる方法は数多く存在することを覚えていてください。

最適化の概要

システムの速度を上げる際に最も重要な要素は基本設計です。また、使用するシステムの用途およびそのボトルネックを認識しておく必要もあります。最も一般的なボトルネックは下記のとおりです。

- ディスクシーク。ディスクが1つのデータを検索するには時間がかかります。最新のディスクでは、通常これにかかる平均時間が10 ms未満であるため、理論的には1秒間に100のシークを実行できることになります。新しいディスクではこの時間の改善が緩やかで、1つのテーブルの最適化が非常に困難です。これを最適化する方法として、複数のディスクにデータを分散することが挙げられます。
- ディスクの読み取りと書き込み。ディスクが適切な位置にある場合、データの読み取りが必要になります。最新のディスクでは、1つのディスクで約10-20MB/sの読み取りが可能になります。これは、複数のディスクから並行した読み取りが可能であるため、シークに比較して最適化が容易に行えます。
- CPU サイクル。メインメモリにデータがある場合（またはすでにそこに存在している場合）、結果を得るためには処理が必要になります。メモリと比較してテーブルが小さい場合は、最も一般的な制限要因になります。しかし、テーブルが小さくても、一般に速度上の問題は発生しない。
- メモリ帯域幅。CPU キャッシュの適正量より多く CPU がデータを必要とする場合、メインメモリの帯域幅がボトルネックになる。これは、ほとんどのシステムに一般的な問題ではないが、認識しておく必要がある。

MySQL の設計上の制約とトレードオフ

[MyISAM](#)ストレージエンジンの使用時に、MySQL では非常に高速のテーブルロック（複数リーダ/単一ライタ）が使用されます。このテーブル型の最大の問題は、同じテーブルに対して複数の UPDATE と遅い SELECT が混在する場合に発生します。テーブルでこのような問題が発生した場合は、別のテーブル型を使用してもかまいません。[13章ストレージエンジンとテーブルタイプ](#)を参照してください。

MySQL はトランザクションテーブル、非トランザクションテーブルの両方で機能します。非トランザクションテーブル（何らかのエラー発生した場合にロールバックができない）での動作をスムーズにするため、MySQL には次のルールがあります。このルールが適用されるのは正確にSQLモードで作動しているとき、あるいは[IGNORE](#)スペシフィックアヤを[INSERT](#)もしくは[UPDATE](#)に使用しているときのみです。

- すべてのカラムにデフォルト値がある。
- カラムに対して「正しくない」値を挿入した場合や、数値列の数値が大きすぎる場合、MySQL ではエラーを発生するのではなく、「とりうる可能な値のうちの最適値」に値を設定する。数値の場合は0で、可能な最小値が最大値になる。文字列の場合は、空白文字がカラムの最大長さにあわせた文字列になる。
- 計算式はすべて、エラー状態を表示するのではなく、使用可能な値を返す。たとえば、1/0 の場合は、[NULL](#)を返す。

SQLのサーバモードの設定をすることで、さらに制御されたデータの扱いを有効にします。このことにより、以前の動作反応を変更することができます。この詳細については、[MySQL における制約の処理](#)、[SQL モード](#)、そして[INSERT 構文](#)を参照してください。

移植性のためのアプリケーション設計

SQL サーバは SQL のさまざまな部分を実装しているので、移植可能な SQL アプリケーションの作成が可能となります。非常に単純な SELECT や INSERT は容易ですが、必要なことが増えれば増えるほど、作成が難しくなります。多数のデータベースを使用しながら素早い速度が要求されるアプリケーションの場合は、さらに難度が上がります。

どのデータベースシステムにも欠点があります。それはつまり、データベースのすべてに何らかの弱点があります。言い換えると、動作の相違を招くさまざまな設計上の障害があります。

複雑なアプリケーションを移植可能にするには、ともに稼動する必要のある SQL サーバ数を選択する必要があります。MySQL [crash-me](#)プログラムを使用すると、データベースサーバの選択に使用できる関数、データ型、制約を調べることができます。現在の [crash-me](#)は可能なことすべてのテストを実行できるとは決して言えませんが、約 450 項

目のテストが幅広く行われています。たとえば、[crash-me](#)が提供する情報の例として、Infomix や DB2 の使用を可能にするには、18 文字を超えるカラム名は使用できません、といったものがあります。

MySQL ベンチマークと [crash-me](#) プログラムはいずれもデータベースへの依存度が非常に低くなっています。これらのプログラムがどのように処理されているかを調べることによって、データベースに依存しないアプリケーションを作成する際に必要なことに関する感覚を得ることができます。ベンチマーク自体は、MySQL ソースディストリビューションの [sql-bench](#) ディレクトリにあります。これらは Perl で書かれ、DBI データベースインターフェースを使用する。DBI の使用それ自体で移植性の問題の1部分を解決できます。これはデータベースから独立したアクセス方法をとるからです。[MySQL ベンチマークスイート](#)を参照してください。

データベースの独立性の獲得を目指す場合は、SQL サーバそれぞれのボトルネックを正しく理解する必要があります。MySQL では、非常に高速に [MyISAM](#) レコード (テーブルや行) の取り出しと更新が行われますが、1 つのテーブル上に低速のリーダとライタが混在することに問題があります。異なり、Oracle では、更新直後のレコードがディスクに保存される前にそのレコードにアクセスしようとする際に大きな問題があります。一般にトランザクションデータベースの場合、ログテーブルからのサマリテーブルの生成時は行ロックがほとんど役に立たず、問題が生じやすくなっています。

アプリケーションを実際にデータベース非依存にするには、データ操作に使用する簡単な拡張可能インタフェースを定義する必要があります。例えば、C++ ほとんどのシステムでは C++ が使用できるため、データベースに C++ クラスインタフェースを使用することは道理になっています。

あるデータベースに固有の機能を使用する場合 (MySQL の [REPLACE](#) コマンドなど) は、他の SQL サーバでその機能を実装できるようにするメソッドをコード化する必要があります (ただし低速化します)。低速化しますが、他のサーバで同じ作業を実行できるようにします。

MySQL を使用すると、`/*! */` 構文を使用して MySQL 固有のキーワードをクエリに追加できます。`/*! */` 内のコードは、その他の SQL サーバのほとんどでコメントとして処理 (無視) されます。コメントの挿入に関する情報は、[コメント構文](#)を参照してください。

一部の Web アプリケーションのように正確性よりパフォーマンスが重視される場合は、すべての結果をキャッシュするアプリケーションレイヤを作成すると、さらにパフォーマンスを改善できます。一定の期間後に古い結果を '期限切れ' することで、キャッシュを適度に最新の状態で保持できます。これにより、キャッシュを動的に拡大し、通常の状態に戻るまでタイムアウト期限を高速に設定して、高負荷のスパイクを処理するメソッドが提供されます。

この場合、テーブル作成情報にキャッシュの初期サイズと通常時にテーブルがリフレッシュされる頻度に関する情報が組み込まれます。

アプリケーションを実行する代わりにMySQLクエリのキャッシュを使用するのは、効果的です。クエリキャッシュを有効にすることで、クエリ結果が再利用できるかどうかの判断の詳細をサーバに一任します。これによりユーザのアプリケーションは単純化します。[MySQL クエリ キャッシュ](#)を参照してください。

MySQL 使用実績

このセクションではMySQL開発当初のアプリケーションを説明します。

MySQL の初期開発当時は、最大顧客に合わせて MySQL の機能が開発されてきました。この機能は、スウェーデンの最大小売商数社向けにデータウェアハウスを処理するものです。

すべての店舗からボーナスカード取引すべてのサマリを毎週取得し、店舗の所有者が顧客に対する広告キャンペーンの効果を調べる際に役立つ情報を提供するように求められています。

このデータは非常に大量 (1 か月に約 700 万のサマリ取引) で、ユーザへの提示に必要な 4-10 年間のデータを保有しています。このデータから新しいレポートに '即時' アクセスできるようにしたいという顧客からの要求が毎週ありました。

1 か月ごとにすべての情報を圧縮「トランザクション'テーブル」に格納することでこの要求を解決しました。トランザクションテーブルからさまざまな基準 (製品グループ、顧客 ID、店舗など) によって分類されたサマリテーブルを生成する単純なマクロ (スクリプト) セットを開発しています。レポートは Web ページ形式で、Web ページを解析し、SQL ステートメントを実行して、結果を挿入する、短い Perl スクリプトから動的に生成されます。[mod_perl](#) の使用のほうに適しているとも言えますが、その当時は利用できませんでした。

グラフィカルデータについては、SQL クエリの結果 (この結果に処理を加えて) から GIF を生成する簡単なツールを C で作成しました。これも HTML ファイルを解析する Perl スクリプトから動的に実行されます。

ほとんどの場合、既存のスクリプトをコピーし、その SQL クエリを修正することで新規のレポートを簡単に実行することができます。状況によっては、既存のサマリテーブルにフィールドを追加したり、新規のテーブルを生成することが必要な場合もありますが、これもディスク上にすべてのトランザクションテーブルを保存しているため非常に容易なことです。（現在、少なくとも 50 G のトランザクションテーブルとその他の 200 G の顧客データを保持しています）。

顧客は、ODBC によってサマリテーブルに直接アクセスすることができ、上級ユーザであれば各自でデータを処理することができます。

非常に適度な規模の Sun Ultra SPARCstation (2x200 Mhz) を使用した処理では何も問題が発生していません。徐々にシステムは Linux に移植されていきました。

MySQL ベンチマークスイート

このベンチマークスイートは、SQL 実装のパフォーマンスを向上または低下させる操作をユーザに示すことを目的としています。MySQL ソースディストリビューションの [sql-bench](#) ディレクトリにあるコードと結果を確認することでベンチマークに関するヒントが得られます。

このベンチマークはシングルスレッドであるため、実行される操作の最短時間が測定されていることに注意してください。将来はこのベンチマークスイートにマルチスレッドのテストも多数追加する予定です。

ベンチマークスイートを使用するには、以下の要件を満たす必要があります。

- ベンチマークスイートは、MySQL ソースディストリビューションによって提供されます。<http://dev.mysql.com/downloads/> のサイトからリリースされているディストリビューションをダウンロードするか、現在の開発ツリーを使用できる。（詳しくは [開発ソース ツリーからのインストール](#) をご確認ください。）
- ベンチマークスクリプトは Perl で作成され、データベースサーバへのアクセスには Perl DBI モジュールを使用しているため、DBI のインストールが必要である。テスト対象のサーバのそれぞれにサーバ専用の DBD ドライバも必要である。たとえば、MySQL、PostgreSQL、DB2 をテストするには、[DBD::mysql](#)、[DBD::Pg](#)、[DBD::DB2](#) のモジュールをインストールする必要がある。[Perl のインストールに関する注釈](#) を参照してください。

MySQL ソースディストリビューション入手後、ベンチマークスイートは、MySQL ソースディストリビューションの [sql-bench](#) ディレクトリにあります。ベンチマークテストを実行するには、ロケーションをそのディレクトリ [sql-bench](#) に変更し、[run-all-tests](#) スクリプトを実行します。

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

[server_name](#) はサポートされるサーバの 1 つを表します。すべてのオプションとサポート対象サーバの一覧を取得するには、このコマンドを呼び出してください。

```
shell> perl run-all-tests --help
```

[sql-bench](#) ディレクトリ内に [crash-me](#) があります。[crash-me](#) では、データベースがサポートする機能と、実際のクエリを実行した場合の機能と制約の判定が試行されます。たとえば、以下についての判定が行われます。

- サポートされるデータ型
- サポートされるインデックス数
- サポートされる関数
- 使用可能なクエリのサイズ
- 使用可能な [VARCHAR](#) カラムのサイズ

独自のベンチマークの使用

確実にアプリケーションとデータベースのベンチマークを行い、ボトルネックを検出しておく必要があります。これを修正（または、ボトルネックを「dummy」に置換）することによって、次のボトルネック（など）の確認が容易になります。現在のアプリケーションの総合的なパフォーマンスが許容できるものであっても、実際にパフォーマンスの強化が迫られる場合に備えて、少なくともボトルネックそれぞれに対して計画を立て解決方法を判定しておく必要があります。

移植可能なベンチマークプログラムの例として、MySQL ベンチマークスイートを取り上げます。詳しくは[MySQL ベンチマークスイート](#)を参照してください。このスイートから任意のプログラムを選び、必要に合わせて修正することができます。これによって、それぞれの問題に対して複数の解決方法を試行して、実際に最も高速が得られるのはどれであるかについてテストすることができます。

これ以外の無料のベンチマークスイートに Open Source Database Benchmark があり、これは <http://osdb.sourceforge.net/>で入手できます。

一般的には、システムの負荷が非常に高い状況にのみ問題が発生します。負荷の問題が（テスト済の）本稼働のシステムで発生したと問い合わせてくる顧客が多数いました。ほとんどの場合、パフォーマンスに関わる問題は基本的な設計上の問題（高負荷時のテーブルスキャンの不良）かオペレーティングシステムやライブラリの問題が原因だと判明しています。たいていは、システムがまだ本稼働に入っていない場合のほうが問題の修正がはるかに容易です。

このような問題を回避するには、想定可能な最悪の負荷でアプリケーション全体のベンチマークにある程度力を注ぐ必要があります。

- 複数のクライアントが同時にクエリを発行したときに生じる高負荷状態をシミュレートするには、[mysqslap](#)プログラムが効果的です。[mysqslap — クライアント負荷エミュレーション](#)を参照してください。

- Super Smack も試してください。これは、<http://jeremy.zawodny.com/mysql/super-smack/>で入手できます。

その名（Smack = 打ちこわし）のとおり、システムに限界まで負荷をかけることができるため、必ず開発システムでのみ使用するようにしてください。

SELECTステートメントおよびその他のクエリの最適化

第 1 にすべてのクエリに影響を及ぼすことが 1 つあります。アクセス権システムのセットアップの複雑性が増すほど、オーバーヘッドも増加します。[GRANT](#)ステートメントを発行する際に単純なアクセス権を使用することで、クライアントがステートメントを事項する際のMySQLにアクセス権確認オーバーヘッドを減らすことができます。例えば、テーブルレベルやカラムレベルの特権を許可したくない場合、サーバは[tables_priv](#)と[columns_priv](#)テーブルの内容を確認する必要はまったくありません。同じように、どのアカウントにもリソース制限を設けない場合、サーバは性能リソースカカウンティングを行う必要がありません。大量の処理が必要なときは、[GRANT](#)を使用しないことで時間を節約できる場合もあります。

明示的なMySQL関数に関わる問題がある場合は、常に[mysql](#)クライアントで[BENCHMARK\(\)](#)関数の計時を行うことができます。その構文は[BENCHMARK\(loop_count,表現\)](#)。返される値は常に0ですが、[mysql](#)はステートメントの実行にどの程度の時間を要したかを表示するラインをプリントします。例：

```
mysql> SELECT BENCHMARK(1000000,1+1);
+-----+
| BENCHMARK(1000000,1+1) |
+-----+
|          0 |
+-----+
1 row in set (0.32 sec)
```

これは、PentiumII 400MHz 上でMySQLによって1,000,000の+式を0.32秒間に実行できることを示しています。

MySQL関数はすべて最適化されていますが、例外も若干あります。[BENCHMARK\(\)](#)はクエリに関数上の問題があるかどうかを調べる際に最適のツールです。

EXPLAINを使用して、クエリを最適化する

```
EXPLAIN tbl_name
```

または

```
EXPLAIN [EXTENDED | PARTITIONS] SELECT select_options
```

[EXPLAIN](#)ステートメントは[DESCRIBE](#)のシノニムとして使用するが、MySQLがどのように[SELECT](#)ステートメントを実行するかの情報が得られます。

- [EXPLAIN tbl_name](#)は[DESCRIBE tbl_name](#) または[SHOW COLUMNS FROM tbl_name](#)のシノニムです。

- キーワード **EXPLAIN** を **SELECT** ステートメントの前に置いた場合、MySQL によってテーブルの結合状況と順序に関する情報が提供され、テーブルの **SELECT** の処理方法が説明されます。
- **EXPLAIN PARTITIONS** は MySQL 5.1.5 から提供されています。区割りされたテーブルのクエリを調べるときに便利です。詳細については、[パーティション情報の取得](#) をご参照ください。

このセクションでは、クエリ実行情報を得るための **EXPLAIN** の2つめの使用方法を記述します。**DESCRIBE** と **SHOW COLUMNS** ステートメントの詳細については、[DESCRIBE 構文](#) と [SHOW COLUMNS 構文](#) を参照してください。

EXPLAIN を利用すると、より速くレコードを検索する **SELECT** を得るために、どの時テーブルにインデックスを追加しなければならないかを確認できます。また、**EXPLAIN** を使用して、オプティマイザがテーブルを最適な順序で結合しているのかも確認することができます。オプティマイザが特定の順番で結合を行うように強制するにはただ **SELECT** でステートメントをはじめるのではなく、**SELECT** ステートメントに **SELECT STRAIGHT_JOIN** 節を追加します。

最適化方法の選択に影響を及ぼすキーの、カーディナリティなどのテーブル統計を更新するために、**ANALYZE TABLE** を定期的に行う必要があります。[ANALYZE TABLE 構文](#) を参照してください。

EXPLAIN は **SELECT** ステートメントで使用する各テーブルに関する情報を返します。テーブルは、読み取られた順序に従って一覧表示されます。MySQL は、単一スweep多結合メソッドを使用してすべての結合を解決します。これは、MySQL が最初のテーブルからレコードを読み取ってから、第2のテーブル、第3のテーブルといった順序で、一致するレコードの検索を行うことを意味します。すべてのテーブルの処理が終わると、選択したカラムと、さらに一致レコードがあるテーブルが検索されるまでのテーブル一覧のバックトラックが出力されます。次のレコードはこのテーブルから読み取られ、処理が次のテーブルから続行されます。

EXTENDED キーワードが使用された時、**EXPLAIN** は **SHOW WARNINGS** ステートメントを **EXPLAIN** ステートメントの後で発行することで閲覧できる余分な情報を表示する。この情報は、**SELECT** ステートメント内でオプティマイザがどのようにテーブル名とカラム名を認証するか、**SELECT** が再書き込みと最適化ルールの適用後どのように表示されるか、そして最適化プロセスの他の注意点なども表示します。**EXPLAIN EXTENDED** は MySQL 5.1.12 以降、**filtered** カラムも表示します。

注:**EXTENDED** と **PARTITIONS** キーワードを、同じ **EXPLAIN** ステートメントで使用することはできません。

EXPLAIN の各出力行は1つのテーブルの情報を提供し、各行は以下のカラムを含んでいます。

- **id**

SELECT 識別子。クエリ内におけるこの **SELECT** の順序番号。

- **select_type**

SELECT 節の種類、次のいずれかが示される。

SIMPLE	単純な SELECT (UNION やサブクエリを使用しない)。
PRIMARY	最外部の SELECT 。
UNION	内の第2およびそれ以降の SELECT ステートメント。
DEPENDENT UNION	UNION 内の第2およびそれ以降の SELECT ステートメント内の UNION 、外側のサブクエリに依存する。
UNION RESULT	UNION の結果。
SUBQUERY	サブクエリ内の第1 SELECT 。
DEPENDENT SUBQUERY	第1 SELECT 、外側のサブクエリに依存する。
DERIVED	派生テーブル SELECT (FROM 節内のサブクエリ)
UNCACHEABLE SUBQUERY	結果がキャッシュされず、外側のクエリの各行ごとに再評価されるサブクエリ。

DEPENDENT は主に、相互に関係するサブクエリの使用を表します。[相関サブクエリ](#) を参照してください。

「依存型サブクエリ」の評価は **UNCACHEABLE SUBQUERY** 評価とは異なります。「**DEPENDENT SUBQUERY**」に関しては、外側コンテキストの変数の値が異なるたびに、一回のみサブクエリの再評価が行われます。**UNCACHEABLE SUBQUERY** に関しては、サブクエリは外側コンテキストの各行ごとに再評価されます。サブ

クエリのキャッシュアビリティは[クエリ キャッシュの動作](#)で記述される制限によります。例えば、ユーザ変数に参照することでサブクエリがキャッシュできなくなります。

- **テーブル**

結果を得るために参照するテーブル。

- **type**

結合型。各結合型を最適なものから順に紹介する。

- **システム**

1レコードのみで構成されるテーブル (= システムテーブル)。これは、[const](#)結合型の特殊なケースである。

- **const**

テーブルに、一致するレコードが最大で1つあり、クエリの開始時に読み取られる。レコードが1つしかないため、このレコードのカラムの値はオプティマイザによって定数と見なされる。[const](#)テーブルは、1回しか読み取られないため、非常に高速である。

[const](#)はPRIMARY KEY/UNIQUEキーを定数と比較する場合に使用される。下記のクエリでは、tbl_nameは[const](#)テーブルとして使用できる。

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- **eq_ref**

前のテーブルのレコードの組み合わせのそれぞれに対して、このテーブルから1レコードずつ読み取られる。これは、[system](#)と[const](#)型以外で最適な結合型である。結合でインデックスのすべての部分が使用され、このインデックスが [UNIQUE](#)または [PRIMARY KEY](#)である場合に使用される。

=演算子と比較されるインデックスの張られたカラムには、[eq_ref](#)を使用できる。較対象のアイテムは定数でも、このテーブル以前に読み取られたテーブルのカラムを使用する式でもかまわない。下記の例では、[ref_table](#)で[eq_ref](#)が使用される。

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- **ref**

前のテーブルのレコードの組み合わせのそれぞれに対して、インデックス値にマッチするすべてのレコードがこのテーブルから読み取られる。[ref](#)は、インデックスの左端の先頭部分のみが結合で使用される場合、またはインデックスが [UNIQUE](#)や [PRIMARY KEY](#)ではない場合 (すなわち、この結合において、インデックス値から1つのレコードをSELECTできない場合) に使用される。この結合型は、使用されるインデックスと一致するレコードが数レコードしかない場合に適している。

=あるいは<=>演算子と比較されるインデックスの張られたカラムには、[ref](#)を使用できる。下記の例では、MySQLは[ref_table](#)で[ref](#)が使用される。

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- **ref_or_null**

refと同様だが、NULLを使用したレコードの補足検索も追加で実行される。この結合型の最適化は主としてサブクエリを解決する場合に使用される。下記の例では、MySQLはref_tableでref_or_nullが使用される。

```
SELECT * FROM ref_table
WHERE key_column=expr OR key_column IS NULL;
```

IS NULL最適化参照。

- [index_merge](#)

この結合型はインデックス併合最適化が使用されたことを示しています。この場合、出力行のkeyカラムは使用されたインデックスのリストが含まれ、key_lenには使用されたインデックスの最長キー部分が含まれます。詳細は[インデックス結合最適化](#)をご覧ください。

- [unique_subquery](#)

この型は、下記のフォームでINサブクエリの代わりに、refを使用します。

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

unique_subqueryは、効率化のためサブクエリの代わりにつとめるインデックスルックアップ関数です。

- [index_subquery](#)

この結合型はunique_subqueryに似ています。INサブクエリの代わりに使用されますが、下記のサブクエリのフォームでユニークではないインデックスで使用できます。

```
value IN (SELECT key_column FROM single_table WHERE some_expr)
```

- [range](#)

インデックスを使用して、一定の範囲にあるレコードのみが取り出される。keyカラムに使用されるインデックスが表示される。key_lenには使用される最長のインデックス部分が記載される。この型ではrefカラムがNULLになる。

range は、インデックスを張っているカラムが =、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN、およびINを使用して定数と比較される場合に使用される。

```
SELECT * FROM tbl_name
WHERE key_column = 10;

SELECT * FROM tbl_name
WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
WHERE key_part1= 10 AND key_part2 IN (10,20,30);
```

- [index](#)

これは、インデックスツリーのみがスキャンされる点を除いてALLと同じである。一般にインデックスファイルはデータファイルより小さいため、通常はALLより高速である。

MySQL は、クエリで1インデックスの構成部分であるカラムのみが使用される場合にのみ使用できます。

- [ALL](#)

前のテーブルのレコードの組み合わせのそれぞれに対して、フルテーブルスキャンが実行される。一般に、テーブルがconstの指定がない第1テーブルの場合には適さず、その他の場合はすべて非常に不適である。通常は、さらにインデックスを追加することでALLを回避し、定数値または以前のテーブルのカラム値を基準にレコードを取り出すようにすることができる。

- [possible_keys](#)

`possible_keys`カラムは、このテーブル内のレコードの検索に MySQL で使用可能なインデックスを示す。このカラムはEXPLAINからの出力により表示されたテーブルの順序にはまったく依存しないことに注意する。すなわち、`possible_keys`のキーの一部は、生成されたテーブルの順序では事実上使用できないことになる。

このカラムが `NULL` の場合は、対応するインデックスがない。この場合は、`WHERE` 節でインデックス作成に適するカラムを 1 つ以上参照しているかどうかを調べることでクエリのパフォーマンスを改善できる。参照している場合は適切なインデックスを作成し、再度 EXPLAIN を使用してクエリをチェックする。ALTER TABLE 構文を参照してください。

テーブルにあるインデックスを調べるには `SHOW INDEX FROM tbl_name` を使用する。

- `key`

`key`カラムは、MySQL が実際に使用を決定したキー（インデックス）を示す。MySQL が行をルックアップするため `possible_keys` インデックスを使用した場合、キー値としてそのインデックスがリストされる。

`key` は `possible_keys` 値に存在しないインデックスを指名する可能性もあります。これは `possible_keys` インデックスのうちどれも行をルックアップするのに適していない場合におこりますが、クエリに選択された全てのカラムは他のインデックスのカラムになります。つまり、指名されたインデックスが選択されたカラムをカバーします。どの行を取得するか判別するのに使用されていなくとも、データ行スキャンよりもインデックススキャンの方が効率的です。

InnoDBでは、クエリがプライマリーキーを選択していてもセカンダリインデックスが選択されたカラムをカバーするかもしれません。これはクエリがプライマリーキーを選択した場合もありえるのは、InnoDBが各セカンダリインデックスと共にプライマリーキー値も保存するからです。MySQLがクエリを効率的に実行するインデックスを見つけられなかった場合、この`key`は `NULL` になる。

MySQLで `possible_keys`カラムに記載されたキーが使用されるように強制するには、クエリで `FORCE INDEX`、`USE INDEX`、または `IGNORE INDEX` を使用する。SELECT 構文を参照してください。

MyISAMテーブルには、`ANALYZE TABLE` を実行することでオプティマイザでより適したインデックスを選択する際役立つ。MyISAMテーブルに関しても、`myisamchk --analyze` は同じことをします。ANALYZE TABLE 構文、テーブル保守とクラッシュリカバリ を参照して下さい。

- `key_len`

`key_len`カラムは、MySQL が実際に使用を決定したキーの長さを示す。`key`が `NULL` の場合、この長さは `NULL` になる。`key_len`の値によって、複合キーで MySQL が実際に使用するパート数が示されることに注意する。

- `ref`

`ref`カラムは、テーブルからレコードを選択する際に `key`とともに使用されるカラムまたは定数を示す。

- `rows`

`rows`カラムは、クエリの実行に際して調べる必要があると MySQL によって判定されたレコードの数を示す。

- `filtered`

`filtered`カラムはテーブルの状態によってフィルターされるテーブル行のパーセンテージ（予想）を表示します。つまり、`rows`は検査された行の予想数を表示し、 $\text{rows} \times \text{filtered} / 100$ は前のテーブルと結合する行の数を表示します。EXPLAIN EXTENDEDを使用すると、このカラムが表示されます。(MySQL 5.1.12の新しい機能です。)

- `Extra:`

このカラムには、MySQL でどのようにクエリが解決されるかに関する追加情報が記載される。下記のリストはこのカラムで表示される可能性のある値を説明する。クエリを速度をできる限り上げたい場合は、`Using filesort` と `Using temporary` の `Extra` 値に注目してください。

- `Distinct`

マッチした最初のレコードが検索されると、MySQL は現在のレコードの組み合わせによるその後のレコード検索を続行しないことを示す。

- [const tablesを読んだ後 Impossible WHERE発見](#)

MySQL は全てのconst (あと、system) テーブルを読んだ後、WHERE節が常に偽となります。

- [No tables](#)

クエリにはFROM節がないか、FROM DUAL節があります。

- [Not exists](#)

MySQL でクエリに対する LEFT JOIN最適化が実行でき、LEFT JOINに一致するレコードが 1 つ検索されると、前のレコードの組み合わせによるその後のテーブルのレコードについては調べないことを示す。このように最適化できるクエリの例を以下に示します。

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

t2.idが NOT NULL で定義されているとする。この場合、MySQL で t1 がスキャンされ、t1.id で t2 内のレコードのルックアップが行われる。MySQL によって t2 内のマッチするレコードが検索されると、t2.id は NULL ではないと認識され、t2 内の同じ id を持つ残りのレコードのスキャンは行われない。言い換えると、t2 にあるマッチするレコードの数に関わらず、MySQL で実行が必要なことは t1 のレコードのそれぞれに対して、t2 のルックアップを 1 回実行することだけである。

- [range checked for each record \(index map: N\)](#)

MySQL で使用に適した実際のインデックスを検索できなかったことを示す。代替として、先行テーブルのレコードの組み合わせのそれぞれに対して、使用するインデックス (存在する場合) range または index_merge のチェックが実行され、このインデックスがテーブルからのレコードの取り出しに使用される。非常に高速ではないが、インデックスなしの結合と比較すると高速である。適用基準は Range 最適化とインデックス結合最適化で説明されています。ただし、これは前テーブルの全てのカラム値が知られており、定数であるという前提においてです。

- [Select tables optimized away](#)

クエリは MyISAM 用に、インデックスで解決された集約ファンクション (MIN(), MAX()) そして COUNT(*) があり、GROUP BY 節は含みませんでした。オプティマイザは 1 つの行のみが返されるべきと判断しました。

- [Using filesort](#)

レコードをソートして取り出す方法を決定するには、MySQL はパスを余分に実行しなくてはならないことを示す。join type に従ってすべてのレコードをスキャンし、WHERE 条件に一致する全てのレコードに、ソートキー + 行ポインタを格納して、ソートは実行される。その後キーがソートされる。最後に、ソートされた順にレコードが取り出される。ORDER BY 最適化を参照してください。

- [Using index](#)

インデックスツリーの情報のみを使用してカラム情報がテーブルから取り出され、実際の行を読み取るその後の検索を実行する必要がないことを示す。MySQL は、クエリで 1 インデックスの構成部分であるカラムのみが使用される場合にのみ使用できます。

- [Using temporary](#)

クエリの解決に MySQL で結果を保持するテンポラリテーブルの作成が必要であることを示す。これは一般に、GROUP BY を実行したカラムセットと異なるカラムセットに対して ORDER BY を実行した場合に発生する。

- [Using where](#)

次のテーブルとの一致が調べられるレコードまたはクライアントに送信されるレコードの限定に WHERE 節が使用されることを示す。この情報がなく、Extra の値が Using where ではなく、テーブルの型が ALL または index である場合はクエリが正常に実行されないことがある (テーブルのすべてのレコードの取得や検査を意図していない場合) 。

- [Using sort_union\(...\)](#)、[Using union\(...\)](#)、[Using intersect\(...\)](#)

これらは[index_merge](#)結合型でインデックススキャンがどのように併合されるかを示しています。詳細は [インデックス結合最適化](#) を参照。

- [Using index for group-by](#)

[Using index](#)を使用してテーブルをアクセスする方法に似て、[Using index for group-by](#)はMySQLが余分なディスクアクセスを実際のテーブルに行うことなく、[GROUP BY](#)または[DISTINCT](#)クエリのカラムを全て取得することができるインデックスを見つけたことを意味します。加えて、インデックスは各グループにとって最も効率的に使われるので、数種類のインデックスしか読まれません。詳細については、[GROUP BY最適化](#)をご参照ください。

- [Using where with pushed condition](#)

このアイテムは[NDB Cluster](#)テーブルにのみ適用されます。それはMySQL クラスタがcondition pushdownを使用して行う、インデックスのないカラムと定数を直接比較(=)の効率化を図ることを意味します。その場合、状態はクラスタのデータノードに「押し戻され」ており、全てのパーティションで同時に評価されます。これはマッチしない行をネットワーク上で送る必要を無くし、コンディションプッシュダウンが使える状態にあり、使用しないケースでそのようなクエリを5乗から10乗に増やします。

以下のように定義されたクラスタテーブルがあるとします。

```
CREATE TABLE t1 (
  a INT,
  b INT,
  KEY(a)
) ENGINE=NDBCLUSTER;
```

この場合、コンディションプッシュダウンは下記のようなクエリで使用できます。

```
SELECT a,b FROM t1 WHERE b = 10;
```

これは[EXPLAIN SELECT](#)の出力で見られます。例えば

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
***** 1. row *****
   id: 1
 select_type: SIMPLE
   table: t1
   type: ALL
possible_keys: NULL
   key: NULL
  key_len: NULL
   ref: NULL
  rows: 10
 Extra: Using where with pushed condition
```

コンディションプッシュダウンは下記の2つのクエリと一緒に使用できません。

```
SELECT a,b FROM t1 WHERE a = 10;
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

この二つのクエリのうち最初のものに関しては、インデックスがaカラムに存在するため、コンディションプッシュダウンは適用できません。2番目のクエリの場合、インデックスのないカラムbに関する比較は直接的でないため、コンディションプッシュダウンが適用できません。(ただし、[b + 1 = 10](#)をWHERE 節内で**b = 9**に減らす場合は適用されます。)

ただし、>または<演算子を使用している定数とインデックスカラムが比較された場合、コンディションプッシュダウンが使用される場合もあります。

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE a < 2\G
***** 1. row *****
   id: 1
 select_type: SIMPLE
   table: t1
   type: range
possible_keys: a
```

```

key: a
key_len: 5
ref: NULL
rows: 2
Extra: Using where with pushed condition

```

コンディションプッシュダウンに関して、以下のことに留意してください。

- コンディションプッシュダウンはMySQL クラスタにのみ関連しており、他の保存エンジンを使用するテーブルに対してクエリを実行するときは起こりえません。
- コンディションプッシュダウン機能はデフォルトでは使用されません。起動するには、`mysqld` を`--engine-condition-pushdown` オプションで使用するか、以下のステートメントを実行してください。

```
SET engine_condition_pushdown=On;
```

注:コンディションプッシュダウンはBLOB やTEXTタイプのどのカラムに対してもサポートされていません。

EXPLAIN出力の `rows` カラムのすべての値を掛け算することで、結合がどの程度適しているかを示す指針を取得できます。Thisこれは、クエリの実行時にMySQL で調べる必要があるレコード数の概要を示します。この数値は、`max_join_size`変数でクエリを制限する際にも使用される他、どのマルチテーブルSELECTステートメントを実行するか、あるいはアボートするかを判別します。[サーバパラメータのチューニング](#)を参照してください。

下記の例は、EXPLAINによって得られた情報を使用して、マルチテーブルjoinを累進的に最適化する方法を示しています。

ここでは、EXPLAINを使用して、SELECTステートメントを調べるとします。

```

EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
              tt.ProjectReference, tt.EstimatedShipDate,
              tt.ActualShipDate, tt.ClientID,
              tt.ServiceCodes, tt.RepetitiveID,
              tt.CurrentProcess, tt.CurrentDPPerson,
              tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
              et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
AND tt.ActualPC = et.EMPLOYID
AND tt.AssignedPC = et_1.EMPLOYID
AND tt.ClientID = do.CUSTNMBR;

```

この例では以下のように想定しています。

- 比較対象のカラムは以下のように宣言されます。

テーブル	カラム	データ型
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- テーブルには以下のインデックスがあります。

テーブル	インデックス
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (プライマリキー)
do	CUSTNMBR (プライマリキー)

- `tt.ActualPC`値の分布が均一ではない。

当初、最適化の実行前は、EXPLAINステートメントで次の情報が生成されました。

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
do ALL PRIMARY NULL NULL NULL 2135
et_1 ALL PRIMARY NULL NULL NULL 74
tt ALL AssignedPC, NULL NULL NULL 3872
    ClientID,
    ActualPC
range checked for each record (key map: 35)
```

各テーブルで **type** が **ALL** であるため、この出力は MySQL がすべてのテーブルのデカルト積を生成すると示しています。各テーブルのレコードの数の積の分量を調べる必要があるため、これは非常に時間がかかります。この例の場合は、レコードの数が $74 \times 2135 \times 74 \times 3872 = 45,268,558,720$ になります。テーブルがこれより大きい場合は、さらに時間がかかると考えられます。

ここでの問題の 1 つは、宣言の方法が異なると MySQL でカラムのインデックスを効率的に使用できないことにあります。この例では、**VARCHAR** と **CHAR** が異なる長さで宣言されていなければ同じになります。**tt.ActualPC** が **CHAR(10)** として、**et.EMPLOYID** が **CHAR(15)** として宣言されているため、長さの不一致が発生します。

カラムの長さの不一致を修正するため、**ALTER TABLE** を使用して **ActualPC** を 10 文字から 15 文字にします。

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

これで **tt.ActualPC** と **et.EMPLOYID** はいずれも **VARCHAR(15)** になりました。ここでまた **EXPLAIN** を実行してみると、以下の結果が得られました。

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC, NULL NULL NULL 3872 Using
    ClientID, where
    ActualPC
do ALL PRIMARY NULL NULL NULL 2135
range checked for each record (key map: 1)
et_1 ALL PRIMARY NULL NULL NULL 74
range checked for each record (key map: 1)
et eq_ref PRIMARY PRIMARY 15 tt.ActualPC 1
```

これも完全ではありませんが、かなり改善されています (**rows** 値の積が 74 の係数分だけ減少)。このバージョンの場合実行に数秒かかります。

第 2 の変更を加えると、**tt.AssignedPC = et_1.EMPLOYID** と **tt.ClientID = do.CUSTNMBR** の比較でのカラム長の不一致を解消できます。

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
-> MODIFY ClientID VARCHAR(15);
```

ここでは、**EXPLAIN** から以下の出力が生成されます。

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
tt ref AssignedPC, ActualPC 15 et.EMPLOYID 52 Using
    ClientID, where
    ActualPC
et_1 eq_ref PRIMARY PRIMARY 15 tt.AssignedPC 1
do eq_ref PRIMARY PRIMARY 15 tt.ClientID 1
```

これでほとんど改善されています。残りの問題は、MySQL ではデフォルトで **tt.ActualPC** カラムの値の分布が均一であると想定されますが、**tt** テーブルはこれにあてはまらないことです。これは容易に MySQL に示すことができます。

```
mysql> ANALYZE TABLE tt;
```

この追加インデックス情報で、結合が完全になり、**EXPLAIN** で以下の結果が生成されます。

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC NULL NULL NULL 3872 Using
    ClientID, where
    ActualPC
et eq_ref PRIMARY PRIMARY 15 tt.ActualPC 1
et_1 eq_ref PRIMARY PRIMARY 15 tt.AssignedPC 1
do eq_ref PRIMARY PRIMARY 15 tt.ClientID 1
```

EXPLAINの出力の **rows** カラムは、MySQL 結合オプティマイザの学習による推測であることに注意してください。クエリを最適化するには、この数値が実際に近いものであるかどうかを確認するために **rows** のプロダクトとクエリが実際に返す行の数を比較する必要があります。実際とかけ離れている場合は、**SELECT** ステートメントで **STRAIGHT_JOIN** を使用し、**FROM** 節でテーブルの順序を変えて一覧表示してみるとパフォーマンスを改善できます。

MySQL Enterprise

MySQL ##### <http://www-jp.mysql.com/products/enterprise/advisors.html#####>

クエリパフォーマンスの推定

ほとんどの場合、ディスクシークをカウントしてパフォーマンスを推定できます。小さいテーブルの場合は一般に1つのディスクシークでレコードを検索できます(インデックスがキャッシュされることが多いため)。大きいテーブルの場合の推定では、(B++ ツリーインデックスを使用して) $\log(\text{row\_count}) / \log(\text{index\_block\_length} / 3 \times 2 / (\text{index\_length} + \text{data\_pointer\_length})) + 1$ のシークがレコードの検索に必要になります。

MySQL では、インデックスブロックが通常 1,024 バイトで、データポインタは通常 4 バイトです。インデックスの長さが 3 (**MEDIUMINT**) の 500,000 レコードのテーブルの場合は以下ようになります。 $\log(500,000) / \log(1024/3 \times 2 / (3+4)) + 1 = 4$ シーク)

上のインデックスでは約 $500,000 \times 7 \times 3/2 = 5.2\text{M}$ が必要になるため(一般的な状況としてインデックスバッファの 2/3 が使用されていると想定)、メモリにインデックスの多くがあり、OS からデータを読み取り、レコードを検索するには、1 回か 2 回の呼び出しで済むと推定されます。

ただし、書き込みについては、上記の例で新規インデックスの配置場所を探し出すのに 4 シークの要求が、また、インデックスの更新とレコードの書き込みに通常 2 シークが必要になります。

Note that このことは、アプリケーションが対数 **N** の分だけ低速になるという意味ではないことに注意してください。OS または SQL サーバですべてがキャッシュされている限り、テーブルが拡大しても速度の低下はわずかです。データがキャッシュできないほど増加すると、ディスクシーク(対数 **NN** の分だけ増加する)によって最終的にアプリケーションがバインドされるまで大幅に速度の低下が始まります。)これを回避するには、データの増加に合わせてインデックスキャッシュも拡大します。MyISAM テーブルに関しては、キーキャッシュサイズは **key_buffer_size** システム変数に制御されます。[サーバパラメータのチューニング](#) を参照してください。

SELECT クエリの速度

一般に、低速の **SELECT ... WHERE** の速度を上げる必要がある場合は、まず、インデックスを追加できるかどうかをチェックします。一般に複数のテーブル間の参照はすべてインデックスを使用して実行する必要があります。EXPLAIN コマンドを使用して、**SELECT** に使用されるインデックスを判定できます。EXPLAIN を使用して、クエリを最適化すると [MySQL におけるインデックスの使用](#) を参照してください。

MyISAM テーブルのクエリ速度を上げる一般的なヒント。

- To MySQL によるクエリの最適化を容易にするには、関連データをロードした後にテーブルに対して **ANALYZE TABLE** あるいは **myisamchk --analyze** を実行する。これはインデックスのために、同じ値があるレコードの平均値を更新する(ユニークインデックスの場合、これは常に 1 になる)。MySQL はこれを使用して、2 つのテーブルを '非定数式' で接続する際に選択するインデックスを判定する。**SHOW INDEX FROM tbl_name** を実行し **Cardinality** 値を調べると **myisamchk --description --verbose** はインデックスの分布情報を表示する。
- インデックスに従ってインデックスとデータをソートするには **myisamchk --sort-index --sort-records=1** (インデックス 1 でソートする場合) を使用する。速度を上げるには、すべてのレコードの読み取りにユニークインデックスを使用し、そのインデックスに従った順序で読み取りを行うように推奨される。ただし、このソートでは書き込みの最適化はできず、テーブルが大きい場合は時間がかかる。

WHERE 節最適化

このセクションでは **WHERE** 節をプロセスする際の最適化方法を記述します。WHERE の最適化は、ほとんどの場合 **SELECT** とともに使用されるため、**SELECT** 部分に適用されますが **DELETE** や **UPDATE** のステートメントの **WHERE** にも同じ最適化が適用されます。

また、このセクションは完全なものではないため、注意が必要です。MySQL は多様な最適化を実行するため、すべてを文書化するには時間が足りませんでした。

MySQL によって実行される最適化の一部をここに紹介します

- 不要なかつこの削除。

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- 定数の折りたたみ。

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- 定数条件の削除 (定数の折りたたみに必要) 。

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- インデックスが使用する定数式が評価されるのは、1 回に限られる。
- **COUNT(*) WHERE**がない単一テーブルの **COUNT(*)** は、**MyISAM**と **HEAP** テーブルのテーブル情報から直接取り出される。これは、テーブル 1 つのみで使用する場合はすべての **NOT NULL**式でも実行される。
- 無効定数式の早期検出。MySQL は実行不可能な **SELECT**ステートメントがある場合、それを迅速に検出し、結果としてレコードを返さない。
- **GROUP BY**またはグループ関数 (**COUNT()**、**MIN()**) を使用しない場合、**HAVING**は **WHERE**とマージされる。
- サブ結合のそれぞれに、単純な **WHERE**が構造化され、サブ結合ごとに迅速に **WHERE**評価を取得し、可能な限り迅速にレコードをスキップする。
- クエリ内の他のすべてのテーブルの前に、まず、すべての定数テーブルが読み込まれる。定数テーブルとは以下のものを指す。
 - 空白テーブルまたは 1 行のみのテーブル。
 - **UNIQUE**インデックスまたは **PRIMARY KEY**を使う **WHERE**節とともに使用されるテーブルで、インデックス部分のすべてが定数式とともに使用され、そのインデックス部分が **NOT NULL**として定義されている場合。

以下のテーブルはすべて定数テーブルとして使用される。

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- テーブルを結合する最適な結合の組み合わせは、すべての可能性を試行することで発見される。**ORDER BY**および **GROUP BY**内の全てのカラムが 1 つのテーブルに存在する場合、結合を行う時は第一にこのテーブルが選ばれる。
- **ORDER BY**節とそれと異なる **GROUP BY**節がある場合、あるいは、**ORDER BY**または **GROUP BY**に結合キューの第 1 テーブルとは異なるテーブルのカラムが含まれている場合は、テンポラリテーブルが作成される。
- **SQL_SMALL_RESULT**を使用する場合、MySQL ではメモリ内のテンポラリテーブルが使用される。
- オプティマイザがテーブルスキャンをしたほうが効率的と判断しない限り、テーブルインデックスごとにクエリが行われ、スパンがレコードの 30% 以上である最適インデックスが使用される。しかし、現在では一定の率でインデックスかスキャンを使用するか判断されます。今バージョンのオプティマイザはより複雑で、テーブルサイズ、行数、そして I/O ブロックサイズを基準に推定します。
- 状況によっては、MySQL でデータファイルの参照もせずにインデックスからレコードを読み取る場合もある。インデックスから使用されるカラムのすべてが数値型の場合、クエリの解決にはインデックスツリーのみが使用される。
- レコードのそれぞれが出力される前に、**HAVING**節と一致しないレコードはスキップされる。

非常に高速なクエリのサンプルをいくつか紹介します。

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQLは次のクエリで、インデックスツリーのみを使用して解決します（インデックスのあるカラムが数値型であると想定）。

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

次のクエリは、ソートのパスを分けることなく、ソートしたレコードを取り出すためにインデックスを使用します。

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

Range 最適化

rangeアクセスメソッドはシングルインデックスを使用して、1つもしくは複数のインデックス値インターバルに含まれるテーブル行のサブセットを取得します。シングルパートかマルチパートインデックスに使用できます。以下のセクションでは**WHERE** 節からどのようにしてインターバルが抽出されるかの詳細説明を記述します。

シングルパートインデックスのためのRangeアクセスメソッド

シングルパートインデックスでは、インデックス値インターバルは**WHERE**節内の対応する状態で表現されます。よって「インターバル」よりも**range** 状態が使用されます。

シングルパートインデックスのための**range**状態の定義は以下のとおりです。

- **BTREE**と**HASH**の両インデックスにとって、**=**、**<=>**、**IN**、**IS NULL**、または**IS NOT NULL**演算子を使用した際の定数値の主部分の比較は**range**状態です。
- **BTREE**インデックスにとって、**>**、**<**、**>=**、**<=**、**BETWEEN**、**!=**、**<>**演算子、または**LIKE 'pattern'** ('pattern' がワイルドカードで始まらないとき)を使用しての定数値の主部分の比較は、**range**状態です。
- 全インデックスの種類にとって、**OR**または**AND**で結合されたマルチレンジ状態は、あるレンジ状態を形成します。

先の説明の「定数値」は以下の1つを意味しています。

- クエリ文字列の定数
- 同じ結合の**const**または**system**テーブルのカラム
- 関連しないサブクエリの結果
- 以前の型のサブ表現からのみ生成された表現

ここに**WHERE**節内でレンジ状態のクエリの例を以下に示します。

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;
```

```
SELECT * FROM t1
WHERE key_col = 1
OR key_col IN (15,18,20);

SELECT * FROM t1
WHERE key_col LIKE 'ab%'
OR key_col BETWEEN 'bar' AND 'foo';
```

非定数値が定数伝播フェーズ中に定数に変換されることに留意してください。

MySQLはWHERE節から、各インデックスよりレンジ状態を抽出しようとします。抽出プロセスの最中、レンジ状態を生成することができない状態は破棄され、重複するレンジを生成する状態は結合され、そして空のレンジを生成する状態は取り除かれます。

key1がインデックスカラムでnonkeyがインデックスカラムでない、下記のステートメントを考慮に入れてください。

```
SELECT * FROM t1 WHERE
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z');
```

key1の抽出プロセスは以下のとおりです。

1. オリジナルのWHERE節で始めてください。

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z')
```

2. nonkey = 4とkey1 LIKE '%b'はレンジスキャンに使用できませんので、取り除いてください。TRUEと置き換えることが、正しく取り除く方法です。こうすることによって、レンジスキャン中に適合する行を見落としません。TRUEで置き換えたとき、以下になります。

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
(key1 < 'bar' AND TRUE) OR
(key1 < 'uux' AND key1 > 'z')
```

3. 常に真か偽のコラップス状態

- (key1 LIKE 'abcde%' OR TRUE)は常に真です
- (key1 < 'uux' AND key1 > 'z')は常に偽です

定数でこれらの状態を置き換えると、以下になります。

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

不必要なTRUEとFALSE定数を取り除くことで、以下になります。

```
(key1 < 'abc') OR (key1 < 'bar')
```

4. 重複するインターバルを1つに結合することでレンジスキャンで使える最終状態が生成されます。

```
(key1 < 'bar')
```

一般的にレンジスキャンで使用する状態は(そして前の例で説明されたとおり)、WHERE節よりも制限されません。MySQLはレンジ条件を満たすが、WHERE節を完全に満たさない行にフィルタをかけるために追加でチェックを行います。

レンジ条件抽出アルゴリズムは任意のデプスの入れ子 AND/OR 生成子を取り扱うことができ、その出力はWHERE節内でどの順序で条件が現れるかに影響されません。

複合パートインデックスのためのレンジアクセスメソッド

複合パートインデックスのレンジ条件はシングルパートインデックスのレンジ条件の拡張です。複合パートインデックスのレンジ条件はインデックス行が1つもしくは複数のキータプルインターバルに含まれるよう制限します。キータプルインターバルはキータプルセット上で、インデックスからの順序づけを使用して定義されます。

例えば、複合パートインデックスが`key1(key_part1, key_part2, key_part3)`として定義され、以下のキータブルのセットがキー順序でリストされたとします。

key_part1	key_part2	key_part3
NULL	1	'abc'
NULL	1	'xyz'
NULL	2	'foo'
1	1	'abc'
1	1	'xyz'
1	2	'abc'
2	1	'aaa'

`key_part1 = 1`の条件は以下のインターバルを定義します。

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

インターバルは前データセットの4、5、そして6番目タブルをカバーし、レンジアクセスメソッドで使用できます。

それに引き換え、`key_part3 = 'abc'`条件はシングルインターバルを定義せず、レンジアクセスメソッドで使用できません。

以下はレンジ条件が複合パートインデックスでどのように働くかの詳細を説明します。

- **HASH**インデックスでは、同値を含む各インターバルが使用できます。これは以下のフォームをとる条件のためのみインターバルが生成できることを意味しています。

```
key_part1 cmp const1
AND key_part2 cmp const2
AND ...
AND key_partN cmp constN;
```

ここでは、`const1`、`const2`、... は定数で、`cmp` は`=`、`<=>`、または`IS NULL`比較演算子の1つであり、条件は全てのインデックスパートをカバーします。(つまり、`N`条件があり、各`N`-パートインデックスごとに1つあります。)例えば、以下は3パート**HASH**インデックスのレンジ条件です。

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

定数の定義は、[シングルパートインデックスのためのRangeアクセスメソッド](#)を参照してください。

- **BTREE**インデックスでは、インターバルは**AND**と結合された条件に使用できることがあります。これは、各条件がキーパートと定数値を、`=`、`<=>`、`IS NULL`、`>`、`<`、`>=`、`<=`、`!=`、`<>`、`BETWEEN`、または`LIKE 'pattern'`を使用して比較した場合です ('pattern' がワイルドカードで始まらない場合)。条件にマッチする行を全て含むシングルキータブルを判別できるかぎり、インターバルが使用できます。(あるいは、`<>` or `!=`が使用されたいれば2インターバル)。例えば、以下の条件では：

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

シングルインターバルは：

```
('foo',10,10) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
```

作成されたインターバルが初期条件よりも行が多い可能性があります。例えば、初期条件を満たさない前インターバルが('foo', 11, 0)の値を含んでいます。

- インターバルに含まれる行のセットをカバーする条件が**OR**で結合された場合、インターバルの結合に含まれる行のセットをカバーする条件を生成します。**AND**で条件が結合された場合、インターバルの交差点に含まれる行のセットをカバーする条件を生成します。例えば、この2パートインデックスの条件について：

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

インターバルは：

```
(1,-inf) < (key_part1,key_part2) < (1,2)
(5,-inf) < (key_part1,key_part2)
```

この例では、最初のラインのインターバルはレフトバウンドに1キーパートを使用し、ライトバウンドには2キーパートを使用します。2番目のラインのインターバルは1キーパートのみ使用します。**EXPLAIN** 出力の`key_len`カラムは、使用されたキープリフィックスの最大長を示しています。

場合によって、`key_len`はキーパート使用されたことを示すことがありますが、ユーザが期待していたとおりではないかも知れません。`key_part1`と`key_part2`がNULLだったとします。`key_len`カラム次の条件の2キーパート長を表示します。

```
key_part1 >= 1 AND key_part2 < 2
```

しかし、実際条件以下に変換されます。

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

シングルパートインデックスのためのRangeアクセスメソッドはシングルパートインデックスでレンジ条件のインターバルを結合か削除の際の最適化がどのように実行されるかを記述しています。マルチパートインデックスのレンジ条件にはアナログステップが実行されます。

インデックス結合最適化

Index Mergeメソッドは、複数のrangeスキャンを有する行を取得と、それぞれの結果を1つに結合するのに使用されます。この結合によって、基礎スキャンの結合、共通集合、あるいは交差点の結合が生成されます。

EXPLAIN出力では、インデックスメソッドはtype カラムでindex_mergeとして現れます。この場合、keyカラムは使用されたインデックスのリストが含まれ、key_lenはインデックスの最長キー部分が含まれます。

例：

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;
```

```
SELECT * FROM tbl_name
WHERE (key1 = 10 OR key2 = 20) AND non_key=30;
```

```
SELECT * FROM t1, t2
WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
AND t2.key1=t1.some_col;
```

```
SELECT * FROM t1, t2
WHERE t1.key1=1
AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

インデックス結合メソッドは複数のアクセスアルゴリズムがあります。(ExtraフィールドのEXPLAIN出力で見られます。)

- Using intersect(...)
- Using union(...)
- Using sort_union(...)

以下のセクションはこれらのメソッドの詳細を記述しています。

注:インデックス結合最適化アルゴリズムには以下の欠点があります。

- あるキーでレンジスキャンが可能な場合、インデックス結合は考慮されません。例えば、以下のクエリでは：

```
SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

このクエリでは、2つのプランが考えられます。

- (goodkey1 < 10 OR goodkey2 < 20)条件を使用したインデックス結合スキャン。
- badkey < 30条件を使用したレンジスキャン。

ただし、オプティマイザは2つ目のプランしか考慮しません。

- ユーザのクエリに複雑なAND/OR入れ子を持つWHERE節があり、MySQLが最適なプランを選択しない場合、以下のID法を使用して定義を分布してみてください。

```
(x AND y) OR z = (x OR z) AND (y OR z)
```

```
(x OR y) AND z = (x AND z) OR (y AND z)
```

- インデックス結合は、フルテキストインデックスには適用されません。将来的にリリースされるMySQLのバージョンでカバーできるよう、拡張する予定です。

種々のインデックス結合メソッドや他のアクセスメソッドの選択に関しては、選択枝のコスト予想によります。

インデックス結合共通集合アルゴリズム

このアクセスアルゴリズムが使用できるのは、**WHERE**節が異なるキーの複数のレンジ条件に変換、**AND**で結合され、各コンディションは以下の1つ：

- このフォームでは、インデックスパーツはN個あります(つまり、全てのインデックスパーツがカバーされています。)

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- InnoDBテーブルのプライマリキーをカバーするレンジ条件。

例：

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_col1=20;
```

```
SELECT * FROM tbl_name  
WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

インデックス共通集合アルゴリズムは全ての使用されたインデックスの同時スキャンを実行し、結合インデックススキャンから受信した行シーケンスの共通集合を生成します。

クエリで使用される全てのカラムが使用済みインデックスでカバーされている場合、完全なテーブル行は取得されません。(EXPLAIN 出力はExtra フィールド内のUsing indexを含んでいます)。そのようなクエリの例です：

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

クエリ内で使用されたカラムを使用されたインデックスがカバーしない場合、完全な行が取得されるのは全てのキーのレンジ条件が満たされたときのみです。

結合条件のうち1つがInnoDBテーブルのプライマリキー上の条件である場合、行取得には使用されず、他の条件を使用して取得した行にフィルターをかけるのに使用されます。

インデックス結合ユニオンアクセスアルゴリズム

このアルゴリズムの適用基準はインデックス結合メソッド共通集合アルゴリズムと似ています。このアクセスアルゴリズムが使用できるのは、**WHERE**節が異なるキーの複数のレンジ条件に変換、**OR**で結合され、各コンディションは以下の1つ：

- このフォームでは、インデックスパーツはN個あります(つまり、全てのインデックスパーツがカバーされています。)

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- InnoDBテーブルのプライマリキーをカバーするレンジ条件。
- インデックス結合共通集合アルゴリズムが適用できる条件

例：

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;
```

```
SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR  
(key3='foo' AND key4='bar') AND key5=5;
```

インデックス結合ソートユニオンアクセスアルゴリズム

このアクセスアルゴリズムは、**OR**により**WHERE** 節が複数のレンジ条件に変換されるが、インデックス結合メソッドユニオンアルゴリズムが適用できない場合に使用します。

例：

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

ソートユニオンアルゴリズムとユニオンアルゴリズムの違いは、ソートユニオンアルゴリズムはまず、行を返すまえに全ての行のIDを取得しソートしなければならないところにあります。

IS NULL最適化

MySQL では、`col_name = constant_value`の場合と同じ最適化を `col_name IS NULL`に対しても実行できます。たとえば、MySQL では、インデックスと範囲を使用して、`IS NULL`で `NULL`を検索できます。

例：

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

`WHERE`節内で`col_name IS NULL`で定義されたものを`NOT NULL`と使用する場合、その式は消去して最適化されます。この最適化は、結果的にカラムが`NULL`を生成する場合には生じません。たとえば、`LEFT JOIN`の右側のテーブルからきている場合。

MySQL は`col_name = expr AND col_name IS NULL`の組み合わせを最適化する機能が追加されています。これは解決されたサブクエリではよくあるフォームです。この最適化が使用される場合は、`EXPLAIN`は `ref_or_null`を表示します。

この最適化は、すべてのキー部分で `IS NULL`を 1 つ処理できます。

最適されたクエリのサンプルをいくつか紹介します (`t2`のキーを (`a,b`) とします)。

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null`はまずリファレンスキーの読み取りを行い、その後 `NULL`キー値のあるレコードの別検索を実行します。

この最適化では、1 つの `IS NULL`レベルしか処理できないことに注意が必要です。以下のクエリではMySQL は (`t1.a=t2.a AND t2.a IS NULL`)の部分に対してキーのルックアップを実行するのみで、`b`のキー部分は使用できません。

```
SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL)
OR (t1.b=t2.b AND t2.b IS NULL);
```

DISTINCT最適化

`DISTINCT`が `ORDER BY`と組み合わせられて用いられると、多くの場合はテンポラリテーブルが必要になります。

`DISTINCT`は `GROUP BY`をとまなう可能性が高いので、選択されないカラムを`ORDER BY`または `HAVING`した時に、どのように MySQL が機能するかを認識しておく必要があります。非常時フィールドとの `GROUP BY` および `HAVING`を参照してください。

ほとんどの場合、`DISTINCT`節は`GROUP BY`の特殊ケースと考えられます。例えば、下記の2クエリは等価です

```
SELECT DISTINCT c1, c2, c3 FROM t1 WHERE c1 > const;
```

```
SELECT c1, c2, c3 FROM t1 WHERE c1 > const GROUP BY c1, c2, c3;
```

等価であることによって、**GROUP BY**クエリに適用できる最適化は**DISTINCT**節のあるクエリにも適用できます。さらなる**DISTINCT**クエリ最適化の可能性については、**GROUP BY最適化**を参照してください。

LIMIT row_countを **DISTINCT**とともに使用した場合、MySQL は一意のレコードを **row_count**行検索するとただちに検索を停止します。

使用するテーブル内のカラムを使用しない場合、MySQL は最初にマッチするレコードを検索するとただちに未使用テーブルのスキャンを停止します。ここでは、**t1**が **t2**の前に使用され (**EXPLAIN**によるチェック)、**t2**で最初のレコードが検索されると **t2**からの読み取り (**t1**の特定のレコード) を停止します。

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

LEFT JOINとRIGHT JOIN最適化

MySQL の **AB join_condition**は以下のように実装されます。

- テーブル**B**はテーブル**A**と **A**が依存するすべてのテーブルに依存するように設定される。
- テーブル**A**は、**LEFT JOIN**条件で 사용되는すべてのテーブル (**B**を除く) に依存するように設定される。
- **LEFT JOIN**条件は、テーブル**B**からのレコードの取り出し方法の判定に使用される。(言い換えると、**WHERE**節の条件はいずれも使用されない)。
- あるテーブルが全てのテーブルの後に読み取られる場合を除き、通常の実装全てが行われる。依存関係が循環している場合は、MySQL からエラーが出力される。
- 標準の **WHERE**最適化すべてが実行される。
- **A**に**WHERE**節の条件にマッチするレコードがあり、**B**に **ON**条件にマッチするレコードがない場合、**B**のカラムの値が **NULL**に設定されたレコードが生成される。
- テーブルのいずれかに存在しないレコードを検索する際に **LEFT JOIN**を使用した場合: **col_name IS NULL**の**WHERE**節内で、**NOT NULL**と定義した **col_name** を column_name IS NULL で評価した場合、MySQL は **LEFT JOIN**条件に一致するレコードを 1つ検索すると、その後はレコードの検索 (特定のキー組み合わせの) を停止する。

RIGHT JOINの実装は **LEFT JOIN**と類似しています。

結合最適化がテーブルの結合順序を計算します。テーブル読み取り順序は **LEFT JOIN**と **STRAIGHT_JOIN**によって強制されるため、チェック対象のテーブル順列が減少し、結合最適化 (テーブルの結合順序を計算する) の動作の速度がさらに上がります。**LEFT JOIN**が **d**の前に読み取るように強制するため、MySQL では **b**の完全スキャンが実行されることに注目してください。

```
SELECT *
FROM a JOIN b LEFT JOIN c ON (c.key=a.key)
LEFT JOIN d ON (d.key=a.key)
WHERE b.key=d.key;
```

この場合の修正は逆に行われ、**a**と**b**が**FROM**節でリストされています。

```
SELECT *
FROM b JOIN a LEFT JOIN c ON (c.key=a.key)
LEFT JOIN d ON (d.key=a.key)
WHERE b.key=d.key;
```

LEFT JOINにとって、生成された **NULL**行で **WHERE**条件が常にfalseである場合、**LEFT JOIN**は通常の結合に変更されます。たとえば、**t2.column1**カラムが **NULL**であるとする、以下のクエリの **WHERE**節は falseになります:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

よって、通常の結合に変換しても問題ありません。

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

これでクエリが改善できる場合、MySQL がテーブル **t1** を読み取る前にテーブル **t2** を使用できるようになるためスピードが向上します。テーブルの順序を指定して強制する場合は **STRAIGHT_JOIN** を使用します。

入れ子結合最適化

結合を表す構文は入れ子結合を許可します。以下は **JOIN 構文** で記述された結合構文に関連します。

table_factor 構文は SQL 標準と比較して拡張されています。後者は **table_reference** のみ受付、かつこのリストは受け付けません。これは、**table_reference** アイテムのリスト内の点 (、) が内部結合と等価とする場合、この拡張は控えめです。例：

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

は以下と等価です。

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
      ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

MySQL では、**CROSS JOIN** は **INNER JOIN** と構文上等価です (置き換え可能です)。標準 SQL 上等価ではありません。**INNER JOIN** は **ON** 節と一緒に使用されます。**CROSS JOIN** は他の使用方法があります。

一般的に、inner join オペレーションを含む join 表現のかっこは無視できます。かっこを取り除きグルーピングオペレーションを左に移動させた後、join 表現は：

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
      ON t1.a=t2.a
```

以下の表現に変換されます。

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
      ON t2.b=t3.b OR t2.b IS NULL
```

しかし、二つの表現は等価ではありません。たとえば、**t1**、**t2**、そして **t3** が以下の状態であるとして。

- テーブル **t1** は (1)、(2) を含む
- テーブル **t2** は (1,101) 行を含む
- テーブル **t3** は (101) 行を含む

この場合、最初の表現は (1,1,101,101)、そして (2,NULL,NULL,NULL) を含む行の結果セットを返します。2番目の表現は (1,1,101,101)、(2,NULL,NULL,101) を含む行を返します。

```
mysql> SELECT *
-> FROM t1
-> LEFT JOIN
-> (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
-> ON t1.a=t2.a;
+-----+-----+
| a | a | b | b |
+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | NULL |
+-----+-----+

mysql> SELECT *
-> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
-> LEFT JOIN t3
-> ON t2.b=t3.b OR t2.b IS NULL;
+-----+-----+
| a | a | b | b |
+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | 101 |
+-----+-----+
```

下記の例では、外側 join オペレーションが内側 join オペレーションと一緒に使用されます。

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

その表現は以下の表現に変換できません。

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

既存のテーブル状態では、以下の2表現は異なる行セットを返します。

```
mysql> SELECT *
-> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+-----+
| a | a | b | b |
+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | NULL |
+-----+

mysql> SELECT *
-> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+-----+
| a | a | b | b |
+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | 101 |
+-----+
```

よって、外側 join 演算子を含む join 表現のかっこを取り除いた場合、元の表現の結果セットを変える可能性があります。

正確には、左外側 join オペレーションの右演算子のかっこを、そして右側 join オペレーションの左演算子のかっこを無視することができません。言い換えれば、外側 join オペレーションの内側テーブル表現のかっこを無視することはできません。他のオペランド (外側テーブルのオペランド) のかっこは無視できます。

以下の表現：

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

はこの表現と等価です：

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

テーブル`t1,t2,t3`と条件`P`属性`t2.b` and `t3.b`.

join 表現(`join table`)のjoin オペレーション実行順序が左から右でない場合、入れ子 join の話が出てきます。以下のクエリを考慮してください。

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
WHERE t1.a > 1

SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

上記クエリは以下の入れ子 join が含まれると考えられています。

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

最初のクエリでは、左 join オペレーションを使用して入れ子 join が生成されます。二番目のクエリでは内側 join オペレーションで生成されます。

最初のクエリでは、かっこは取り除いてもかまいません。join 表現の文法構成はjoin オペレーションと同じ実行順序を指令します。2番目のクエリでは、かっこなしでも join 表現があいまいに解釈されますが、かっこは取り除くことができません。(拡張された構文では、理論上はかっこなしでもパースされますが、2番目クエリの`(t2, t3)`のかっこは必要です。まだ、クエリはあいまいでない構文構成になります。これは`LEFT JOIN`と`ON`が左と右のデリミタの役割りを右の表現`(t2,t3)`で果たすからです。)

前述の例でこれらの点を証明しています。

- インナーjoinsのみ関する表現 (アウターjoins は不可) については、かっこは取り除けます。かっこを取り除いて左から右に評価を行うことができます(あるいは、テーブルの評価は好きな順序で行えます)。

- 一般的に、そとがわjoinや外がわjoinと併合された内側joinにとっては、同じではありません。かっこを取り除くことで結果を変えることがあるかもしれません。

入れ子外側joinsを含むクエリは内側joinを含むクエリと同じように、パイプライン形式で実行されます。正確には、入れ子ループjoinアルゴリズムが利用されます。入れ子ループjoinがクエリを実行する際利用するアルゴリズムスキーマを思い出してください。例えば、3つのテーブルT1,T2,T3に関するjoinクエリが、以下のフォームであるとしします。

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
      INNER JOIN T3 ON P2(T2,T3)
WHERE P(T1,T2,T3).
```

ここでは、P1(T1,T2)とP2(T3,T3)はjoin条件です(表現につく)。それに引き換え、P(t1,t2,t3)はテーブルT1,T2,T3カラム上の条件です。

入れ子ループjoinアルゴリズムはこのクエリを次のように実行します。

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

t1||t2||t3を使用した表記法は「t1、t2、そしてt3行を含むカラムを連鎖させることで作成された行」を意味します。以下の例では、行の名前があらわれる箇所にNULLとある場合、NULLはその行の各カラムに使用されることを意味します。例えば、t1||t2||NULLは「t3の各カラム毎のt1、t2、そしてNULL行のカラムを連鎖させることで作成された行。」

入れ子のある外側joinクエリを見てみましょう。

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON P2(T2,T3))
      ON P1(T1,T2)
WHERE P(T1,T2,T3).
```

このクエリでは、入れ子ループパターンを改良することで以下を取得します。

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

一般的に、外側joinオペレーションの最初の内側テーブル入れ子ループにとって、ループ前に消され、ループ後にチェックされるフラグが導入されます。フラグがオンになるのは、内側オペランドを表すテーブルから外側テーブルの現在行にマッチが見つかったときです。ループサイクルの最後でフラグがOFFの場合は、外側テーブルの現在行でマッチが見つからなかったときです。この場合、行がインナーテーブルのカラムNULL値で補われています。結果行は次の入れ子ループが出力へ、最終確認のため渡されますが、これは行が組み込まれた全ての外側joinの条件を満たしている場合のみです。

この例では、次の表現で表された外側joinテーブルは組み込まれています。

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

内側joinを含むクエリにとって、オプティマイザは以下のような異なる順序の入れ子ループが選択できることに注目してください。

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

外側テーブルを含むクエリに関しては、オプティマイザは外側テーブルのループが内側テーブルのループの前になる順序のみ選択可能です。よって、外側joinのクエリにとって、1つの入れ子順序のみ可能となります。以下のクエリでは、2つの異なる入れ子を評価します。

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
WHERE P(T1,T2,T3)
```

以下が入れ子です。

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
    f1:=TRUE
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

そして

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
    f1:=TRUE
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

両の入れ子にとって、**T1**は外側joinで使用されているため、外側ループでプロセスされなければいけません。**T2**と**T3**は内側joinで使用されているため、そのjoinは内側ループで処理されなければいけません。ただし、joinが内側joinのため、**T2**と**T3**はどちらの順序でも処理できます。

内側joinの入れ子ループアルゴリズムについては、クエリ実行性能に関する、重大な詳細を省きました。いわゆる、「後入れ先出し」条件に関することには触れませんでした。例えば、**WHERE**条件**P(T1,T2,T3)**が接続法によって表されるとします。

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

この場合、MySQLは内側joinを含むクエリの実行には以下の入れ子ループスキーマを使用します。

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2) {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

ここで、書く接続詞C1(T1)、C2(T2)、C3(T3)は評価が可能なよう、最も内側にあるループから最も外側にあるループまで押し出されます。もしC1(T1)が制限力の高い条件である場合、この条件の後入れ先出しはテーブルT1から内側ループに渡される行の数を大幅に減らします。結果的に、クエリの実行時間が大きく短縮できます。

外側joinを含むクエリについては、外側テーブルの現在行に内側テーブルからのマッチがあることが確認できてからWHERE条件がチェックされます。よって、内側入れ子ループの後だし先入れ条件最適化は外側joinを含むクエリには直接適用できません。ここではマッチが見つかった時に起動するフラグに守られた、条件つき後出し先入れの述語を紹介しなければいけません。

例えば、外側joinでは

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

ガードされた後だし先入れ条件を使用した入れ子ループスキーマは以下のようになります。

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
    such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
      BOOL f2:=FALSE;
      FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
          IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
            t:=t1||t2||t3; OUTPUT t;
          }
          f2=TRUE;
        }
      f1=TRUE;
    }
  IF (!f2) {
    IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
      t:=t1||t2||NULL; OUTPUT t;
    }
    f1=TRUE;
  }
}
IF (!f1 && P(t1,NULL,NULL)) {
  t:=t1||NULL||NULL; OUTPUT t;
}
}
```

一般的に、後だし先入れ述語はP1(T1,T2)やP(T2,T3)といったjoin条件から抽出できます。この場合、後だし先入れ述語は対応する外側joinオペレーションNULLに補われた行のチェックを妨げるフラグによって守られています。

ここで、1つの内側テーブルから同じ入れ子joinへのアクセスキーは、WHERE条件からの述語に誘導されている場合、禁止されています。(この場合条件付きのキーアクセスを使用することはできませんが、MySQLではこのテクニックはまだ使われていません5.1。)

外側Join 単純化

クエリのFROM節内テーブル表現は多くの場合単純化されています。

パーサ段階で、右外側joinオペレーションを含むクエリは左joinオペレーションを含む等価のクエリに変換されます。一般的には、変換は以下のルールに従って実行されます。

```
(T1, ...) RIGHT JOIN (T2,...) ON P(T1,...,T2,...) =
(T2, ...) LEFT JOIN (T1,...) ON P(T1,...,T2,...)
```

T1 INNER JOIN T2 ON P(T1,T2) フォームの全ての内側join表現は **T1,T2**、**P(T1,T2)** 結合された **WHERE** 条件によって置き換えられます。(あるいは、組み込まれたjoinのjoin条件が存在する場合は、それに置き換えられます)。

オプティマイザが外側joinオペレーションのjoinクエリ計画を評価する際、各オペレーション時、外側テーブルが内側テーブルより前にアクセスされます。そのようなプランは、入れ子ループスキーマによる外側joinオペレーションを含むクエリの実行のみ可能なため、オプティマイザ選択枝は制限されています。

例えば、以下のようなフォームのクエリがあるとします。

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
WHERE P(T1,T2) AND R(T2)
```

R(T2) がテーブル **T2** からマッチする行を大幅に狭めます。クエリをそのまま実行した場合、オプティマイザは **T1** より前に **T2** をアクセスする以外に選択枝が与えられず、非常に非効率的な実行プランになることがあります。

幸い、MySQL **WHERE** 条件が null-rejected の場合、そのようなクエリを外側joinオペレーションを含まないクエリに変換します。もし外側join オペレーションが **NULL**-によって補われたオペレーションのために作成された行の **FALSE** または **UNKNOWN** に評価される場合は、null-rejected と呼ばれます。

よって、この外側joinでは：

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

以下のような条件は null-rejected です：

```
T2.B IS NOT NULL,
T2.B > 3,
T2.C <= T1.C,
T2.B < 2 OR T2.C > 1
```

以下のような条件は null-rejected ではありません：

```
T2.B IS NULL,
T1.B < 3 OR T2.B IS NOT NULL,
T1.B < 3 OR T2.B > 3
```

条件が外側joinオペレーションにとって null-rejected か否かを確認する一般的なルールは単純です。以下の場合、条件は null-rejected になります。

- フォームが **A IS NOT NULL** で、**A** が内側テーブルのどれかの属性である場合
- **UNKNOWN** のアークギュメントの内1つが **NULL** で、且つ **UNKNOWN** と評価する内側テーブルへのリファレンスを含む述語である場合
- null-rejected 条件をコンジャンクトとして含んでいる結合子の場合。
- null-rejected 条件のディスジャンクションの場合。

条件は1つクエリでの外側joinオペレーションで null-rejected となり、他のクエリで not null-rejected になりえます。以下のクエリでは：

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
LEFT JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

WHERE 条件は、2つ目の外側joinオペレーションでは null-rejected であり、1つ目の外側joinオペレーションでは not null-rejected です。

もし **WHERE** 条件がクエリの外側joinオペレーションで null-rejected である場合、外側joinオペレーションは内側joinオペレーションに置き換えられます。

例えば、前のクエリは以下のクエリに置き換えられます。

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
INNER JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

オリジナルクエリには、1つのアクセス順序 **T1,T2,T3** と対応するプランをオプティマイザが評価します。クエリを置き換える場合は、**T3,T1,T2** アクセスシーケンスを追加で考慮します。

1つの外側joinオペレーションの変換は別のオペレーションの変換を引き起こす場合があります。よって、以下のクエリでは：

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      LEFT JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

は最初にこのクエリに変換されます。

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      INNER JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

それはこのクエリと等価です。

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

条件が **T3.B=T2.B** null-rejected であることと、外側joinを含まないクエリを取得したため、残る外側joinオペレーションは内側joinに置き換えることができます。

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

時折、組み込まれた外側joinオペレーションを置き換えることができて、組み込まれた外側joinが変換できない場合があります。以下のクエリでは：

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A
WHERE T3.C > 0
```

は以下変換されます。

```
SELECT * FROM T1 LEFT JOIN
      (T2 INNER JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A
WHERE T3.C > 0,
```

それは組み込まれた外側joinオペレーションを含むフォームにのみ書き換えることができます。

```
SELECT * FROM T1 LEFT JOIN
      (T2,T3)
      ON (T2.A=T1.A AND T3.B=T2.B)
WHERE T3.C > 0.
```

組み込まれた外側joinオペレーションをクエリに変換する場合、**WHERE**条件とともに組み込まれた外側joinのjoin条件を考慮しなければいけません。以下のクエリでは：

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A AND T3.C=T1.C
WHERE T3.D > 0 OR T1.D > 0
```

WHERE は組み込まれた外側joinではnot null-rejected ですが、組み込まれた外側join **T2.A=T1.A AND T3.C=T1.C** のjoin条件はnull-rejectedになります。よって、クエリは以下に変換されます。

```
SELECT * FROM T1 LEFT JOIN
      (T2, T3)
      ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
WHERE T3.D > 0 OR T1.D > 0
```

ORDER BY最適化

余分なソートを行わずに **ORDER BY** の要求に応じるために、MySQL はインデックスを使用する場合があります。

全ての使用されていないインデックス部分と他の部分が **WHERE** 節内で定数であるカラムである場合、**ORDER BY** がインデックスに完全にマッチしない場合でもこのインデックスを使用できます。次のクエリではインデックスを使用して **ORDER BY** 部分を解決します。

```

SELECT * FROM t1
ORDER BY key_part1, key_part2, ...;

SELECT * FROM t1
WHERE key_part1=constant
ORDER BY key_part2;

SELECT * FROM t1
ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
WHERE key_part1=1
ORDER BY key_part1 DESC, key_part2 DESC;

```

MySQL で **ORDER BY** の解決にインデックスを使用できない場合は以下のとおりです (この場合も MySQL は **WHERE** 節の条件に一致するレコードの検索にインデックスを使用します)。

- 複数のキーに対して **ORDER BY** を実行する場合。

```
SELECT * FROM t1 ORDER BY key1, key2;
```

- 連続しないキー部分に対して **ORDER BY** を実行する場合。

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
```

- **ASC** と **DESC** が混在している場合。

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
```

- 行の取り出しに使用されるキーが **ORDER BY** の実行に使用されるキーと異なる場合。

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- **ORDER BY** で多くのテーブルとカラムを結合していて、それら全てがレコードの取り出しに使用される最初の非 const テーブルではない場合 (これは **EXPLAIN** で出力される最初のテーブルで、かつ、**const** メソッドを使用していないテーブル)。
- **ORDER BY** と **GROUP BY** 式が異なる場合。
- 使用されたテーブルインデックスが、並び順にレコードを格納していないインデックスタイプの場合。
(**MEMORY** テーブルの **HASH** インデックスなど)。

EXPLAIN SELECT ... ORDER BY を使用すると、MySQL でインデックスを使用してクエリを解決できるかどうかをチェックできます。Extraカラムに **Using filesort** が出力された場合は、MySQL で **ORDER BY** の解決にインデックスを使用できません。 **EXPLAIN** を使用して、クエリを最適化することを参照してください。

ソートキー値と行ポジションだけでなく、クエリに必要なカラムまで記憶する **filesort** 最適化が使用されます。これにより行の2度読みを避けられます。 **filesort** アルゴリズムは以下のように実行されます。

1. **WHERE** 節とマッチする行を読む。
2. 各行ごとに、クエリに必要なカラムとソートキー値と行ポジションを含むタプル値を記憶する。
3. ソートキー値でタプルを並べ替える
4. 並べ替えられた順序で行を取得しますが、テーブルに2度アクセスするよりも、並べ替えられたタプルから必要なカラムを読み取ります。

MySQL の旧バージョンで使用されていたアルゴリズムよりも格段に改良されています。

遅滞を避けるため、この最適化は **max_length_for_sort_data** システム変数の値をソートタプル内の余分なカラムのトータルサイズを超えない場合使用されます。(この変数が高く設定されると、活発なディスクアクティビティに対して低い CPU 活動といった状態が発生します。)

ORDER BY 速度を上げたい場合、MySQL が余分な並び替えフレーズよりもインデックスを使用できるか確認してください。これが不可能な場合、以下の手段を試してみてください。

- **sort_buffer_size** 変数のサイズを大きくしてください。

- `read_rnd_buffer_size`変数のサイズを大きくしてください。
- `tmpdir`を変更することで、秋スペースの要領が多い専用のファイルシステムを示してください。この選択肢はラウンドロビン方式で複数のパスを受領します。Unixでは、パスはコロンを含む文字(:)で分けられ、ウィンドウズ、Netware、そしてOS/2ではセミコロンを含む文字(;)で分けられるべきです。この特性を利用して複数のディレクトリに渡り負荷を分散することができます。注:パスは、同ディスクのパーティションで分けられた領域ではなく、異なる物理的なディスクのファイルシステム内のディレクトリに通じます。

MySQLはデフォルトで、`GROUP BY col1, col2, ...`の全クエリを`ORDER BY col1, col2, ...`で指定したかのよう
に、クエリをソートします。同じカラムリストを含む`ORDER BY`節を明示的に取り入れた場合、ソートが実行される
とはいえ、MySQLは速度ペナルティなしに最適化します。クエリに`GROUP BY`が含まれていながら、結果のソー
トに費やすオーバーヘッドを避けたい場合、`ORDER BY NULL`を指定することでソートを実行しないようにするこ
とができます。例:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

GROUP BY最適化

`GROUP BY`節を満たす最も一般的な方法は、テーブル全体をスキャンし、各グループの全ての行が連続する新しいテンポラリテーブルを作成することです。それにより、このテンポラリテーブルを使用してグループの発見や集約ファンクション (の適用が可能になります。場合により、MySQLはインデックスアクセスを使用することでテンポラリテーブルの作成を回避することが可能です。

`GROUP BY`インデックスを使用するための最も重要な前提条件は、全ての`GROUP BY`カラムは同じインデックスから属性を紹介することと、そしてインデックスがキーを正しい順序で保存することです。(例えば、`BTREE`はインデックスであり、`HASH`インデックスではありません)。テンポラリテーブルの使用がインデックスアクセスに置き換えることが可能かは、クエリ内でどのインデックス部分が使用されるか、その部分を指定する条件、そして選択された集約ファンクションにもよります。

インデックスアクセスを通して`GROUP BY`クエリを実行するには以下に記述された、2つの方法があります。最初の方法では、グルーピングオペレーションは全てのレンジ前提(これが有る場合に限り)と共に適用されます。2つ目の方法は、まずレンジスキャンを実行し、その後結果タプルをグループします。

ルースインデックススキャン

`GROUP BY`を処理する最も効率的な方法は、グループフィールドを直接取得するのにインデックスが使用されるときです。このアクセスメソッドで、MySQLはキーが順序づけられている、インデックス型の特性を利用します。(例えば、`BTREE`)。この特性は全ての`WHERE`条件を満たすインデックス内のキーを考慮せずとも、インデックス内のルックアップグループの仕様を可能にします。このアクセスメソッドはインデックス内のほんの一部のキーを考慮するため、`loose index scan`と呼ばれています。`WHERE`節がない場合、ルースインデックススキャン (`loose index scan`) はナンバーグループの数だけキーを読みます。これはキーの総数よりも少ない数かもしれません。もし`WHERE`節がレンジ前提を含む場合、(`EXPLAIN`を使用して、クエリを最適化する内に記述された `range join`型に関するディスカッションを参照してください)、a `loose index scan` はレンジ条件を満たす各グループの最初のキーを参照し、最低限のキーの数を読みます。これは以下の条件下で可能になります。

- クエリはシングルテーブル上にある。
- `GROUP BY`はインデックスの最初の連続部分を含む。(もし、`GROUP BY`の代わりにクエリが`DISTINCT`節を含む場合、全ての異なる属性はインデックスの最初を参照する。)
- 使用される集約ファンクション (ある場合) は`MIN()`と`MAX()`であり、全て同じカラムを参照します。
- クエリで参照された`GROUP BY`以外のインデックス部分は定数でなければいけません(つまり、定数と等価であるよう参照されなければいけません) が、`MIN()`あるいは`MAX()`ファンクションは例外です。

そのようなクエリの`EXPLAIN`出力は`Extra` カラム内の`Using index for group-by`を示します。

テーブル`t1(c1,c2,c3,c4)`にインデックス`idx(c1,c2,c3)`があることを前提に、以下のクエリがこのカテゴリに属します。

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
```

```
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

以下の理由により以下のクエリはこのクイックセレクトメソッドを使用しての実行はできません。

- **MIN()**や**MAX()**以外の集約ファンクションが存在します。例えば：

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

- 以下で示されるよう、**GROUP BY**節内のフィールドはインデックスの最初の部分に属するものではありません。

```
SELECT c1,c2 FROM t1 GROUP BY c2, c3;
```

- クエリは、定数と等価ではない、**GROUP BY**部分の後にくるキー部分を参照しています。

```
SELECT c1,c3 FROM t1 GROUP BY c1, c2;
```

タイトインデックススキャン

タイトインデックススキャンは、クエリ条件によって、フルインデックススキャンもしくはレンジインデックススキャンのいずれかとなります。

ルースインデックスに見合う条件がなければ、**GROUP BY**クエリの一時的なテーブル作成は拒否されます。**WHERE**節にレンジ条件がある場合、このメソッドはこれらの条件を満たすキーだけを読み取ります。もしくは、インデックススキャンとして実行されます。この方法は**WHERE**節に定義された各レンジ内の全てのキーを読むか、もしくはレンジ条件がなければ全てのインデックスをスキャンするため、タイトインデックススキャンと呼ばれています。レンジ条件を満たす全てのキーが認識された後でのみ、タイトインデックススキャンを用いてグループ演算が実行されることに注意してください。

このメソッドを起動させるには、クエリ内にある全てのカラムに対する定数同等条件があれば十分です。ここでいうクエリとは、**GROUP BY**キーの前もしくは間にくるキー部分を参照するものです。同等条件の定数は検索キーの「ギャップ」を埋めるため、インデックスの完全なプレフィックスを形成できます。これらインデックスのプレフィックスは、インデックスルックアップに使用できます。**GROUP BY**結果のソートを要求した場合、またはインデックスのプレフィックスである検索キーを形成できる場合、MySQLもまた余分なオペレーションのソートを拒否します。これは、すでに整頓されたインデックス内でプレフィックスを用いて探索することは、全てのキーを順番に検索することになるからです。

以下のクエリは、前述のルースインデックススキャンアクセス方法では機能しません。しかし、タイトインデックススキャンアクセス方法 (テーブル`t1(c1,c2,c3,c4)`上にインデックス`idx(c1,c2,c3)`があると仮定する)では、機能します。

- **GROUP BY**にはギャップがありますが、条件`c2 = 'a'`によってカバーされます。

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- **GROUP BY**は、キーの最初の部分では開始されませんが、その部分に対してある定数を与える条件があります。

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

LIMITの最適化

HAVINGを使用するのではなく **LIMIT row_count**を使用している場合、MySQL によるクエリの処理方法が異なる場合があります。

- **LIMIT**を使用して数行しか選択していないと、フルテーブルスキャンが行われそうな場合に、MySQL はインデックスを使うことがある。
- **ORDER BY**とともに **LIMIT row_count**を使用している場合、MySQL ではすべてのテーブルがソートされるのではなく、最初の **row_count**行の検索が行われた時点でただちにソートを終了する。インデックスを用いて整頓されている場合、これはとても速い方法です。ファイルソートが実行されなければならない場合、最初の**row_count**行が検索されたことを確認する前に、**LIMIT**節を用いないクエリに当てはまる全ての行が選択されなければなりません。そして、それらのほとんど、もしくは全部がソートされなければなりません。いずれの場合でも最初の行が検索された後では、どの結果セットリマインダもソートする必要はありません。また、MySQLもその必要はありません。

- `LIMIT row_count`を `DISTINCT` とあわせて使用した場合、MySQL は一意の `row_count`行を検索するとただちに停止します。
- `GROUP BY`がキーを順番に読む (またはキーのソートを実行して読む) ことで解決でき、キーの値が変わるまで サマリが計算される場合もあります。この場合、`LIMIT row_count`では不要な `GROUP BY` 値の計算がすべて行われなくなります。
- MySQL が要求された行数をクライアントに送信すると、クエリが中止されます (`SQL_CALC_FOUND_ROWS`を使用していない場合)。
- `LIMIT 0`は常に迅速に空のセットを返します。これは、クエリの妥当性チェックに役立ちます。MySQL APIの1つを使用している場合、結果カラム型の獲得も可能です。(この方法は、MySQLモニター(`mysql`プログラム)では、`Empty set`と表示されるだけで働きません。このため、代わりに`SHOW COLUMNS`または`DESCRIBE`を使用する必要があります。)
- サーバでテンポラリテーブルを使用してクエリが解決される場合、`LIMIT row_count`節が必要な領域の計算に使用されます。

テーブルスキャンを避ける方法

MySQL がクエリを解決するのにテーブルスキャンを使用すると、`EXPLAIN`からの出力は、`type`カラムの`ALL`を表示します。これはたいてい以下の条件下で生じます。

- テーブルが小さすぎて、時間のかかるキールックアップよりもテーブルスキャンの実行のほうが遅くなります。このことは、10行以下の行や短い行をもつテーブルにはよく起こることです。
- インデックスカラムに対して、`ON`または`WHERE`節内に使用できる制限はありません。
- インデックスカラムを定数値と比較し、MySQLは (インデックスツリーに基づいて) その定数がテーブルの大きすぎる部分をカバーしているか、またテーブルスキャンが高速に行われるかを計算します。`WHERE` 節最適化を参照してください。
- 他のカラムをとおして、低濃度 (多数の行がキー値に当てはまる) でキーを使用します。この場合、MySQLは、キーを使用して多数のキールックアップが実行され、またテーブルスキャンもより速く行われるであろうと認識します。

小さいテーブルに対しては、テーブルスキャンはたいてい適切であり、実行の際の影響は無視されます。大きいテーブルに対しては、オプティマイザが間違ったテーブルスキャンを選択しないように、以下の方法を試してください。

- スキャンされたテーブルのキー配置を更新するには、`ANALYZE TABLE tbl_name` を使用してください。`ANALYZE TABLE` 構文を参照してください。
- MySQL に、テーブルスキャンは既存インデックスを使用するのに比べて大変時間がかかることを示すには、`FORCE INDEX`をスキャンされたテーブルに使用してください。

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

`SELECT` 構文を参照してください。

- オプティマイザに1,000キーシーク以上のキースキャンがないことを認識させるには、`--max-seeks-for-key=1000` オプションで `mysqld`を起動させるか、もしくは`SET max_seeks_for_key=1000`を使用してください。`システム変数`を参照してください。

INSERTステートメントの速度

行挿入の時間は、以下の要因によって決定されます。(数はおよその割合を示します。)

- 接続: (3)
- サーバへのクエリの送信: (2)
- クエリの解析: (2)

- 行挿入:(1 × 行サイズ)
- インデックス挿入:(1 × インデックス数)
- クローズ: (1)

テーブルを開く初期オーバヘッドは算入されていません (これは同時実行クエリのそれぞれで1回実行されます)。

テーブルのサイズによって対数Nの分だけインデックス挿入の速度が低下します (B ツリー)。

挿入の速度を上げる方法

- 1つのクライアントから同時に多数の行を挿入する場合は、マルチプルVALUESリストでINSERTステートメントを使用します。これで独立したINSERTステートメントの使用時と比較して大幅に (場合によっては数倍) 速度が上がります。空ではないテーブルにデータを追加する場合は、さらに速度を上げるためにbulk_insert_buffer_size変数を調整します。システム変数を参照してください。
- 異なる複数のクライアントから大量のレコードを挿入する場合は、INSERT DELAYEDステートメントを使用すると速度を上げることができます。INSERT DELAYED 構文を参照してください。
- MyISAM テーブルでは、テーブルに削除された行がない場合、SELECTの実行と同時に行を挿入できることに注意してください。同時挿入を参照してください。
- テキストファイルからテーブルをロードする場合はLOAD DATA INFILEを使用します。通常、これはINSERTステートメントを使用する場合と比較して、20倍速度が上がります。LOAD DATA INFILE 構文を参照してください。
- テーブルにインデックスが多数ある場合、操作を少し追加するだけでMyISAMテーブルのLOAD DATA INFILEの実行速度をさらに上げることができます。以下の手順を使用してください。
 1. CREATE TABLEを使用して、テーブルを作成します。
 2. FLUSH TABLESステートメントまたはmysqladmin flush-tablesコマンドを実行します。
 3. myisamchk --keys-used=0 -rq /path/to/db/tbl_nameを使用します。これでテーブルからすべてのインデックスの使用が削除されます。
 4. LOAD DATA INFILEを使用して、テーブルにデータを挿入します。これはインデックスをまったく更新しないため、非常に高速になります。
 5. テーブルを読み取り専用にする場合は、myisampackを実行してテーブルを小さくします。圧縮テーブルの特徴を参照してください。
 6. myisamchk -rq /path/to/db/tbl_nameを使用してインデックスを作成しなおします。これは、ディスクに書き込む前にメモリにインデックスツリーを作成してディスクシークを回避するため、LOAD DATA INFILE中のインデックス更新が非常に高速になります。生成されたインデックスツリーは完全にバランスが取られています。
 7. FLUSH TABLESステートメントまたはmysqladmin flush-tablesコマンドを実行します。

データを挿入したMyISAMテーブルが空の場合は、LOAD DATA INFILEは上記の最適化を自動的に実行します。上記手順との主な相違点は、LOAD DATA INFILEステートメントの実行中にサーバにインデックスの再作成を割り当てる場合より、myisamchkにインデックス作成用のテンポラリメモリ割り当てるほうが、より大幅に割り当てることができる点です。

myisamchkよりも以下のステートメントを使用して、MyISAMのインデックス利用を可能にしたり、不可能にしたりできます。これらのステートメントを使用すると、FLUSH TABLEオペレーションをスキップできます。

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- 非トランザクショナルテーブル上で、複数ステートメントを使用して実行されるINSERT速度を上げるには、テーブルをロックしてください。

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
```

```
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

主な速度の相違点は、すべての **INSERT** ステートメントの完了後にインデックスバッファが 1 回のみディスクにフラッシュされることです。通常は、**INSERT** ステートメントの数と同じだけ、インデックスバッファのフラッシュが行われます。すべての行を 1 つの **INSERT** で挿入できる場合はロックの必要がありません。

トランザクショナルテーブルの場合は、**LOCK TABLES**ではなく **START TRANSACTION**および**COMMIT** を使用して速度の改善を図ります。

ロックは複数の同時接続テストの合計時間も短縮するが、一部のスレッドの最大待機時間は長くなります。(ロックの際に待機するため) 例:

1. 接続1は1000行を挿入
2. 接続2, 3, 4は1行を挿入
3. 接続5は1000行を挿入

ロックを使用しない場合、2、3、4 は 1 と 5 の前に終了します。ロックを使用した場合は、2、3、4 は 1 と 5 の前には終了しない確率が高くなりますが、合計時間は約 40% 短縮されます。

MySQL では、**INSERT**、**UPDATE**、および **DELETE** の演算が非常に速いため、約 5 つより多い挿入や更新をする前にロックを追加すると、総合的なパフォーマンスを改善できます。1 行で非常に多数の挿入を実行する場合は、ときどき (約 1,000 行ごと) **LOCK TABLES**に**UNLOCK TABLES** を続けて実行して、他のスレッドからのテーブルへのアクセスを可能にすることができます。それでもパフォーマンスの増加が得られます。

アウトラインストラテジー使用時でも、データのロードには**LOAD DATA INFILE**のほうが**INSERT**よりも大幅に高速です。

- **MyISAM**テーブルおよび**LOAD DATA INFILE**と**INSERT**の両方に対するパフォーマンスの向上には、**key_buffer_size**システム変数値を上げてキーキャッシュを拡張します。**サーバパラメータのチューニング**を参照してください。

MySQL Enterprise

```
#####MySQL Network Monitoring and Advisory Service#####
##### http://www-jp.mysql.com/products/enterprise/advisors.html#
#####
```

UPDATEステートメントの速度

更新ステートメントは、**SELECT**クエリと同様に最適化されますが、書き込みオーバーヘッドが加算されます。書き込みの速度は更新対象のデータのサイズおよび更新対象のインデックス数によって異なります。変更がないインデックスは更新されません。

更新の速度を上げるもう 1 つの方法は、更新を遅延して 1 行で多数の更新を後から行うことです。1 行での多数の更新は、テーブルをロックすると同時に行う場合と比較して大幅に高速に実行できます。

可変長テーブル**MyISAM**の場合は、合計の長さが今よりも長いものに行を更新すると、行が分割される場合があることに注意します。このため、頻繁にこれを実行する場合は、ときどき **OPTIMIZE TABLE**することが重要になります。**OPTIMIZE TABLE** 構文を参照してください。

DELETEステートメントの速度

行の削除に要する時間は、完全にインデックス数に比例します。行削除の速度を上げるには、**key_buffer_size**システム変数を上げて、キーキャッシュのサイズを拡大します。**サーバパラメータのチューニング**を参照してください。

テーブル内のすべての行を削除する場合は、**TRUNCATE TABLE tbl_name**のほうが**DELETE FROM tbl_name**を使用するより高速です。**TRUNCATE** 構文を参照してください。

その他の最適化のヒント

このセクションでは、クエリ処理高速化のためのヒントを挙げます。

- 接続オーバーヘッドを回避するには、データベースに対して永続的な接続を使用します。永続的な接続を使用せずにデータベースに対して多数の新規接続を実行する場合は、`thread_cache_size`変数の値の変更が必要になることがあります。[サーバパラメータのチューニング](#)を参照してください。
- 常にすべてのクエリがテーブル内に作成したインデックスを実際に使用していることを確認します。MySQLでは、`EXPLAIN`ステートメントでこれを実行できます。[EXPLAINを使用して、クエリを最適化する](#)を参照してください。
- 大量に更新された `MyISAM` テーブルに対して複雑な `SELECT` クエリを使用しないようにします。これで、読み手と書き手間の競合から生じるテーブルロックを回避します。
- 削除されたレコードがない `MyISAM` テーブルの場合は、別のクエリでそのテーブルからの読み取りが行われるのと同時に行を挿入できます。これがあなたにとって重要ならば、行削除の回避をおこなうテーブルの使用を検討します。また、大量の行削除後の `OPTIMIZE TABLE` の実行を検討します。[MyISAM ストレージエンジン](#)を参照してください。

この動作は、`concurrent_inserts`変数設定をとおして、変更されます。行が削除されたテーブル上であっても、新しい行を付加できます。(またその結果、同時に挿入することが可能です。)[同時挿入](#)を参照してください。

- `ARCHIVE` テーブルで生じるデータ圧縮問題を修復するのに、`OPTIMIZE TABLE` を使用できます。[ARCHIVE ストレージエンジン](#)を参照してください。
- 通常 `expr1`、`expr2`、... の順で行を読み取る場合は、`ALTER TABLE ... ORDER BY expr1, expr2, ...` を使用してください。テーブルが大幅に変更された後にこのオプションを使用すると、パフォーマンスを改善できます。
- 他のカラムの情報を基にした「ハッシュされた」カラムを導入することが役立つ場合があります。このカラムが短いもので、一意性がある場合は、多数のカラムに「大きな」インデックスを使用するより大幅に高速化できます。MySQLでは、追加カラムの使用が以下のように非常に容易です。

```
SELECT * FROM tbl_name
WHERE hash_col=MD5(CONCAT(col1,col2))
AND col1='constant' AND col2='constant';
```

- 頻繁に変わる `MyISAM` テーブルでは、全ての可変長カラム(`VARCHAR`、`BLOB`、そして `TEXT`)の使用を避けてください。たった1つの可変長カラムを含む場合でも、テーブルではダイナミック行フォーマットが使用されます。[13章ストレージエンジンとテーブルタイプ](#)を参照してください。
- 一般に、1つのテーブルを複数のテーブルに分割することは、行が大きくなるだけで高速化の役には立ちません。行にアクセスする際の、最も大きなパフォーマンス要因は、レコードの最初のバイトを見つけるためのディスクシークです。データの検索後、ほとんどの新規ディスクでは、大多数のアプリケーションに十分な速度で行全体を読み取ることができます。テーブルの分割が実際に有効な状況は、固定長テーブルへの変更が可能な可変長 `MyISAM` テーブルの場合、テーブルのスキャンを非常に頻繁に必要としながらもほとんどのカラムに必要としない場合のみです。[13章ストレージエンジンとテーブルタイプ](#)を参照してください。
- 多数の行の情報から計算する頻度を非常に高くする必要がある場合(カウントの場合など)、新たなテーブルを導入し、リアルタイムでカウンタを更新するほうがはるかに適しています。以下のような更新は非常に高速にできます。

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
```

これは、`MyISAM` のようにテーブルロック(複数リーダ/単一ライタ)のみのMySQLストレージエンジンを使用する場合に、非常に重要です。また、このような場合は行ロックマネージャで必要な作業が少なくなるため、ほとんどのデータベースでパフォーマンスが改善されます。

- 大きなログテーブルから統計を収集する必要がある場合は、テーブル全体をスキャンするのではなく、サマリテーブルを使用します。サマリの管理は、「リアルタイム」で統計を実行する場合と比較して非常に高速になります。何らかの変更がある(業務上の決定に応じて)場合は、ログから新規にサマリテーブルを再生成したほうが、実行アプリケーションの変更よりはるかに高速です。
- 可能であれば、レポートを「リアルタイム」か「集計」かのいずれかに分類するように推奨します。集計レポートに必要なデータは、サマリテーブルから生成され、サマリテーブルは実データから生成されます。

- カラムにデフォルト値がある利点を生かします。挿入対象の値がデフォルト値と相違する場合のみ明示的に値を挿入します。これで、MySQL が要する解析作業が軽減され、挿入の速度が改善されます。
- 状況によっては、データをBLOBにバックし、格納したほうが便利です。このような場合は、BLOB へのバックおよびバック解除を行うコードをアプリケーションに追加する必要がありますが、あるステージにおける大量のアクセスを省略できることになります。これは、固定長テーブル構造に準拠しないデータがある場合に実用的です。
- 通常は、すべてのデータが冗長にならないようにする必要があります。(データベースセオリの第3正規化)。しかし、高速化を図る必要がある場合はデータなどの複製やサマリテーブルの作成をためらうべきではありません。
- ストアドルーチンや UDF (ユーザ定義関数) はパフォーマンスの向上に役立つ手段です。詳しくは、[17章ストアドプロシージャとファンクション](#)および[Adding New Functions to MySQL](#)を参照してください。
- アプリケーションのクエリと応答をキャッシュすること、および挿入と更新の同時実行を試行することは必ず高速化に役立ちます。データベースでロックテーブルがサポートされる場合 (MySQL や Oracle など) は、これによって確実にすべての更新後にインデックスキャッシュが 1 回だけフラッシュされるようにできます。MySQL のクエリキャッシュも、同様の結果を得るために利用できます。詳しくは[MySQL クエリ キャッシュ](#)を参照してください。
- データの書き込みするタイミングを知る必要がない場合はINSERT DELAYEDを使用します。多数の行が 1 回のディスクへの書き込みで書き込まれるため、これで高速化が図れます。
- SELECTの優先を上げる場合は、INSERT LOW_PRIORITY を使用します。
- キューをジャンプするようにする場合は、SELECT HIGH_PRIORITY を使用します。言い換えると、書き込み待機中のユーザがいる場合でも、SELECTを実行できるようになる。
- 1 つの SQLステートメントで多数の行を格納するには、複数行のINSERT ステートメントを使用します。これは、MySQLを含む多数の SQL でサポートされています。
- 大量のデータをロードする場合はLOAD DATA INFILEを使用します。これは通常のINSERTより高速になります。
- 一意の値にするには、AUTO_INCREMENTカラムを使用します。
- 一定の間隔で OPTIMIZE TABLE を使用して、動的MyISAMテーブルの断片化を回避します。[MyISAM テーブルストレージフォーマット](#)を参照してください。
- さらに高速化が可能であれば、MEMORYテーブルを使用します。詳しくは[MEMORY \(HEAP\) ストレージエンジン](#)を参照してください。頻繁にアクセスされる非クリティカルデータ (クッキーなしでユーザに最後に表示されたバナーの情報など) にはMEMORYテーブルを使用します。多くのWebアプリケーション環境では、揮発性データの処理にユーザセッションも使用できます。
- Webサーバでは、画像と他のバイナリアセットを通常ファイルとして格納します。言い換えると、データベース内にはファイル参照のみを格納します。この主な理由は、通常の Web サーバのほうがデータベースコンテンツと比較してファイルのキャッシュに優れているためです。このため、ファイルを使用したほうがシステムの高速化を容易に図れます。
- 別のテーブルで同一情報を扱うカラムは、同じ宣言をし、同じデータ型を持つようにします。この結果、一致カラムに基づく結合速度が速くなります。
- カラム名はなるべく単純なものに保持します。たとえば、customerテーブルではcustomer_nameではなく name を使用します。他の SQL サーバに移植可能にすることを考慮するなら、名前を 18 文字未満にする。
- 高速化が大きく必要とされる場合は、複数の SQL サーバがサポートするデータストレージの低レベルインタフェースを調べる必要があります。たとえば、MySQL MyISAMストレージエンジンに直接アクセスすることによって、SQL インタフェース使用時と比較して 2~5 倍の速度が得られることもあります。これを実行可能にするには、データをアプリケーションと同じサーバに配置し、また通常は 1 プロセスのみからアクセスするようにする必要があります (外部ファイルロックが非常に低速なため)。上記の問題は、MySQL サーバに低レベルの MyISAM コマンドを導入することで解消できます (必要に応じてパフォーマンスを改善する容易な手段の 1 つとなるのです)。データベースインタフェースを慎重に設計することで、この種の最適化を容易にサポートできる。
- 多くの場合、テキストファイルにアクセスするのと比較して、データベースからデータにアクセスしたほうが高速である。この理由は一般にテキストファイル (数値データ使用時) よりデータベースのほうがよりコンパクトで、必要なディスクアクセスが少ないことによる。また、テキストファイルを解析してレコードとカラムの境界を検索する必要がないため、コードも節約できる。

- レプリケーションはオペレーションによって、性能向上を図ります。負荷を分散させるため、クライアント修正をレプリケーションサーバに分布できる。バックアップを作成する間マスタの速度が低下するのを避けるため、スレーブサーバを作成することができる。5章レプリケーションを参照してください。
- `DELAY_KEY_WRITE=1` オプションで `MyISAM` テーブルを定義すると、ファイルが閉じられるまでディスクにログが記録されないためインデックス更新の速度が上がる。この欠点は、途中で `mysqld` の強制終了が発生した場合にテーブルに問題がないことを確認するため、`mysqld` を開始する前に、テーブルに対して `myisamchk` を実行するか、`--myisam-recover` でサーバを作動させる必要があるということである。キー情報は常にデータから生成可能であるため、`DELAY_KEY_WRITE` を使用しても何も消失はしない。

ロック関連の問題

MySQL のテーブルロック方法

MySQL は `MyISAM` と `MEMORY` テーブルにはテーブルレベルロックを使用し、`InnoDB` テーブルには行レベルロックを使用します。

ほとんどの場合、どのロックタイプがアプリケーションに適しているか推察することが可能ですが、一概にどのロックタイプが優れているかを判断するのは困難です。全てはアプリケーションに依存しており、かつアプリケーションの部分毎に異なるロック型があります。

行レベルロックで保存エンジンを使用するか判断する場合、ユーザはアプリケーションの役割と、使用されている `select` と `update` ステートメントを確認する必要があります。例えば、大抵の Web アプリケーションは多くの選択を実行し、相対的に削除はほとんど行わず、主にキー値にもとづいた更新を行い、かつ特定のテーブルに挿入します。ベース MySQL `MyISAM` セットアップはよくチューニングされています。

MySQL Enterprise

MySQL Network Monitoring and Advisory Service #####
#####<http://www-jp.mysql.com/products/enterprise/advisors.html>#####

MySQL のテーブルロックは、テーブルレベルロックを使用するストレージエンジンに対して、デッドロックフリーです。デッドロックは、クエリの開始時に、同時に必要とされる全てのロックを常に要求すること、そして同じ順序のテーブルを常にロックすることで回避されます。

MySQLでのテーブルロックメソッドは `WRITE` ロックを用いて以下のように機能します。

- テーブルにロックがない場合、`write` ロックをつけてください。
- もしくは、`write` ロックキューでロック要求を行ってください。

MySQLでのテーブルロックメソッドは `READ` ロックを用いて以下のように機能します。

- テーブルに `write` ロックがない場合、`read` ロックをつけてください。
- もしくは、`read` ロックキューでロック要求を行ってください。

ロックが開放されるとき、`write` ロックキューのスレッドに対してロックは有効であり、それから `read` ロックキューのスレッドに対しても有効です。これはテーブルの更新を頻繁に行う場合、`SELECT` ステートメントは更新が行われなくなるまで待機することを意味します。

`Table_locks_waited` および `Table_locks_immediate` ステータス変数をチェックすることでシステム上でテーブルロック競合を分析できます。

```
mysql> SHOW STATUS LIKE 'Table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 1151552 |
| Table_locks_waited | 15324 |
+-----+-----+
```

`MyISAM` テーブルが中心に空きブロックを持たない場合、行は必ずデータファイルの後部に挿入される。この場合、併発する `INSERT` および `SELECT` ステートメントをロックなしで `MyISAM` テーブルに対して自由に混合して使用できま

す。つまり、他のクライアントが読むと同時にMyISAM テーブルに行を挿入できます。テーブルの途中で行を更新もしくは削除した場合欠落が生じます。欠落がある場合、同時挿入はできませんが、全ての欠落が新しいデータで満たされた場合は、自動的に再使用可能となります。

この機能は`concurrent_inserts`システム変数によって変更されます。[同時挿入](#)を参照してください。

テーブルに対して多数の `INSERT` および `SELECT` 操作を行う必要がある場合、このような待機を回避するには、テンポラリテーブルに行を挿入し、一定の間隔でテンポラリテーブルからの行で実テーブルを更新します。これは以下のコードで実行できます。

```
mysql> LOCK TABLES real_table WRITE, insert_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM insert_table;
mysql> TRUNCATE TABLE insert_table;
mysql> UNLOCK TABLES;
```

InnoDBは行ロックを使用する。InnoDBでは、SQLステートメントのトランザクションの初めではなく、プロセス中に自動的に入手するため、デッドロックが可能です。

行レベルロックの利点

- 多数のスレッドで異なる行をアクセスする際に、ロックコンフリクトが少なくて済みます。
- ロールバックの変更が少なくて済みます。
- 1つの行を長時間ロックすることが可能です。

行レベルロックの欠点

- テーブルレベルロックよりもメモリを要します。
- テーブルの大部分で使われる際、より多くのロックが必要となるためテーブルレベルロックよりも処理速度が遅くなります。
- データの大部分に対して`GROUP BY`オペレーションを実行する場合、もしくは全テーブルを頻繁にスキャンする場合他のロックよりもはるかに遅いです。

以下の場合、行レベルロックに対してテーブルロックが優勢になります。

- テーブルのほとんどのステートメントは `read` です。
- 1つのキーリードで取得される1つの行に対して、`write` が更新もしくは削除される場合、`read` と `write` が混合となります。

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- `SELECT` が同時`INSERT`ステートメントとごく少数の`UPDATE`もしくは`DELETE`ステートメントと混合されます。
- `writer` を使用しない、全てのテーブルの`GROUP BY`オペレーションや多くのスキャン。

高レベルロックで、異なるタイプのロックをサポートすることによって、より簡単にアプリケーションのチューニングが行えます。というのも、ロックオーバーヘッドは低レベルロックに対して少ないからです。

行レベルロック以外のオプション

- バージョニング (同時挿入でMySQLに使用されるような) : 同時に 1 writer と多数の reader が存在する場合です。これはアクセス開始時に応じたデータに対して、データベースやテーブルは異なるビューをサポートします。これに対する他の共通用語は「タイムトラベル」、「writeのコピー」または「コピーオンデマンドです。」
- 多くの場合コピーオンデマンドは行レベルロックよりも優勢です。しかし、最悪の場合、通常のロックを使用するよりも、メモリ容量が多く必要となり得ます。
- 行レベルロックを使用する代わりに、MySQLでは`GET_LOCK()`や`RELEASE_LOCK()`といったアプリケーションレベルロックを利用できます。これらはアドバイザリロックで、特に問題のないアプリケーションでのみ機能します。(詳しくは[その他の関数](#)をご確認ください。)

テーブルロック関連の問題

高速なロックスピードを実現するため、MySQL [InnoDB](#)以外の全てのテーブルロックを全てのストレージエンジンに使用します。

[InnoDB](#)テーブルの場合は、MySQL で[LOCK TABLES](#)によって明示的テーブルをロックした場合のみテーブルロックが使用されます。[InnoDB](#)は自動行レベルロックを使用してトランザクションの独立を確実にするため、これらのテーブル型には、[LOCK TABLES](#)をまったく使用しないように推奨します。

大きいテーブルには、ほとんどのアプリケーションでテーブルロックの方が行ロックより適切ですが、落とし穴もあります。

- テーブルロックにより、同時に多数のスレッドがテーブルからの読み取りを行うことができますが、あるスレッドがテーブルへの書き込みを行うときは、まず排他処理をする必要があります。更新時は、特定のテーブルにアクセスしようとする他のすべてのスレッドが、更新の準備ができるまで待機します。
- 一般にテーブルの更新はテーブル検索より重要だと見なされるため、テーブルを更新するステートメントは優先度が高くなります。これにより、更新では特定のテーブルに対して重い[SELECT](#)アクティビティが使用されるため、更新が「資源枯渇」にさらされないことが確実にになります。
- テーブルロックによって、ディスク容量がいっぱいのため、スレッドが待機するような問題が引き起こされます。またスレッドが処理される前に開きスペースを利用可能にする必要があります。この場合、問題テーブルにアクセスを求める全てのスレッドも、より多くのディスクスペースが利用可能になるまで待機中となります。

ただし、テーブルロックは以下のシナリオには適していません。

- クライアントが実行に長時間かかる [SELECT](#)を使用します。
- その後、別のクライアントが使用テーブルに対して [UPDATE](#)を使用する。このクライアントは [SELECT](#)が完了するまで待機が必要になる。
- 別のクライアントが同一テーブルに対してさらに [SELECT](#)ステートメントを使用します。[UPDATE](#)は[SELECT](#)より優先度が高いため、この [SELECT](#)は [UPDATE](#)が完了するまで待機が必要になります。また、最初の [SELECT](#)の完了を待つ必要もあります。

以下のアイテムはテーブルロックによる競合を軽減または回避する方法を記述します。

- [SELECT](#)ステートメントの実行の高速化を試行する。これにはサマリテーブルの作成が必要な場合もあります。
- [--low-priority-updates](#)のオプションで [mysqld](#)を開始する。これは、テーブルを更新 (変更)するすべてのステートメントの優先度を [SELECT](#)ステートメントの優先度より低くします。この場合、前シナリオの2つ目の[SELECT](#)ステートメントは[UPDATE](#)ステートメントの前に実行され、一番目の[SELECT](#)が完了するまで待機する必要はありません。
- [SET LOW_PRIORITY_UPDATES=1](#)ステートメントを使用すると、特定の接続からの更新すべてが低い優先度で実行されるように指定できます。[SET 構文](#)を参照してください。
- [LOW_PRIORITY](#)属性を使用して、特定の[INSERT](#)、[UPDATE](#)、または [DELETE](#)ステートメントの優先度を低く設定できます。
- [HIGH_PRIORITY](#)性を使用すると、特定の [SELECT](#) の重要度を高く指定できます。[SELECT 構文](#)を参照してください。
- [max_write_lock_count](#)システム変数の値を低くして[mysqld](#)を開始し、MySQLに全ての[SELECT](#)ステートメント (特定数をテーブルに挿入した後に待機しているステートメント) の優先度を一時的に引き上げさせます。これは、一定数の[WRITE](#)ロックの後に[READ](#)ロックを設定します。
- [SELECT](#)と結合した[INSERT](#) に問題がある場合は、[SELECT](#)ステートメントと[INSERT](#) ステートメントの同時サポートが可能になるため、新規の [MyISAM](#)テーブルを使用するように切り替えます。(詳しくは[同時挿入](#)をご確認ください。)
- 同じテーブル上で insertとdeleteステートメントの混在が多い場合、[INSERT DELAYED](#)が非常に役立つ可能性があります。[INSERT DELAYED 構文](#)を参照してください。

- `SELECT`と`DELETE`ステートメントに問題がある場合、`DELETE`に`LIMIT`オプションを使用すると解決できる場合があります。`DELETE` 構文を参照してください。
- `SELECT`ステートメントで`SQL_BUFFER_RESULT`を使用すると、テーブルロックの持続を短縮することができます。`SELECT` 構文を参照してください。
- 単一エリを使用するために、`mysys/thr_lock.c`でロックコードを変更できることもあります。この場合、writeロックとreadロックは同等の優先度をもち、いくつかのアプリケーションにとって役立つものとなります。

以下はMySQLにおけるテーブルロックについてのヒントです。

- 更新と、同じテーブルの多くの行を調べるセレクトを混合しない限り、同時ユーザは問題を引き起こしません。
- シングルロックをしようしてのアップデイトはロックなしでのアップデイトよりも速いため、`LOCK TABLES`を使用して速度を上げることができます。テーブルのコンテンツを別々のテーブルに分けるのも効果的です。
- MySQLでテーブルロック関連の問題が生じたとき、テーブルをInnoDBに変換することで性能向上を図ることができます。`InnoDB ストレージ エンジン`を参照してください。

MySQL Enterprise

```
#####MySQL Network Monitoring and Advisory Service #####
#####http://www-jp.mysql.com/products/enterprise/advisors.html#####
```

同時挿入

MyISAM テーブルでは、テーブルに削除された行がない場合、`SELECT`の実行と同時に行を挿入できることに注意してください。

上記がデフォルト作動で、`concurrent_inserts`システム変数で制御できます。1に設定された場合、削除された行を含むMyISAMテーブルで同時挿入が起こります。2に設定された場合、削除された行があっても、全ての新しい行がテーブル後部に付加されることで同時挿入が強制的に行われます。??? [167]を参照してください。

同時挿入が使用できる状況下では、`INSERT`ステートメントの`DELAYED`修飾子を使用する必要はほとんどありません。`INSERT DELAYED` 構文を参照してください。

バイナリログを使用している場合、同時挿入は`CREATE ... SELECT`もしくは`INSERT ... SELECT`ステートメントの一般的な挿入に変換されます。このことで、バックアップ演算中にログを適用することでテーブルの正確なコピーを再作成することができます。

`LOAD DATA INFILE`で同時挿入（つまり、途中で空きブロックを含まない）の条件を満たすMyISAMテーブルを使用して`CONCURRENT`を指定する場合、他のスレッドは`LOAD DATA`実行中にテーブルからデータを取得することができます。このオプションを使用することで、たとえ他のスレッドが同時にテーブルを使用しているとしても、`LOAD DATA`のパフォーマンスに幾分かの影響が与えられます。

データベース構造の最適化

設計上の選択

MySQLはローデータとインデックスデータを別のファイルに格納します。その他のデータベースの多く（ほとんど）は、ローデータとインデックスデータが同じファイルに混在しています。現在の非常に多くのシステムでMySQLの選択のほうが優れていると確信しています。

行データの格納方法には、各カラムの情報を独立した領域に格納する方法もあります（例: SDBM、Focus など）。これは、複数のカラムにアクセスするすべてのクエリでパフォーマンスに影響を及ぼします。パフォーマンスは複数のコラムへのアクセスを開始するとただちに低速化するため、このようなモデルは汎用データベースには適さないと確信しています。

一般的にインデックスとデータと一緒に格納されている場合も多くあります（Oracle、Sybase などの場合）。この場合は、レコード情報をインデックスのリーフページで検索します。このレイアウトで優れている点は、多くの場合インデックスのキャッシュ方法次第でディスクの読み取りを節約できることにあります。このレイアウトの欠点は以下のとおりです。

- データの取得時にインデックス全体を読み取る必要があるため、テーブルスキャンの速度が大幅に下がる。
- クエリでデータを取り出す際にインデックステーブルのみの使用ができない。
- ノードからインデックスを複製する必要があるため（レコードはノードに格納できないことによる）、大量の領域が消費される。
- 削除があるとテーブルの速度が次第に低下する（通常、削除ではノードのインデックスが更新されないため）。
- インデックスデータのみのキャッシュが困難である。

データの小型化

最も基本的な最適化の1つにデータ（およびインデックス）が占めるディスク領域を可能な限り少なくすることがあります。これで、ディスクの読み取りが高速化し、使用メモリも一般に減少するため、大幅な改善が図れます。カラムが小さければインデックス作成で消費されるリソースも少なくなります。

MySQL では多様なテーブル型とレコード形式がサポートされます。適切なテーブル形式を選択することで、パフォーマンスを大幅に改善できます。[13章ストレージエンジンとテーブルタイプ](#)を参照してください。

ここで紹介する技法を使用すると、テーブルのパフォーマンス改善とストレージ領域の最小化を図ることができます。

- できる限り効率性の高い（最小）の型を使用する。MySQL にはディスク領域とメモリを節約できる専用の型がある。可能な場合は、小さなテーブルの取得には小さな整数型を使用する。たとえば、`INT`より、25%スペース使用量の少ない`MEDIUMINT`のほうが適している場合もしばしばあります。
- できる限り、カラムに `NOT NULL`を宣言します。これですべてが高速化され、1カラム当たり1ビットを節約できます。アプリケーションで実際に `NULL`が必要な場合は、必ず使用する必要があるため、注意が必要です。デフォルトですべてのカラムにこれを設定することは避けます。
- `MyISAM`テーブルでは、可変長カラム(`VARCHAR`、`TEXT`、あるいは`BLOB`など)がまったくない場合は固定長レコード形式を使用します。これで速度が上りますが、領域の消費も増えます。詳しくは[MyISAM テーブルストレージフォーマット](#)を参照してください。たとえば`CREATE TABLE`オプション`ROW_FORMAT=FIXED`が使用された`VARCHAR`カラムがあっても、固定長行が必要な場合のヒントが得られます。
- `InnoDB`はコンパクトストレージフォーマットを使用します。MySQL 5.0.3以前のバージョンでは、`InnoDB`行は固定サイズカラムにたいしても、カラム数やカラム長さといった冗長な情報を含みます。デフォルトでは、コンパクトフォーマット(`ROW_FORMAT=COMPACT`)でテーブルが作成されます。MySQLの旧バージョンにダウングレードをしたい場合、`ROW_FORMAT=REDUNDANT`で古いフォーマットを要求できます。

コンパクト`InnoDB`フォーマットはUTF-8データを含む`CHAR`カラムがどのように格納されるかも変更します。最長UTF-8エンコードキャラクタが3バイトであることを前提に、`ROW_FORMAT=REDUNDANT`では、UTF-8 `CHAR(N)`は $3 \times N$ バイトを使用します。多くの言語は主にシングルバイトUTF-8キャラクタを用いてかかれるため、固定保存長はスペースを無駄に使用します。`ROW_FORMAT=COMPACT`フォーマットでは、`InnoDB`は`N`から $3 \times N$ バイトのストレージ可変容量をこれらカラムに割り当てます（必要であればトレールスペースを取り除いて行います）。最小のストレージ長は、ある場合に適切な更新が行われるように、`N`バイトとして保持されます。

- テーブルのプライマリインデックスを可能な限り短くします。これで、レコードの識別が容易になり効率化が図れます。
- インデックスの作成は必要なものだけに限定します。インデックスは取り出しに優れていますが、高速保存が必要な場合は適されません。カラムの組み合わせを使用してテーブルを検索し、テーブルにアクセスする場合はほとんどであれば、インデックスを作成します。インデックスの最初の部分は、最も使用頻度の高いカラムにする必要があります。テーブルの検索時に、常に常に多数のカラムを使用する場合は、より重複しているカラムを先に使用するとインデックスの圧縮を改善できます。
- 文字列の最初の数文字に、一意のプリフィックスがあるカラムが多い場合は、このプリフィックスのみをインデックス化したほうがよりよくなります。MySQL はカラムの最も左側の部分インデックス作成をサポートします（[CREATE INDEX 構文](#)を参照してください）。短いインデックスの速度が速い理由は、占有ディスク領域が小さいことだけでなく、インデックスキャッシュでのヒットが多くなり、所要ディスクシークが少なくなることにもよります。[サーバパラメータのチューニング](#)を参照してください。

- ・状況によっては、スキヤンの頻度が高いテーブルを2つに分割したほうが有利な場合もあります。これは特に、動的テーブルで、テーブルスキヤンの際に対応する行の検索に小さな静的テーブルの使用が可能である場合に当てはまります。

カラムインデックス

MySQL の全てのカラム型にはインデックスを張ることができます。SELECT操作のパフォーマンスの改善には、対応するカラムにインデックスを使用することが最善の方法です。

テーブルあたりの最大インデックス数とインデックスの最大長は、ストレージエンジンごとに定義されます。13章 [ストレージエンジンとテーブルタイプ](#) を参照してください。ストレージエンジンのすべてで、1 テーブルあたり 16 以上のインデックスと 256 バイト以上のインデックス長がサポートされます。大抵のストレージエンジンでは、インデックス最大長がより高く設定されています。

インデックス指定で `col_name(N)` 構文を用いて、文字列カラムの最初の `N` キャラクタのみを使用したインデックスを作成できます。このようにカラム値のプレフィックスのみをインデックス化すると、インデックスファイルをかなり小さくすることができます。BLOB もしくは TEXT カラムにインデックスを張る場合、インデックスに対してプレフィックス長を指定しなければなりません。例：

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

プレフィックスは、最大1000バイト長(InnoDBテーブルに対しては767バイト)まで可能です。プレフィックスの最大長はバイトで評価されます。一方、CREATE TABLE ステートメント内のプレフィックス長は文字数として解釈されます。プレフィックス長を、マルチバイトキャラクタセットを使用するカラムに対して指定するときこのことを考慮に入れなければいけません。

FULLTEXT インデックスの作成も可能です。これらはフルテキスト検索に使用されます。FULLTEXT インデックスをサポートするのは MyISAM ストレージエンジンだけで、CHAR、VARCHAR、そして TEXT カラムについてのみサポートされます。インデックスの作成は常にカラム全体を対象として、先頭部分 (プリフィックス) のインデックス化は行われません。詳細については、[全文検索関数](#) をご参照ください。

空間データ型上でインデックスの作成もできます。現在、MyISAM のみが空間型 R-tree インデックスをサポートしています。他のストレージエンジンは空間型のインデックス化に B-trees を使用します。(ただし ARCHIVE や NDBCLUSTER といった空間型インデックスをサポートしないものを除きます)

MEMORY ストレージエンジンはデフォルトで HASH インデックスを使用しますが、BTREE インデックスもサポートしています。

複合インデックス

MySQL では複数のカラムに対するインデックスを作成できます。インデックスは最大 16 カラムで構成できます。特定のデータ型に関しては、カラムのプリフィックスをインデックス上で検索することができます。([カラムインデックス](#) を参照してください)

複数カラムのインデックス (複合インデックス) は、インデックス化されたカラムの値を連結することによって生成された値が含まれ、ソート化された配列と見なすことができます。

MySQL では、WHERE 節内でインデックスの第 1 カラムを指定する場合、他のカラムの値を指定しなくても、クエリが高速化できるように複合インデックスが使用されます。

次のようなテーブルが定義されているとします。

```
CREATE TABLE test (
  id      INT NOT NULL,
  last_name CHAR(30) NOT NULL,
  first_name CHAR(30) NOT NULL,
  PRIMARY KEY (id),
  INDEX name (last_name,first_name)
);
```

ここで、インデックス `name` は、`last_name` と `first_name` に対するインデックスです。このインデックスは、`last_name` の範囲、または `last_name` と `first_name` の両方の範囲の値を指定するクエリに使用できます。したがって、`name` インデックスは次のようなクエリに使用されます。

```
SELECT * FROM test WHERE last_name='Widenius';

SELECT * FROM test
  WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius'
    AND (first_name='Michael' OR first_name='Monty');

SELECT * FROM test
  WHERE last_name='Widenius'
    AND first_name >='M' AND first_name < 'N';
```

しかし、次のクエリには [name](#) インデックスが使用されません。

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

MySQL でインデックスを使用してクエリパフォーマンスを改善する方法の詳細については、[MySQLにおけるインデックスの使用](#)を参照してください。

MySQLにおけるインデックスの使用

インデックスは通常迅速に特定のカラム値の行を検索するのに使用されます。インデックスがない場合、関連する行を検索する場合MySQLは最初の行から始め全テーブルを読まなければなりません。テーブルが大きいほど、コストがかかります。検索しているカラムのインデックスをテーブルが含んでいる場合、MySQLは全てのデータを検索せずに、データファイルの途中でシークポジションを迅速に決定します。テーブルに1000行ある場合、連続して読む場合よりも少なくとも100倍の速度で処理が行われます。行のほとんどにアクセスしなければならない場合、連続して読むほうがより高速です。というのも、これはディスクシークの最小化が行われるためです。

ほとんどのMySQLインデックスは(PRIMARY KEY、UNIQUE、INDEX、そしてFULLTEXT)はB-treesに保存されています。R-treesを使用する空間データ型インデックスは例外で、MEMORYテーブルでもハッシュインデックスがサポートされています。

文字列のプリフィックスとエンドスペースは自動的に圧縮されます。[CREATE INDEX 構文](#)を参照してください。

一般的に、インデックスは以下のように使用されます。ハッシュインデックスの特徴は(MEMORYテーブルで使われる)このセクションの終わりに説明されています。

MySQLはこれらの演算についてインデックスを使用します。

- [WHERE](#)節にマッチする行を迅速に検索する場合。
- 行を考慮に入れない場合。複数のインデックス間を選択できる場合、MySQLは通常最小行数を検索するインデックスを使用します。
- 結合実行時に、他のテーブルから行を取得する場合。
- 特定のインデックス化されたカラムkey_colに対して、MIN()あるいはMAX()値を検索する場合。これはインデックス内でkey_colより前に発生する全てのキーパーツで、WHERE key_part_N = constantが使用されているかをチェックするプリプロセッサによって最適化されます。この場合MySQLはMIN()もしくはMAX()表現それぞれに対して単一キールックアップを行い、定数で置き換えます。全ての表現が定数で置き換えられた場合、クエリは一度に返されます。例：

```
SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
```

- 使用可能キーの最も左側のプリフィックスでグループ化やソート化が行われる際、テーブルをソートもしくはグループ化する場合に使用します。(例えば ORDER BY key_part1、key_part2)全てのキー部分にDESCが後続する場合、キーは逆の順序で読まれます。[ORDER BY最適化](#)を参照してください。
- データ行を参照せず値を取得するためにクエリが最適化される場合もあります。クエリがあるキーの最も左側のプリフィックスを形成し、かつテーブル内で数値のみのカラムを使用する場合、選択された値は高速化を図るため、インデックスツリーから取得されることもあります。

```
SELECT key_part3 FROM tbl_name
WHERE key_part1=1
```

例えば次のSELECTステートメントを発行したとします。

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

col1とcol2上で複合カラムインデックスが存在する場合、適当な行は直接取得されます。col1とcol2に別々のシングルカラムインデックスが存在する場合、オプティマイザはどのインデックスがより少ない行を検索するかを決定することで、最も制限力のあるインデックスを検索します。また、そのインデックスを使用して行を取得します。

テーブルに複合インデックスがある場合、オプティマイザではインデックスの左端の先頭部分のいずれかをレコードの検索に使用できます。たとえば、(col1, col2, col3)に3カラムのインデックスがある場合、(col1)、(col1, col2)そして(col1, col2, col3)に対して、インデックスの検索機能を使用できます。

カラムがインデックスの左端の先頭部分を構成していない場合、MySQLでは、部分インデックスを使用できなくなります。以下のSELECTステートメントがあります。

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

インデックスが(col1, col2, col3)に存在する場合、最初のクエリだけがインデックスを使用できます。3つめと4つめのクエリには、インデックス化したカラムが必要ですが、(col2)と(col2, col3)は(col1, col2, col3)の左端のプリフィックスではありません。

=, >, >=, <, <=あるいはBETWEEN演算子を使用する表現のカラム比較に、B-treeインデックスが使用可能です。LIKEがワイルドカードキャラクタで始まらない定数文字列の場合、インデックスはLIKE比較にも使用できます。例えば、以下のSELECTステートメントはインデックスを使用します。

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

最初のステートメントでは、'Patrick' <= key_col < 'Patricl'を含む行のみ考慮されます。2つ目のステートメントでは、'Pat' <= key_col < 'Pau'を含む行のみ考慮されます。

以下のSELECTステートメントはインデックスを使用しません。

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

最初のステートメントでは、LIKE値はワイルドカードキャラクタで始まります。2つ目のステートメントでは、LIKE値は定数ではありません。

もし... LIKE '%文字列%'そして文字列は3文字より長い場合、MySQLはTurbo Boyer-Moore アルゴリズムを使用して、文字列のパターンを初期化してから、このパターンを使用して検索をすばやく実行します。

col_name IS NULLを使用した検索では、col_nameにインデックスが張られている場合にインデックスが使用されます。

WHERE節内の全てのANDにかかっていないインデックスは、クエリの最適化に使用されません。言い換えると、インデックスの使用を可能にするには、インデックスの先頭部分がすべてのANDグループで使用されている必要があります。

次のWHERE節ではインデックスが使用されます。

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
/* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
/* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
/* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

次のWHERE節ではインデックスが使用されません。

```

/* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

/* Index is not used in both parts of the WHERE clause */
... WHERE index=1 OR A=10

/* No index spans all rows */
... WHERE index_part1=1 OR index_part2=10

```

MySQL では利用可能な場合でもインデックスが使用されないことがあることに注意してください。この一例として、インデックスの使用によって、MySQL がテーブルの 30% を超えるレコードにアクセスする必要がある場合が挙げられます（この場合は、必要なシークが大幅に減少するため、テーブルスキャンのほうが高速になる可能性が高くなります）。ただしこのクエリに、レコードの一部のみを取り出す **LIMIT** が使用されている場合、結果で返される少数のレコードを迅速に検索できるため、MySQL はインデックスを使用します。

ハッシュインデックスは先ほど挙げたインデックスとは特徴が異なります。

- **=** or **<=>** 演算子を使用する等価比較にのみ使用されます。（ただし非常に 高速です）**<** のように値の範囲を検索する比較演算子には使用されません。
- オプティマイザは **ORDER BY** オペレーション速度を上げるためにハッシュインデックスを使用することはできません。（このようなインデックスは順序どおりに次のエントリを検索することはできません）
- MySQL は 2 つの値の間にある行の数を判別することはできません。（どのインデックスを使用するかを決定する上、範囲オプティマイザによって使用されます）**MyISAM** テーブルをハッシュインデックスを含む **MEMORY** テーブルに変換した場合、これは一部のクエリに影響を与えるかもしれません。
- 行の検索には自然キーのみが使用できます。（B-tree インデックスでは、どのキーの左端の先頭部を使用しても行を検索できます。）

MySQL Enterprise

— ##### MySQL Network Monitoring
and Advisory Service ##### [http://www-jp.mysql.com/products/enterprise/
advisors.html](http://www-jp.mysql.com/products/enterprise/advisors.html) #####

MyISAMキーキャッシュ

ディスク I/O を最小化するために、**MyISAM** ストレージエンジンは多数のデータベースマネジメントシステムで使用される戦略を利用します。メモリ内で最も頻繁にアクセスされるテーブルブロックを保持するために、キャッシュメカニズムが使用されます。

- インデックスブロックには、key cache（あるいは key buffer）という特別な構造が保持されています。その構造には最も頻繁に使用されるインデックスがおかれた多数のブロックバッファが含まれます。
- データブロックに対して、MySQL は特別なキャッシュを使用しません。代わりに、ネイティブオペレーティングシステムファイルシステムキャッシュに依存します。

このセクションでは最初に **MyISAM** キーキャッシュの基本的な演算について説明しています。キーキャッシュパフォーマンスを向上させる特徴や、キャッシュオペレーションをよりよくコントロールできる特徴について、説明されています。

- 複合スレッドはキャッシュを同時にアクセスできます。
- 複合キーキャッシュのセットアップおよび特定のキャッシュに対するテーブルインデックスの割り当てが可能です。

キーキャッシュのサイズを制限するには、**key_buffer_size** システム変数を使用します。この変数がゼロにセットされた場合、利用できるキーキャッシュはありません。**key_buffer_size** 値が小さすぎてブロックバッファの最小値が割り当てられない場合、キーキャッシュも使用できません（8）。

MySQL Enterprise

key_buffer_size ##### MySQL Network Monitoring and Advisory Service #####
<http://www-jp.mysql.com/products/enterprise/advisors.html>

キーキャッシュが作動していない場合、インデックスファイルはオペレーティングシステムによるネイティブファイルシステムバッファのみを使用してアクセスされます。(つまり、テーブルインデックスブロックはテーブルデータブロック利用と同様の方法でアクセスされます)

インデックスブロックはMyISAMインデックスファイルにアクセスする隣接ユニットです。通常、インデックスブロックのサイズは、インデックスB-treeのノードサイズと等価です。(インデックスはB-tree データ構造を使用してディスク上で表現されます。ツリーの元にあるノードはリーフノードです。リーフノードの上層にあるノードは非リーフノードです。)

キーキャッシュ構造内のブロックバッファは全て同サイズです。このサイズは、テーブルインデックスブロックサイズと比べて、等しい/大きい/小さい場合があります。通常これら二つの値のうち一方は、他方の複合となっています。

あるテーブルインデックスブロックのデータアクセスが必要な場合、サーバはまずキーキャッシュのどれかのブロックバッファ内で利用可能かどうかをまずチェックします。可能である場合、サーバはディスク上ではなく、キーキャッシュのデータにアクセスします。つまり、ディスクからではなく、キャッシュから読むかもしくはキャッシュに書きこみます。でなければ、異なるテーブルインデックスブロックを含むキャッシュブロックバッファを選択し、そのデータを要求されたテーブルインデックスブロックのコピーと置き換えます。新しいインデックスブロックがキャッシュ内におかれるとすぐ、インデックスデータはアクセス可能となります。

たまたま置き換えのために選択されたブロックが改良されていた場合、ブロックは「汚染されたモノ」とみなされます。この場合、置き換えられる前に、内容は元のテーブルインデックスにフラッシュされます。

通常サーバはLRU (Least Recently Used)戦略に従います。置換のためブロックを選択した場合、最近使用した中で最も初期に使用されたインデックスブロックを選択します。この選択を簡単にするために、キーキャッシュモジュールは、使用された全てのブロックの特別なキュー(LRU chain) を保持します。ブロックがアクセスされた場合、キューの最後尾に置かれます。ブロックを置換する必要がある場合、キューの最前部にあるブロックは、最近使用した中で最も初期に使用されたものであり、かつ最初の除去対象となります。

共有キーキャッシュアクセス

スレッドはキーキャッシュバッファに同時にアクセス可能であり、以下の条件に従います。

- 更新されていないバッファは複合スレッドによってアクセス可能です。
- 更新中のバッファは更新が完了するまでスレッドを待機させます。
- 複合スレッドは、お互いに衝突しない限り、キャッシュブロック置換を引き起こす要求をします (つまり、異なるインデックスブロックを必要とする限り、異なるキャッシュブロックを置換します。)

キーキャッシュの共有アクセスによって、サーバ処理能力が大幅に向上します。

複合キーキャッシュ

キーキャッシュへの共有アクセスはパフォーマンスを向上させますが、スレッド間の競合を完全には削除しません。キーキャッシュバッファへアクセスするためのコントロール構造をめぐって競合が行われます。キーキャッシュアクセス競合をさらに軽減するために、MySQLは複合キーキャッシュも提供しています。この特性によって、異なるキーキャッシュに対して各テーブルインデックスの割り当てが可能となります。

複合キーキャッシュがある場合、サーバは既存のMyISAMテーブルに対してクエリ処理を行う際に、どのキャッシュを使用すべきか知らされていなければなりません。デフォルトでは、全てのMyISAMテーブルインデックスはデフォルトキーキャッシュ内にキャッシュされます。テーブルインデックスを特定のキーキャッシュに割り当てるには、**CACHE INDEX**ステートメントを使用してください。(CACHE INDEX 構文を参照してください。)例えば、以下のステートメントはt1、t2、そしてt3テーブルから、hot_cacheと名づけられたキーキャッシュにインデックスを割り当てます。

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | assign_to_keycache | status | OK |
| test.t2 | assign_to_keycache | status | OK |
| test.t3 | assign_to_keycache | status | OK |
+-----+-----+-----+-----+
```


CACHE INDEXステートメント内の参照キーキャッシュは、SET GLOBALパラメータ設定ステートメントを用いてサイズを設定するか、もしくはサーバスタートアップオプションを使用して作成できます。例：

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

キーキャッシュを破壊するにはサイズをゼロに設定してください。

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

デフォルトキーキャッシュを破壊することはできない点に注意してください。この破壊に関してはいずれの試みも無視されます。

```
mysql> SET GLOBAL key_buffer_size = 0;
```

```
mysql> SHOW VARIABLES LIKE 'key_buffer_size';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 8384512 |
+-----+-----+
```

キーキャッシュ変数は名前と部位のある構造システム変数です。keycache1.key_buffer_sizeにとって、keycache1はキャッシュ変数名であり、key_buffer_sizeはキャッシュ部位です。構造キーキャッシュシステム変数に対して使用される構文の詳細については、[構造化システム変数](#)を参照してください。

デフォルトではテーブルインデックスは、サーバ起動時に、メイン（デフォルト）キーキャッシュに割り当てられます。キーキャッシュが破壊された場合、それに割り当てられた全てのインデックスはデフォルトキーキャッシュに再度割り当てられます。

サーバが込んでいる場合、以下の3つのキーキャッシュを使用する戦略をお勧めします。

- 全てのキーキャッシュに割り当てられたスペースの20%以上を占める「hot」キーキャッシュ。検索に使用される頻度が高く、かつ更新されていないテーブルに対してはこれを使用してください。
- 全てのキーキャッシュに割り当てられたスペースの20%以上を占める「cold」キーキャッシュ。このキャッシュは、テンポラリテーブルのような中位の集中的に改良されたテーブルに使用してください。
- キーキャッシュスペースの60%以上を占める「warm」キーキャッシュ。これをデフォルトで全ての他のテーブルに使用されるように、このキーキャッシュをデフォルト設定してください。

上記3つのキーキャッシュを使用する理由のひとつの利点は、1つのキーキャッシュ構造は他二つのキーキャッシュへのアクセスを阻害しない点です。あるキャッシュに割り当てられたテーブルにアクセスするステートメントは、他のキャッシュに割り当てられたテーブルにアクセスするステートメントとは競合しません。パフォーマンス向上には他の理由もあります。

- hotキャッシュはクエリの取得にのみ使用されるため、内容は決して改良されません。結果、インデックスブロックがディスクから引き抜かれなければならない場合でも、置換のために選択されたキャッシュブロック内容は最初にフラッシュされる必要はありません。
- キャッシュに割り当てられたインデックスにとって、インデックススキャンを必要とするクエリがなければ、インデックスB - treeの非リーフノードに一致するインデックスブロックがキャッシュに残っている可能性が高くなります。
- テンポラリテーブルに対する最も頻度が高い更新オペレーションは、更新ノードがキャッシュ内にあり、最初にディスクから読まれる必要がない場合、処理速度がはるかに向上します。テンポラリテーブルのインデックスサイズがcoldキーキャッシュのサイズと同等の場合、更新ノードがキャッシュ内にある可能性が高くなります。

CACHE INDEXはテーブルとキーキャッシュの関連性を設けますが、サーバが再起動するたびにその関連性は失われます。サーバが再起動するたびに関連性を有効にしたい場合、オプションファイルを使用することで実行できます。キーキャッシュをコンフィギュアする変数設定と、CACHE INDEXステートメントを実行するファイルに名前をつけるinit-fileオプションを含んでください。例：

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
```



```
init_file=/path/to/data-directory/mysql_init.sql
```

MySQL Enterprise

```
my.cnf/my.ini #####MySQL Network Monitoring and Advisory
Service##### http://www.jp.mysql.com/products/
enterprise/advisors.html#####
```

サーバが再起動するたびにmysql_init.sqlでのステートメントは実行されている。ファイルはラインごとの1つのsqlのステートメントを含むべきである。次の例はhot_cache と cold_cacheに複数のテーブルをそれぞれ割り当てる。

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

ミッドポイント挿入戦略

デフォルトでは、キーキャッシュ管理システムは除去されるキーキャッシュブロックの選択に、LRU戦略が用いられますが、ミッドポイント挿入戦略と呼ばれるより複雑なメソッドもサポートします。

ミッドポイント挿入戦略を使用している間は、LRU チェーンは2つに分かれます。hotサブチェーンとwarmサブチェーン。2部位間の分け目は固定されていませんが、warm部位が「短すぎない」ように、最低でもkey_cache_division_limit%のキーキャッシュブロックを含むようにします。key_cache_division_limitは構成されたキーキャッシュ変数の構成要素であるため、その値はキャッシュ毎に設定可能なパラメータとなります。

インデックスブロックがテーブルからキーキャッシュに読まれる際、warmサブチェーンの後尾に置かれます。ある特定のヒット数後(ブロックへのアクセス後)、hotサブチェーンに昇格されます。現在では、ブロックを昇格させるために必要なヒット数は(3) 全てのインデックスブロックで同じです。

サブチェーンに昇格されたブロックはチェーンの後部に置かれます。それからブロックはこのsubチェーン内をめぐります。もしブロックが十分長い時間subチェーンの前部にとどまった場合、warmチェーンに降格されます。この時間はキーキャッシュのkey_cache_age_threshold構成要素値によって決定されます。

境界値は、Nブロックを含むキーキャッシュにとって、最後の $N \times \text{key\_cache\_age\_threshold} / 100$ ヒットでアクセスされなかったhotサブチェーン前部のブロックは、warmサブチェーンの前部に移動されます。そこから除去の第一候補となります。というのも、置換ブロックは常にwarmサブチェーンの前部から取得されるからです。

ミッドポイント挿入戦略によってキャッシュ内に常に高価値なブロックを保持することができます。プレーンLRU 戦略の使用を好む場合は、key_cache_division_limit値をデフォルトである100に設定しておいてください。

ミッドポイント挿入戦略によって、インデックススキャンで必要とされるクエリの実行が、高価値なハイレベルB-tree ノードと一致するインデックスブロックを、効果的にキャッシュから押し出される際のパフォーマンスを向上させます。これを回避するには、100以下に設定されたkey_cache_division_limitを用いた、ミッドポイント挿入戦略を使用しなければなりません。この結果、ノードに頻繁にヒットする変数は、インデックススキャンオペレーション中もhotサブチェーンに保存されます。

インデックスプレロード

全てのインデックスのブロックを保持するためのキーキャッシュ内に十分なブロックがある場合、もしくは少なくとも非リーフノードと一致するブロックがある場合、使用する前に、インデックスブロックでキーキャッシュをプレロードするのは理にかなっています。プレロードによって、テーブルインデックスブロックを最も効果的な方法でキーキャッシュバッファ内におくことができます。ディスクから連続してインデックスブロックを読む方法。

プレロードなしでは、ブロックは、クエリの必要に応じてキーキャッシュ内におかれます。ブロックはキャッシュ内にとどまりますが、バッファは足りているため、ディスクからランダムかつ不連続に取得されます。

インデックスをキャッシュにプレロードするにはLOAD INDEX INTO CACHEステートメントを使用してください。例えば、以下のステートメントはt1およびt2テーブルのインデックスノード (インデックスブロック) をプレロードします。

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
```

Table	Op	Msg_type	Msg_text
test.t1	preload_keys	status	OK
test.t2	preload_keys	status	OK

IGNORE LEAVES モディファイヤはインデックスの非リーフノードに対してはブロックのみがプレロードされます。したがって、表示されたステートメントは **t1** から全てのインデックスブロックをプレロードしますが、**t2** からは非リーフノードに対するブロックのみとなります。

インデックスが **CACHE INDEX** ステートメントを用いてキーキャッシュに割り当てられた場合、プレロードはインデックスブロックをキャッシュ内に置きます。もしくは、インデックスはデフォルトキーキャッシュにロードされません。

キーキャッシュブロックサイズ

key_cache_block_size 変数を使用して、個々のキーキャッシュのブロックバッファサイズを指定することができます。これによって、インデックスファイルに対して、I/O オペレーションパフォーマンスのチューニングが許可されます。

I/O オペレーションに対する最適なパフォーマンスは、read バッファサイズがネイティブオペレーティングシステム I/O バッファサイズと同等の場合に向上します。しかし、キーノードサイズを I/O バッファサイズと等しく設定しても、最適な全体的パフォーマンスを常に保証するわけではありません。ビッグリーフノードを読む場合、サーバは多数の不要なデータを吸収し、実質的に他のリーフノードが読まれるのを阻止します。

現在、テーブル内でインデックスブロックのサイズを変更することはできません。このサイズは、**MYI** インデックスファイルが作成される際に、サーバによって設定されます。これは、テーブル定義内に存在するインデックスのキーサイズに依存します。ほとんどの場合、I/O バッファサイズと等価に設定されます。

キーキャッシュの再構築

キーキャッシュはパラメータ値を更新することで、随時、再構築されます。例：

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

key_buffer_size または **key_cache_block_size** キーキャッシュ構成部位に構成部位の現在値と異なる値を割り当てた場合、サーバはキャッシュの旧構造を破壊し、新しい値に基づいた新構造を作成します。キャッシュに汚染されたブロックが存在する場合、サーバはキャッシュを破壊/再作成する前にディスクにそれを保存します。他のキーキャッシュパラメータを変更した場合再構築は起きません。

キーキャッシュを再構築する場合、サーバはディスクの汚染されたバッファ内容を最初にフラッシュします。その後、キャッシュ内容は無効となります。しかし、再構築はキャッシュに割り当てられたインデックス使用を必要とするクエリをブロックしません。その代わり、サーバはネイティブファイルシステムキャッシングを使用するテーブルインデックスに直接アクセスします。ファイルシステムキャッシングはキーキャッシュを使用した場合ほど、効率が良くありません。クエリの実行がなされますが、速度の低下が予想されます。キャッシュが再構築された後、キャッシュインデックスが割り当てられる毎に有効となり、インデックスのファイルシステムキャッシングの使用が停止します。

MyISAM インデックス統計コレクション

ストレージエンジンはオプティマイザに使用されるテーブルの統計を集めます。テーブル統計は値グループに基づき、値グループは同じキープリフィックス値を持つ行のセットです。オプティマイザの目的に関しては、標準グループサイズは重要な統計となります。

MySQL は標準値グループサイズを以下のように使用します。

- 各 **ref** アクセスごとに読まれなければならない行の数を推測
- 一部 join が生成する行の数を推測するため: つまり、このフォームのオペレーションが生成する行の数になります。

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

インデックスの標準値グループサイズが増加する毎に、その2つの目的に関してはインデックスは役に立たなくなり、というのも、ルックアップあたりの標準行数は増加するためです。最適化の目的のためにインデックスが有用であるため、各インデックス値はターゲットとするテーブル内の行の数は少なくなければいけません。既存のインデックス値が大量の行を生成する際、インデックスは役に立たなくなり、MySQL はそれを使用しなくなります。

標準値グループサイズはテーブルカーディナリティと関連しています。これは値グループの数を指します。SHOW INDEX ステートメントは N/S に基づくカーディナリティ値を表示します。これは N がテーブル内の行数であり、S が標準値グループサイズとなります。その比率はテーブル内の値グループの概数を生成します。

`<=>` 比較演算子に基づく join に対して、NULL は他の値と同様に扱われます。NULL `<=>` NULL、ちょうど N `<=>` N が他の N であるように。

ただし、`=` 演算子に基づく join に対して、NULL は非 NULL 値と異なります。expr1 = expr2 は expr1 あるいは expr2 (もしくは両方) が NULL である場合、適切ではありません。このことは、フォーム `tbl_name.key = expr` の比較に対する ref アクセスに影響を与えます。expr の現在値が NULL の場合、MySQL はテーブルにアクセスしません。というのも、比較は適切でないからです。

`=` 比較に対して、NULL 値の数がテーブル上にいくらであっても、関係ありません。最適化のためには、関連する値は非 NULL 値グループの標準サイズにします。ただし、現在 MySQL は標準サイズを収集もしくは使用することを許可しません。

MyISAM テーブルに対して、`myisam_stats_method` システム変数を使用することで、テーブル統計コレクションが管理されます。この変数には 2 つの可能値があり、これらは以下のとおり異なります。

- `myisam_stats_method` が `nulls_equal` の場合、全ての NULL 値は等価として扱われます。(つまり、それらは全てシングル値グループを形成します。)

NULL 値グループサイズは標準非 NULL 値グループサイズよりはるかに大きくなります。このメソッドは標準値グループサイズを大きくゆがませます。これによりオプティマイザから見たインデックスは非 NULL 値を検索する join に対して役に立たないように見えます。結果的に、`nulls_equal` メソッドはオプティマイザに ref アクセスに対してインデックスを使用すべきときでも使用しないままにする可能性があります。

- `myisam_stats_method` が `nulls_unequal` の時、NULL 値は等価として扱われません。代わりに、各 NULL 値はサイズ 1 の別値グループを生成します。

NULL 値が多い場合、このメソッドは標準値グループサイズを小さくゆがませます。もし標準非 NULL 値グループサイズが大きい場合、NULL 値をサイズ 1 のグループとして取り扱うと、オプティマイザに非 NULL 値を探させる join のインデックス値を過大評価します。結果的に、`nulls_unequal` メソッドは、他のメソッドの方が適しているにもかかわらず、オプティマイザにこのインデックスを ref ルックアップに使用させることがあるかもしれません。

`=` よりも `<=>` を使用する join を多くユーザが使用している場合、NULL 値は比較では特別ではなく、1 つの NULL は他のものと等価です。この場合、`nulls_equal` は適切な統計メソッドです。

`myisam_stats_method` システム変数はグローバルとセッション値を保持しています。グローバル値の設定は MyISAM テーブルに対する MyISAM 統計収集に影響を与えます。セッション値を設定することで、現在のクライアント接続のみに対する統計収集に影響を与られます。これは、既存のメソッドで他のクライアントに影響を与えず、テーブルの統計をセッション値 `myisam_stats_method` に設定することで再生することが可能です。

テーブル統計を再生するために、以下のメソッドを使用してください。

- `myisam_stats_method` を設定し、CHECK TABLE ステートメントを発行します。
- `myisamchk --stats_method=method_name --analyze` を実行します。
- テーブルを変更し統計を旧値とし(例えば、行を挿入し、それから削除します)、そして `myisam_stats_method` を設定し ANALYZE TABLE ステートメントを発行します。

`myisam_stats_method` の使用に関する警告。

- 以下で説明されているように、強制的にテーブル統計を明示的に収集させることができます。ただし、MySQL も自動的に統計を収集する可能性があります。例えば、テーブルステートメント実行時、ステートメントの中にはテーブルを改良するものもあり、MySQL は統計を収集する可能性があります。(例えば、これは大量挿入や削除時、もしくは ALTER TABLE ステートメントの際に起こる可能性があります。)これが生じた場合、統計は `myisam_stats_method` がその時々で持っている値を使用して統計が収集されます。よって、あるメソッドで統計を収集したが `myisam_stats_method` が自動的にテーブル統計が収集された後他のメソッドに設定されている場合、他のメソッドが使用されます。
- 既存の MyISAM テーブルに対してどのメソッドが統計生成に使用されたかを知ることはできません。

- `myisam_stats_method`はMyISAMテーブルにのみ適用されます。他のストレージエンジンはテーブル統計を収集するメソッドが1つしかありません。通常は、`nulls_equal`メソッドに近いです。

MySQL でのテーブルのオープンとクローズの方法

`mysqladmin status`を実行すると、以下の出力が表示されます。

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

テーブルが6つしかない場合に `Open tables`値が12と表示されることに、当惑する場合があります。

MySQLはマルチスレッド化されているため、多数のクライアントが同時に同じものに対してクエリを使用することがあります。2つのクライアントスレッドで1つのファイルに異なるステータスが発生する問題を最小にするため、同時に実行しているスレッドがそれぞれで無関係にテーブルを開きます。これはメモリの消費を増やしますが、一般にパフォーマンスは向上します。MyISAMテーブルの場合は、テーブルを開いたそれぞれのクライアントにデータファイルに対するファイル記述子が必要になります。このテーブル型では、インデックスファイルに対するファイル記述子がすべてのスレッドで共有されます。

`table_open_cache`、`max_connections`、および `max_tmp_tables`サーバ変数は、サーバが開いた状態で保持できるファイルの最大数に影響します。これらの値の1つ以上を増加すると、OSによって制限されている1プロセスが持つことができるファイル記述子の最大数まで実行が可能になります。システムごとに方法は多様ですが、多数のオペレーティングシステムでオープンファイルの制限値を上げることができます。制限値の拡大が可能かどうかの判定、およびその実行方法については、使用するオペレーティングシステムの文書を参照してください。

`table_open_cache`は `max_connections`と関係します。たとえば、同時接続数が200の場合、最低 $200 \times N$ のテーブルキャッシュサイズが必要です。この `N`は結合で使用するテーブル数の最大値を示します。また、テンポラリテーブルとファイル用のファイル記述子も必要です。

あなたのオペレーティングシステムが `table_open_cache`の設定に従ったファイル記述子の数を処理できることを確認してください。`table_open_cache`の設定が高すぎると、MySQLがファイル記述子を使い果たして接続を拒否し、クエリの実行ができなくなり、信頼性が大幅に低下します。また、MyISAMストレージエンジンでは1つのテーブルごとに2つのファイル記述子が必要であることも考慮に入れる必要があります。`--open-files-limit`スタートアップオプションを使用すると、`mysqld`で使用可能なファイル記述子数を拡大できます。`'File' Not Found and Similar Errors`を参照してください。

オープンテーブルのキャッシュは、`table_open_cache`エントリレベルに保持されます。デフォルト値は64です。これは、`--table_open_cache`オプション `mysqld`に与えることで変更できます。`mysqld`は一時的にさらに多くのテーブルを開いてクエリの実行を実現することがあります。

MySQL Enterprise

`table_cache`#####MySQL Network
Monitoring and Advisory Service##### [http://www-jp.mysql.com/products/enterprise/
advisors.html](http://www-jp.mysql.com/products/enterprise/advisors.html)#####

以下の状況では、MySQLは使用されていないテーブルが閉じられ、テーブルキャッシュから削除されます。

- キャッシュが満杯のときに、キャッシュにないテーブルをスレッドが開こうとした場合。
- キャッシュに `table_open_cache`を超えるエントリがあり、あるスレッドがテーブルの使用を終えた場合。
- テーブルフラッシュオペレーションが起きたとき。いずれかのユーザが `FLUSH TABLES`、`mysqladmin flush-tables`または `mysqladmin refresh`を実行した場合。

テーブルキャッシュが満杯になると、サーバでは以下の手順に従って使用するキャッシュエントリを割り当てます。

- 現在使用中でないテーブルは、最後に使用した時が古いものから順にリリースされる。
- キャッシュが満杯でリリース可能なテーブルがなく、新たにテーブルを開く必要がある場合は、必要に応じてキャッシュが一時的に拡張される。

キャッシュが一時的に拡張された状況で、使用中のテーブルが使用されなくなったときは、そのテーブルが閉じられ、キャッシュからリリースされる。

テーブルは同時アクセスのそれぞれで開かれます。つまり、2 つのスレッドで同じテーブルにアクセスする場合、または 1 つのスレッドが同一クエリでテーブルに 2 回アクセスする場合 (テーブルを同一テーブルに結合する場合など) は、テーブル を 2 回開く必要があることになります。いずれかの **MyISAM** テーブルを最初に開く際に 2 つのファイル記述子が割り当てられ、その後さらにそのテーブルを使用する場合はファイル記述子が 1 つのみ割り当てられます。最初のオープン時の 2 つめの記述子は、インデックスファイルに使用され、この記述子はすべてのスレッドで共有されます。

HANDLER tbl_name OPEN ステートメントを使用してテーブルを開く場合は、専用テーブルオブジェクトがスレッドに割り当てられます。このテーブルオブジェクトは他のスレッドと共有されず、スレッドが **HANDLER tbl_name CLOSE** を呼び出すか、スレッドが終了するまで閉じられません。この場合はテーブルがテーブルキャッシュに戻されます (キャッシュが満杯でない場合)。 **HANDLER** 構文を参照してください。

テーブルキャッシュが小さすぎるかどうかは、**mysqld** の **Opened_tables** 変数のチェックで確認できます。

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 2741  |
+-----+-----+
```

たとえ多くの **FLUSH TABLES** を実行していない場合でも、この値が非常に大きい場合は、テーブルキャッシュサイズを拡張する必要があります。 **システム変数**、**ステータス変数** を参照して下さい。

1 つのデータベースに大量のテーブルを作成した場合の欠点

ディレクトリに **MyISAM** テーブルが多数ある場合、オープン、クローズ、および作成の動作が低速になります。多数のテーブルに対して **SELECT** ステートメントを実行した場合、必要なテーブルを開くごとに、他のテーブルを閉じることが必要になるため、テーブルキャッシュが満杯の場合にオーバーヘッドが少し発生します。このオーバーヘッドは、テーブルキャッシュを拡大することで軽減できます。

MySQL サーバの最適化

システム、コンパイル時間およびスタートアップパラメータのチューニング

システムレベルの要素は、その一部を初期段階に決定する必要があるため、この話から始めます。これに該当しない場合は、システムを大きく変えることが重要でないのであれば、このセクションは簡単に目を通せば十分です。ただし、このレベルで変更を行うことでどの程度改善できるのかを自覚しておくことは必ず役に立ちます。

使用するオペレーティングシステムは非常に重要です。複数 CPU のコンピュータを使用するなら、Solaris (スレッド実装機能が優れている) または Linux (2.2 カーネルの SMP サポートが優れている) が良いでしょう。また、旧バージョンの Linux カーネルのデフォルトには 2G ファイルサイズの制限があります。このカーネルで 2G より大きいファイルがどうしても必要な場合は、ext2 ファイルシステムの LFS (Large File System) パッチを入手する必要があります。これ以外の ReiserFS や XFS などには 2G の制限がありません。

MySQL を本番稼働させていないため、可能であれば選択前に候補のプラットフォームのテストを実行することを推奨します。

他のヒント：

- RAM が十分にある場合は、スワップデバイスすべてを削除できる。オペレーティングシステムによっては、空きメモリがある場合でもスワップデバイスが使用されることがある。
- 外部ロックを回避する。MySQL 4.0以降、外部ロックはデフォルトで全てのシステムで使用不可能となっています。 **--external-locking** と **--skip-external-locking** オプションは外部ロックの使用可能および不可能を明示的に宣言します。

外部ロックを使用しなければ、1つのサーバのみを起動している場合、MySQL の機能性には影響しません。 **myisamchk** を起動する前にサーバを停止してください (または関連するテーブルをロック/フラッシュしてください)。いくつかのシステム上では外部ロックを必ず使用不可に設定します。というのも、機能しないからです。

外部ロックが使用不可である唯一の状況は、同一データに対して複数の MySQL servers (クライアントではない) を実行している場合と、サーバに対して初めにテーブルのフラッシュとロックを行う指示を出さずに、テーブルに

対して `myisamchk` を実行する場合に限られる。MySQL クラスタを使用しているとき以外で、同じデータを同時にアクセスするのに複数の MySQL サーバを使用することは推奨されていません。

`LOCK TABLES` と `UNLOCK TABLES` ステートメントは内部ロックを使用するため、外部ロックが不可となっている場合でも使用できます。

サーバパラメータのチューニング

mysql サーバで使われるデフォルトのバッファサイズは次のコマンドで確認できます。

```
shell> mysql --verbose --help
```

このコマンドによって、mysql オプションと設定可能な変数すべての一覧が生成されます。この出力には、デフォルトの変数値も記載され、以下のように表示されます。

```
help                TRUE
abort-slave-event-count 0
allow-suspicious-udfs  FALSE
auto-increment-increment 1
auto-increment-offset 1
automatic-sp-privileges  TRUE
basedir              /home/mysql/
bind-address          (No default value)
character-set-client-handshake TRUE
character-set-server  latin1
character-sets-dir    /home/mysql/share/mysql/charsets/
chroot                (No default value)
collation-server      latin1_swedish_ci
completion-type       0
concurrent-insert     1
console               FALSE
datadir               /home/mysql/var/
default-character-set  latin1
default-collation      latin1_swedish_ci
default-time-zone     (No default value)
disconnect-slave-event-count 0
enable-locking         FALSE
enable-pstack          FALSE
engine-condition-pushdown FALSE
external-locking       FALSE
gdb                    FALSE
large-pages            FALSE
init-connect           (No default value)
init-file              (No default value)
init-slave             (No default value)
innodb                 TRUE
innodb_checksums      TRUE
innodb_data_home_dir  (No default value)
innodb_doublewrite    TRUE
innodb_fast_shutdown  1
innodb_file_per_table FALSE
innodb_flush_log_at_trx_commit 1
innodb_flush_method   (No default value)
innodb_locks_unsafe_for_binlog FALSE
innodb_log_arch_dir   (No default value)
innodb_log_group_home_dir (No default value)
innodb_max_dirty_pages_pct 90
innodb_max_purge_lag  0
innodb_status_file    FALSE
innodb_table_locks    TRUE
innodb_support_xa     TRUE
isam                   FALSE
language               /home/mysql/share/mysql/english
local-infile           TRUE
log                    /home/mysql/var/master1.log
log-bin                /home/mysql/var/master1
log-bin-index          (No default value)
log-bin-trust-routine-creators FALSE
log-error              /home/mysql/var/master1.err
log-isam               myisam.log
log-queries-not-using-indexes FALSE
```



```

log-short-format      FALSE
log-slave-updates     FALSE
log-slow-admin-statements  FALSE
log-slow-queries      (No default value)
log-tc                tc.log
log-tc-size           24576
log-update             (No default value)
log-warnings          1
low-priority-updates  FALSE
master-connect-retry  60
master-host           (No default value)
master-info-file       master.info
master-password       (No default value)
master-port           3306
master-retry-count    86400
master-ssl            FALSE
master-ssl-ca         (No default value)
master-ssl-capath     (No default value)
master-ssl-cert       (No default value)
master-ssl-cipher     (No default value)
master-ssl-key        (No default value)
master-user           test
max-binlog-dump-events 0
memlock              FALSE
myisam-recover        OFF
ndbcluster            FALSE
ndb-connectstring     (No default value)
ndb-mgmd-host         (No default value)
ndb-nodeid            0
ndb-autoincrement-prefetch-sz 32
ndb-distribution      KEYHASH
ndb-force-send        TRUE
ndb_force_send        TRUE
ndb-use-exact-count   TRUE
ndb_use_exact_count   TRUE
ndb-shm               FALSE
ndb-optimized-node-selection TRUE
ndb-cache-check-time  0
ndb-index-stat-enable TRUE
ndb-index-stat-cache-entries 32
ndb-index-stat-update-freq 20
new                   FALSE
old-alter-table       FALSE
old-passwords         FALSE
old-style-user-limits FALSE
pid-file              /home/mysql/var/hostname.pid1
port                  3306
relay-log             (No default value)
relay-log-index       (No default value)
relay-log-info-file   relay-log.info
replicate-same-server-id FALSE
report-host           (No default value)
report-password       (No default value)
report-port           3306
report-user           (No default value)
rpl-recovery-rank     0
safe-user-create      FALSE
secure-auth           FALSE
server-id             1
show-slave-auth-info  FALSE
skip-grant-tables     FALSE
skip-slave-start      FALSE
slave-load-tmpdir     /tmp/
socket                /tmp/mysql.sock
sporadic-binlog-dump-fail FALSE
sql-mode              OFF
symbolic-links        TRUE
tc-heuristic-recover  (No default value)
temp-pool             TRUE
timed_mutexes        FALSE
tmpdir                (No default value)
use-symbolic-links    TRUE
verbose               TRUE
warnings              1
back_log              50

```

```

binlog_cache_size      32768
bulk_insert_buffer_size 8388608
connect_timeout        5
date_format            (No default value)
datetime_format        (No default value)
default_week_format    0
delayed_insert_limit   100
delayed_insert_timeout 300
delayed_queue_size     1000
expire_logs_days       0
flush_time             0
ft_max_word_len        84
ft_min_word_len        4
ft_query_expansion_limit 20
ft_stopword_file       (No default value)
group_concat_max_len   1024
innodb_additional_mem_pool_size 1048576
innodb_autoextend_increment 8
innodb_buffer_pool_awesome_mem_mb 0
innodb_buffer_pool_size 8388608
innodb_concurrency_tickets 500
innodb_file_io_threads 4
innodb_force_recovery 0
innodb_lock_wait_timeout 50
innodb_log_buffer_size 1048576
innodb_log_file_size   5242880
innodb_log_files_in_group 2
innodb_mirrored_log_groups 1
innodb_open_files      300
innodb_sync_spin_loops 20
innodb_thread_concurrency 20
innodb_commit_concurrency 0
innodb_thread_sleep_delay 10000
interactive_timeout     28800
join_buffer_size        131072
key_buffer_size         8388600
key_cache_age_threshold 300
key_cache_block_size    1024
key_cache_division_limit 100
long_query_time         10
lower_case_table_names  0
max_allowed_packet      1048576
max_binlog_cache_size   4294967295
max_binlog_size         1073741824
max_connect_errors      10
max_connections         100
max_delayed_threads     20
max_error_count         64
max_heap_table_size     16777216
max_join_size           4294967295
max_length_for_sort_data 1024
max_relay_log_size      0
max_seeks_for_key       4294967295
max_sort_length         1024
max_tmp_tables          32
max_user_connections    0
max_write_lock_count    4294967295
multi_range_count       256
myisam_block_size       1024
myisam_data_pointer_size 6
myisam_max_extra_sort_file_size 2147483648
myisam_max_sort_file_size 2147483647
myisam_repair_threads   1
myisam_sort_buffer_size 8388608
myisam_stats_method     nulls_unequal
net_buffer_length       16384
net_read_timeout        30
net_retry_count         10
net_write_timeout       60
open_files_limit        0
optimizer_prune_level   1
optimizer_search_depth  62
preload_buffer_size     32768
query_alloc_block_size  8192
query_cache_limit       1048576

```

```

query_cache_min_res_unit    4096
query_cache_size            0
query_cache_type            1
query_cache_wlock_invalidate FALSE
query_prealloc_size        8192
range_alloc_block_size     2048
read_buffer_size            131072
read_only                   FALSE
read_rnd_buffer_size        262144
div_precision_increment     4
record_buffer               131072
relay_log_purge             TRUE
relay_log_space_limit       0
slave_compressed_protocol   FALSE
slave_net_timeout           3600
slave_transaction_retries   10
slow_launch_time            2
sort_buffer_size            2097144
sync_binlog                 0
sync_frm                    TRUE
sync_replication            0
sync_replication_slave_id   0
sync_replication_timeout    10
table_open_cache            64
table_lock_wait_timeout     50
thread_cache_size           0
thread_concurrency          10
thread_stack                196608
time_format                  (No default value)
tmp_table_size               33554432
transaction_alloc_block_size 8192
transaction_prealloc_size   4096
updatable_views_with_limit  1
wait_timeout                 28800

```

現在実行中の `mysqld` サーバがある場合は、次のステートメントで変数に実際に使用されている値を調べることができます。

```
mysql> SHOW VARIABLES;
```

また、次のステートメントでは、実行中のサーバの統計やステータスインジケータを調べることができます。

```
mysql> SHOW STATUS;
```

システム変数とステータス情報は、`mysqladmin`でも入手できます。

```

shell> mysqladmin variables
shell> mysqladmin extended-status

```

全てのシステムとステータス変数については、本マニュアルの[システム変数](#)と[ステータス変数](#)を参照してください。

MySQL は非常にスケーラブルなアルゴリズムを使用しているため、通常は実行時のメモリ消費が非常に小さくなります。しかし、MySQL に対するメモリを多く割り当てると、通常はパフォーマンスが向上します。

MySQL サーバをチューニングする際に使用される最も重要な変数は `key_buffer_size` と `table_open_cache` の 2 つです。他の変数の変更を行う前にこの変数をあらかじめ適切に設定しておくことで自信がつきます。

以下に典型的な変数を実行時に設定している例を示します。

- 最小 256M のメモリで多数のテーブルがあり、中程度のクライアントで最大のパフォーマンスを得るには、次のように使用します。

```

shell> mysql_safe --key_buffer_size=64M --table_open_cache=256 \
--sort_buffer_size=4M --read_buffer_size=1M &

```

- メモリが 128M で、テーブルは少数で大量のソートの実行が必要な場合は、次のように使用できます。

```
shell> mysql_safe --key_buffer_size=16M --sort_buffer_size=1M
```

同時接続が多数ある場合、`mysqld` が各接続ごとに最小限のメモリを使用するよう設定されていない限り、スワップ問題が起こります。全ての接続にメモリが十分であれば `mysqld` のパフォーマンス性は向上します。

- メモリがほとんどなく大量の接続がある場合は、次のように使用します。

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
--read_buffer_size=100K &
```

また、次のようにもできます。

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
--table_open_cache=32 --read_buffer_size=8K \
--net_buffer_length=1K &
```

使用可能メモリより大幅に大きいテーブルで **GROUP BY** または **GROUP BY** を実行する場合は **read_rnd_buffer_size** の値を大きくしてソート操作後のレコードの読み取りの速度を上げる必要があります。

ユーザの **MySQL** ディストリビューションで、例オプションファイルを使用する場合は、**あらかじめ形成されたオプション・ファイル** を参照してください。

mysqld または **mysqld_safe** のコマンドラインでオプションを指定した場合、そのサーバの呼び出しでしか有効性が保持されないことに注意してください。サーバ実行のたびにオプションを使用する場合は、オプション設定ファイルに配置します。

パラメータ変更の有効性を調べるには、次のように実行します。

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

変数値は出力の最後付近で一覧表示されます。**--verbose** と **--help** オプションが残っていることを確認してください。でなければ、コマンドラインでそれらの後に表示されるオプションの効果は出力に反映されません。

InnoDB ストレージエンジンのチューニングに関する情報は、**InnoDB パフォーマンス チューニング ヒント** を参照してください。

MySQL Enterprise

```
#####MySQL Network Monitoring and Advisory Service#####
##### http://www.jp.mysql.com/products/enterprise/advisors.html#####
```

クエリオプティマイザパフォーマンスの管理

クエリオプティマイザの役割は **SQL** クエリの実行に際し、最適なプランを検索することです。「良い」と「悪い」プランのパフォーマンス性の違いは、マグニチュードの順序(つまり、秒に対する時間、または日にち)であるため、**MySQL** を含む大抵のクエリオプティマイザは全ての評価可能なプランの中から最適なプランを検索します。join クエリに対して、**MySQL** オプティマイザによって確認される可能なプランの数は、クエリで参照されるテーブル数と共に、急激に増加します。少ない数のテーブル(通常から7-10以下)に対して、これは問題とはなりません。しかし、大きなクエリが提出されたとき、クエリの最適化に要する時間はサーバのパフォーマンスを低下させる主な原因となります。

フレキシブルなクエリ最適化メソッドは、最適なクエリ評価プランを、オプティマイザがどれほど徹底的に検索するかを管理します。基本的には、オプティマイザによって確認されるプランが少なければ少ないほど、クエリをコンパイルするのに要する時間も少なくてすむと考えられています。一方で、オプティマイザはいくつかのプランをスキップするため、最適プランを見逃す可能性もあります。

評価するプラン数に対するオプティマイザの動作は二つのシステム変数を通して管理されます。

- optimizer_prune_level** 変数はオプティマイザに、各テーブルにアクセスされた行数の見積もりに基づくプランをスキップさせます。経験上この種類の「推測」は最適プランを見過ごすことはほとんどなく、劇的にクエリをコンパイルする時間を減少させます。(**optimizer_prune_level=1**) のオプションがデフォルトなのはこのためです。ただし、オプティマイザがより適したクエリプランを見逃したと思う場合は、クエリのコンパイルにかなりの時間を要するリスクを伴いますが、このオプション(**optimizer_prune_level=0**) は停止できます。この発見的方法を使用しても、オプティマイザは、指数的なプランの数を検索します。
- optimizer_search_depth** 変数はオプティマイザが各不完全プランの「先」をどのくらい見通すかを示すほか、さらに拡張が必要かを評価します。**optimizer_search_depth** のさらに小さい値は、マグニチュード順序あるいはクエリのコンパイルに要する時間が劇的に減少します。例えば、12, 13, もしくはそれ以上のテーブルのクエリは、**optimizer_search_depth** がクエリ内のテーブル数に近い場合、コンパイルするのに数時間、時には数日間を容易に必要とします。同時に、3が4と等価の **optimizer_search_depth** でコンパイルされた場合、オプティマイザは同

クエリでは1分以下でコンパイル可能な場合があります。`optimizer_search_depth`にとってリーズナブルな値が不明確な場合、変数を0に設定することで、オブティマイザに自動的に値を決定させることができます。

MySQL の速度に対するコンパイルとリンクの影響

以下のテストのほとんどは、MySQL ベンチマークを使用した Linux で実行されていますが、これ以外のオペレーティングシステムおよびワークロードに対しても一定の指針になります。

`-static`とリンクした場合に最速のバイナリが得られます。

Linux 上では、`pgcc`および `-O3`でコンパイルした場合に最速のコードが得られます。これらのオプションで `sql_yacc.cc`をコンパイルする場合は、`gcc/pgcc`で関数のすべてをインラインにする際に大量のメモリが要求されるため約 200M のメモリが必要です。MySQL のコンフィギュア時に `CXX=gcc` も設定して、`libstdc++`ライブラリ (これは不要です) が含まれないようにします。`pgcc`の一部のバージョンでは、生成されたコードを x86 タイプのプロセッサ (AMD など) すべてで動作可能にするコンパイラオプションを使用しても、コードが純正 Pentium プロセッサでしか実行できないため注意が必要です。

適切なコンパイラおよびコンパイラオプションを使用することで、アプリケーションの速度が 10–30% 改善されます。これは各自で SQL サーバをコンパイルする場合に特に重要です。

Cygnus CodeFusion と Fujitsu コンパイラの両方をテストしましたが、いずれもバグフリーではなく、最適化をオンにして MySQL をコンパイルするには不十分でした。

標準の MySQL バイナリディストリビューションは、すべてのキャラクタセットをサポートするようにコンパイルされています。MySQL のコンパイル時は、使用するキャラクタセットのサポートのみを含めます。これは `--with-charset`オプションから `configure`によって管理されます。

以下に実施した測定結果の一部を紹介します。

- `pgcc`を使用し、すべてを `-O6`でコンパイルした場合、`mysqld`サーバは `gcc2.95.2`と比較して 1% 速度が上がる。
- 動的にリンクした場合 (`-static`なし) は、結果が Linux 上で 13% 遅くなった。クライアントアプリケーションには動的リンクの MySQL ライブラリを使用できることに注意する。これは、サーバのパフォーマンス上重大である。
- 同一ホスト上で実行されるクライアントからサーバへの接続で、Unix ソケットファイルではなく、TCP/IP で接続すると、7.5% パフォーマンスが遅くなった (Unix では `localhost`に接続する場合、MySQL ではデフォルトでソケットファイルが使用される)。
- TCPクライアントからサーバへの TCP/IP 接続で別のホストにあるリモートサーバに接続した場合、100M イーサネットによる接続でも、同一ホスト上のローカルサーバに接続した場合と比較して、8–11% 遅くなった。
- 暗号化した接続 (内部 SSL サポートによるすべてのデータの暗号化) を使用してベンチマークテストを実行した場合、パフォーマンスが 55% 遅くなった。
- `--with-debug=full`でコンパイルすると、ほとんどのクエリが 20% 遅くなる。一部のクエリはかなり長くかかった (たとえば MySQL ベンチマークは 35% の速度低下)。 `--with-debug=full`なしで) を使用すると、この速度低下は 15% で済む。 `--with-debug=full`でコンパイルされた `mysqld`バージョンは、 `--skip-safemalloc`オプションで起動すると実行時のメモリチェックを無効化できる。この場合の最終的な結果は、 `--with-debug`で構成した場合に非常に近くなる。
- Sun UltraSPARC-IIe, Forte 5.0 は、 `gcc3.2` より 4% 速度が上がった。
- Sun UltraSPARC-IIe, Forte 5.0 では、64 ビットモードより 32 ビットモードのほうが 4% 速かった。
- `gcc2.95.2` for UltraSPARC にオプション `-mcpu=v8 -Wa,-xarch=v8plusa`を付けてコンパイルすると、パフォーマンスが 4% 改善した。
- Solaris 2.5.1, MIT-pthreads は、単一プロセッサ上で Solaris ネイティブスレッドより 8–12% 遅かった。CPU の負荷が増加するとこの差はさらに拡大する。
- フレームポインタ `-fomit-frame-pointer`または `-fomit-frame-pointer -ffixed-ebp`なしで `gcc`を使用して Linux-x86 でコンパイルすると、`mysqld`が 1–4% 遅くなった。

`pgcc`によるコンパイルに MySQL AB 提供の バイナリMySQL-Linux ディストリビューションを使用したか、AMD で実行されないコードを生成するバグが `pgcc`にあったため、通常の `gcc` の使用に戻さざるを得ませんでした。このバグ

が解決されるまで `gcc` の使用を続行します。ただし、AMD 以外のコンピュータを使用する場合は、`pgcc` でコンパイルすると高速なバイナリが得られます。標準の MySQL Linux バイナリは、速度および移植性を高めるため静的にリンクされています。

MySQL でのメモリの使用

以下の一覧は、`mysqld` サーバでのメモリの使用方法の一部を示しています。可能な場合は、メモリ使用に関連するサーバ変数名も記載されています。

- The キーバッファ (変数 `key_buffer_size`) はすべてのスレッドで共有される。サーバが使用するこれ以外のバッファは必要に応じて割り当てられる。 [サーバパラメータのチューニング](#) を参照してください。
- それぞれの接続はいくつかのスレッド固有領域を使用します。以下にそれらの領域およびどの変数がサイズをコントロールするかをリストアップします。
 - スタック(デフォルト192KB, 変数 `thread_stack`)
 - 接続バッファ(変数 `net_buffer_length`)
 - 結果バッファ(変数 `net_buffer_length`)

接続バッファと結果バッファ (ともに `net_buffer_length` で与えられるサイズで始まる) は必要に応じて `max_allowed_packet` まで動的に拡張される。結果バッファは各 SQL ステートメントの後 `net_buffer_length` に縮小される。クエリの実行中は現在のクエリ文字列のコピーも割り当てられる。

- すべてのスレッドで同じベースメモリが共有される。
- スレッドが必要ない場合、それに割り当てられたメモリはリリースされ、スレッドがスレッドキャッシュに戻るまでシステムに返されます。この場合、メモリは割り当てられた状態のままです。
- MySQL 5.1.4以前では、圧縮された `MyISAM` テーブルのみがメモリーマップされました。MySQL 5.1.4以降では、`mysam_use_mmap` システム変数は全ての `MyISAM` テーブルに対してメモリマッピングを可能にするために、1 に設定できます。 [システム変数](#)。
- テーブルの順次スキャンを行う要求はそれぞれ、read buffer (変数 `read_buffer_size`) を割り当てる。
- レコードを "ランダムな" 順序で読み取る場合 (ソート後など)、random-read buffer が割り当てられディスクシークが回避される (変数 `read_rnd_buffer_size`)。
- 結合はすべて 1 回の受け渡しで実行され、ほとんどの結合はテンポラリテーブルを使用せずに実行される。テンポラリテーブルのほとんどはメモリーベース (HEAP) テーブルである。レコード長の大きなテンポラリテーブル (すべてのカラム長の合計として算出) や `BLOB` カラムが含まれるテンポラリテーブルはディスク上に格納される。

メモリ内の heap テーブルのサイズが `tmp_table_size` を超えた場合、MySQL はディスクベース `MyISAM` テーブルに変更されることで自動的に処理される。この問題を回避するには、`tmp_table_size` オプションを `mysqld` に設定するか、クライアントプログラムで `SQL_BIG_TABLES` を設定することで、テンポラリテーブルのサイズを拡張する。 [SET 構文](#) を参照してください。

MySQL Enterprise

MySQL Network Monitoring and Advisory Service#####`tmp_table_size`#####
#####`tmp_table_size`#####MySQL Network Monitoring and Advisory
Service#####<http://www.jp.mysql.com/products/enterprise/advisors.html>#####

- ソートを実行する要求のほとんどで、ソートバッファおよび結果セットサイズに応じた 0 から 2 つのテンポラリファイルが割り当てられる。 [Where MySQL Stores Temporary Files](#) を参照してください。
- 解析および計算のほとんどすべてが、ローカルメモリストアで実行される。小さいアイテムにはメモリーオーバーヘッドが不要で、通常の低速メモリの割り当ておよび解放は回避される。メモリは、予測外の規模の文字列の場合のみ割り当てられ、これは、`malloc()` および `free()` で実行される。
- 開かれる `MyISAM` テーブルにはそれぞれ 1 回開かれ、データファイルは、同時実行スレッドごとに 1 回開かれる。同時スレッドのそれぞれに対して、テーブル構造、各カラムのカラム構造、サイズ `3 × N` のバッファが割り当てられる (`N` は、レコードの最大長、ただし `BLOB` カラムは計算外)。`BLOB` カラムは、5 から 8 バイトに `BLOB` データ

の長さを加算したバイト数を使用する。**MyISAM**ストレージエンジンは、内部使用のための追加レコードを 1 つ使用する。

- **BLOB**カラムがあるテーブルのそれぞれで、大きな **BLOB**値を読み込むためにバッファが動的に拡張される。テーブルをスキャンする場合は、最大 **BLOB**値と同じ大きさのバッファが割り当てられる。
- 使用中テーブルすべてのハンドラ構造がキャッシュに保存され、FIFO 形式で管理される。一般にキャッシュには 64 のエントリがある。テーブルが同時に 2 つの実行スレッドで使用されている場合、キャッシュにはそのテーブルのエントリが 2 つ配置される。**MySQL** での**テーブルのオープンとクローズの方法**を参照してください。
- **FLUSH TABLES**コマンド (または **mysqladmin flush-tables**ステートメント) によって、使用中でないテーブルすべてが閉じられ、現在実行中のスレッドの終了時に使用中のテーブルすべてが閉じられるように指定される。これで効率的に使用中メモリに空きを作ることができる。**FLUSH TABLES**は全てのテーブルが閉じられるまで返されない。

ps およびその他のステータスプログラムによって、**mysqld**が大量のメモリを使用していることを示すレポートが行われることがあります。これは、複数のメモリアドレスでのスレッドスタックによって発生します。たとえば、Solaris バージョンの **ps**ではスタック間の使用していないメモリが使用メモリにカウントされます。これは、**swap -s**で使用可能スワップをチェックすることで検証できます。市販のメモリリーク検出装置で **mysqld**をテストし、メモリリークがないと判明しています。

MySQLの DNS の使用

新たなクライアントが **mysqld**に接続すると、**mysqld**によって要求を処理する新規のスレッドが作成されます。このスレッドでは、まずホスト名がホスト名キャッシュにあるかどうかチェックされます。ない場合は、ホスト名の解決が試行されます。

- オペレーティングシステムがスレッドセーフの **gethostbyaddr_r()**と**gethostbyname_r()**の呼び出しをサポートしている場合、スレッドではこれを使用してホスト名の解決が実行される。
- オペレーティングシステムがスレッドセーフの呼び出しをサポートしていない場合、スレッドでは相互排除ロックを行い、代わりに **gethostbyaddr()**と **gethostbyname()**が呼び出される。この場合、他のスレッドでは最初のスレッドが相互排除ロックを解除するまでホスト名キャッシュ内のホスト名を解決できなくなることに注意する。

--skip-name-resolveを **mysqld**オプションを指定して起動すると、DNS ホスト名ルックアップを無効化できます。ただし、この場合は、MySQL 権限テーブルで IP 番号しか使用できなくなります。

非常に低速の DNS と多数のホストがある場合は、**--skip-name-resolve**で DNS ルックアップを無効化するか、**HOST_CACHE_SIZE**の定義 (デフォルト値: 128) を拡張し、**mysqld**を再コンパイルすることで、パフォーマンスを改善できます。

--skip-host-cache オプションを使用してサーバを起動すると、ホスト名キャッシュを無効化できます。ホスト名のキャッシュをクリアするには、**FLUSH HOSTS**ステートメントを使用するか、**mysqladmin flush-hosts**コマンドを実行します。

TCP/IP 接続すべてを認めない場合は、**--skip-networking**オプションを指定して **mysqld**を開始します。

ディスク関連の問題

- ディスクシークはパフォーマンスに対する大きなボトルネックである。この問題は、データが拡大し、効率的なキャッシュが実行不能になるほど大きくなるにつれて明白になる。大規模データベースで、事実上ランダムにデータにアクセスする場合、読み取りでは最低 1 回、書き込みでは最低 2 回のディスクシークが必要になることがわかる。この問題を最小にするには、シーク回数を減らすようにディスクを使用する。
- 複数のディスクに対してファイルをシンボリックリンクするか、ストライピングを行って、利用可能なディスクスピンドル数を増加する (およびそれによるシークのオーバーヘッドを軽減する) 。
- シンボリックリンクの使用

MyISAMテーブルの場合、通常のデータディレクトリ内の位置から別のディスクへのインデックスファイルやデータファイルのシンボリックリンクを行う (ストライピングも可能)。これによって、ディスクが他の用途に

使用されていなければ、シークと読み取り時間がいずれも改善される。[シンボリックリンクの使用](#)を参照してください。

- ストライピング

ストライピングは、ディスクが多数ある場合に、第 1 ブロックを第 1 ディスクに、第 2 ブロックは第 2 ディスクに、第 N ブロックは $(N \text{ MOD } \text{number\_of\_disks})$ ディスクにといった配置を意味する。これにより、通常のデータサイズがストライプサイズより小さい（または完全に一致している）場合にパフォーマンスが大幅に改善する。ストライピングはオペレーティングシステムとストライプサイズへの依存性が非常に高いため、ストライプサイズをさまざまに変えながらアプリケーションのベンチマークを行う。[独自のベンチマークの使用](#)を参照してください。

ストライピングの速度はパラメータによって大きく異なることに注意する。ストライピングパラメータの設定方法とディスク数によって桁ちがいの差異が発生する。ランダムアクセスが順次アクセスのいずれの最適化を行うかの選択が必要なことに注意する。

- 信頼性を高めるため、RAID 0+1（ストライピング + ミラーリング）の使用が必要な場合、 N 個のドライブのデータの保持に $2 \times N$ 個のドライブが必要になる。財務上の余裕がある場合はこれが最適な選択肢になる。しかし、処理の効率化にボリューム管理ソフトウェアへの投資が必要なこともある。
- データの種類の重大性に応じて RAID レベルを変える選択も推奨される。たとえば、ホスト情報やログなどの重要度の高いデータは、RAID 0+1 または RAID N ディスクに格納し、再生成が可能で重要性が中程度のデータは RAID 0 ディスクに格納することなどができる。RAID N は、パリティビットの更新に時間がかかるため、書き込みが多いと問題になる場合がある。
- Linux の場合、[hdparm](#) を使用してディスクのインタフェースを構成することでパフォーマンスを大幅に改善できる（負荷時に 100% 改善できることも珍しくない）。次の例は、MySQL（およびその他の多数のアプリケーション）に非常に適した [hdparm](#) オプションである。

```
hdparm -m 16 -d 1
```

上記を使用した場合のパフォーマンスと信頼性は使用ハードウェアに依存するため、[hdparm](#) の使用後はシステムを総合的にテストするように強く推奨する。詳細については、[hdparm](#) のページを参照。[hdparm](#) の使用が適切でない場合は、ファイルシステムの損傷が発生することがあるため、テストの際はあらかじめすべてのバックアップを取っておく必要がある。

- データベースが使用するファイルシステムのパラメータを設定することもできる。

if ファイルへの最終アクセス時を認識する必要がない（データベースサーバでは重要度が低い）場合、[-o noatime](#) オプションを使用してファイルシステムをマウントできる。これで、ファイルシステムの i ノードへの最終アクセス時間の更新がスキップされ、一部のディスクシークを回避できる。

多数のオペレーティングシステムで、[-o async](#) オプションを使用してディスクをマウントし、ファイルシステムが非同期で更新されるように設定できる。使用しているコンピュータが適度に安定している場合は、信頼性を損なわずにさらにパフォーマンスを改善できる（Linux ではこのフラグがデフォルトでオンになっている）。

シンボリックリンクの使用

テーブルとデータベースをデータベースディレクトリから他の位置に移動し、新しい位置へのシンボリックリンクに置換することができます。これは、たとえば、データベースを空き領域の多いファイルシステムに移動し、テーブルを別のディスクに分散することでシステムの速度を上げる場合などに実行できます。

この推奨される実行方法は、データベースだけ別のディスクへのシンボリックリンクを行い、最後の手段としてのみテーブルのシンボリックリンクを行うことです。

Unix 上のデータベースに対するシンボリックリンクの使用

Unix の場合、データベースのシンボリックリンクは、まず、空き領域のあるディスクにディレクトリを作成し、次に MySQL データベースディレクトリからそのディレクトリへのシンボリックリンクを作成します。

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL は、1 つのディレクトリに対して複数のデータベースをリンクさせることをサポートしていません。データベースディレクトリをシンボリックリンクに置換すると、複数のデータベースへシンボリックリンクを張らない限り、問題なく機能します。仮に MySQL データディレクトリにデータベース `db1` がある場合に、`db1` を指すシンボリックリンク `db2` を作成するとします。

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

これで、`db1` のテーブル `tbl_a` が、`db2` のテーブル `tbl_a` としても表示されます。あるスレッドで `db1.tbl_a` が更新され、他のクライアントが `db2.tbl_a` に更新すると、問題が発生します。

ただし、実際にこれを実行しなければいけないとき、ソースファイルである `mysys/my_symlink.c` を変更することで可能となります。その場合、以下のステートメントを参照してください。

```
if (!(MyFlags & MY_RESOLVE_LINK) ||
    (!stat(filename,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

このステートメントを以下に変えます。

```
if (1)
```

Unix 上のテーブルに対するシンボリックリンクの使用

`realpath()` の呼び出しの機能が完全でないシステムではテーブルのシンボリックリンクを行わないでください。(少なくとも、Linux と Solaris では `realpath()` がサポートされています)。 `SHOW VARIABLES LIKE 'have_symlink'` ステートメントを発行することで、ユーザのシステムがシンボリックリンクをサポートするかチェックできます。

MySQL 4.0 では `MyISAM` テーブルでのみシンボリックリンクが完全サポートされています。これ以外のテーブル型で上記のコマンドを使用すると、予想外の問題の発生の恐れがあります。

MySQL 4.0 での `MyISAM` テーブルのシンボリックリンクの処理は次のように機能します。

- データディレクトリには常にテーブル定義ファイル(`.frm`)、データファイル(`.MYD`) およびインデックスファイル(`.MYI`) がある。データファイルとインデックスファイルは、別の場所に移動し、データディレクトリ内でシンボリックリンクによって置換できる。定義ファイルはこれができない。
- データファイルとインデックスファイルは、それぞれ独立して別のディレクトリにシンボリックリンクを作成できる。
- シンボリックリンクは、オペレーティングシステムレベル (`mysqld` が実行されていない場合)、または SQL で `CREATE TABLE` に `DATA DIRECTORY` および `INDEX DIRECTORY` オプションを指定して実行できる。詳しくは `CREATE TABLE` 構文を参照してください。あるいは、シンボリックリンクは `ln -s` を使用してコマンドラインから手動で行えますが、これは `mysqld` が作動していない場合に限りです。
- `myisamchk` は、データファイルやインデックスファイルのシンボリックリンクを置き換えない。`myisamchk` はリンクで指し示されているファイルに直接作用する。テンポラリファイルはすべてデータファイルやインデックスファイルが配置されているのと同じディレクトリに作成されます。同様のことが `ALTER TABLE`、`OPTIMIZE TABLE`、そして `REPAIR TABLE` ステートメントでいえます。
- 注:シンボリックリンクを使用しているテーブルをドロップすると、シンボリックリンクとシンボリックリンクが指しているファイルの両方がドロップされる。このため、`root` として `mysqld` を実行すべきではなく、また、MySQL データベースディレクトリへの書き込みアクセスをユーザに許可するべきでもない。
- `ALTER TABLE ... RENAME` を使用してテーブルの名前を変更し、テーブルを他のデータベースに移動しない場合、データベースディレクトリのシンボリックリンクの名前が新しい名前に変更され、データファイルとインデックスファイルもそれに従って名前が変更される。
- `ALTER TABLE ... RENAME` を使用してテーブルを別のデータベースに移動すると、テーブルが別のデータベースディレクトリに移動され、それまであったシンボリックリンクとそれが指すファイルが削除される (新規テーブルのシンボリックリンクは作成されない)。
- シンボリックリンクを使用していない場合は、`mysqld` に `--skip-symbolic-links` オプションを指定して使用し、確実に誰もデータディレクトリの外でファイルのドロップや名前の変更を行う `mysqld` を使用できないようにする。

サポートされていないテーブルシンボリックリンクオペレーション

- `ALTER TABLE`では `DATA DIRECTORY`と `INDEX DIRECTORY` テーブルオプションが無視される。
- `BACKUP TABLE`と`RESTORE TABLE`ではシンボリックリンクが考慮されない。
- `.frm`ファイルはシンボリックリンクにすることがまったくできない (前述のように、データファイルとインデックスファイルのみシンボリックリンクにできる)。これを実行した場合 (シノニム作成など)、正しい結果が得られなくなる。MySQL データディレクトリにデータベース `db1`があり、このデータベースにはテーブル `tbl1`が、`db1`ディレクトリには `tbl1`を指すシンボリックリンク `tbl2`があるとする。

```
shell> cd /path/to/datadir/db1
shell> ln -s tbl1.frm tbl2.frm
shell> ln -s tbl1.MYD tbl2.MYD
shell> ln -s tbl1.MYI tbl2.MYI
```

あるスレッドで `db1.tbl1`が読み取られ、別のスレッドで `db1.tbl2`が更新されると、問題が発生する。

- クエリキャッシュが「欺かれ」 (`tbl1`が更新されていないと判断され、最新でない結果が返される)。
- `tbl2`に対する`ALTER`ステートメントもエラーになる。

上のデータベースに対するシンボリックリンクの使用

シンボリックリンクはデフォルトで全てのWindowsサーバのため有効になっています。これにより、シンボリックリンクを設定することでデータベースディレクトリを別のディスクにセットすることが可能となります。リンクを設定するプロセスは異なりますが、データベースシンボリックリンクがUnix上で作動するのと似ています。シンボリックリンクが必要な場合、`--skip-symbolic-links`オプションを使用して動作しないように設定できます。

On Windows では、対象ディレクトリへのパスが記載されたファイルを作成して MySQL データベースに対するシンボリックリンクを作成します。ファイル名 `db_name.sym`を使用してデータディレクトリにファイルを保存します。この `db_name`はデータベース名です。

たとえば、MySQL データディレクトリが `C:\mysql\data`で、データベース `foo`を `D:\data\foo`に配置したい場合。このプロセスを使用してsymlinkをセットしてください。

1. この作業には `D:\data\foo`ディレクトリが存在している必要があります。言い換えると、すでに `foo`という名前のデータベースディレクトリがデータディレクトリにある場合、これを `D:\data`に移動しないとシンボリックリンクが有効にならないことになります (問題を回避するため、データベースディレクトリの移動時はサーバを実行しないでください)。
2. `D:\data\foo`パスネームを含むテキストファイル`C:\mysql\data\foo.sym`を作成してください。

この後、`foo`データベースで作成される全てのテーブルは`D:\data\foo`内に作成されます。MySQLデータディレクトリに同じ名前のディレクトリがある場合、シンボリックリンクは使用されません。注意してください。 .

第7章 クライアントプログラムとユーティリティ プログラム

目次

クライアントプログラムとユーティリティ プログラムの概要	435
my_print_defaults — オプション ファイルから オプションを表示する	437
myisam_ftdump — フル テキスト インデックス情報を表示する	438
myisamchk — MyISAM テーブル メンテナンス ユーティリティ	439
myisamchk 一般的なオプション	440
myisamchk チェック オプション	442
myisamchk 修復オプション	442
他の myisamchk オプション	444
myisamchk メモリ使用量	445
myisamlog — Display MyISAM Log File Contents	445
myisampack — 圧縮された、読み取り専用MyISAM テーブルを作成する。	446
mysql — MySQL コマンド ライン ツール	452
mysql オプション	452
mysql Commands	457
mysql サーバサイドヘルプ	460
テキストファイルからSQLステートメントを実行する	461
mysql ヒント	461
mysqlaccess — アクセス権限をチェックするクライアント	462
mysqladmin — MySQL サーバの管理を行うクライアント	464
mysqlbinlog — バイナリログファイルを処理するためのユーティリティ	468
mysqlcheck — テーブル メンテナンスと修復プログラム	474
mysqldump — データベースバックアッププログラム	477
mysqlhotcopy — データベースバックアッププログラム	485
mysqlimport — データインポートプログラム	487
mysqlshow — データベース、テーブル、カラム情報を表示します。	490
mysqlslap — クライアント負荷シミュレーション	491
mysql_zap — パターンとマッチする処理を消去します。	494
perror — エラーコードの説明	494
replace — 文字列置き換えユーティリティ	495

MySQL クライアント プログラムには、データベースにアクセスしたり管理タスクを実行するためにサーバに接続するものが多々あります。他のユーティリティも利用できます。これらはサーバとクライアントを接続するようなことはしませんが、MySQL関連のオペレーションを実行します。

この章ではこれらのプログラムを大きく分け概要を提供した上で、各プログラムの詳細を説明します。各プログラムの詳細説明は起動構文とそのプログラムによって理解されるオプションを示しています。プログラム オプションの指定と、プログラムの起動に関する一般的な情報は、[3章MySQL プログラムの使用](#)を参照してください。

クライアントプログラムとユーティリティ プログラムの概要

以下のリストに手短にMySQL クライアント プログラムとユーティリティ プログラムを記します。

- [my_print_defaults](#)

オプション ファイルのオプション グループに存在するユーティリティです。[my_print_defaults — オプション ファイルから オプションを表示する](#) を参照してください。

- [myisam_ftdump](#)

MyISAMテーブル内のフルテキスト インデックスの情報を表示するユーティリティです。[myisam_ftdump — フルテキスト インデックス情報を表示する](#) を参照してください。

- [myisamchk](#)

MyISAMテーブルを修理、最適化、チェック、そして説明するユーティリティです。[myisamchk — MyISAM テーブル メンテナンス ユーティリティ](#) を参照してください。

- [myisamlog](#)

MyISAMログ ファイルの内容を処理するユーティリティです。[myisamlog — Display MyISAM Log File Contents](#) を参照してください。

- [myisampack](#)

読み取り専用の小型テーブルを生成するためにMyISAM テーブルを圧縮するユーティリティです。[myisampack — 圧縮された、読み取り専用MyISAM テーブルを作成する](#)。 を参照してください。

- [mysql](#)

インタラクティブにSQL ステートメントを書き込む、もしくはバッチモード内のファイルを使用してステートメントを実行するためのコマンドライン ツールです。[mysql — MySQL コマンドライン ツール](#) を参照してください。

- [mysqlaccess](#)

ホスト名、ユーザ名、そしてデータベース コンビネーションのアクセス権限をチェックするスクリプトです。[mysqlaccess — アクセス権限をチェックするクライアント](#) を参照してください。

- [mysqladmin](#)

データベースの作成や削除、グラント テーブルのリロード、ディスクへのテーブルのフラッシュ、そしてログ ファイルの再オープンなどの管理オペレーションを実行するクライアント。[mysqladmin](#)はサーバからバージョン、処理、そしてステータス情報の取得ができます。[mysqladmin — MySQL サーバの管理を行うクライアント](#) を参照してください。

- [mysqlbinlog](#)

バイナリ ログからステートメントを読み取るためのユーティリティ。.クラッシュ状態からリカバーするために、バイナリ ログ ファイルに含まれる実行ステートメントのログを使用することができます。[mysqlbinlog — バイナリログ ファイルを処理するためのユーティリティ](#) を参照してください。

- [mysqlcheck](#)

テーブルのチェック、修理、分析、そして最適化を行うテーブル メンテナンスのクライアント。[mysqlcheck — テーブル メンテナンスと修復プログラム](#) を参照してください。

- [mysqldump](#)

MySQL データベースをSQL、テキスト、もしくはXML としてダンプするクライアント。[mysqldump — データベースバックアッププログラム](#) を参照してください。

- [mysqlhotcopy](#)

サーバ作動中に MyISAM テーブルのバックアップを作成するユーティリティ。[mysqlhotcopy — データベースバックアッププログラム](#) を参照してください。

- [mysqlimport](#)

LOAD DATA INFILEを使用してそれぞれのテーブルにテキスト ファイルをインポートするクライアント。[mysqlimport — データインポートプログラム](#) を参照してください。

- [mysqlshow](#)

データベース、カラム、そしてインデックスの情報を表示するクライアント。[mysqlshow — データベース、テーブル、カラム情報を表示します](#)。 を参照してください。

- [mysqlslap](#)

MySQL サーバのクライアント負荷をエミュレートし、各ステージのタイミングを報告するクライアント。複数のクライアントがサーバにアクセスしているかのように作動します。

- `mysql_zap`

パターンと合致するプロセスを抹消するユーティリティです。 [mysql_zap — パターンとマッチする処理を消去します。](#)

- `perror`

システムあるいはMySQL エラー コードの意味を表示するユーティリティです。 [perror — エラーコードの説明](#) を参照してください。

- `replace`

インプット テキストで文字列を置換するユーティリティ プログラムです。 [replace — 文字列置き換えユーティリティ](#) を参照してください。

MySQL AB は管理や MySQL サーバとの使用のためにいくつかの GUI ツールも提供しています。これらについての基本的な情報に関しては、[3章MySQL プログラムの使用](#) を参照してください。

各MySQL プログラムが多くのさまざまなオプションを使用します。ほとんどの MySQL プログラムには `--help` オプションがあり、このオプションを使用することにより、そのプログラムのすべてのオプションの説明を表示することができます。例えば、`mysql --help` を試してみてください。

MySQL クライアント・サーバ ライブラリを使用してサーバと通信する MySQL クライアント プログラムは以下の環境変数を使用しています。

<code>MYSQL_UNIX_PORT</code>	<code>localhost</code> への接続に使用される、デフォルト Unix ソケットファイル
<code>MYSQL_TCP_PORT</code>	TCP/IP 接続に使用されるデフォルトのポート番号
<code>MYSQL_PWD</code>	デフォルトパスワード
<code>MYSQL_DEBUG</code>	デバッグ中にトレースオプションをデバッグ
<code>TMPDIR</code>	テンポラリテーブルやテンポラリファイルが作成されるディレクトリ

`MYSQL_PWD` の使用は危険です。 [パスワードのセキュリティ](#) を参照してください。

オプション ファイルやコマンドラインでオプションを指定することで、全ての標準プログラムで指定される環境変数値やデフォルト オプション値を重ね処理することができます。 [プログラム・オプションの指定](#) を参照してください。

my_print_defaults — オプション ファイルから オプションを表示する

`my_print_defaults` オプション ファイルのオプション グループ内にあるオプションを表示します。出力は特定のオプション グループを読むプログラムに使用されるオプションを示します。例えば、`mysqlcheck` プログラムは `[mysqlcheck]` と `[client]` のオプショングループを読みます。標準オプション ファイル内のグループに存在するオプションを確認するには、`my_print_defaults` を以下のように起動してください。

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=secret
--host=localhost
```

コマンドラインで特定されているフォーム通りに、1行につき1つのオプションによって、出力が構成されています。

`my_print_defaults` は以下のオプションを理解します。

- `--help, -?`

ヘルプ メッセージを表示し、閉じます。

- `--config-file=file_name, --defaults-file=file_name, -c file_name`

与えられたオプション ファイルのみ読みこみます。

- `--debug=debug_options, -# debug_options`

デバッグのログを書き込みます。debug_options 文字列は大抵 'd:t:o,file_name' になります。デフォルトは 'd:t:o,/tmp/my_print_defaults.trace' になります。

- --defaults-extra-file=file_name, --extra-file=file_name, -e file_name

このオプション ファイルはグローバル オプション ファイルの後に、ただし (Unix では) ユーザ オプション ファイルの前に読み込んでください。

- --defaults-group-suffix=suffix, -g suffix

コマンド ラインで名づけられたグループのほかに、以下の接尾辞を与えられたグループを読み込んでください。

- --no-defaults, -n

殻の文字列を返してください。

- --verbose, -v

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。

- --version, -V

バージョン情報を表示し、閉じます。

myisam_ftdump — フル テキスト インデックス情報を表示する

myisam_ftdump は MyISAM テーブル内の FULLTEXT インデックスに関する情報を表示します。MyISAM インデックス ファイルを直接読み込みますので、テーブルが存在しているサーバ ホストで起動しなければいけません。

myisam_ftdump は以下のように起動してください。

```
shell> myisam_ftdump [options] tbl_name index_num
```

tbl_name アーギュメントは MyISAM テーブルの名前であるべきです。インデックス ファイルに名前をつけることでテーブルを特定することができます。(MYI 接尾辞のついたファイル)。テーブル ファイルが存在しているディレクトリ内で myisam_ftdump を起動しない場合、テーブル データベース ディレクトリへのパスを含んだファイル名が、テーブルがインデックス ファイルの名前より先行していなければいけません。インデックス ナンバーは0で始まりま

す。

例:例えば test データベースに、以下の定義を持つ mytexttable というテーブルが含まれるとします。

```
CREATE TABLE mytexttable
(
  id INT NOT NULL,
  txt TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
);
```

id のインデックスは0であり、txt の FULLTEXT インデックスは1になります。ワーキングディレクトリが test データベースディレクトリである場合、以下の様に myisam_ftdump を起動してください。

```
shell> myisam_ftdump mytexttable 1
```

test データベース ディレクトリへのパスを含んだファイル名が /usr/local/mysql/data/test の場合、そのパスを含んだファイル名を使用して、テーブル ネーム アーギュメントを特定することができます。これは、データベース ディレクトリ内で myisam_ftdump を起動しない場合役に立ちます。

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

myisam_ftdump は以下のオプションを理解します。

- --help, -h -?

ヘルプ メッセージを表示し、閉じます。

- `--count, -c`

一語ごとの統計を計算します。(カウントとグローバル ウェイト)

- `--dump, -d`

データオフセットやワードウェイトを含むインデックスをダンプします。

- `--length, -l`

長さの分布をレポートします。

- `--stats, -s`

グローバル インデックス統計をレポートします。他にオペレーションが特定されていない場合、これがデフォルト オペレーションになります。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに出力をプリントアウトする。

myisamchk — MyISAM テーブル メンテナンス ユーティリティ

myisamchk ユーティリティはユーザのデータベース テーブルの情報を収集し、チェック、修復、もしくは最適化します。**myisamchk** は **MyISAM** テーブルとともに作動します(データやインデックスを記憶するための **.MYD** や **.MYI** テーブル)。

myisamchk は以下のように起動してください。

```
shell> myisamchk [options] tbl_name ...
```

options は **myisamchk** にどういった作業をさせたいか特定します。以下のセクションで説明されています。**myisamchk --help** を起動することでオプションのリストを取得することができます。

オプションがない場合、**myisamchk** をデフォルト オペレーションとして、ユーザのテーブルをチェックします。さらに情報を取得もしくは **myisamchk** に修復行動をとらせるには、以下のディスカッションで説明されているようにオプションを特定してください。

tbl_name はユーザがチェックもしくは修復したいデータテーブルになります。データベース ディレクトリ内以外で **myisamchk** を起動させた場合、必ずデータベース ディレクトリへのパスを特定しなければいけません。これは、**myisamchk** にはデータベース ディレクトリの場所がまったくわからないからです。実際、**myisamchk** にとっては作業中のファイルがデータベース ディレクトリに含まれていようがいまいが、とくに関係ありません。データベース テーブルに対応するファイルは他の場所へコピーして、そこでリカバリ オペレーションをそれらファイルにかけることができます。

myisamchk コマンド ライン上の複数のテーブルに名前をつけることができます。インデックス ファイルに名前をつけることでテーブルを特定することができます。(**.MYI** 接尾辞のついたファイル)。 ***.MYI** パターンを使用することによって、ディレクトリ内の全てのテーブルを特定できるようになります。例えば、データベース ディレクトリ内にいるとき、以下の様にしてそのディレクトリ内の **MyISAM** テーブルをチェックすることができます。

```
shell> myisamchk *.MYI
```

データベース ディレクトリ内にいない場合、ディレクトリへのパスを特定することで全てのテーブルをチェックすることができます。

```
shell> myisamchk /path/to/database_dir/*.MYI
```

MySQL データ ディレクトリへのパスと共にワイルドカードを特定することで、全てのデータベースの全てのテーブルをチェックすることができます。

```
shell> myisamchk /path/to/datadir/*.MYI
```

全ての**MyISAM**テーブルをチェックするお勧めの方法は以下のとおりです。

```
shell> myisamchk --silent --fast /path/to/datadir/*.*.MYI
```

全ての **MyISAM** テーブルをチェックし、破壊されているものを修復したい場合、以下のコマンドを使用してください。

```
shell> myisamchk --silent --force --fast --update-state \
--key_buffer_size=64M --sort_buffer_size=64M \
--read_buffer_size=1M --write_buffer_size=1M \
/path/to/datadir/*.*.MYI
```

このコマンドの使用は 64MB 以上がフリーであることが前提です。**myisamchk** とメモリのアロケーションについては、詳しくは **myisamchkメモリ使用量** を参照してください。

myisamchk を起動中に、他のプログラムがテーブルを使用していないことを確認してください。でなければ、**myisamchk** を起動したとき、以下のエラーが表示されます。

```
warning: clients are using or haven't closed the table properly
```

これはまだファイルが閉じられていない、もしくは閉じる前に破壊されてしまった他のプログラムによって、更新されたテーブルをチェック使用としていることを意味しています。(たとえば **mysqld** サーバ)。

もし **mysqld** が起動している場合、**FLUSH TABLES** を使用して、メモリにバッファされている全てのテーブルに加えられた修正を、強制的に一気に消去し鳴けリアいけません。**myisamchk** を起動中に、他のユーザがテーブルを使用していないことを確認してください。この問題を避けるには、テーブルのチェックには **myisamchk** の代わりに **CHECK TABLE** を使用してください。

myisamchk 一般的なオプション

このセクションで紹介されているオプションは **myisamchk** によって実行される全てのテーブル メンテナンス オペレーションに使用することができます。このセクション以降のセクションは特定のオペレーションに関するオプションのみを説明します。例えば、テーブルのチェックや修復。

- **--help, -?**

ヘルプ メッセージを表示し、閉じます。

- **--debug=debug_options, -# debug_options**

デバッグのログを書き込みます。**debug_options** 文字列は大抵 **'d:t:o,file_name'** になります。

- **--silent, -s**

サイレントモード。エラーが発生したときのみアウトプットを書き込みます。**-s** は **myisamchk** を非常にサイレントにするため 2 回(**-ss**) 使用することができます。

- **--verbose, -v**

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。これは **-d** と **-e** と共に使用することができます。さらに出力を得るため、**-v** を複数回(**-vv**、**-vvv**) 使用してください

- **--version, -V**

バージョン情報を表示し、閉じます。

- **--wait, -w**

テーブルがロックされている場合に、エラーとして消去するよりも、再開はテーブルがアンロックされるまで待つてください。外部ロックを無効の状態 **mysqld** を起動している場合、テーブルは **myisamchk** コマンドを使用することでのみ、ロックすることができます。

--var_name=value 構文を使用することで以下の構文をセットすることができます。

変数	デフォルト値

decode_bits	9
ft_max_word_len	version-dependent
ft_min_word_len	4
ft_stopword_file	built-in list
key_buffer_size	523264
myisam_block_size	1024
read_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
stats_method	nulls_unequal
write_buffer_size	262136

可能な `myisamchk` 変数とデフォルト値は `myisamchk --help` で確認することができます。

`sort_buffer_size` はソートキーでキーが修復された場合に使用されます。これは `--recover` を使用した場合は普通です。

`key_buffer_size` は `--extend-check` でテーブルをチェックするとき、もしくはテーブルに行ごとにキーを入力することでキーを修復する際に使用されます(普通にインサートする場合の様に)。キーバッファを通しての修復は以下の場合使用されます。

- `--safe-recover` を使用する場合。
- キーのソートに必要なテンポラリ ファイルは直接キーファイルを作成する際よりも、倍の大きさとなります。これは `CHAR`、`VARCHAR`、もしくは `TEXT` カラムに大きなキー値が与えられている場合によくあります。これは、進行するにつれてソート オペレーションは完全なキー値を記憶する必要があるからです。テンポラリ スペースに余裕があり、`myisamchk` を使用して強制的にソートすることで修復する場合、`--sort-recover` オプションを使用することができます。

キーバッファを使用して修復するのはソートよりもはるかにディスクの空き容量に余裕を持たせることができますが、速度も落ちます。

修復スピードを早くする場合、`key_buffer_size` と `sort_buffer_size` 変数を使用できるメモリの25%にセットしてください。同時に使用されることは無いので、双方の変数の値を大きくセットすることができます。

`myisam_block_size` はインデックス ブロックに使用されるサイズです。

`stats_method` は `NULL` 値が、`--analyze` オプションを与えられたときに、インデックス統計の集計の際にどう扱われるかに影響します。`myisam_stats_method` システム変数のような働きをします。詳しくは、in [システム変数](#) と [MyISAMインデックス統計コレクション](#) に含まれる `myisam_stats_method` の詳細を参照してください。

`ft_min_word_len` と `ft_max_word_len` は `FULLTEXT` インデックスの最低と最高文字長を指します。`ft_stopword_file` はストップワードファイルに名前をつけます。以下の状態でこれらをセットしなければいけません。

`myisamchk` を使用してテーブル インデックスを改良するオペレーションを実行する場合、(例えば分析や修復)他に特定されない場合、`FULLTEXT` インデックスはデフォルト フル テキスト パラメータ値を、最低・最高文字長とストップワードファイルを使用して再構築されます。これはクエリの失敗につながる可能性があります。

この問題は、パラメータがサーバのみに理解されていることにより発生します。`MyISAM` インデックスファイルには記憶されていません。サーバ内の最低・最高文字長もしくはストップワードファイルを改良した上で、問題を回避するには、`ft_min_word_len`、`ft_max_word_len`と`ft_stopword_file` の値を `mysqld` で使用する `myisamchk` に特定してください。例えば、最低文字長を3にセットした場合、テーブルは `myisamchk` で以下の様に修復できます。

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

`myisamchk` とサーバがフル テキスト パラメータに同じ値を確実に使用するには、それぞれオプションファイルの `[mysqld]` と `[myisamchk]` セクションに置いてください。

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

`myisamchk` を使用する代わりに、`REPAIR TABLE`、`ANALYZE TABLE`、`OPTIMIZE TABLE`、もしくは `ALTER TABLE` を使用することができます。これらのステートメントは、正しいフル テキスト パラメータ値を理解しているサーバによって実行されます。

myisamchk チェック オプション

`myisamchk` はテーブルチェックオペレーション用の以下のオプションをサポートしています。

- `--check, -c`

テーブルにエラーが無いかチェックします。明示的にオペレーションタイプを選択しない場合、これがデフォルトオペレーションとなります。

- `--check-only-changed, -C`

前チェックより変更されたテーブルのみをチェックする。

- `--extend-check, -e`

テーブル チェックを入念に行います。テーブルにインデックスが多数ある場合、この作業には時間がかかります。このオプションは極端な処置として使用するべきです。通常、`myisamchk` と `myisamchk --medium-check` を使用してテーブル内のエラーの有無を確認することができます。

`--extend-check` を使用していてメモリ容量も十分な場合、`key_buffer_size` 値を大きくセットすれば、修復オペレーションのスピードを上げることができます。

- `--fast, -F`

正しく閉じられていないテーブルのみをチェックする。

- `--force, -f`

`myisamchk` がエラーをテーブル内で発見した場合、自動的に修復オペレーションを実行する。修復タイプは `--recover` や `-r` オプションで特定されたものと同一です。

- `--information, -i`

チェックされたテーブルの統計情報をプリントします。

- `--medium-check, -m`

`--extend-check` オペレーションよりも速いチェックを行います。これはエラーの99.99%をチェックし、ほとんどの場合において十分な成果を発揮します。

- `--read-only, -T`

テーブルをチェックされたものとして処理しません。`myisamchk` を使用して、ロックを使用しないアプリケーションによって使用されているテーブルをチェックする際に便利です。例えば、外部ロックを無効にした状態で `mysqld` が作動している場合。

- `--update-state, -U`

情報を `.MYI` ファイルに記憶し、いつテーブルがチェックされたか、そしてテーブルがクラッシュしているかをチェックします。これを使用することによって `--check-only-changed` オプションの利便性を最大限に引き出すことができますが、`mysqld` サーバがテーブルを使用し、且つ外部ロックを無効にした状態で起動している場合、このオプションを使用しないでください。

myisamchk修復オプション

myisamchk はテーブル修復オペレーション用の以下のオプションをサポートしています。

- `--backup, -B`

.MYD ファイルのバックアップを `file_name-time.BAK` として作成する。

- `--character-sets-dir=path`

キャラクタ セットがインストールされるディレクトリです。データおよびソート用キャラクタ セット を参照してください。

- `--correct-checksum`

テーブルのチェックサム情報を修正する。

- `--data-file-length=len, -D len`

データファイルの最高長さ(「full」時データファイルを再作成する場合)。

- `--extend-check, -e`

データファイルから全ての行をリカバーする修復を試みる。これは通常、役に立たない行も取り込んでしまう。よほど切羽詰っていない限り、このオプションは使用しないでください。

- `--force, -f`

アボートする代わりに、古い中間ファイルをオーバーライトする(`tbl_name.TMD` などの名前を持つファイル)

- `--keys-used=val, -k val`

myisamchk では、オプション値はどのインデックスを更新すべきかを指し示す、ビット値です。オプション値の各バイナリビットは、最初のインデックスがビット0のところ、テーブルインデックスと対応しています。0のオプション値は全てのインデックスへの更新を無効にし、これによりインサートのスピードを上げることができます。無効化されたインデックスは `myisamchk -r` を使用して有効化することができます。

- `--max-record-length=len`

myisamchk が記憶するためのメモリを確保できない場合、ある一定の長さの行をスキップします。

- `--parallel-recover, -p`

`-r` と `-n` 同じテクニックを使いますが、異なるスレッドを使用して全てのキーをパラレルで作成します。これはベータ級コードです。ユーザの責任で使用してください。

- `--quick, -q`

データファイルを改良しないことで、修復のスピードを上げられます。このオプションを2回指定することで、複製キーの場合 `myisamchk` を使用して強制的にオリジナルデータファイルを改良させることができます。

- `--recover, -r`

ユニークではなユニークキー以外のほぼ全ての問題を解決することのできる修復を実行します(MyISAM テーブルでこのエラーが発生する可能性は非常に低いです)。テーブルをリカバーしたい場合、このオプションを先に試してください。`--safe-recover`は、`myisamchk` が `--recover` を使用してもテーブルがリカバーできないと報告した場合のみ、試すべきです。(非常に稀ではありますが、`--recover` が失敗した場合、データファイルは無事なままです。)

メモリ容量に余裕がある場合、`sort_buffer_size` の値を増やすことをお勧めします。

- `--safe-recover, -o`

発見した行に基づいて、全ての行を読みこみ全てのインデックストリーを更新する古いリカバリ手段をとります。この方法は `--recover` よりも格段に遅くなりますが、`--recover` では対応できない稀なケースにも対応できます。このリカバリメソッドは `--recover` よりもはるかに少ないディスクスペースを使用します。通常、まず `--recover` を使用して修復を行い、`--recover` が失敗した場合、`--safe-recover` を使用してください。

メモリ容量に余裕がある場合、`key_buffer_size` の値を増やすことをお勧めします。

- `--set-collation=name`

テーブル インデックスのソートに使用する照合順序を特定します。照合順序名の初めの部位がキャラクタセット名を示しています。

- `--sort-recover, -n`

テンポラリ ファイルのサイズが大きくなりますが、キーの解決に強制的に `myisamchk` にソートを使用させます。

- `--tmpdir=path, -t path`

テンポラリ ファイルのソートに使用されるディレクトリのパスです。これがセットされていない場合、`myisamchk` は `TMPDIR` 環境変数の値を使用します。`tmpdir` はテンポラリファイルの作成のためラウンドロビン形式でディレクトリ パスのリストをセットするために用いられます。ディレクトリ名の間にあるキャラクタは、ユニックス上ではコロン(:)であり、ウィンドウズ、Netware、OS/2 ではセミコロン(';') になります。

- `--unpack, -u`

`myisamchk` でバックされたテーブルをアンパックします。

他の `myisamchk` オプション

`myisamchk` はテーブルチェックや修復以外のアクションを行う、以下のオプションをサポートしています。

- `--analyze, -a`

キー値の分布を分析します。ジョイン最適マイザに、どのテーブルを結合し、どのインデックスを使用するかより効率的に選択させることで結合パフォーマンスを向上させます。To キー分布の情報を取得するためには、`myisamchk --description --verbose tbl_name` コマンドか `SHOW INDEX FROM tbl_name` ステートメントを使用してください。

- `--block-search=offset, -b offset`

ある特定のオフセットのブロックが属する記録を見つけなさい。

- `--description, -d`

テーブルの説明を含む情報をプリントします。

- `--set-auto-increment[=value], -A[value]`

`AUTO_INCREMENT` ナンバリングを強制して新しい行がある値で始まるようにします(あるいは、存在する行の `AUTO_INCREMENT` 値が大きい場合、さらに大きい値で始まるようにします。)もし 値 が特定されていない場合、新しい行の `AUTO_INCREMENT` 数字は現在テーブル内の最も高い値 + 1 になります。

- `--sort-index, -S`

高い順にインデックストリーブロックをソートします。これによりシークが最適化され、インデックスを使用するテーブルスキャンのスピードが上がります。

- `--sort-records=N, -R N`

特定のインデックスに基づいて行をソートします。これによりデータがさらにローカライズされ、このインデックスを使用する、レンジに基づく `SELECT` や `ORDER BY` オペレーションのスピードが上がります。(初めてテーブルをソートするのにこのオプションを使用する場合、かなり遅い場合があります。)テーブルのインデックス数字を決定するには、`myisamchk` が見た同じ順序でテーブルのインデックスを表示する `SHOW INDEX` を使用してください。インデックスは1から番号がふられます。

キーがバックされていない場合、(`PACK_KEYS=0`) 同じ長さになります。よって、`myisamchk` が行を移動もしくはソートするとき、インデックスの行オフセットを上書きします。キーがバックされている場合、(`PACK_KEYS=1`) `myisamchk` はまずキーブロックをアンパックし、それからインデックスを再作成、キーブロック

をバックしなければいけません。(この場合、各インデックスのオフセットを更新するよりも、インデックスを再作成するほうが早いです。)

myisamchkメモリ使用量

myisamchkを起動させるとき、メモリの割り当ては重要です。myisamchkはセットされたメモリ関連の変数以上のメモリを使用しません。myisamchkを大きなテーブルで使用する場合は、まずどのくらいのメモリを使用するか決定しなければいけません。デフォルトで、修復には3MBくらいしか使用しないように設定されています。さらに大きな値を設定することで、myisamchkのオペレーション速度を上げることができます。例えば、32MB RAMよりも多くメモリがある場合、以下のオプション(他に特定したオプションのほかに)を使用することができます。

```
shell> myisamchk --sort_buffer_size=16M --key_buffer_size=16M \
--read_buffer_size=1M --write_buffer_size=1M ...
```

--sort_buffer_size=16Mを使用すれば、ほとんどの場合ことたります。

myisamchkはTMPDIR内でテンポラリファイルを使用することに注意してください。もしTMPDIRがメモリファイルシステムを指した場合、メモリエラーから簡単に脱出できます。これが起こった場合、スペースに余裕のあるファイルシステム上のディレクトリを指定するよう、myisamchkを--tmpdir=pathオプションと使用してください。

修復時、myisamchkもディスクスペースを大量に必要とします。

- データファイルの大きさを2倍にしてください(オリジナルファイルと複製を作成してください)。--quickで修復を行った場合スペースは必要ありません。この場合、再作成されるのはインデックスファイルのみです。このスペースはオリジナルのデータファイルと同じシステムで必要になります！(コピーはオリジナルと同じディレクトリで作成されます。)
- 古いインデックスファイルの代わりとなる新しいインデックスファイルのスペース修復オペレーションの始めに古いインデックスファイルは切り捨てられるため、大抵はこのスペースは無視されます。このスペースはオリジナルのインデックスファイルと同じシステムで必要になります！
- recoverあるいは--sort-recoverを使用しているとき、(しかし--safe-recoverは使用していないとき)ソートバッファのスペースが必要となります。以下の式は必要なスペースの量を生み出します。

```
(largest_key + row_pointer_length) × number_of_rows × 2
```

キーの長さとしてrow_pointer_lengthをmyisamchk -dv tbl_nameでチェックすることができます。このスペースはテンポラリディレクトリにアロケートされています。(TMPDIRが--tmpdir=pathによって特定されています)。

修復中、ディスクスペースが足りなくなった場合、--recoverの代わりに--safe-recoverを使用してみてください。

myisamlog — Display MyISAM Log File Contents

myisamlogはMyISAMログファイルの内容を処理します。

myisamlogは以下のように起動してください。

```
shell> myisamlog [options] [log_file [tbl_name] ...]
```

デフォルトオペレーションは更新(-u)になります。リカバリが実行された場合、(-r), 全ての書き込み、そして更新や削除も実行され、エラーは数えられるだけとなります。デフォルトログファイル名は、log_fileアーギュメントが存在しない場合、myisam.logとなります。コマンドライン上でテーブルに名前が与えられた場合、そのテーブルのみが更新されます。

myisamlogは以下のオプションを理解します。

- ?, -l
ヘルプメッセージを表示し、閉じます。
- c N
Nコマンドのみ実行します。
- f N

開かれているファイルの最大数を特定します。

- `-i`

閉じる前に余分な情報を表示します。

- `-o offset`

始まりのオフセットを特定します。

- `-p N`

パスからNコンポーネントを取り除きます。

- `-r`

リカバリオペレーションを実行します。

- `-R record_pos_file record_pos`

行ポジションファイルと行ポジションを特定します。

- `-u`

更新オペレーションを実行します。

- `-v`

Verbose モードプログラムの動作についてさらに出力をプリントアウトする。このオプションをさらに多くの出力を生成するために複数回提供することができます。

- `-w write_file`

書き込みファイルを特定します。

- `-V`

バージョン情報を表示します。

myisampack — 圧縮された、読み取り専用MyISAM テーブルを作成する。

myisampackユーティリティはMyISAMテーブルを圧縮します。myisampackはテーブル内の各カラムを別個に圧縮することで作動します。通常、myisampackは40%-70%データファイルをバックします。

テーブルが後で使用される場合、カラムの解凍に必要な情報をサーバがメモリ内に読み込みます。これにより個別の行をアクセスする際のパフォーマンスが向上します。これは 1つの行のみ解凍するだけでことたりるからです。

MySQLは圧縮されたテーブルでメモリのマッピングが可能な場合、`mmap()`を使用します。もし`mmap()`が使用できない場合、MySQLは普通のファイル読み込み・書き込みオペレーションを使用します。

以下の点に注意してください。

- もしmysqldサーバが外部ロックが無効化された状態で起動された場合、バックの最中にテーブルが更新される可能性があるため、myisampackの起動はお勧めできません。サーバが停止している最中にテーブルを圧縮するのが安全です。
- テーブルをバックした場合、読み取り専用となります。通常これは意図して行います(たとえばCD内のバックされたテーブルにアクセスする際)。バックされたテーブル内で書き込みを許容できるようにする予定はありますが、優先順位は高くありません。
- myisampackはBLOBあるいはTEXTカラムをバックできます。(ISAMテーブルの古いpack_isamプログラムにはこの機能はありませんでした。)

myisampackは以下のように起動してください。

```
shell> myisampack [options] file_name ...
```

各ファイル名アーギュメントはインデックスファイル(.MYI) 名であるべきです。ユーザがデータベースディレクトリにいない場合、ファイルへのパスネームを特定してください。[.MYI拡張の省略は許容されています。](#)

myisampackを使用してテーブルを圧縮した後、インデックスを再生するため[myisamchk -rq](#)を使用してください。
[myisamchk — MyISAM テーブル メンテナンス ユーティリティ](#)。

myisampackは次のオプションをサポートします。

- `--help, -?`

ヘルプ メッセージを表示し、閉じます。

- `--backup, -b`

`tbl_name.OLD`を使用して各テーブルのデータファイルのバックアップを作成します。

- `--character-sets-dir=path`

キャラクタ セットがインストールされるディレクトリです。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。`debug_options` 文字列は大抵 'd:t:o,file_name'になります。

- `--force, -f`

元のテーブルより大きくなるか、以前のmyisampack起動からのインターミディエートファイルが存在する場合も、バックされたテーブルを生成します。(myisampackは、テーブル圧縮時、`tbl_name.TMD`と名づけられたインターミディエートファイルをデータベースディレクトリ内に作成します。myisampackを抹消した場合、`.TMD`ファイルは削除されていないかもしれません。)通常、myisampackは`tbl_name.TMD`が存在する場合、エラーが発生し閉じます。`--force`では、myisampackはかまわずテーブルをバックします。

- `--join=big_tbl_name, -j big_tbl_name`

コマンドライン上の全テーブルを1つのテーブルbig_tbl_nameに結合します。結合されるテーブルは、必ず同一の構成でなければいけません(同一カラム名、型、同一インデックス等など)。

- `--packlength=len, -p len`

バイトで行長さ保存サイズを特定します。値は1、2、あるいは3であるべきです。myisampackは全ての行を1、2、あるいは3バイトの長さポインタで記憶します。通常、myisampackはファイルのバックをはじめる前に正しい長さの値を割り出せますが、さらに短い長さを使用することができた場合でも、バック処理の最中に気づかない可能性がありあます。この場合、myisampackは次回同じファイルをバックする際、さらに短い行長さを使用することができるとを示すノートをプリントします。

- `--silent, -s`

サイレントモード。エラーが発生したときのみアウトプットを書き込みます。

- `--test, -t`

実際にテーブルをバックせず、バックのテストを行います。

- `--tmpdir=path, -T path`

名づけられたディレクトリを、myisampackがテンポラリファイルを作成する場所に使用します。

- `--verbose, -v`

Verbose モードバックオペレーションの進捗具合と結果を示す情報を書き出します。

- `--version, -V`

バージョン情報を表示し、閉じます。

- `--wait, -w`

テーブルが使用中の場合、待ってから再度試みます。もしmysqldサーバが外部ロックが無効化された状態で起動された場合、パッキングの最中にテーブルが更新される可能性があるため、myisampackの起動はお勧めできません。

以下のコマンドシーケンスは典型的なテーブル圧縮セッションを表しています。

```
shell> ls -l station.*
-rw-rw-r-- 1 monty my 994128 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 monty my 53248 Apr 17 19:00 station.MYI
-rw-rw-r-- 1 monty my 5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file: station
Isam-version: 2
Creation time: 1996-03-13 10:08:58
Recover time: 1997-02-02 3:06:43
Data records: 1192 Deleted blocks: 0
Datafile parts: 1192 Deleted data: 0
Datafile pointer (bytes): 2 Keyfile pointer (bytes): 2
Max datafile length: 54657023 Max keyfile length: 33554431
Recordlength: 834
Record format: Fixed length

table description:
Key Start Len Index Type Root Blocksize Rec/key
1 2 4 unique unsigned long 1024 1024 1
2 32 30 multip. text 10240 1024 1

Field Start Length Type
1 1 1
2 2 4
3 6 4
4 10 1
5 11 20
6 31 1
7 32 30
8 62 35
9 97 35
10 132 35
11 167 4
12 171 16
13 187 35
14 222 4
15 226 16
16 242 20
17 262 20
18 282 20
19 302 30
20 332 4
21 336 4
22 340 1
23 341 8
24 349 8
25 357 8
26 365 2
27 367 2
28 369 4
29 373 4
30 377 1
31 378 2
32 380 8
33 388 4
34 392 4
35 396 4
36 400 4
37 404 1
```



```

38 405 4
39 409 4
40 413 4
41 417 4
42 421 4
43 425 4
44 429 20
45 449 30
46 479 1
47 480 1
48 481 79
49 560 79
50 639 79
51 718 79
52 797 8
53 805 1
54 806 1
55 807 20
56 827 4
57 831 4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal: 20 empty-space: 16 empty-zero: 12 empty-fill: 11
pre-space: 0 end-space: 12 table-lookups: 5 zero: 7
Original trees: 57 After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> ls -l station.*
-rw-rw-r-- 1 monty my 127874 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 monty my 55296 Apr 17 19:04 station.MYI
-rw-rw-r-- 1 monty my 5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file: station
Isam-version: 2
Creation time: 1996-03-13 10:08:58
Recover time: 1997-04-17 19:04:26
Data records: 1192 Deleted blocks: 0
Datafile parts: 1192 Deleted data: 0
Datafile pointer (bytes): 3 Keyfile pointer (bytes): 1
Max datafile length: 16777215 Max keyfile length: 131071
Recordlength: 834
Record format: Compressed

table description:
Key Start Len Index Type Root Blocksize Rec/key
1 2 4 unique unsigned long 10240 1024 1
2 32 30 multip. text 54272 1024 1

Field Start Length Type Huff tree Bits
1 1 1 constant 1 0
2 2 4 zerofill(1) 2 9
3 6 4 no zeros, zerofill(1) 2 9
4 10 1 3 9
5 11 20 table-lookup 4 0
6 31 1 3 9
7 32 30 no endspace, not_always 5 9
8 62 35 no endspace, not_always, no empty 6 9
9 97 35 no empty 7 9
10 132 35 no endspace, not_always, no empty 6 9
11 167 4 zerofill(1) 2 9
12 171 16 no endspace, not_always, no empty 5 9
13 187 35 no endspace, not_always, no empty 6 9
14 222 4 zerofill(1) 2 9
15 226 16 no endspace, not_always, no empty 5 9
16 242 20 no endspace, not_always 8 9
17 262 20 no endspace, no empty 8 9
18 282 20 no endspace, no empty 5 9
19 302 30 no endspace, no empty 6 9

```

20	332	4	always zero	2	9
21	336	4	always zero	2	9
22	340	1		3	9
23	341	8	table-lookup	9	0
24	349	8	table-lookup	10	0
25	357	8	always zero	2	9
26	365	2		2	9
27	367	2	no zeros, zerofill(1)	2	9
28	369	4	no zeros, zerofill(1)	2	9
29	373	4	table-lookup	11	0
30	377	1		3	9
31	378	2	no zeros, zerofill(1)	2	9
32	380	8	no zeros	2	9
33	388	4	always zero	2	9
34	392	4	table-lookup	12	0
35	396	4	no zeros, zerofill(1)	13	9
36	400	4	no zeros, zerofill(1)	2	9
37	404	1		2	9
38	405	4	no zeros	2	9
39	409	4	always zero	2	9
40	413	4	no zeros	2	9
41	417	4	always zero	2	9
42	421	4	no zeros	2	9
43	425	4	always zero	2	9
44	429	20	no empty	3	9
45	449	30	no empty	3	9
46	479	1		14	4
47	480	1		14	4
48	481	79	no endspace, no empty	15	9
49	560	79	no empty	2	9
50	639	79	no empty	2	9
51	718	79	no endspace	16	9
52	797	8	no empty	2	9
53	805	1		17	1
54	806	1		3	9
55	807	20	no empty	3	9
56	827	4	no zeros, zerofill(2)	2	9
57	831	4	no zeros, zerofill(1)	2	9

myisampackは以下の種類の情報を表示します。

- ノーマル

余分なバックが使用されていないカラムの数。

- empty-space

スペースの値のみ含むカラムの数これらは1ビットの容量を占めています。

- empty-zero

バイナリゼロの値のみ含むカラムの数これらは1ビットの容量を占めています。

- empty-fill

それぞれの型のバイトレンジを全て占領しない整数カラムの数。これらは小さいタイプに変えられます。例えば、BIGINT カラム (8バイト)の全ての値が-128 から 127 の範囲内にある場合は、このカラムを TINYINT カラム(1バイト)として格納する事ができます。

- pre-space

リードスペースで記憶される小数点カラムの数。この場合、各値はリードスペースの数に対して数値を含んでいます。

- end-space

トレールスペースを多く含むカラムの数。この場合、各値はトレールスペースの数に対して数値を含んでいます。

- table-lookup

ハフマン圧縮の前にENUMに変換された、少量の異なる値が絡むにありました。

- [zero](#)

全ての値がゼロのカラムの数です。

- [オリジナルトリー](#)

もともとのハフマントリーの数です。

- [アフタージョイン \(結合 \)](#)

ヘッダースペース温存のため、トリーの結合の後に残った明確なハフマントリーの数です。

テーブルが圧縮された後、[myisamchk -dvv](#)は各カラムの追加情報をプリントします。

- [タイプ](#)

データ型。この値は以下のどれかの記述子を含んでいます。

- [constant](#)

全ての行は同じ値を持っています。

- [no endspace](#)

エンドスペースを記憶しません。

- [no endspace, not_always](#)

エンドスペースを記憶せず、また全ての値にエンドスペース圧縮を行いません。

- [no endspace, no empty](#)

エンドスペースを記憶しません。空の値を記憶しません。

- [table-lookup](#)

カラムは[ENUM](#)に変換されています。

- [zerofill\(N\)](#)

値の中の最も重要なNバイトは常に0であり、記憶されません。

- [no zeros](#)

ゼロを記憶しません。

- [always zero](#)

1ビットを使用してゼロ値が記憶されます。

- [Huff tree](#)

カラムに関連しているハフマントリーのナンバーです。

- [Bits](#)

ハフマントリーで使用されているビット数です。

[myisampack](#)起動後、インデックスを再作成するには[myisamchk](#)を起動しなければいけません。このとき、MySQLオプティマイザの作動効率化を図りたい場合、インデックスブロックのソートと統計の作成を行うことができます。

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

MySQLデータベースディレクトリにパックされたテーブルをインストールした後、[mysqld](#)に新しいテーブルの使用を強制するため、[mysqladmin flush-tables](#)を実行してください。

パックされたテーブルをアンパックする場合、`myisamchk`に対して`--unpack`オプションを使用してください。

mysql — MySQL コマンド ライン ツール

`mysql`は単純な(GNU `readline`機能を装備した。)SQLシェルです。インタラクティブ・ノンインタラクティブ使用の両方をサポートします。インタラクティブの場合、ASCII-テーブルフォーマットでクエリの結果が提示されます。ノンインタラクティブの場合(例えばフィルターとして)、タブによって分けられたフォーマットで結果が提示されます。出力フォーマットはコマンドオプションを使用することで変更することができます。

大きな結果セット用のメモリが足りないことで問題が発生している場合、`--quick`オプションを使用してください。これにより、`mysql`は全結果セットを取得、メモリ内でバッファ後表示といった一連の作業を一気にこなさず、サーバから1行ずつ結果を取得します。これは、`mysql_store_result()`よりも、クライアントサーバ内の`mysql_use_result()` C API ファンクションを使用して結果セットを返すことで実行できます。

`mysql`の使用は簡単です。以下の様に、コマンドインタプリタのプロンプトから起動してください。

```
shell> mysql db_name
```

または

```
shell> mysql --user=user_name --password=your_password db_name
```

そこでSQLステートメントを書き、`;`、`\g`、または`\G`で終わらせEnterを押してください。

MySQL 5.1.10以降、コントロールCを押すことで`mysql`に現在のステートメントの消去を命令します。これが実行できない場合、あるいはステートメントが消去される前にコントロールCが押された場合、`mysql`は閉じます。以前では、コントロールCを押すと`mysql`は全ての場合において閉じました。

SQLステートメントは以下の様に、スクリプトファイル (バッチファイル) 形式で実行できます。

```
shell> mysql db_name < script.sql > output.tab
```

mysql オプション

`mysql` は次のオプションをサポートします。

- `--help, -?`

ヘルプ メッセージを表示し、閉じます。

- `--auto-rehash`

自動リハッシュを有効化します。このオプションはデフォルトではオンに設定されており、テーブル・カラム名の終了を可能にしています。リハッシュを無効化するには、`--skip-auto-rehash`を使用してください。これにより`mysql`の起動が早くなりますが、テーブル・カラム名を終了させたい場合、`rehash`コマンドを発行しなければいけません。

- `--batch, -B`

タブとして、カラムセパレータを使用することで、各行が新しいライン上に配置されるように、結果をプリントします。このオプションでは、`mysql`はヒストリファイルを使用しません。

- `--character-sets-dir=path`

キャラクタ セットがインストールされるディレクトリです。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--column-names`

結果にカラム名を記述します。

- `--compress, -C`

双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。

- `--database=db_name, -D db_name`

使用されるべきデータベースです。これは基本的には、オプションファイルで便利です。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。`debug_options` 文字列は大抵 `'d:t:o,file_name'` になります。`'d:t:o,/tmp/mysql.trace'` がデフォルトになります。

- `--debug-info, -T`

プログラムが閉じるときに、デバッグ情報をプリントします。

- `--default-character-set=charset_name`

`charset_name` をデフォルトキャラクタセットとして使用します。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--delimiter=str`

ステートメントデリミタをセットします。デフォルトはセミコロン(';') キャラクタになります。

- `--execute=statement, -e statement`

ステートメントを実行し、やめます。デフォルトの出力フォーマットは、`--batch` で生成されるものと類似しています。例については、[コマンドラインにおけるオプションの使用](#) をご参照してください。

- `--force, -f`

SQLエラーが発生しても続けます。

- `--host=host_name, -h host_name`

与えられたホスト上でMySQLサーバに接続します。

- `--html, -H`

HTML出力を生成します。

- `--ignore-spaces, -i`

ファンクション名の後のスペースを無視します。この効果は `IGNORE_SPACE` SQLモード([SQL モード](#) を参照してください)のディスカッションで記述されています。

- `--line-numbers`

エラーの際ライン番号を書き出します。`--skip-line-numbers` を使用することで無効化することができます。

- `--local-infile[={0|1}]`

`LOAD DATA INFILE` 上で `LOCAL` 能力を有効化・無効化する。値がない場合、オプションは `LOCAL` を有効化します。オプションは `LOCAL` を明示的に有効化・無効化するため、`--local-infile=0` か `--local-infile=1` として提供されている場合があります。`LOCAL` を有効化しても、サーバがサポートしていない場合、効果はありません。

- `--named-commands, -G`

名をつけられているmysqlコマンドを有効化します。ショートフォーマットコマンドだけでなく、ロングフォーマットコマンドも許容されています。例えば、`quit` と `\q` は両方認識されます。名前つきコマンドを無効化するには、`--skip-named-commands` を使用してください。[mysql Commands](#) を参照してください。

- `--no-auto-rehash, -A`

`--skip-auto-rehash` の反対のフォーム。`--auto-rehash` の説明を参照してください。

- `--no-beep, -b`

エラー音を発生させません。

- `--no-named-commands, -g`

名前のついたコマンドを無効化します。`\*`フォームか、名前のついたコマンドはセミコロン(`;`)で終わる行の始めのみで使用してください。`mysql`はデフォルトでenabledこのオプションで起動します。ただし、このオプションを使用しても、ロングフォーマットコマンドは最初の行から効果を発揮します。[mysql Commands](#) を参照してください。

- `--no-pager`

`--skip-pager`の反対のフォーム。`--pager`オプションを参照してください。

- `--no-tee`

アウトプットをファイルへコピーしません。[mysql Commands](#), discusses tee files further.

- `--one-database, -o`

コマンドライン上で名づけられたデフォルトのデータベースのステートメント以外を無視します。これは他のバイナリログ内のデータベースの更新をスキップする場合に便利です。

- `--pager[=command]`

ページングクエリ出力にこのコマンドを使用してください。このコマンドが取り除かれている場合、ページングのデフォルトはPAGER環境変数の値となります。有効なページングは`less`、`more`、`cat [> filename]`などです。このオプションはUnix上でしか作動しません。パッチモードでは作動しません。ページングを無効化するには、`--skip-pager`を使用してください。[mysql Commands](#), には出力ページングの詳細説明があります。

- `--password[=password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(`-p`)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で`--password`あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#) を参照してください。

- `--port=port_num, -P port_num`

コネクションに使用するTCP/IPポート番号です。

- `--prompt=format_str`

プロンプトを特定のフォーマットにセットします。そのデフォルトは`mysql>`です。プロンプト内で存在しえる特別なシーケンスは、[mysql Commands](#)で紹介されています。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--quick, -q`

各クエリ結果をキャッシュせず、受け取ったとおりに各行をプリントします。出力が遅延された場合、サーバのスピードを下げる可能性があります。このオプションでは、`mysql`はヒストリファイルを使用しません。

- `--raw, -r`

エスケープ変換なしでカラム値を書きます。主に`--batch`オプションと併用されます。

- `--reconnect`

サーバとの接続が失われたとき、再接続を自動的に試みます。接続が失われるたびに一度再接続が試みられます。再接続行為を抑制するには、`--skip-reconnect`を使用してください。

- `--safe-updates, --i-am-a-dummy, -U`

キー値を使用してどの行を改良するか特定する、`UPDATE`や`DELETE`ステートメントを許容する。このオプションをオプションファイル内でセットした場合、`--safe-updates`をコマンドライン上で使用することで重ね処理することができます。これらのオプションについては、[mysql ヒント](#) をご参照してください。

- `--secure-auth`

古い(4.1.1以前)フォーマットでサーバへパスワードを送りません。接続を新しいパスワードフォーマットを使用するサーバに限定します。

- `--show-warnings`

警告が存在する場合、各ステートメント後に表示させます。このオプションはインタラクティブとバッチモードにのみ対応しています。

- `--sigint-ignore`

SIGINTシグナルを無視します(Control-Cを押すことで現れる主な結果)。

- `--silent, -s`

サイレントモード。出力生成を少なくします。このオプションをさらに少ない出力を生成するために複数回提供することができます。

- `--skip-column-names, -N`

結果にカラム名を記述しません。

- `--skip-line-numbers, -L`

エラーの際ライン番号を書き出しません。エラーメッセージを含む結果ファイルを比較したい場合に便利です。

- `--socket=path, -S path`

`localhost`の接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- `--ssl*`

`--ssl`で始まるオプションは、SSLを介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。[SSL コマンド オプション](#) を参照してください。

- `--table, -t`

出力をテーブルフォーマットで評します。インタラクティブの場合これがデフォルトになりますが、テーブル出力をバッチモードで生成するのに使用することもできます。

- `--tee=file_name`

ファイル上で出力の複製をアペンドします。このオプションはバッチモードでは作動しません。[mysql Commands](#)でteeファイルの説明を記述しています。

- `--unbuffered, -n`

各クエリ後にバッファをフラッシュします。

- `--user=user_name, -u user_name`

サーバに接続する際使用するMySQLユーザ名です。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに出力を生成します。このオプションをさらに多くの出力を生成するために複数回提供することができます。(例えば、`-v -v -v`はバッチモードでもテーブル出力フォーマットを生成します。)

- `--version, -V`

バージョン情報を表示し、閉じます。

- `--vertical, -E`

クエリ出力行を縦にプリントします。(カラム値ごとに一行)。このオプションを使用しない場合、`\G`で消去することで個々のステートメントの縦の出力を特定することができます。

- `--wait, -w`

接続ができない場合、アボートせずに休止してから再トライします。

- `--xml, -X`

XML出力を生成します。

注:MySQL 5.1.12以前では、カラム内にNULL値を含むカラムと文字列リテラルをカラム内に含む'NULL'では、出力に違いは在りませんでした。双方とも、以下のように表現されていました。

```
<field name="column_name">NULL</field>
```

MySQL 5.1.12に始まり、`--xml`がmysqlと使用されたときの出力はmysqldump `--xml`の出力とマッチします。マニュアルの[セクションを参照してください](#)。mysqldump [\[484\]](#)の詳細について、`--xml`オプションが紹介されています。

`--var_name=value` 構文を使用することで以下の構文をセットすることができます。

- `connect_timeout`

接続タイムアウトまでの秒数。(そのデフォルトは0です。)

- `max_allowed_packet`

サーバから送・受信される最大パケット長。(そのデフォルト値は16MBです。)

- `max_join_size`

自動的に設定される`--safe-updates`使用時の結合内にある行のリミットです。(そのデフォルト値は1,000,000です。)

- `net_buffer_length`

TCP/IPとソケット通信のバッファサイズ。(そのデフォルト値は16KBです。)

- `select_limit`

自動的に設定される`--safe-updates`使用時のSELECTステートメントのリミットです。(そのデフォルト値は1,000です。)

`--set-variable=var_name=value` or `-O var_name=value`構文を使用することで、変数をセットすることも可能です。構文は反対語となっています。

Unixでは、mysqlクライアントはヒストリファイルに実行されたステートメントのレコードを書きます。デフォルトによりそのヒストリファイルは`.mysql_history`と名づけられており、ホームディレクトリ内で作成されます。異なるファイルを特定したい場合、`MYSQL_HISTFILE`環境変数値をセットしてください。

ヒストリファイルを保持したくない場合、まず`.mysql_history`が存在する場合消去し、以下の手段を用いてください。

- `MYSQL_HISTFILE`変数を`/dev/null`にセットしてください。ログインするたびにこのセッティングが効果を表すようにするには、このセッティングをシェルのスタートアップファイルのいずれかに置いてください。
- `.mysql_history`を`/dev/null`に対してのシンボリックリンクとして作成してください。

```
shell> ln -s /dev/null $HOME/.mysql_history
```

これは一度だけ実行すれば事足ります。

mysql Commands

mysqlは各SQLステートメントの実行のため、ユーザが発行するSQLステートメントをサーバに送ります。**mysql**自体が理解するコマンドもあります。これらコマンドのリストが必要な場合、**mysql>**プロンプトで**help**あるいは**\h**とタイプしてください。

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?      (\?) Synonym for 'help'.
charset (\C) Switch to another charset. Might be needed for processing
        binlog with multi-byte charsets.
clear   (\c) Clear command.
connect (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter. NOTE: Takes the rest of the line as
        new delimiter.
edit     (\e) Edit command with $EDITOR.
ego      (\G) Send command to mysql server, display result vertically.
exit     (\q) Exit mysql. Same as quit.
go       (\g) Send command to mysql server.
help     (\h) Display this help.
nopager  (\n) Disable pager, print to stdout.
notee    (\t) Don't write into outfile.
pager    (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print    (\p) Print current command.
prompt   (\R) Change your mysql prompt.
quit     (\q) Quit mysql.
rehash   (\#) Rebuild completion hash.
source   (\.) Execute an SQL script file. Takes a file name as an argument.
status   (\s) Get status information from the server.
system   (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given
        outfile.
use       (\u) Use another database. Takes database name as argument.
warnings (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.

For server side help, type 'help contents'
```

各コマンドにはそれぞれロングとショートフォームがあります。ロングフォームは大文字小文字の区別をしません、ショートフォームは影響されます。ロングフォームはセミコロンによる終端器が後に続くこともありますが、ショートフォームではありません。

helpコマンドにアーギュメントを提供した場合、**mysql**はMySQLリファレンスマニュアルの内容からサーバサイドヘルプをアクセスするための検索文字列としてアーギュメントを使用します。詳細は [mysqlサーバサイドヘルプ](#) をご覧ください。

charsetコマンドは、デフォルトキャラクタセットを変更し**SET NAMES**ステートメントを発行します。これにより、**mysql**が自動再接続が有効化された状態で作動中でも、クライアント・サーバ間のキャラクタセットがシンクロしている状態を保つことができるのは、変更されたキャラクタセットが再接続に使用されるからです(これは推奨されていません)。このコマンドはMySQL 5.1.12.で追加されました。

delimiterコマンドを使用する時、MySQLに対してエスケープキャラクタとなるので、バックスラッシュ (\) キャラクタの使用を避けてください。

edit、**nopager**、**pager**、そして**system**コマンドはUnix上でのみ機能します。

statusコマンドは、ユーザが使用している接続とサーバに関する情報を提供します。**--safe-updates**モードで作動中の場合、**status**はクエリに影響する**mysql**変数の値をプリントします。

クエリとクエリの出力をログするには、**tee**コマンドを使用してください。画面上に表示されるデータは全てあるファイルにアペンドされます。これはデバッグを行う際にも非常に便利です。この機能をコマンドライン上で有効化する場合**--tee**オプションを使用するか、インタラクティブの場合は**tee** コマンドを使用してください。**tee**ファイルは**notee**コマンドでインタラクティブに無効化することができます。**tee**を実行することでロギングを再有効化します。パラメータがない場合、以前のファイルが使用されます。**mysql**が次のプロンプトをプリントする前に、**tee**は各ステートメントの後にクエリの結果をフラッシュすることに注意してください。

--pagerオプションを使用することで、インタラクティブモードではクエリ結果をブラウズ、検索することがless, moreUnix、もしくは他の似たようなプログラムで可能です。オプションで値を特定しない場合、mysqlはPAGER環境変数の値をチェックし、ページャーをその値にセットします。出力ページングはpagerコマンドでインタラクティブに有効化することができ、nopagerで無効化することができます。コマンドはオプションアルアーギュメントを採用します。与えられている場合、ページプログラムはそれにセットされます。アーギュメントがない場合、ページャーはコマンドラインにセットされているものが、ページャーが特定されていない場合stdoutになります。

出力ページングはUnix上でしか作動しません。これはWindowsでは存在しないpopen()ファンクションを使用するからです。Windowsでは、クエリ出力を記憶するのにteeオプションが使用できますが、これは場合によってはブラウジング出力用のpagerほど便利ではありません。

pagerコマンドのヒントを以下に記します。

- これを使用してファイルに書き込み、結果はファイルにのみ送られます。

```
mysql> pager cat > /tmp/log.txt
```

ユーザのページャーとして使用したいプログラムのオプションをパスすることができます。

```
mysql> pager less -n -i -S
```

- 前の例の、-Sオプションに注目してください。幅広いクエリ結果のブラウズの際便利です。スクリーン上であまりにも幅広い結果セットは読みにくい場合があります。lessに対する-Sオプションは、左右の方向キーを使用して横にスクロールすることができますので、結果セットを読みやすくします。-Sはless内で、横向きブラウズモードをオン・オフする際にインタラクティブに使用することができます。追加情報に関しては、lessマニュアルのページを参照してください。

```
shell> man less
```

- クエリアウトプットの取り扱いに関する複雑なページャーコマンドを特定することができます。

```
mysql> pager cat | tee /dr1/tmp/res.txt \
| tee /dr2/tmp/res2.txt | less -n -i -S
```

この例では、クエリ結果を/dr1と/dr2上の2つの異なるファイルシステム、ディレクトリ、そしてファイルに送信しますが、lessを介して結果を画面に表示します。

teeとpagerファンクションも合わせる事ができます。teeファイルを有効化し、pagerをlessにセットしてあれば、lessプログラムを使い結果をブラウズしつつ、同時に全てをファイルにアペンドすることができます。pagerコマンドと一緒に使用されたUnix teeとmysqlビルトインteeコマンドの違いは、teeはUnix teeが提供されていない場合でも作動します。ビルトインteeはスクリーンにプリントされているもの全てをログしますが、pagerと一緒に使用されるUnix teeはそこまでログしません。さらに、teeファイルロギングはmysql内からインタラクティブにオン・オフすることができます。これは一部のクエリだけファイルにログしたいときに有効です。

デフォルトmysql>プロンプトは再コンフィギャ可能です。プロンプトを定義する文字列は以下の特別なシーケンスを含んでいる場合があります。

オプション	説明
\v	サーババージョン
\d	デフォルトデータベース
\h	サーバホスト
\p	現TCP/IPポートかソケットファイル
\u	ユーザネーム
\U	フルのuser_name@host_nameアカウント名
\l	リテラル'\'バックスラッシュキャラクタ
\n	ニューラインキャラクタ
\t	タブキャラクタ
\	スペース(バックスラッシュの後のスペース)
_	スペース

\R	現時刻、軍隊表記(0-23)
\r	現時刻、標準表記(1-12)
\m	現時刻の分
\y	現年、2桁
\Y	現年、4桁
\D	日付 (フルで)
\s	現時刻の秒
\w	曜日名を頭3文字で(Mon, Tue, ...)
\P	am/pm
\o	月名を数字で
\O	月明を頭3文字で(Jan, Feb, ...)
\c	発行されたステートメントごとに増加するカウンター
\l	現デリミタ(5.1.12で新規追加)
\S	セミコロン
\'	シングルクォート
\"	ダブルクォート

\'他の文字が後続する場合、文字はその後続する文字に変わります。

アーギュメント無しでpromptコマンドを特定した場合、mysqlはプロンプトをmysql>のデフォルトにセットします。プロンプトはいくつかの方法でセットできます。

- 環境変数を使用してください。MYSQL_PS1環境変数をプロンプト文字列にセットすることができます。例:

```
shell> export MYSQL_PS1="(u@h) [d]> "
```

- コマンドラインオプションを使用してください。コマンドライン上の--promptオプションをmysqlにセットすることができます。例:

```
shell> mysql --prompt="(u@h) [d]> "
(user@host) [database]>
```

- オプションファイルを使用してください。どのMySQLオプションファイルの[mysql]グループにも、promptオプションをセットすることができます。例えば、ホームディレクトリにある/etc/my.cnf or the .my.cnfファイル。例:

```
[mysql]
prompt="(u@h) [d]> \_
```

この例では、バックスラッシュが2つあることに注意してください。オプションファイルでpromptオプションを使用してプロンプトをセットした場合、特別なプロンプトオプションを使用するときはバックスラッシュをダブルで使用するをお勧めします。オプションファイルに認識される、許容可能なプロンプトオプションと特別なエスケープシーケンスのセットには、重複部分があります。(これらシーケンスはオプションファイルの使用で記されます。)シングルのバックスラッシュのみを使用している場合、重複は問題となる可能性があります。例えば、\sは現秒値としてよりも、スペースとして解釈されます。以下の例ではオプションファイルを使用してプロンプトを定義することで、HH:MM:SS>フォーマットで現時刻を含む方法を示しています。

```
[mysql]
prompt="(r:\m:\s> "
```

- インタラクティブでプロンプトをセットしてください。prompt (あるいは\R)コマンドを使用することでインタラクティブにプロンプトを変更することができます。例:

```
mysql> prompt '(u@h) [d]> \_
PROMPT set to '(u@h) [d]> \_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
```

mysql>

mysqlサーバサイドヘルプ

mysql> [help search_string](#)

[help](#) コマンドにアーギュメントを提供した場合、[mysql](#) は MySQL リファレンスマニュアルの内容からサーバサイドヘルプをアクセスするための検索文字列としてアーギュメントを使用します。このコマンドの正当なオペレーションは [mysql](#) データベース内のヘルプテーブルがヘルプトピック情報で初期化されていることを要求します([サーバサイドヘルプ](#)を参照してください)。

検索文字列にマッチがない場合、検索は失敗に終わります。

mysql> [help me](#)

Nothing found
Please try to run 'help contents' for a list of all accessible topics

ヘルプカテゴリのリストを閲覧するには[help contents](#)を使用してください。

mysql> [help contents](#)

You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the following categories:
Account Management
Administration
Data Definition
Data Manipulation
Data Types
Functions
Functions and Modifiers for Use with GROUP BY
Geographic Features
Language Structure
Plugins
Storage Engines
Stored Routines
Table Maintenance
Transactions
Triggers

検索文字列にマッチが多数ある場合は、[mysql](#) はマッチするトピックのリストを表示します。

mysql> [help logs](#)

Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
SHOW
SHOW BINARY LOGS
SHOW ENGINE
SHOW LOGS

トピックのヘルプ情報を閲覧するには、トピックを検索文字列として使用してください。

mysql> [help show binary logs](#)

Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as part of the procedure described in [purge-master-logs], that shows how to determine which logs can be purged.

mysql> SHOW BINARY LOGS;

```
+-----+-----+
| Log_name | File_size |
+-----+-----+
| binlog.000015 | 724935 |
| binlog.000016 | 733481 |
+-----+-----+
```


テキストファイルからSQLステートメントを実行する

`mysql`クライアントは、以下の様にインタラクティブに使用されます。

```
shell> mysql db_name
```

しかし、SQLステートメントをファイルに入れ、`mysql`にはそのインプットをファイルから読み取るように指示することも可能です。そうすることで、実行したいステートメントを含む`text_file`を作成することができます。それから、以下で示されるように`mysql`を起動してください。

```
shell> mysql db_name < text_file
```

ファイルの最初のステートメントとして`USE db_name`ステートメントを置いた場合、コマンドライン上でデータベース名の特定は不必要になります。

```
shell> mysql < text_file
```

すでに`mysql`が作動中の場合は、`source`が`\. command`:を使用してSQLスクリプトを実行することができます。

```
mysql> source file_name
mysql> \. file_name
```

ユーザに進行状況を表示するスクリプトを使用したい場合があります。以下のようなステートメントを使用してください。

```
SELECT '<info_to_display>' AS ' ';
```

表示されたステートメントは`<info_to_display>`を出力します。

バッチモードの詳細については[Using mysql in Batch Mode](#)を参照してください。

mysqlヒント

このセクションでは`mysql`をさらに効果的に使用するテクニックを紹介します。

クエリ結果を縦に表示する

クエリ結果の中にはたて表示のほうが、横テーブルフォーマットよりも読みやすいものがあります。セミコロン（;）の代わりに`\G`がついたクエリを消去することで、クエリは縦に表示できます。例えば、新しい行を含む長いテキスト値は立て表示のほうがはるかに読みやすいです。

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
***** 1. row *****
msg_nro: 3068
date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
reply: monty@no.spam.com
mail_to: "Thimble Smith" <tim@no.spam.com>
sbj: UTF-8
txt: >>>>> "Thimble" == Thimble Smith writes:

Thimble> Hi. I think this is a good idea. Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
file: inbox-jani-1
hash: 190402944
1 row in set (0.09 sec)
```

--safe-updatesオプションを使用する

初心者にとって、使いやすいスタートアップオプションは`--safe-updates` (あるいは同じ効果のある`--i-am-a-dummy`)です。これは`DELETE FROM tbl_name`ステートメントを発行したが、`WHERE`節を忘れてしまった場合に便利です。通

常、このようなステートメントはテーブルから全ての行を消去します。`--safe-updates`では、行を表すキー値を特定することでのみ、行を消去することができます。これにより、間違いや事故を予防します。

`--safe-updates`オプションを使用するとき、`mysql`はMySQLサーバに接続した際以下のステートメントを発行します。

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=1000,SQL_MAX_JOIN_SIZE=1000000;
```

詳しくはこちらをを参照してください[SET 構文](#)。

SETステートメントには以下の効果があります。

- `UPDATE`や`DELETE`ステートメントは、`WHERE`節内でキー制限を特定するか、`LIMIT`節を提供するか(あるいは両方)しないかぎり、実行は許容されません。例:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;
```

```
UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- サーバは全ての大きい`SELECT`結果を1,000行に限定します。ただし、ステートメントに`LIMIT`節が含まれていない場合にのみです。
- サーバは、1,000,000の行コンビネーションをチェックしなければいけない、複数テーブル`SELECT`ステートメントをアボートします。

1,000 と1,000,000以外の制限を特定するには、`--select_limit`と`--max_join_size`オプションを使用することでデフォルトを重ね処理することができます。

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

mysql自動再接続の無効化

もし`mysql`クライアントが、ステートメントの送信中にサーバとの接続が遮断された場合、直ちに自動的に再接続し、ステートメントの送信を再度試みます。ただし、`mysql`が再接続に成功しても、最初の接続が遮断された時点で前セッションオブジェクトと設定は失われています。この中には、テンポラリテーブル、オートコミットモード、ユーザによって定義された変数やセッション変数も含まれています。加えて、現トランザクションロールバックも。この動作は危険な場合があります。例えば、以下の例ではサーバはユーザの了解なしに、最初のステートメントと二番目のステートメントの間に終了し、リスタートさせられています。

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+-----+
| a |
+-----+
| NULL |
+-----+
1 row in set (0.05 sec)
```

`@a`ユーザ変数は接続と共に失われ、再接続語は定義されていません。接続が失われた際に、`mysql`がエラーで終了することが望ましい場合、`mysql`クライアントを`--skip-reconnect`オプションでスタートさせることができます。

自動再接続と再接続時の状態と効果に関する追加情報は、[自動再接続挙動の管理](#)を参照してください。

mysqlaccess — アクセス権限をチェックするクライアント

`mysqlaccess`はMySQL分布のために、Yves Carlierが提供した診断ツールです。ホスト名、ユーザ名、そしてデータベースコンビネーションのアクセス権限をチェックします。`mysqlaccess`はアクセスをチェックする際、`user`、`db`、

そして`host`テーブルのみを使用します。`tables_priv`、`columns_priv`、そして`procs_priv` テーブルで特定されるテーブル、カラム、そしてルーチン権限はチェックしません。

`mysqlaccess`は以下のように起動してください。

```
shell> mysqlaccess [[host_name [user_name [db_name]]] [options]
```

`mysqlaccess`は以下のオプションを理解します。

- `--help, -?`
ヘルプ メッセージを表示し、閉じます。
- `--brief, -b`
シングル・ライン・タブフォーマットでレポートを生成します。
- `--commit`
テンポラリテーブルから元のグラントテーブルへ新しいアクセス権限をコピーします。新しい権限が発動するには、グラントテーブルはフラッシュされなければいけません。(例えば、`mysqladmin reload`コマンドを実行してください。)
- `--copy`
オリジナルから、テンポラリグラントとテーブルを再ロードします。
- `--db=db_name, -d db_name`
データベース名を特定します。
- `--debug=N`
デバッグレベルを特定します。`N`は0から3までの整数になります。
- `--host=host_name, -h host_name`
アクセス権限で使用するホスト名です。
- `--howto`
`mysqlaccess`の使用方法を示す例を表示します。
- `--old_server`
サーバが、`WHERE`節の取り扱い方を完全に理解していない、古い(MySQL 3.21以前) MySQLサーバであると仮定します。
- `--password[=password], -p[password]`
サーバに接続する際使用するパスワードです。コマンドライン上で`--password` あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。
コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#) を参照してください。
- `--plan`
未来のリリースのためのアイデアや提案を表示する。
- `--preview`
テンポラリグラントテーブルに変更を加えた後、権限の差異を表示します。
- `--relnotes`
リリースノートを表示します。

- `--rhost=host_name, -H host_name`

与えられたホスト上でMySQLサーバに接続します。

- `--rollback`

テンポラリグラントテーブルへの最も最近の変更を戻します。

- `--spassword[=password], -P[password]`

サーバに接続する際、スーパーユーザとして使用するパスワードです。コマンドライン上で`--password` あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#)を参照してください。

- `--superuser=user_name, -U user_name`

スーパーユーザとして接続する際のユーザ名を特定します。

- `--table, -t`

テーブルフォーマットでレポートを生成します。

- `--user=user_name, -u user_name`

アクセス権限で使われるユーザ名です。

- `--version, -v`

バージョン情報を表示し、閉じます。

ユーザのMySQLディストリビューションが通常とは違うロケーションにインストールされている場合、`mysqlaccess`が`mysql`クライアントを発見できる場所を変更する必要があります。`mysqlaccess` スクリプトを行18で編集してください。以下のような行を探してください。

```
$MYSQL = '/usr/local/bin/mysql'; # path to mysql executable
```

ロケーションを示すように、パスを変更してください。`mysql`は実際にユーザのシステムに記憶されています。これをしなければ、`mysqlaccess`を起動したときに`Broken pipe`エラーが発生します。

mysqladmin — MySQL サーバの管理を行うクライアント

`mysqladmin`は管理オペレーションを実行するためのクライアントです。サーバコンフィグや現在ステータスのチェックのほか、データベースの作成・破棄、他にもさまざまな用途があります。

`mysqladmin`は以下のように起動してください。

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

`mysqladmin`は以下のリストで紹介されているコマンドをサポートしています。コマンドの中にはコマンド名の後にアークギュメントが続きます。

- `create db_name`

`db_name`という名前の新しいデータベースを作成します。

- `debug`

エラーログにデバッグ情報を書き込むようにサーバに知らせます。

MySQL 5.1.12に始まり、イベントスケジューラーの情報も含まれています。[Event Scheduler Status](#)を参照してください。

- `drop db_name`

`db_name`という名前のデータベースとそのテーブルを全て削除します。

- `extended-status`

サーバステータス変数とその値を表示します。

- `flush-hosts`

ホストキャッシュ内の情報を全てフラッシュします。

- `flush-logs`

ログを全てフラッシュします。

- `flush-privileges`

グラントテーブルを再ロードします(`reload`と同じ)。

- `flush-status`

ステータス変数をクリアします。

- `flush-tables`

テーブルを全てフラッシュします。

- `flush-threads`

スレッドキャッシュをフラッシュします。

- `kill id,id,...`

サーバスレッドを消去します。複数スレッドID値が提供されている場合、リストにはスペースが存在しないことになります。

- `old-password new-password`

これは`password`コマンドと似ていますが、古い(4.1以前)パスワードハッシュフォーマットを使用してパスワードを記憶します。(詳しくは[MySQL 4.1 のパスワードハッシュ](#)をご確認ください。)

- `password new-password`

新しいパスワードをセットします。これによりサーバへの接続に使う`mysqladmin`のアカウントパスワードを`new-password`に変更します。よって、同じアカウントを使用して`mysqladmin` (あるいは他のクライアントプログラム) を起動するとき、新しいパスワードを指定しなければいけません。

もし`new-password` 値がスペースやコマンドインタープリタにとって特別なキャラクタを含んでいる場合、クォートで囲まなければいけません。Windowsでは、シングルクォートではなくダブルで囲むようにしてください。シングルクォートはパスワードの一部として認識されてしまいます。例:

```
shell> mysqladmin password "my new password"
```

- `ping`

サーバが作動しているかをチェックします。サーバが作動している場合`mysqladmin`のリターンステータスは0になり、作動していない場合1になります。`Access denied`のようなエラーの場合でも0となります。これは、サーバは作動しているが接続を拒否した形になり、サーバが作動していない状態とは異なるからです。

- `processlist`

アクティブなサーバスレッドのリスト表示します。これは`SHOW PROCESSLIST`ステートメントの出力に似ています。`--verbose`オプションが提供されている場合、出力は`SHOW FULL PROCESSLIST`と似ています。(詳しくは[SHOW PROCESSLIST 構文](#)をご確認ください。)

- `reload`

グラントテーブルを再ロードします。

- [refresh](#)
全テーブルをフラッシュし、ログファイルを閉じて、開きます。
- [shutdown](#)
サーバを停止させます。
- [start-slave](#)
スレーブサーバの複製を開始します。
- [status](#)
短いサーバステータスメッセージを表示します。
- [stop-slave](#)
スレーブサーバの複製を停止します。
- [variables](#)
サーバシステム変数とその値を表示します。
- [version](#)
サーバよりバージョン情報を表示します。

全てのコマンドは独特のプリフィックスで省略できます。例:

```
shell> mysqladmin proc stat
+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host | db | Command | Time | State | Info |
+-----+-----+-----+-----+-----+-----+-----+
| 51 | monty | localhost | | Query | 0 | | show processlist |
+-----+-----+-----+-----+-----+-----+-----+
Uptime: 1473624 Threads: 1 Questions: 39487
Slow queries: 0 Opens: 541 Flush tables: 1
Open tables: 19 Queries per second avg: 0.0268
```

[mysqladmin status](#) コマンド結果は以下の値を表示します。

- [Uptime](#)
MySQLサーバが作動している秒数です。
- [Threads](#)
アクティブスレッド(クライアント)の数です。
- [Questions](#)
サーバが起動して以来クライアントからよせられた質問 (クエリ) の数です。
- [Slow queries](#)
[long_query_time](#)秒よりも時間を要したクエリの数です。[スロー クエリ ログ](#) を参照してください。
- [Opens](#)
サーバによって開かれていないテーブルの数です。
- [Flush tables](#)
サーバが実行した[flush-*](#)、[refresh](#)、そして[reload](#)コマンドの数です。
- [オープンテーブル](#)

現在開いているテーブルの数です。

- `Memory in use`

`mysqld`によって直接割り当てられたメモリの量です。この値は、MySQLが`--with-debug=full`でコンパイルされたときのみ表示されます。

- `maximum memory used`

`mysqld`によって直接割り当てられたメモリの最大量です。この値は、MySQLが`--with-debug=full`でコンパイルされたときのみ表示されます。

Unixソケットファイルを使用してローカルサーバに接続する際`mysqladmin shutdown`を実行した場合、`mysqladmin`はサーバのプロセスIDが取り除かれるまで待ちます。これはサーバが正しく停止したことを確認するためです。

`mysqladmin`は次のオプションをサポートします。

- `--help, -?`

ヘルプメッセージを表示し、閉じます。

- `--character-sets-dir=path`

キャラクタセットがインストールされるディレクトリです。[データおよびソート用キャラクタセット](#)を参照してください。

- `--compress, -C`

双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。

- `--count=N, -c N`

コマンドの実行反復回数。これは`--sleep`オプションでのみ作動します。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。`debug_options`文字列は大抵'`d:t:o,file_name`'になります。`.'d:t:o,/tmp/mysqladmin.trace'`がデフォルトになります。

- `--default-character-set=charset_name`

`charset_name`をデフォルトキャラクタセットとして使用します。[データおよびソート用キャラクタセット](#)を参照してください。

- `--force, -f`

`drop db_name`コマンドの確認を要求しません。複数のコマンドでは、エラーが発生しても続けます。

- `--host=host_name, -h host_name`

与えられたホスト上でMySQLサーバに接続します。

- `--password[=password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(`-p`)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で`--password`あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#)を参照してください。

- `--port=port_num, -P port_num`

コネクションに使用するTCP/IPポート番号です。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--relative, -r`

`--sleep` オプションと使用された場合の現在値と以前の値の差異を表します。現時点では、このオプションは `extended-status` コマンドのみと併用できます。

- `--silent, -s`

サーバとの接続が確立できないときは閉じてください。

- `--sleep=delay, -i delay`

コマンドは反復実行、`delay` 秒間スリープしてください。`--count` オプションは反復回数を決定します。

- `--socket=path, -S path`

`localhost` の接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。[SSL コマンド オプション](#) を参照してください。

- `--user=user_name, -u user_name`

サーバに接続する際使用する MySQL ユーザ名です。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。

- `--version, -V`

バージョン情報を表示し、閉じます。

- `--vertical, -E`

縦に出力をプリントします。これは `--relative` に似ていますが、出力が縦にプリントされます。

- `--wait[=count], -w[count]`

接続が確立できない場合、アボートせずに休止してから再トライします。`count` 値が提供されている場合、再トライ回数を表しています。デフォルトは1回です。

`--var_name=value` 構文を使用することで以下の構文をセットすることができます。

- `connect_timeout`

接続タイムアウトまでの最大秒数。デフォルトは43200秒 (12時間)。

- `shutdown_timeout`

サーバがシャットダウンするまで待つ最大秒数。デフォルトは3600秒 (1時間) です。

`--set-variable=var_name=value` or `-O var_name=value` 構文を使用することで、変数をセットすることも可能です。構文は反対語となっています。

mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ

サーバが生成するバイナリログファイルはバイナリフォーマットで書かれています。これらのファイルをテキストフォーマットで確認するには、[mysqlbinlog](#) ユーティリティを使用してください。それに加えて、複製セットアップ内のスレーブサーバによって書き出されたリレイログファイルを読み取るのに [mysqlbinlog](#) を使用することもできます。リレイログはバイナリログファイルと同じフォーマットです。

mysqlbinlogは以下のように起動してください。

```
shell> mysqlbinlog [options] log_file ...
```

例えば、binlog.000003と名づけられたバイナリログファイルの内容を表示するにはこのコマンドを使用してください。

```
shell> mysqlbinlog binlog.000003
```

出力にはbinlog.000003内の全てのイベントが含まれています。イベント情報は実行されたステートメント、実行に必要な時間、発行したクライアントのスレッドID、実行時のタイムスタンプ等が含まれています。

ログ内にステートメントを再付加するためにmysqlbinlogの出力は再実行することができます(例えば、mysqlの入力として使用することで)。これはサーバがクラッシュした再のリカバリオペレーションとして便利にです。他の使用例は、後にこのセクションに登場するディスクッションを参照してください。

通常、バイナリログファイルを直接読み取りにmysqlbinlog使用し、ローカルMySQLサーバに付加します。--read-from-remote-serverオプションを使用してリモートサーバからバイナリログを読み取ることも可能です。リモートバイナリログを読み取る場合、接続パラメータオプションを提供することでどのようにしてサーバと接続すればよいか示しています。これらのオプションは--host、--password、--port、--protocol、--socket、と--userになります。--read-from-remote-serverオプションを使用した場合以外では、無視されます。

バイナリログやリレイログはバイナリ ログとレプリケーション リレーとステータス ファイルでさらに詳細を参照することができます。

mysqlbinlogは次のオプションをサポートします。

- --help, -?

ヘルプ メッセージを表示し、閉じます。

- --base64-output

ベース64 エンコードを使用して全てのバイナリログエントリをプリントします。これはデバッグ専用です。このオプションを使用して生成したログはプロダクションシステムには付加すべきではありません。このオプションはMySQL 5.1.5 で追加されました。

- --character-sets-dir=path

キャラクタ セットがインストールされるディレクトリです。データおよびソート用キャラクタ セット を参照してください。

- --database=db_name, -d db_name

このデータベース (ローカルログのみ) のエントリをリストします。このオプションを使用することで1つのデータベースのみを指定することができます。複数の--databaseオプションを指定した場合、最後のオプションのみ使用されます。このオプションはmysqlbinlogに、デフォルトデータベースの場所からバイナリログのエントリを出力しよう強制します。(つまり、USEに選択された)db_name。これによって、異なるデータベースやデータベース自体を選択せずとも、UPDATE some_db.some_table SET foo='bar'といったクロスデータベースステートメントを複製しません。

- --debug=[debug_options], -# [debug_options]

デバッグのログを書き込みます。debug_options文字列は大抵'd:t:o,file_name'になります。

- --disable-log-bin, -D

バイナリロギングを無効化します。これは、--to-last-logオプションを使用して同じMySQLサーバに対して出力を送信している場合、エンドレスループを回避するのに便利です。このオプションは、クラッシュ後、すでにログ下ステートメントの複製を生成するのを回避するのに便利です。

このオプションはSUPER権限を保持していることを要求します。残りの出力のバイナリロギングを無効化するため、mysqlbinlog出力にSET SQL_LOG_BIN=0ステートメントを含ませます。SETステートメントは、SUPER権限がない場合向こうです。

- `--force-read, -f`

このオプションでは、`mysqlbinlog`が認識できないバイナリログイベントを読み込んだ場合、警告をプリント、イベントを無視し続行します。このオプションなしでは、`mysqlbinlog`はそのようなイベントを読み込んだ時点で停止します。

- `--hexdump, -H`

コメントでログの16進法ダンプを表示します。この出力は複製デバッグの際に便利です。16進法ダンプフォーマットは後ほどこのセクションで説明されています。このオプションはMySQL 5.1.2.で追加されました。

- `--host=host_name, -h host_name`

あるホストでMySQLサーバからバイナリログを取得する。

- `--local-load=path, -l path`

特定のディレクトリ内から`LOAD DATA INFILE`のローカルテンポラリファイルを準備する。

- `--offset=N, -o N`

ログの最初のNエントリをスキップする。

- `--password=[password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(-p)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で`--password`あるいは-pに続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#)を参照してください。

- `--port=port_num, -P port_num`

リモートサーバ接続時に使用するTCP/IPポート番号です。

- `--position=N, -j N`

Deprecated.代わりに`--start-position`を使用してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--read-from-remote-server, -R`

バイナリログをローカルファイルから読み取らずにMySQLサーバから読み取ります。このオプションも提供されていない限りどの接続パラメータオプションも無視されます。これらのオプションは`--host`、`--password`、`--port`、`--protocol`、`--socket`、そして`--user`になります。

- `--result-file=name, -r name`

提供されているファイルに出力を導きます。

- `--server-id=id`

特定のサーバIDを持つサーバによってのみ作成されたイベントだけを抽出します。このオプションはMySQL 5.1.4.以降から提供されています。

- `--set-charset=charset_name`

ログファイルの処理のために使用する文字列を特定するため、出力に`SET NAMES charset_name`ステートメントを追加してください。このオプションはMySQL 5.1.12.で追加されました。

- `--short-form, -s`

ログに含まれるステートメントのみを表示します。

- `--socket=path, -S path`

`localhost`の接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- `--start-datetime=datetime`

`datetime`アークギュメントと等価、もしくは遅いタイムスタンプを持つ最初のイベントからバイナリログの読み取りを始めます。`datetime`値は`mysqlbinlog`を作動させた場合、ローカルタイムゾーンに相対的です。値は`DATETIME`や`TIMESTAMP`データ型に受け付けられるフォーマットでなければいけません。例:

```
shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
```

このオプションはpoint-in-timeリカバリを使用する際便利です。[バックアップとリカバリ手法の例示](#)を参照してください。

- `--stop-datetime=datetime`

`datetime`アークギュメントと等価、もしくは遅いタイムスタンプを持つ最初のイベントからバイナリログの読み取りを停止します。このオプションはpoint-in-timeリカバリを使用する際便利です。`datetime`値に関する詳細については`--start-datetime`オプションの説明を参照してください。

- `--start-position=N`

`N`アークギュメントと等価、もしくは遅いタイムスタンプを持つ最初のイベントからバイナリログの読み取りを始めます。このオプションはコマンドライン上で最初に名づけられるログファイルに対してのみ適用されます。

- `--stop-position=N`

`N`アークギュメントと等価、もしくは遅いタイムスタンプを持つ最初のイベントからバイナリログの読み取りを停止します。このオプションはコマンドライン上で最後に名づけられるログファイルに対してのみ適用されます。

- `--to-last-log, -t`

MySQLサーバから要求されるバイナリログの後部で終了せず、最後のバイナリログまで続けてプリントします。同じMySQLサーバに出力を送信した場合、エンドレスループになる場合があります。このオプションは`--read-from-remote-server`を要求します。

- `--user=user_name, -u user_name`

リモートサーバに接続する際使用するMySQLユーザ名です。

- `--version, -V`

バージョン情報を表示し、閉じます。

`--var_name=value` 構文を使用することで以下の構文をセットすることができます。

- `open_files_limit`

リザーブするオープンファイルディスクリプタの数を特定します。

`--set-variable=var_name=value` or `-O var_name=value` 構文を使用することで、変数をセットすることも可能です。構文は反対語となっています。

`mysqlbinlog`の出力を`mysql`クライアントにパイプすることができます。これはバイナリログに含まれるステートメントを実行するために行います。バックアップが古い([データベースのバックアップ](#)を参照してください)場合、クラッシュからのリカバリに使用します。例:

```
shell> mysqlbinlog binlog.000001 | mysql
```

または

```
shell> mysqlbinlog binlog.[0-9]* | mysql
```

`mysqlbinlog`の出力をテキストファイルにリダイレクトすることもできます。これは、先にステートメントログを改変する必要がある場合に行います。(例えば、実行したくないステートメントを取り除かなければいけないときに)。ファイル編集後、`mysql`プログラムの入力として使用することで含まれているステートメントを実行してください。

mysqlbinlogには--start-positionオプションがあります。ある特定のポジションよりも等価もしくは大きいオフセットを持つ、バイナリログ内のステートメントのみをプリントします(そのポジションは1つのイベントのスタートをマッチしなければいけません)。それはある特定の日付と時刻を与えられているイベントをスタート・ストップさせるオプションも有しています。これにより、--stop-datetimeオプションを使用してpoint in timeリカバリの実行を有効化します(例えば、「データベースを10:30分現在の状態まで巻き戻す」といったことが可能になります)。

MySQLサーバ上で1つ以上のバイナリログを実行しなければいけない場合、サーバに対して1つの接続で処理するのが安全です。以下に危険な例を示します。

```
shell> mysqlbinlog binlog.000001 | mysql # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql # DANGER!!
```

最初のログファイルにCREATE TEMPORARY TABLEステートメントが含まれており、2番目のログにはテンポラリテーブルを使用するステートメントが含まれている場合、サーバに対して異なる接続を使用してバイナリログを処理すると問題が発生します。最初のmysql処理が停止したとき、サーバはテンポラリテーブルをドロップします。2番目のmysql処理でテーブルの使用を試みると、サーバは「認識されていないテーブル」として報告します。

このような問題を回避するには、処理したいバイナリログの内容を実行するために1つの接続を使用してください。以下に1例を示します。

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql
```

また、ログを全て1つのファイルに書き込み、ファイルを処理するのも手です。

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

mysqlbinlogはオリジナルデータファイルなしでLOAD DATA INFILEオペレーションを再生する出力を生成することができます。mysqlbinlogはデータをテンポラリファイルにコピーし、ファイルに関連するLOAD DATA LOCAL INFILEステートメントを書きます。これらのファイルが書かれているディレクトリ内のデフォルトロケーションはシステムごとに異なります。ディレクトリを明示的に指定したい場合は、--local-loadオプションを使用してください。

mysqlbinlogがLOAD DATA INFILEステートメントをLOAD DATA LOCAL INFILEステートメントに変換するため、(つまり、LOCALを追加する)ステートメントの処理に使用するクライアントとサーバ双方がLOCAL機能を許容するようにコンフィギャされていなければいけません。LOAD DATA LOCALのセキュリティ関連事項を参照してください。

警告：LOAD DATA LOCALステートメント用に作成されたテンポラリファイルは、自動的に消去されません。これは、それらステートメントを実行するまで必要になるからです。ステートメントログがなくなった時点でユーザがテンポラリファイルを消去する必要があります。これらのファイルはテンポラリファイルディレクトリに存在し、original_file_name-##といった名前がついています。

--hexdumpオプションはコメント内にログ内容の16進性ダンプを生成します。

```
shell> mysqlbinlog --hexdump master-bin.000001
```

上記のコマンドを使用すれば、出力は以下のようになります。

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1 end_log_pos 98
# Position Timestamp Type Master ID Size Master Pos Flags
# 00000004 9d fc 5c 43 0f 01 00 00 00 5e 00 00 00 62 00 00 00 00 00
# 00000017 04 00 35 2e 30 2e 31 35 2d 64 65 62 75 67 2d 6c |..5.0.15.debug.1|
# 00000027 6f 67 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |log.....|
# 00000037 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
# 00000047 00 00 00 00 9d fc 5c 43 13 38 0d 00 08 00 12 00 |.....C.8.....|
# 00000057 04 04 04 04 12 00 00 4b 00 04 1a |.....K...|
# Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
# at startup
ROLLBACK;
```

16進性ダンプ出力は、現在以下のエレメントを含んでいます。このフォーマットは将来変更される可能性があります。

- **ポジション**: ログファイル内のバイトポジション。

- **Timestamp**: イベントのタイムスタンプ以下の例では、'9d fc 5c 43'は'051024 17:24:13'を16進変換で表します。
- **Type**: ログイベントのタイプ以下の例では、'0f'は例のイベントが**FORMAT_DESCRIPTION_EVENT**であることを意味しています。以下のテーブルが可能なタイプをリストします。

タイプ	名	意味
00	UNKNOWN_EVENT	このイベントはログ内に存在してはいけません。
01	START_EVENT_V3	MySQL 4より前のバージョンによって書き出されたログファイルの始めを示しています。
02	QUERY_EVENT	最も一般的なイベントのタイプです。これらはマスタで実行されるステートメントを含んでいます。
03	STOP_EVENT	マスタが停止したことを示しています。
04	ROTATE_EVENT	マスタが新しいログファイルにスイッチした際に書き出されます。
05	INTVAR_EVENT	一般的に、 AUTO_INCREMENT 値に対して使用されます。これは、 LAST_INSERT_ID() 関数がステートメントで使用されているときです。
06	LOAD_EVENT	MySQL 3.23で LOAD DATA INFILE 用に使用されます。
07	SLAVE_EVENT	将来の使用のため保存されます。
08	CREATE_FILE_EVENT	LOAD DATA INFILE ステートメントに使用されます。これはそのようなステートメントの実行の始まりを示しています。スレーブ上でテンポラリファイルが作成されます。MySQL 4でのみ使用されています。
09	APPEND_BLOCK_EVENT	LOAD DATA INFILE ステートメントで使用されるデータを含んでいます。スレーブ上でデータがテンポラリファイル内に記憶されます。
0a	EXEC_LOAD_EVENT	LOAD DATA INFILE ステートメントに使用されます。スレーブ上のテーブルにテンポラリファイルの内容が記憶されます。MySQL 4でのみ使用されています。
0b	DELETE_FILE_EVENT	LOAD DATA INFILE ステートメントのロールバックです。テンポラリファイルはスレーブ上で消去されます。
0c	NEW_LOAD_EVENT	LOAD DATA INFILE MySQL 4以前のバージョンで使用されます。
0d	RAND_EVENT	RAND() 関数がステートメントで使用されている場合、ランダム値の情報を送信するのに使用されます。
0e	USER_VAR_EVENT	ユーバ変数の複製に使用されます。
0f	FORMAT_DESCRIPTION_EVENT	MySQL 5.1以降のバージョンで書き出されたログファイルの始まりを示しています。
10	XID_EVENT	XAトランザクションのコミットを示すイベントです。
11	BEGIN_LOAD_QUERY_EVENT	MySQL 5以降のバージョンで LOAD DATA INFILE ステートメントに使用されます。
12	EXECUTE_LOAD_QUERY_EVENT	MySQL 5以降のバージョンで LOAD DATA INFILE ステートメントに使用されます。
13	TABLE_MAP_EVENT	テーブル定義の情報です。MySQL 5.1以降使用されています。
14	WRITE_ROWS_EVENT	作成されるはずのシングルテーブルの行データです。MySQL 5.1以降使用されています。
15	UPDATE_ROWS_EVENT	更新されるはずのシングルテーブルの行データです。MySQL 5.1以降使用されています。
16	DELETE_ROWS_EVENT	消去されるはずのシングルテーブルの行データです。MySQL 5.1以降使用されています。

- **Master ID**: イベントを作成したマスタのサーバIDです。
- **Size**: イベントのサイズをバイトで表しています。
- **Master Pos**: オリジナルマスタログファイル内のイベントのポジションです。
- **Flags**: 16 フラグ現在、以下のフラグが使用されています。他のフラグは将来に向けて保存されています。

フラグ	名	意味
01	LOG_EVENT_BINLOG_INFLIB	ログファイルは完全に閉じています。(FORMAT_DESCRIPTION_EVENTでのみ使用されています。)このフラグがセットされている場合(例えば、フラグがFORMAT_DESCRIPTION_EVENT内で'01 00'の場合) ログファイルは完全には閉じられていません。マスタがクラッシュしたことによる場合が最も可能性が高いです(例えば、停電などにより)。
02		将来使用するために保存されています。
04	LOG_EVENT_THREAD_SPECIFIC	mysqldが実行されたときの接続に依存している場合セットしてください(例えば、'04 00')。例えば、イベントがテンポラリテーブルを使用している場合。
08	LOG_EVENT_SUPPRESS_INFO	mysqldがデフォルトデータベースに対して依存していないケースではセットされることがあります。

他のフラグは将来使用するために保存されています。

mysqlcheck — テーブル メンテナンスと修復プログラム

mysqlcheckクライアントはテーブルのチェック、修復、最適化、そして分析を行います。

mysqlcheckのファンクションはmyisamchkと似ていますが、作動方法が異なります。実質的な作動方法の違いは、mysqlcheckはmysqldサーバが作動中の時に使用されなければいけません。myisamchkはこのサーバが作動していない時に使用されなければいけません。mysqlcheckを使用することの利点は、テーブルのチェックや修復時にサーバを停止させなくて済むことです。

mysqlcheckはSQLステートメントCHECK TABLE、REPAIR TABLE、ANALYZE TABLE、そしてOPTIMIZE TABLEをユーザにとって便利な方法で使します。実行したいオペレーションに対してどのステートメントを使用するか決定し、実行のためサーバにステートメントを送信します。各ステートメントがどのストレージエンジンと作動するかは、ステートメントの説明を12章SQL ステートメント構文で参照してください。

MyISAMストレージエンジンは全4ステートメントをサポートしています。よって、mysqlcheckはMyISAMテーブル上で全4オペレーションを実行することができます。他のストレージエンジンは必ずしも全てのオペレーションをサポートしているとは限りません。そのような場合、エラーメッセージが表示されます。例えば、test.tがMEMORYテーブルの場合、チェックしようとする以下の結果が生成されます。

```
shell> mysqlcheck test t
test.t
note : The storage engine for the table doesn't support check
```

一般的に、mysqlcheckを起動するには3つの方法があります。

```
shell> mysqlcheck [options] db_name [tables]
shell> mysqlcheck [options] --databases db_name1 [db_name2 db_name3...]
shell> mysqlcheck [options] --all-databases
```

db_nameに続くテーブルに名前をつけない場合、もしくは--databases、--all-databasesオプションを使用している場合、データベース全体がチェックされます。

他のクライアントプログラムに比べ、mysqlcheckは特別な機能があります。テーブルチェックのデフォルト行為(--check) はバイナリの名前を変更することで変えられます。テーブルをデフォルトで修復するツールが必要な場合、mysqlrepairと名づけたmysqlcheckのコピーを作成するか、mysqlrepairと名づけられたmysqlcheckへのシンボリックリンクを作成してください。mysqlrepairを起動すれば、テーブルを修復します。

以下の名前はmysqlcheckのデフォルト行為を変更するのに使用できます。

mysqlrepair	デフォルトオプションは--repair
mysqlanalyze	デフォルトオプションは--analyze
mysqloptimize	デフォルトオプションは--optimize

mysqlcheckは次のオプションをサポートします。

- `--help, -?`
ヘルプ メッセージを表示し、閉じます。
- `--all-databases, -A`
データベース内のテーブルを全てチェックします。これは`--databases`オプションを使用してコマンドライン上のデータベース全てに名前をつけることと同じです。
- `--all-in-1, -1`
各テーブルのためにステートメントを発行する代わりに、データベースから処理されるテーブルの名前を記載しているデータベースごとにシングルステートメントを実行します。
- `--analyze, -a`
テーブルを分析します。
- `--auto-repair`
チェックされたテーブルが破壊されていた場合、自動的に修復します。必要な修復は全てのテーブルがチェックされた後に実行されます。
- `--character-sets-dir=path`
キャラクタ セットがインストールされるディレクトリです。[データおよびソート用キャラクタ セット](#) を参照してください。
- `--check, -c`
テーブルにエラーが無いかチェックします。これがデフォルトオペレーションになります。
- `--check-only-changed, -C`
最後に行われたチェックより変更されたテーブル、もしくははっきり閉じられていないテーブルをチェックします。
- `--check-upgrade, -g`
`CHECK TABLE`を`FOR UPGRADE`オプションと共に起動し、現サーババージョンとの互換性の無いテーブルがあるかチェックします。これにより`--fix-db-names`と`--fix-table-names`オプションを自動的に有効化します。`--check-upgrade`はMySQL 5.1.7.で追加されました。
- `--compress`
双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。
- `--databases, -B`
名づけられたデータベース内のテーブルを全て処理します。通常、`mysqlcheck`はコマンドライン上の最初のアーギュメント名とそれに続く名をテーブル名として認識します。このオプションを使用することで、名前のついたアーギュメントを全てデータベース名として認識します。
- `--debug[=debug_options], -# [debug_options]`
デバグのログを書き込みます。`debug_options`文字列は大抵`'d:t:o,file_name'`になります。
- `--default-character-set=charset_name`
`charset_name`をデフォルトキャラクタセットとして使用します。[データおよびソート用キャラクタ セット](#) を参照してください。
- `--extended, -e`
テーブルをチェックするのにこのオプションを試用している場合、100%適合していることを保証しますが、時間がかかります。

このオプションを使用してテーブルを修復している場合、修復作業に時間がかかる上、必要な無い無駄な行を生成することもあります。!

- `--fast, -F`

正しく閉じられていないテーブルのみをチェックする。

- `--fix-db-names`

データベース名を5.1フォーマットに変換します。特別な文字を含むデータベース名のみ影響を受けます。このオプションはMySQL 5.1.7.加されました。

- `--fix-table-names`

データベース名を5.1フォーマットに変換します。特別な文字を含むテーブル名のみ影響を受けます。このオプションはMySQL 5.1.7.加されました。

- `--force, -f`

SQLエラーが発生しても続けます。

- `--host=host_name, -h host_name`

与えられたホスト上でMySQLサーバに接続します。

- `--medium-check, -m`

`--extended`オペレーションよりも速いチェックを行います。これはエラーの99.99%をチェックし、ほとんどの場合において十分な成果を発揮します。

- `--optimize, -o`

テーブルを最適化します。

- `--password[=password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(-p)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で`--password`あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#)を参照してください。

- `--port=port_num, -P port_num`

コネクションに使用するTCP/IPポート番号です。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--quick, -q`

このオプションを使用してテーブルをチェックしている場合、正しくないリンクをチェックするために行のスキャンを行いません。これが最速のチェックメソッドです。

このオプションを使用してテーブルを修復している場合、インデックスとリーのみの修復を試みます。これが最速の修復メソッドです。

- `--repair, -r`

ユニークではないユニークキー以外の全てを修復できるリペアを実行します。

- `--silent, -s`

サイレントモード。エラーメッセージのみプリントします。

- `--socket=path, -S path`

`localhost`の接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- `--ssl*`

`--ssl`で始まるオプションは、SSLを介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。[SSL コマンド オプション](#) を参照してください。

- `--tables`

`--databases`が`-B`オプションを重ね処理します。オプションに続く全てのネームアークギュメントはテーブル名として認識されます。

- `--use-frm`

MyISAMの修復オペレーションでは、`.frm`ファイルからテーブルストラクチャを取得することで`.MYI`ヘッダが破壊されていてもテーブルが修復できます。

- `--user=user_name, -u user_name`

サーバに接続する際使用するMySQLユーザ名です。

- `--verbose, -v`

Verbose モードプログラムオペレーションのあらゆるステージの情報をプリントします。

- `--version, -V`

バージョン情報を表示し、閉じます。

mysqldump — データベースバックアッププログラム

`mysqldump`クライアントは元はIgor Romanenkoによって書かれたバックアッププログラムです。バックアップや他のSQLサーバ(MySQLサーバに限りません)への転送のためにデータベースやデータベースのコレクションのダンプに役立ちます。ダンプには一般に、テーブルの作成やそこでのデータ配置、の片方または両方のSQLステートメントが含まれています。また、`mysqldump`はCSVや他の区切り文字のテキスト、あるいはXMLフォーマットでファイルを生成させるために利用することもできます。

もしあなたがサーバのバックアップをしていて、かつテーブルがすべてMyISAMテーブルの場合、代わりに`mysqlhotcopy`の使用をお勧めします。これは、バックアップやリストアのスピードが速くなるからです。[mysqlhotcopy — データベースバックアッププログラム](#) を参照してください。

`mysqldump`を起動する主な方法は3つあります。

```
shell> mysqldump [options] db_name [tables]
shell> mysqldump [options] --databases db_name1 [db_name2 db_name3...]
shell> mysqldump [options] --all-databases
```

`db_name`の後ろにテーブル名を指定しない場合、もしくは`--databases`、`--all-databases`オプションを使用した場合、データベース全体がダンプされます。

使用中のバージョンの`mysqldump`がサポートするオプションのリストを取得するには、`mysqldump --help`を実行してください。

`mysqldump`の中には他オプションをグループ化した略記法となっているものがあります。`--opt`や`--compact`はこれに分類されるものです。例えば、`--opt`を使用することは`--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`を指定したのと同じことです。MySQL 5.1以降、`--opt`が表すオプションは全てデフォルトで有効化されています。これは、`--opt`がデフォルトで有効なためです。

グループオプションの効果を逆転させる場合、オプションの`--skip-xxx`の形式(`--skip-opt`や`--skip-compact`など)を使用してください。グループオプションに続いて特定の機能を有効化・無効化するオプションをつけることで、グループオプションの効果の一部だけを選択することが可能です。以下に例を示します。

- いくつかの機能を除いて `--opt` の効果を選択したい場合、除きたい各機能に対して `--skip` オプションを選択してください。例えば、メモリバッファと拡張インサートを無効化するには、`--opt --skip-extended-insert --skip-quick` を使用してください。(MySQL 5.1では、`--skip-extended-insert --skip-quick` で十分です。これは `--opt` がデフォルトで有効になっているためです。)
- インデックス無効化とテーブルロック機能を生かして 他の `--opt` の機能を無効化したい場合、`--skip-opt --disable-keys --lock-tables` を使用してください。

グループオプションの一部を選択して効果を有効化・無効化する場合、オプションは前から後ろへの順で処理されるため、記述する順番が重要になります。例えば、`--disable-keys --lock-tables --skip-opt` は意図している効果を生みません。単一では `--skip-opt` と同じになります。

`mysqldump` はテーブル内容を一行ずつ取得してダンプするか、テーブルから全ての内容を取得しダンプする前にメモリでバッファすることができます。大きなテーブルをダンプしている場合、メモリへのバッファが問題になる場合があります。一行ずつテーブルをダンプする場合、`--quick` オプションを使用してください(もしくは `--opt` を指定すれば `--quick` が含まれています)。 `--opt` (故に `--quick` も) はMySQL 5.1以降デフォルトで有効化されています。メモリバッファを有効化するには、`--skip-quick` を使用してください。

最新の `mysqldump` を使用してダンプしたものを非常に古いMySQLサーバに再ロードしたい場合、`--opt` または `--extended-insert` オプションの使用は避けてください。代わりに `--skip-opt` を使用してください。

`mysqldump` は次のオプションをサポートします。

- `--help, -?`

ヘルプ メッセージを表示し、閉じます。

- `--add-drop-database`

`DROP DATABASE` ステートメントを `CREATE DATABASE` ステートメントの前に追加します。

- `--add-drop-table`

`DROP TABLE` ステートメントを `CREATE TABLE` ステートメントの前に追加します。

- `--add-locks`

`LOCK TABLES` と `UNLOCK TABLES` ステートメントで各テーブルダンプを囲みます。ダンプファイルを再ロードする際のインサートの速度が向上します。[INSERT ステートメントの速度](#) を参照してください。

- `--all-databases, -A`

すべてのデータベース内のすべてのテーブルをダンプします。これは `--databases` オプションを使用してコマンドラインですべてのデータベース名を指定するのと同じです。

- `--all-tablespaces, -Y`

テーブルダンプに、`NDB Cluster` テーブルに使用されるテーブルスペース作成に必要なSQLステートメントを追加します。でなければ、この情報は `mysqldump` の出力には含まれていません。このオプションは、現在MySQLクラスタテーブルに対してのみ有効です。

このオプションはMySQL 5.1.6で加されました。

- `--allow-keywords`

キーワードであるカラム名の作成を許容します。これは各カラム名のプリフィクスにテーブル名を用いることで可能になります。

- `--character-sets-dir=path`

キャラクタ セットがインストールされるディレクトリです。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--comments, -i`

プログラムバージョン、サーババージョンやホストといった追加情報をダンプファイルに書き込みます。このオプションはデフォルトで有効となっています。追加情報を抑制するには、`--skip-comments`を使用してください。

- `--compact`

verbose生成を少なくします。このオプションはコメントを抑制し、`--skip-add-drop-table`、`--no-set-names`、`--skip-disable-keys`、そして`--skip-add-locks`オプションを有効化します。

- `--compatible=name`

古いMySQLサーバや他のデータベースシステムと互換性のある出力を生成します。`name`の値は`ansi`、`mysql323`、`mysql40`、`postgresql`、`oracle`、`mssql`、`db2`、`maxdb`、`no_key_options`、`no_table_options`、あるいは`no_field_options`となります。複数の値を使用する場合カンマで離してください。これらの値はサーバSQLモードの設定用の対応しているオプションと同じ意味を持っています。[SQL モード](#)を参照してください。

このオプションは他のサーバとの互換性を保証するものではありません。現在提供されている、ダンプ出力の互換性を挙げるためのSQLモード値を有効化するだけです。例えば、`--compatible=oracle`はデータタイプをOracleタイプにマップしたり、Oracleコメント構文を使用したりしません。

- `--complete-insert, -c`

カラム名を含んだ、完全なINSERTステートメントを使用します。

- `--compress, -C`

双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。

- `--create-options`

MySQL独特のオプションをCREATE TABLEステートメントに含みます。

- `--databases, -B`

複数のデータベースをダンプします。通常、`mysqldump`はコマンドライン上の最初のアーギュメント名とそれに続く名をテーブル名として認識します。このオプションを使用するうことで、名前のついたアーギュメントを全てデータベース名として認識します。`CREATE DATABASE`や`USE`ステートメントは新しいデータベースの前の出力に含まれています。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。`debug_options`文字列は大抵`'d:t:o,file_name'`になります。`'d:t:o,tmp/mysqladmin.trace'`がデフォルトになります。

- `--default-character-set=charset_name`

`charset_name`をデフォルトキャラクタセットとして使用します。[データおよびソート用キャラクタ セット](#)を参照してください。文字列が特定されていない場合、`mysqldump`は`utf8`を使用します。

- `--delayed-insert`

INSERTステートメントよりもINSERT DELAYEDステートメントを書き出します。

- `--delete-master-logs`

マスタ複製サーバで、ダンプを実行後バイナリログを消去します。このオプションは自動的に`--master-data`を有効化します。

- `--disable-keys, -K`

各テーブルごとに、INSERTステートメントを`/*!40000 ALTER TABLE tbl_name DISABLE KEYS */;`と`/*!40000 ALTER TABLE tbl_name ENABLE KEYS */;`ステートメントで囲んでください。行が全て挿入された後にインデックスが作成されるため、ダンプファイルのロードが早くなります。このオプションはMyISAMテーブルに対してのみ効果的です。

- `--events, -E`

ダンプされたデータベースからイベントをダンプします。このオプションはMySQL 5.1.8.で追加されました。

- `--extended-insert, -e`

複数のVALUESリストを含む、複数行INSERT構文を使用してください。これにより、ダンプファイルサイズを小さくし、ファイルが再ロードされる際の挿入スピードがあがります。

- `--fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=...`

これらのオプションは-Tオプションと共に使用され、LOAD DATA INFILEに対応する節と同じ意味があります。LOAD DATA INFILE 構文 を参照してください。

- `--first-slave, -x`

Deprecated.現在は--lock-all-tablesと名づけられています。

- `--flush-logs, -F`

ダンプを始める前にMySQLサーバログファイルをフラッシュします。このオプションはRELOAD権限を要求します。このオプションを--all-databases (あるいは-A) オプションと併用した場合、ログはダンプされたデータベースごとにフラッシュされます。例外は、--lock-all-tablesまたは--master-dataを使用しているときです。この場合、ログは全てのテーブルがロックされた瞬間に一度だけフラッシュされます。ログのフラッシュとダンプを同時に行いたい場合、--flush-logsを--lock-all-tablesや--master-dataと併用してください。

- `--flush-privileges`

mysqlデータベースのダンプ後、.FLUSH PRIVILEGESステートメントを発行してください。このオプションはダンプにmysqlデータベースが含まれている場合と、正しいリストアのためにmysqlデータベース内に含まれているデータに依存するデータベースが含まれている場合に使用するべきです。このオプションはMySQL 5.1.12.で追加されました。

- `--force, -f`

テーブルダンプの最中にSQLエラーが発生しても続行します。

このオプションの使用例を挙げると、mysqldumpに無効となったビューに遭遇しても続けて実行させることです。これは、定義が消去されたテーブルを参照するものだからです。--forceなしでは、mysqldumpはエラーメッセージを発し閉じます。--forceを使用した場合、mysqldumpエラーメッセージをプリントしますが、ダンプ出力のビュー定義を含むSQLコメントを書き出し、実行を続けます。

- `--host=host_name, -h host_name`

与えられたホスト上でMySQLサーバからデータをダンプします。デフォルト設定では、localhostがホストになります。

- `--hex-blob`

16進変換表記法を使用しているバイナリカラムをダンプします(例えば、'abc'は0x616263となります)。影響を受けるデータタイプはBINARY、VARBINARY、BLOB、そしてBITになります。

- `--ignore-table=db_name.tbl_name`

データベースとテーブル名の両方を使用して特定されなければいけないテーブルをダンプしないでください。複数テーブルを無視するには、このオプションを複数回使用してください。

- `--insert-ignore`

INSERTステートメントをIGNOREオプションで書いてください。

- `--lines-terminated-by=...`

これらのオプションは-Tオプションと共に使用され、LOAD DATA INFILEに対応する節と同じ意味があります。LOAD DATA INFILE 構文 を参照してください。

- `--lock-all-tables, -x`

データベース内のテーブルを全てロックします。これは全ダンプの期間、グローバルリードロックを取得することで達成されます。このオプションは自動的に`--single-transaction`と`--lock-tables`をオフにします。

- `--lock-tables, -l`

ダンプする前に全てのテーブルをロックします。テーブルは`READ LOCAL`でロックされ、これにより`MyISAM`テーブルの場合同時インサートが許容されます。`InnoDB`といったトランザクションテーブルには、`--single-transaction`はテーブルをロックする必要が無いため、はるかにいいオプションです。

複数データベースをダンプする際は、`--lock-tables`は各データベースのテーブルを個別にロックします。よって、このオプションはダンプファイル内のテーブルがデータベース間で矛盾していないことを保証するわけではありません。異なるデータベース内のテーブルは完全に異なる状態でダンプされることがあります。

- `--master-data[=value]`

出力にバイナリログファイル名とポジションを書きます。このオプションは`RELOAD`権限を要求し、バイナリログが有効化されていなければいけません。オプション値が1と等価の場合、ポジションとファイル名はダンプ出力に書き出され、`CHANGE MASTER`ステートメントのフォームを取ります。ダンプがマスタサーバから行われ、それを利用してスレーブサーバをセットアップする場合、`CHANGE MASTER`ステートメントはスレーブを、マスタのバイナリログ内の正しいポジションからスタートするようにします。オプション値が2と等価の場合、`CHANGE MASTER`ステートメントはSQLコメントとして書かれます。(値が省かれていた場合のデフォルトアクションになります。)

`--master-data`オプションは自動的に`--lock-tables`をオフにします。`--single-transaction`も指定されていなければ、`--lock-all-tables`をオンにもします(この場合、global read lockがダンプの最初に短い間、取得されています)。`--single-transaction`の説明も参照してください。どの場合でも、ログに対するアクションは全てダンプと同時に発生します。

- `--no-autocommit`

各ダンプされたテーブルごとに`INSERT`ステートメントを`SET AUTOCOMMIT=0`と`COMMIT`ステートメントで囲みます。

- `--no-create-db, -n`

このオプションは`--databases`や`--all-databases`オプションが提供されていた場合出力に含まれる`CREATE DATABASE`ステートメントを抑制します。

- `--no-create-info, -t`

各ダンプされたテーブルを再作成する`CREATE TABLE`ステートメントを書かないでください。

- `--no-data, -d`

テーブル行情報を書かないでください(つまりテーブル内容をダンプしないでください)。これはテーブルの`CREATE TABLE`ステートメントのみをダンプしたい場合に非常に便利です。

- `--opt`

このオプションはショートハンドです。`--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`を特定することと同じことです。速いダンプオペレーションを提供し、MySQLサーバにすばやく再ロードできるダンプファイルを生成します。

`--opt`オプションはデフォルトで有効化されています。無効化するには`--skip-opt`を使用してください。`--opt`によって影響されるオプションの部分の一部を有効化・無効化する情報に関しては、このセクションの始めのディスカッションを参照してください。

- `--order-by-primary`

各テーブルの行をプライマリキーが、存在する場合、最初のユニークインデックスでソートします。これは、`InnoDB`テーブルにロードする`MyISAM`テーブルをダンプしているときに便利ですが、ダンプに要する時間がかなり伸びます。

- `--password[=password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(-p)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で`--password`あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#)を参照してください。

- `--port=port_num, -P port_num`

コネクションに使用するTCP/IPポート番号です。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--quick, -q`

このオプションは大きなテーブルのダンプに便利です。これにより、`mysqldump`は全結果セットを取得、メモリ内でバッファ後表示といった一連の作業を一気にこなさず、サーバから1行ずつ結果を取得します。

- `--quote-names, -Q`

``文字でデータベース、テーブル、そしてカラム名をクオートします。`ANSI_QUOTES`SQLモードが有効化されている場合、名前は``キャラクタでクオートされます。このオプションはデフォルトで有効となっています。`--skip-quote-names`で無効化することもできますが、このオプションは`--compatible`のような`--quote-names`を有効化するオプションの後に与えられるべきです。

- `--replace`

`INSERT`ステートメントよりも`REPLACE`ステートメントを書き出します。MySQL 5.1.3より提供されています。

- `--result-file=file, -r file`

提供されているファイルに出力を導きます。このオプションはWindows上で`\n`二ユーライン文字が`\r\n`carriage return/newlineシーケンスに変換されるのを阻止するために使用します。ダンプ生成中にエラーが発生しても、結果ファイルは作成され、内容は上書きされます。以前の内容は失われます。

- `--routines, -R`

記憶されたルーチン(関数とプロシージャ) ダンプされたデータベースからダンプします。このオプションの使用は`mysql.proc`テーブルのための`SELECT`権限を要求します。`--routines`を使用して生成された出力はルーチンの再作成のため、`CREATE PROCEDURE`と`CREATE FUNCTION`ステートメントを含んでいます。ただし、これらのステートメントはルーチン作成や改良タイムスタンプといった属性を含んでいません。つまりルーチンが再ロードされたとき、再ロードに要した時間と等価のタイムスタンプで作成されます。

ルーチンを元のタイムスタンプ属性で再作成しなければいけない場合、`--routines`を使用しないでください。代わりに、`mysql`データベースの正しい権限を持っているMySQLアカウントを使用して`mysql.proc`テーブルの内容を直接ダンプ、再ロードしてください。

このオプションはMySQL 5.1.2で追加されました。これ以前では、記憶されたルーチンはダンプされませんでした。ルーチン`DEFINER`値はMySQL 5.1.8.までダンプされませんでした。つまり、5.1.8.以前でルーチンが再ロードされた場合、再ロードユーザにセットされたデファイナで作成されます。元のデファイナでルーチンを再作成しなければいけない場合、`mysql.proc`テーブルの内容を、以前説明したとおりに、直接ダンプしロードしてください。

- `--set-charset`

出力に`SET NAMES default_character_set`を追加してください。このオプションはデフォルトで有効となっています。`SET NAMES`ステートメントを抑制するには、`--skip-set-charset`を使用してください。

- `--single-transaction`

このオプションはサーバからデータをダンプする前に`BEGIN` SQLステートメントを発行します。`InnoDB`といったトランザクションテーブルに対してのみ便利です。なぜなら、アプリケーションをブロックせずに、`BEGIN`が発行された当時のデータベースの状態をダンプするからです。

このオプションを使用しているときは、一定の状態でダンプされるのはInnoDBテーブルのみだということを留意してください。例えば、このオプションを使用中にダンプされたMyISAMやMEMORYテーブルは状態が変化する可能性があります。

このオプションはMySQLクラスタテーブルではサポートされていません。NDBClusterストレージエンジンがREAD_COMMITTED transaction isolation levelのみをサポートするため、結果が一定である保証がありません。代わりに必ずNDBバックアップを使用し、リストアしてください。

--single-transactionオプションと--lock-tablesは互いに関連していません。これは、LOCK TABLESが待機中のトランザクションを必然的にコミットさせるからです。

大きなテーブルをダンプするには、このオプションを--quickと併用してください。

- --skip-opt

--optオプションの詳細を参照してください。

- --socket=path, -S path

localhostの接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- --skip-comments

--commentsオプションの詳細を参照してください。

- --ssl*

--sslで始まるオプションは、SSLを介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。SSL コマンド オプション を参照してください。

- --tab=path, -T path

タブによって分けられたデータファイルを生成します。各ダンプされたテーブルごとに、mysqldumpはテーブルを作成するCREATE TABLEステートメントを含むtbl_name.sqlファイルと、そのデータを含むtbl_name.txtファイルを作成します。オプション値はファイルを書き込むディレクトリです。

デフォルトで、.txtデータファイルはカラム値と、各行の最後で新しいラインの間にタブキャラクタを使用してフォーマットされます。このフォーマットは明示的に--fields-xxxと--lines-terminated-byオプションを使用することで特定することができます。

注:このオプションはmysqldumpがmysqldサーバと同一のマシンで作動している場合のみ使用されるべきです。FILE権限を保持しており、サーバはユーザの指定してアファイルをディレクトリ内に書き込む権限を与えていなければいけません。

- --tables

--databasesあるいは-Bオプションをオーバーライドしてください。mysqldumpはオプションに続く名前アークユメントをテーブル名として認識しています。

- --triggers

ダンプされたテーブルごとにトリガをダンプします。このオプションはデフォルトで有効化されています。--skip-triggersを使用して無効化してください。

- --tz-utc

ダンプファイルにSET TIME_ZONE='+00:00'を追加してください。これにより、TIMESTAMPカラムは異なるタイムゾーンにあるサーバ間でダンプ・再ロードされます。このオプションなしでは、TIMESTAMPカラムはソースとデスティネーションサーバのタイムゾーンにダンプ・再ロードされ、値が変わる場合があります。--tz-utcはサマータイムによる時間の変更に対してもプロテクトします。--tz-utcはデフォルトで有効化されています。無効化するには、--skip-tz-utcを使用してください。このオプションはMySQL 5.1.2.で追加されました。

- --user=user_name, -u user_name

サーバに接続する際使用するMySQLユーザ名です。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。

- `--version, -V`

バージョン情報を表示し、閉じます。

- `--where='where_condition', -w 'where_condition'`

あるWHERE状態に選択された行のみダンプします。ユーザのコマンドインタプリタにとって特別なキャラクタ、もしくはスペースを含んでいる場合、状態の周りをクオートで囲まなければいけません。

例：

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

- `--xml, -X`

ダンプ出力と、well-formed XMLも書き出します。

NULL, 'NULL', and Empty Values:column_nameと名づけられたカラム、NULL値、空の文字列、文字値'NULL'はこのオプションによって生成された出力では以下の様に差別化します。

値:	XML Representation:
NULL (unknown value)	<field name="column_name" xsi:nil="true" />
" (empty string)	<field name="column_name"></field>
'NULL' (string value)	<field name="column_name">NULL</field>

MySQL 5.1.12に始まり、`--xml`オプションを使用しているときのmysqlクライアントもこれらのルールを守ります。(詳しくはmysql オプションをご確認ください。)

`--var_name=value` 構文を使用することで以下の構文をセットすることができます。

- `max_allowed_packet`

クライアント・サーバ通信のバッファの最大サイズ最大は1GBです。

- `net_buffer_length`

クライアント・サーバ通信のバッファの初期サイズ複数・行・挿入ステートメントを作成する際(`--extended-insert`や`--opt`オプションを使用するとき)、mysqldumpは`net_buffer_length`長さの新しい行を作成します。この変数を増やした場合、MySQLサーバ内の`net_buffer_length`変数も最低同じ大きさでなければいけません。

`--set-variable=var_name=value` or `-O var_name=value`構文を使用することで、変数をセットすることも可能です。構文は反対語となっています。

mysqldumpの最も一般的な用途は、データベース全体のバックアップの作成です。

```
shell> mysqldump db_name > backup-file.sql
```

ダンプファイルをサーバに戻し読みすることが可能です。

```
shell> mysql db_name < backup-file.sql
```

また、次のようにもできます。

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```


`mysqldump`は1つのMySQLサーバからデータをコピーすることでデータベースのpopulatingに便利です。

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

1つのコマンドで複数のデータベースをダンプすることが可能です。

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

全てのデータベースをダンプするには、`--all-databases`オプションを使用してください。

```
shell> mysqldump --all-databases > all_databases.sql
```

InnoDBテーブルに関して、`mysqldump`はオンラインバックアップの作成方法を提供しています。

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

このバックアップはグローバルリードロックをダンプの最初に、全テーブルで取得することだけが必要です(`FLUSH TABLES WITH READ LOCK`を使用して)。このロックが取得されれば、バイナリログの座標は読まれ、ロックが開放されます。`FLUSH`ステートメントが発行されている際、1つの長い更新ステートメントが作動している場合のみ、MySQLサーバはその長いステートメントが終了するまでストールすれば、ダンプがロックフリーとなります。MySQLサーバが受ける更新ステートメントが短い場合(実行時間を指す)、更新の数が多くても最初のロック期間はさほど気にならないはずです。

point-in-timeリカバリは、(もしくは「roll-forward」—これは古いバックアップをリストア、そのバックアップが行われてから発生した変更を再生する場合)、バイナリログを回転する、もしくはダンプが対応しているバイナリログの座標だけでも知っていると便利な場合があります([バイナリ ログ](#)を参照して下さい)。

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

または

```
shell> mysqldump --all-databases --flush-logs --master-data=2
> all_databases.sql
```

`--master-data`と`--single-transaction`オプションは同時に使用することができ、テーブルがInnoDBストレージエンジンを使用して記憶されている場合、point in timeリカバリに合うオンラインバックアップを作成する便利な方法を提供しています。

バックアップ作成の追加情報に関しては、[データベースのバックアップとバックアップとリカバリ手法の例示](#)を参照してください。

ビューのバックアップの際問題が発生した場合、ビューに対する制限を含むセクションを参照してください。権限が不足している事によって失敗した場合の、ビューバックアップ解決策を記しています。[ビューの規制](#)を参照してください。

mysqlhotcopy — データベースバックアッププログラム

`mysqlhotcopy`は元々Tim Bunceによって書かれ、提供されたPerlスクリプトです。データベースバックアップを速やかに作成するため、`LOCK TABLES`、`FLUSH TABLES`、`cp`あるいは`scp`を使用します。データベースやシングルテーブルのバックアップを作成する最速の方法ですが、データベースディレクトリが存在する同じマシン上でしか作動しません。`mysqlhotcopy`はMyISAMとARCHIVEテーブルのバックアップのためのみ作動します。UnixとNetWareで作動します。

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

あるデータベース内で通常の表現とマッチするテーブルをバックアップする。

```
shell> mysqlhotcopy db_name./regex/
```

波線符号をプリフィクスにつけることでテーブル名の通常の表現を取り消すことができます('~'):

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopyは次のオプションをサポートします。

- `--help, -?`

ヘルプメッセージを表示し、閉じます。

- `--addtodest`

ターゲットディレクトリに名前をつけなおさず、(存在する場合)ただファイルを追加します。

- `--allowold`

ターゲットが存在する場合アポートせず、`_old`サフィクスを追加することでリネームします。

- `--checkpoint=db_name.tbl_name`

特定のデータベース`db_name`とテーブル`tbl_name`にチェックポイントエントリを挿入します。

- `--chroot=path`

mysqldがオペレートするchrootジェイルのベースディレクトリ。`path`値はmysqldオプションに与えられる`--chroot`オプションとマッチしなければいけません。

- `--debug`

デバッグ出力を有効化します。

- `--dryrun, -n`

実行せずにアクションを報告します。

- `--flushlog`

全てのテーブルがロックされたあとログをフラッシュします。

- `--host=host_name, -h host_name`

ローカルサーバへのTCP/IP接続を作成するためのローカルホストのホスト名です。デフォルトで、接続はUnixソケットファイルを使用して`localhost`に作成されます。

- `--keepold`

終了後に以前の(リネームされた)ターゲットを消去しません。

- `--method=command`

ファイルの複製メソッド(`cp`あるいは`scp`)。

- `--noindices`

バックアップにフルインデックスファイルを含みません。これによりバックアップを小さく、早くすることができます。再ロードされたテーブルのインデックスは後ほど`myisamchk -rq`を使用して再構築することができます。

- `--password=password, -ppassword`

サーバに接続する際使用するパスワードです。他のMySQLプログラムと違って、このオプションにとってパスワード値は選択できません。コマンドライン上でパスワード提供を回避するためにオプションファイルを使用することができます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#)を参照してください。

- `--port=port_num, -P port_num`

ローカルサーバ接続時に使用するTCP/IPポート番号です。

- `--quiet, -q`

エラー発生時以外音を発しません。

- `--record_log_pos=db_name.tbl_name`

特定のデータベース`db_name`とテーブル`tbl_name`にマスタとスレーブステータスを報告します。

- `--regexp=expr`

与えられた通常の表現とマッチする名前のあるデータベースを全て複製します。

- `--resetmaster`

テーブルを全てロックした後バイナリログをリセットします。

- `--resetslave`

テーブルを全てロックした後`master.info`ファイルをリセットします。

- `--socket=path, -S path`

接続に使用するUnixソケットファイルです。

- `--suffix=str`

複製されたデータベースのサフィックスの名前です。

- `--tmpdir=path`

テンポラリディレクトリ。そのデフォルトは`/tmp`です。

- `--user=user_name, -u user_name`

サーバに接続する際使用するMySQLユーザ名です。

`mysqlhotcopy`はオプションファイルから`[client]`と`[mysqlhotcopy]`オプショングループを読み取ります。

`mysqlhotcopy`を実行するには、バックアップしているテーブルのファイルへのアクセス権、`SELECT`テーブルの権限、`RELOAD`権限(`FLUSH TABLES`の実行のため)そして`LOCK TABLES`権限(それらのテーブルをロックするため)を所持していなければいけません。

追加の`mysqlhotcopy`ドキュメンテーションに関しては、`perldoc`を使用してください。これは`--checkpoint`と`--record_log_pos`オプションに必要なテーブルの構成に関する情報についても、同様です。

```
shell> perldoc mysqlhotcopy
```

mysqlimport — データインポートプログラム

`mysqlimport`クライアントは`LOAD DATA INFILE`SQLステートメントにコマンドラインインターフェースを提供します。`mysqlimport`に対する殆どのオプションは`LOAD DATA INFILE`構文の節に直接対応しています。`LOAD DATA INFILE`構文を参照してください。

`mysqlimport`は以下のように起動してください。

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

コマンドラインで名づけられた各テキストファイルごとに、`mysqlimport`はファイルネームの拡張を取り除き、結果をファイルの内容をインポートするテーブルの名前を決定します。例えば、`patient.txt`、`patient.text`、そして`patient`と名づけられたファイルは全て`patient`と名づけられたファイルにインポートされます。

`mysqlimport`は次のオプションをサポートします。

- `--help, -?`

ヘルプメッセージを表示し、閉じます。

- `--character-sets-dir=path`

キャラクタ セットがインストールされるディレクトリです。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--columns=column_list, -c column_list`

このオプションはカンマによって分けられたカラム名のリストを値とします。カラム名の順序は、データファイルカラムとテーブルカラムをどのようにマッチするか示しています。

- `--compress, -C`

双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。`debug_options` 文字列は大抵 `'d:t:o,file_name'` になります。

- `--default-character-set=charset_name`

`charset_name` をデフォルトキャラクタセットとして使用します。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--delete, -D`

テキストファイルをインポートする前にテーブルを空にします。

- `--fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=...`

これらのオプションは `LOAD DATA INFILE` に対応する節と同じ意味を持っています。[LOAD DATA INFILE 構文](#) を参照してください。

- `--force, -f`

エラーを無視します。例えば、テキストファイルのテーブルが存在しない場合、残ったファイルの処理を続行します。`--force` なしでは、テーブルが存在しない場合 `mysqlimport` は抜けます。

- `--host=host_name, -h host_name`

与えられたホスト上でMySQLサーバからデータをインポートします。デフォルト設定では、`localhost` がホストになります。

- `--ignore, -i`

`--replace` オプションの詳細を参照してください。

- `--ignore-lines=N`

データファイルの最初の `N` ラインを無視します。

- `--lines-terminated-by=...`

これらのオプションは `LOAD DATA INFILE` に対応する節と同じ意味を持っています。例えば、carriage return/linefeed pairs でラインを消去されたWindowsファイルをインポートする場合、`--lines-terminated-by="\r\n"` を使用してください。(コマンドインタプリタのエスケープコンベンションによってはバックスラッシュを二つ加えなければいけない場合があります。) [LOAD DATA INFILE 構文](#) を参照してください。

- `--local, -L`

クライアントホストからインプットファイルをローカルで読み込む。

- `--lock-tables, -l`

テキストファイルを処理する前に、全てのテーブルをロックします。これにより、全てのテーブルがサーバ上でシクロしていることを保証します。

- `--low-priority`

テーブルロード時に `LOW_PRIORITY` を使用してください。

- `--password[=password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(`-p`)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で `--password` あるいは `-p` に続くオプションから `password` 値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#) を参照してください。

- `--port=port_num, -P port_num`

コネクションに使用するTCP/IPポート番号です。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--replace, -r`

`--replace` と `--ignore` オプションは、ユニークキー上に存在する行を複製するインプット行のハンドリングをコントロールします。`--replace` を特定した場合、同じユニークキー値を持つ既存の行は新しい行に取ってかわられます。`--ignore` を特定した場合、ユニークキー値上に存在する行を複製するインプット行はスキップされます。どちらのオプションも特定しなかった場合、複製キー値が発見されたときエラーが発生し、残りのテキストファイルは無視されます。

- `--silent, -s`

サイレントモード。エラーが発生したときのみアウトプットを生成します。

- `--socket=path, -S path`

`localhost` の接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- `--ssl*`

`--ssl` で始まるオプションは、SSLを介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。[SSL コマンド オプション](#) を参照してください。

- `--user=user_name, -u user_name`

サーバに接続する際使用するMySQLユーザ名です。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。

- `--version, -V`

バージョン情報を表示し、閉じます。

`mysqlimport` の使用方法を表すサンプルセッションを以下に記します。

```
shell> mysql -e 'CREATE TABLE impctest(id INT, n VARCHAR(30))' test
shell> ed
a
100 Max Sydow
101 Count Dracula
.
w impctest.txt
32
q
shell> od -c impctest.txt
0000000 1 0 0 \t M a x   S y d o w \n 1 0
0000020 1 \t C o u n t   D r a c u l a \n
```

```
0000040
shell> mysqlimport --local test impctest.txt
test.impctest: Records: 2 Deleted: 0 Skipped: 0 Warnings: 0
shell> mysql -e 'SELECT * FROM impctest' test
+-----+-----+
| id | n |
+-----+-----+
| 100 | Max Sydow |
| 101 | Count Dracula |
+-----+-----+
```

mysqlshow — データベース、テーブル、カラム情報を表示します。

mysqlshow クライアントは、どのデータベース、そのテーブル、あるいはテーブルカラムのインデックスが存在するか確認するために速やかに使用できます。

mysqlshow は複数の SQL **SHOW** ステートメントに対してコマンドラインインターフェースを提供します。**SHOW 構文** を参照してください。それらステートメントを直接使用することで同じ情報を得ることができます。例えば、**mysql** クライアントプログラムから発行することができます。

mysqlshow は以下のように起動してください。

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- データベースが提供されていない場合、データベース名のリストが表示されます。
- テーブルが提供されていない場合、データベース内の全てのマッチするテーブルが表示されます。
- カラムが提供されていない場合、テーブル内の全てのマッチするカラムとカラムタイプが表示されます。

出力は、ユーザがいくつかの権限を所持しているデータベース、テーブル、あるいはカラムの名前のみを表示します。

最後のアーギュメントがシエル、もしくは SQL ワイルドキャラクタを含んでいる場合 ('*', '?', '%', あるいは '_'), ワイルドカードとマッチする名前のみ表示されます。データベース名にアンダースコアが含まれる場合、正しいテーブルやカラムのリストを取得できるように、それらはバックスラッシュで (Unix シエルによつては 2 つ) エスケープされるべきです。 '*' と '?' キャラクタは SQL '%' と '_' ワイルドカードキャラクタに変換されます。これはテーブル名に '_' を含むカラムを表示しようとした際に問題を引き起こす場合があります。なぜなら、**mysqlshow** はパターンにマッチするテーブル名のみを表示するからです。別々のアーギュメントとして '%' をコマンドライン上で追加することで簡単に修正できます。

mysqlshow は次のオプションをサポートします。

- **--help, -?**

ヘルプメッセージを表示し、閉じます。

- **--character-sets-dir=path**

キャラクタ セットがインストールされるディレクトリです。**データおよびソート用キャラクタ セット** を参照してください。

- **--compress, -C**

双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。

- **--count**

テーブルごとの行の数を表示します。これは **MyISAM** テーブルで無ければ、遅い場合があります。

- **--debug[=debug_options], -# [debug_options]**

デバッグのログを書き込みます。**debug_options** 文字列は大抵 'd:t:o,file_name' になります。

- **--default-character-set=charset_name**

`charset_name`をデフォルトキャラクタセットとして使用します。[データおよびソート用キャラクタ セット](#) を参照してください。

- `--host=host_name, -h host_name`

与えられたホスト上でMySQLサーバに接続します。

- `--keys, -k`

テーブルインデックスを表示します。

- `--password[=password], -p[password]`

サーバに接続する際使用するパスワードです。ショートオプションフォーム(-p)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。コマンドライン上で`--password` あるいは`-p`に続くオプションから`password`値を取り除いた場合、パスワード値を求められます。

コマンドライン上でのパスワードの特定は安全ではありません。[パスワードのセキュリティ](#) を参照してください。

- `--port=port_num, -P port_num`

コネクションに使用するTCP/IPポート番号です。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

使用するべき接続プロトコルです。

- `--show-table-type, -t`

テーブルタイプを示すカラムを表示します、例えば`SHOW FULL TABLES`。このタイプは`BASE TABLE`もしくは`VIEW`になります。

- `--socket=path, -S path`

`localhost`の接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。

- `--ssl*`

`--ssl`で始まるオプションは、SSLを介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。[SSL コマンド オプション](#) を参照してください。

- `--status, -i`

各テーブルの追加情報を表示します。

- `--user=user_name, -u user_name`

サーバに接続する際使用するMySQLユーザ名です。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。このオプションは情報量を増加させるために複数回使用することができます。

- `--version, -V`

バージョン情報を表示し、閉じます。

mysqlslap — クライアント負荷エミュレーション

`mysqlslap`はMySQLサーバのクライアント負荷をエミュレートし、各ステージのタイミングを報告する診断プログラムです。サーバにたいして複数のクライアントがアクセスしているかのように作動します。`mysqlslap`はMySQL 5.1.4. から提供されています。

mysqlslapは以下のように起動してください。

```
shell> mysqlslap [options]
```

--createや--queryといったオプションはSQLステートメントを含む文字列やステートメントを含むファイルの特定を許容します。ファイルを特定した場合、デフォルトで各行ごとにステートメントを含んでいなければいけません。(つまり、暗示的なステートメントデリミタは二ユーラインキャラクタになります。)異なるデリミタを特定するのに--delimiterを使用してください。これにより、複数行にわたってステートメントの特定、もしくは1つのライン上で複数のステートメントを置くことができます。ファイルにコメントを含むことはできません。mysqlslapはそれらを理解しません。

mysqlslapは次のオプションをサポートします。

- --help, -?

ヘルプメッセージを表示し、閉じます。

- --auto-generate-sql, -a

ファイルやコマンドオプションを介して提供されていない場合、SQLステートメントを自動的に生成します。

- --compress, -C

双方が圧縮をサポートしている場合、クライアント・サーバ間で行きかう情報を全て圧縮します。

- --concurrency=N, -c N

SELECTステートメントを発行している際、シミュレートするクライアントの数。

- --create=value

テーブル作成の際使用するファイルか文字列。

- --create-schema=value

テストを実行するスキーマ。このオプションはMySQL 5.1.5 . で追加されました。

- --csv[=file]

カンマによって分けられた値のフォーマットで出力を生成します。出力は名づけられたファイルか、ファイルが提供されていない場合標準出力に向かいます。このオプションはMySQL 5.1.5 . で追加されました。

- --debug[=debug_options], -# [debug_options]

デバッグのログを書き込みます。debug_options 文字列は大抵 'd:t:o,file_name'になります。

- --delimiter=str, -F str

ファイルかコマンドオプションを介して提供されたSQLステートメントで使用するデリミタです。

- --engine=engine_name, -e engine_name

テーブル作成の際使用するストレージエンジン。

- --host=host_name, -h host_name

与えられたホスト上でMySQLサーバに接続します。

- --iterations=N, -i N

実行するテストの回数。

- --lock-directory=path

ロックの保存に使用するディレクトリです。このオプションはMySQL 5.1.5 . で追加されました。

- `--number-char-cols=N, -x N`
使用するVARCHARカラムの数 `--auto-generate-sql`が特定されている場合。
- `--number-int-cols=N, -y N`
使用するINTカラムの数 `--auto-generate-sql`が特定されている場合。
- `--number-of-queries=N`
各クライアントをこのクエリの数に限定します。このオプションはMySQL 5.1.5 . で追加されました。
- `--only-print`
データベースに接続しないでください。mysqlslapは実行されるべきであったことだけプリントします。このオプションはMySQL 5.1.5 . で追加されました。
- `--password[=password], -p[password]`
サーバに接続する際使用するパスワードです。ショートオプションフォーム(-p)を使用した場合、オプションとパスワードの間にスペースを置くことはできません。password値を `--password`あるいは-pオプションをコマンドライン上で省いた場合、ここで求められます。

コマンドライン上でのパスワードの特定は安全ではありません。パスワードのセキュリティ を参照してください。
- `--port=port_num, -P port_num`
コネクションに使用するTCP/IPポート番号です。
- `--protocol={TCP|SOCKET|PIPE|MEMORY}`
使用するべき接続プロトコルです。
- `--preserve-schema`
mysqlslapの実行からスキーマを保存します。このオプションはMySQL 5.1.5 . で追加されました。
- `--query=value, -q value`
データ回収のため使用するSELECTステートメントを含むファイルか文字列。
- `--silent, -s`
サイレントモード。出力はありません。
- `--skip-query, -Q`
SELECTを実行しないでください。
- `--slave`
他のmysqlslapクライアントに対してマスタロックをフォローしてください。1つのマスタサーバのまわりでシンク口を試みている場合このオプションを `--lock-directory`とNFS。このオプションはMySQL 5.1.5 . で追加されました。
- `--socket=path, -S path`
localhostの接続用に使用する、ユニックスではソケットファイル、Windowsでは使用する名づけられたパイプ。
- `--ssl*`
`--ssl`で始まるオプションは、SSLを介してサーバに接続し、SSL キーや証明の場所を明示するか否かを指定します。SSL コマンド オプション を参照してください。
- `--use-threads`

Unixでは、デフォルトでは`fork()`を使用します。このオプションを使用することで、代わりに`pthread`コールが使用されます。Windowsでは、デフォルトは`pthread`コールを使用し、オプションは効果がありません。このオプションはMySQL 5.1.6.で加されました。

- `--user=user_name, -u user_name`

サーバに接続する際使用するMySQLユーザ名です。

- `--verbose, -v`

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。

- `--version, -V`

バージョン情報を表示し、閉じます。

mysql_zap — パターンとマッチする処理を消去します。

`mysql_zap`はパターンとマッチする処理を消去します。`ps`コマンドとUnixシグナルを使用しますので、UnixやUnixと似たシステムで作動します。

`mysql_zap`は以下のように起動してください。

```
shell> mysql_zap [-signal] [-?|ft] pattern
```

処理は`ps`コマンドにパターンが含まれている場合に、出力ラインとマッチします。デフォルトで、`mysql_zap`各処理の確認を要求します。処理を消去するには`y`、閉じるには`qmysql_zap`としてください。たのレスポンスに関しては、`mysql_zap`は処理の消去を試みません。

`-signal`オプションが提供されている場合、各処理に送信するシグナルが番号の名前を特定します。でなければ、`mysql_zap`はまず`TERM` (シグナル15)、そして`KILL` (シグナル9)を使用します。

`mysql_zap`は以下のオプションを理解します。

- `--help, -?, -l`

ヘルプ メッセージを表示し、閉じます。

- `-f`

フォースモード。`mysql_zap`は確認なしで各処理の消去を試みます。

- `-t`

テストモード各処理の情報を表示しますが、消去はしません。

perror — エラーコードの説明

殆どのシステムエラーでは、内部テキストメッセージに加えて、MySQLは以下のスタイルでシステムエラーコードを表示します。

```
message ... (errno: #)
message ... (Errcode: #)
```

システムのドキュメンテーションを確認するか、`perror`ユーティリティを使用することでエラーコードの意味を割り出すことができます。

`perror`はストレージエンジン(テーブルハンドラ)エラーコードがシステムエラーコードの説明をプリントします。

`perror`は以下のように起動してください。

```
shell> perror [options] errorcode ...
```

例:

```
shell> perror 13 64
Error code 13: Permission denied
Error code 64: Machine is not on the network
```

MySQL クラスタエラーコードのエラーメッセージを取得するためには、`perror` を `--ndb` オプションと起動してください。

```
shell> perror --ndb errorcode
```

システムエラーメッセージの意味はユーザの OS によって異なる場合があります。エラーコードの意味は異なる OS では違う意味である場合があります。

`perror` は次のオプションをサポートします。

- `--help`, `--info`, `-l`, `-?`

ヘルプ メッセージを表示し、閉じます。

- `--ndb`

MySQL クラスタエラーコードのエラーメッセージをプリントします。

- `--silent`, `-s`

サイレントモード。エラーメッセージのみプリントします。

- `--verbose`, `-v`

Verbose モードエラーコードとメッセージをプリントします。これがデフォルトになります。

- `--version`, `-V`

バージョン情報を表示し、閉じます。

replace — 文字列置き換えユーティリティ

`replace` ユーティリティプログラムはファイル上、もしくは標準インプットの文字列を変更します。

`replace` は以下の様に起動してください。

```
shell> replace from to [from to] ... -- file [file] ...
shell> replace from to [from to] ... < file
```

`from` は探す文字列を表し、`to` はその代わりを表しています。文字列は1つ以上ありえます。

`--` オプションを使用して文字列置き換えリストが終わる場所と、ファイル名が始まる場所を特定してください。この場合、コマンドライン上で名づけられたファイルはその場で改良されるので、変換する前にオリジナルの複製を作成をしたほうがいいかもしれません。`replace` は実際にどのインプットファイルを改良したかを示すメッセージをプリントします。

`--` オプションが与えられていない場合、`replace` は標準インプットを読み、標準出力に書き出します。

`replace` は finite state machine を使用して長い文字列から先にマッチします。文字列の交換に使用できます。例えば、以下のコマンドは `a` と `b` をファイル `file1` と `file2` で交換します。

```
shell> replace a b b a -- file1 file2 ...
```

`replace` プログラムは `msql2mysql` に使用されています。[msql2mysql — MySQL と一緒に使うため、mSQL プログラムを変換してください。](#) を参照してください。

`replace` は次のオプションをサポートします。

- `-?`, `-l`

ヘルプ メッセージを表示し、閉じます。

- `-# debug_options`

デバッグのログを書き込みます。`debug_options` 文字列は大抵 `'d:t:o,file_name'` になります。

- `-s`

サイレントモード。プログラムの動作について、情報を少なくプリントアウトする。

- `-v`

Verbose モードプログラムの動作についてさらに情報をプリントアウトする。

- `-V`

バージョン情報を表示し、閉じます。

第8章 言語構造

目次

リテラル値	497
文字列	497
数値	499
16 進値	499
ブール値	500
ビットフィールド値	500
NULL 値	500
識別子	500
識別子の修飾語	502
識別子の小文字/大文字区別	502
ファイル名への識別子のマッピング	503
関数名の構文解析と名前解決	505
MySQLでの予約語の扱い	507
ユーザによって定義された変数	510
コメント構文	511

この章では、MySQL使用時における、以下のSQLステートメント要素の記述規則について説明します。

- リテラル値：文字列と数値
- 識別子：データベース名、テーブル名、カラム名
- 予約語
- ユーザ変数とシステム変数
- コメント

リテラル値

このセクションでは、MySQLでリテラル値を記述する方法について説明します。リテラル値には、文字列、数値、16進値、ブール値そしてNULLが含まれます。また、MySQLでこれらの基本データ型を処理するときに遭遇するであろう、さまざまなニュアンスと「リョーかい事項」についても扱います。

文字列

文字列は、単一引用符 (') または二重引用符 (") で囲まれたバイトもしくは文字の並び (シーケンス) です。次に例を示します：

```
'a string'
"another string"
```

ANSI_QUOTES SQLモードで実行時は、文字列リテラルは単一引用符でのみ囲まれます。これは、二重引用符で引用された文字列は識別子として解釈されるためです。

バイナリ文字列はキャラクタセットや照合順序を持たないバイト列のことです。バイナリでない文字列は、キャラクタセットや照合順序を持つ文字列のことです。これら両方の文字列タイプは、文字列ユニットの数値に基づいて比較されます。バイナリ文字列にとって、ユニットとはバイトのことです。バイナリでない文字列にとってユニットとは文字であり、マルチバイト文字を認めるキャラクタセットもあります。文字値の順序は、文字列照合順序の関数です。

文字列リテラルでは、オプションとしてキャラクタセットイントロデューサと**COLLATE**節を指定することができます。

```
[charset_name]'string' [COLLATE collation_name]
```

例：

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

これら文字列構文についてさらに詳しく知りたい場合は、次を参照してください。[文字列リテラルのキャラクタセットおよび照合順序](#)

一部のシーケンスは、個々の文字列内で特別な意味を持ちます。これらのシーケンスは、いずれも、エスケープ文字として知られるバックスラッシュ (\) で始まります。MySQLでは、次のエスケープシーケンスが認識されます。

\0	ASCII 0 (NUL) 文字。
\'	単一引用符 (') 文字。
\"	二重引用符 (") 文字。
\b	バックスペース文字。
\n	改行文字 (L F) 。
\r	復帰改行文字。
\t	タブ文字。
\Z	ASCII 26 (Control-Z)。表の下部にある注釈を参照してください。
\\	バックスラッシュ (\) 文字。
\%	'%'文字。表の下部にある注釈を参照してください。
_	'_'文字。表の下部にある注釈を参照してください。

他のすべてのエスケープシーケンスに対して、バックスラッシュは無視されます。つまり、エスケープされた文字がエスケープされていないと解釈されます。例えば、\xはただの'x'となります。

これらのシーケンスは大文字と小文字が区別されます。例えば、\bはバックスペースと解釈されますが、\Bは'B'と解釈されます。

ASCII(26)\Z'。この文字をコード化することによって、ASCII(26) が Windows では END-OF-FILE を表すという問題を回避することができます。ASCII(26) では、mysql db_name < file_name. を使用する場合に問題が発生します)。

エスケーププロセスはcharacter_set_connectionシステム変数により指定されたキャラクタセットに応じて実行されます。で[文字列リテラルのキャラクタセットおよび照合順序](#)説明されたとおり、他キャラクタセットを指定するイントロデューサが先行する文字列に対しても同じことがいえます。

'%'と_シーケンスはパターン照合コンテキスト内で、本来であればワイルドカード文字として解釈されるシーケンス'%'と_のリテラル使用例を検索するのに使われます。[文字列比較関数](#)内のLIKEオペレータに関する記述を参照してください。パターン照合でないコンテキストでは、'\%'または'_'を使用したときに、'%'と'_'の代わりに、文字列'\%'と'_'がそれぞれ返されます。

文字列に引用符を含める方法は、いくつかあります。

- "で囲んだ文字列内で、"を使用する場合、""と記述することができます。
- ""で囲んだ文字列内で、""を使用する場合、""と記述することができます。
- 引用符の直前にエスケープ文字 (\) を使用することができます。
- ""で囲んだ文字列内で""を使用する場合は、""を二つ続けて入力したり、エスケープしたりなどの特別な処置を行う必要はありません。同様に、""で囲んだ文字列内で""を使用する場合も、特別使いする必要はありません。

次のSELECTステートメントは、文字列の引用とエスケープが実際にどのように働くかを示しています。

```
mysql> SELECT 'hello', ""hello", ""hello"", 'hel"lo', '\hello';
+-----+-----+-----+-----+
| hello | "hello" | ""hello"" | hel"lo | \hello |
+-----+-----+-----+-----+

mysql> SELECT "hello", ""hello", ""hello"", "hel""lo", "\"hello";
+-----+-----+-----+-----+
| hello | 'hello' | "hello" | hel"lo | "hello" |
+-----+-----+-----+-----+
```

```
mysql> SELECT 'This\nIs\nFour\nLines';
+-----+
| This
Is
Four
Lines |
+-----+

mysql> SELECT 'disappearing\ backslash';
+-----+
| disappearing backslash |
+-----+
```

文字列のカラム(BLOBなど)にバイナリデータを挿入する場合、次の文字はエスケープシーケンスを使って表現する必要があります。

NULL	NULLASCII 0、バイト。この文字は'\0' (バックスラッシュ+ASCII '0'文字)で表現します。
\	ASCII 92、バックスラッシュ。'\'として表現します。
'	ASCII 39、単一引用符。\'として表現します。
"	ASCII 34、二重引用符。\'として表現します。

アプリケーションプログラムを書く場合、MySQLサーバに送信されるSQLステートメント内で文字がデータ値として使用される前に、これら特殊文字のいずれかを含む可能性のある文字列は正確にエスケープされなければなりません。二通りの実行方法があります。

- 特殊文字をエスケープする関数を使い文字列を処理してください。C コードを書く場合は、文字をエスケープする目的で C API 関数 `mysql_real_escape_string()` を使用できます。[mysql_real_escape_string\(\)](#) を参照してください。Perl DBI インターフェースでは、`quote` メソッドを使用して特殊文字を適切なエスケープシーケンスに変換することができます。[MySQL Perl API](#) を参照してください。他の言語インターフェースでも同様の機能が利用できることがあります。
- または、MySQL API の多くのものが一種のプレースホルダ機能を備えているため、この機能を使ってステートメント文字列に特殊なマーカーを挿入し、ステートメントの発行時にデータ値をそれらのマーカーにバインドすることもできます。この場合、値内の特殊文字のエスケープ処理が API によって自動で行われます。

数値

整数は数字の列として表現されます。浮動小数点は小数を区切るのに '.' を使用します。数値型は正負の値を指定するのに '-' や '+' を先頭に付けることもあります。

有効な整数の例：

```
1221
0
-32
```

有効な浮動小数点の例：

```
294.42
-32032.6809e+10
148.00
```

浮動小数点のコンテキストで整数を使用することもできます。この場合、整数は同等の浮動小数点数として解釈されます。

16 進値

MySQLでは、16進値をサポートしています。数値のコンテキストでは、16進値は整数(64ビット精度)のように動作します。文字列のコンテキストでは、16進値はバイナリ文字列のように動作します。この場合、16進数の各ペアが1文字に変換されます。

```
mysql> SELECT x'4D7953514C';
-> 'MySQL'
```

```
mysql> SELECT 0xa+0;
-> 10
mysql> SELECT 0x5061756c;
-> 'Paul'
```

16 進値のデフォルトのデータ型は文字列です。16 進値の文字列が確実に数値として扱われるようにするには、その文字列に対して `CAST(... AS UNSIGNED)` を使用します。

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
-> 'A', 65
```

`x'hexstring'` 構文は標準 SQL に基づいています。0x 構文は ODBC に基づいています。16 進文字列は、`BLOB` カラムの値を提供する目的で、ODBC によって使用されることがよくあります。

文字列または数値を 16 進形式の文字列に変換するには、`HEX()` 関数を使用できます。

```
mysql> SELECT HEX('cat');
-> '636174'
mysql> SELECT 0x636174;
-> 'cat'
```

ブール値

`TRUE` と `FALSE` の定数はそれぞれ 1 と 0 として評価されます。定数名は大文字/小文字の区別がなされません。

```
mysql> SELECT TRUE, true, FALSE, false;
-> 1, 1, 0, 0
```

ビットフィールド値

ビットフィールド値は `b'値'` 表記方法を用いて表現できます。値は 0 と 1 を用いて書かれたバイナリ値です。

ビットフィールド表記は `BIT` カラムに割り当てる値を指定するのに便利です。

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
+-----+-----+-----+
| b+0 | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+-----+-----+-----+
| 255 | 11111111 | 377      | FF       |
| 10  | 1010     | 12       | A        |
+-----+-----+-----+
```

NULL 値

`NULL` 値は「データなし」を意味します。`NULL` は大文字/小文字どちらでも表記できます。

`NULL` 値は、数値型での 0 や文字列型での空文字列などの値とは異なります。[Problems with NULL Values](#) を参照してください。

テキストファイルのインポートまたはエクスポート形式 `LOAD DATA INFILE` または `SELECT ...INTO OUTFILE` の使用時には、`NULL` は `\N` で表現することができます。[LOAD DATA INFILE 構文](#) を参照してください。

識別子

データベース、テーブル、インデックス、カラム、そしてエイリアスは識別子です。このセクションでは MySQL の識別子で利用できる構文を記述します。

下記テーブルは各識別子の長さ（最長時）記しています。

識別子	最長(バイト)
データベース	64
テーブル	64
カラム	64

インデックス	64
エイリアス	255

識別子に現れる文字には制限があります。

- ・ 識別子内でASCII 0 (0x00) および255のビット値は使用できません。
- ・ 識別子内での識別引用符使用は許可されていますが、必要でなければ使用は避けてください。
- ・ データベース名、テーブル名、そしてカラム名はスペース文字で終われません。
- ・ MySQL 5.1.6前では、データベース名には`'`、```、`~`やディレクトリー名で許可されていない文字を使用することはできませんでした。
- ・ MySQL 5.1.6前では、テーブル名には`'`、```、`~`やファイル名で許可されていない文字を使用することはできませんでした。
- ・ 識別子の長さは文字数ではなく、バイト数で表現されます。識別子名にマルチバイト文字を使用している場合、長さは使用した文字の合計バイト数で表現されます。

MySQL 5.1.6より、データベース名とテーブル名内の特殊文字は[ファイル名への識別子のマッピング](#)で記述されているとおり、対応するファイルシステム名にコード化されています。旧バージョンのMySQLを使用していて、で特殊文字を含むデータベース名やテーブル名が新しいエンコーディングに対応するようアップデートされていない場合、`#mysql50#`が接頭に表示されます。そういった名称を検索、もしくはそれらを新しいエンコーディングに変換するには、そのセクションを参照してください。

識別子は(UTF-8)を使って保存されます。これは`.frm`ファイル内に保存されたテーブル定義の識別子と`mysql`データベース内の許可テーブルに保存された識別子に対応しています。MySQL5.1の許可テーブル内 (そして他のテーブル) の文字列カラムの大きさは文字数でカウントされます。これは前バージョンのMySQLと違って、カラム内の値に充てる文字数を減らさずにマルチバイト文字を使用することができることを意味しています。

識別子を引用符で囲んだり、引用符を取り除くことができます。識別子が予約語、もしくは特殊文字を含む場合、参照する時必ず引用符で囲まなければいけません。(例外: 修飾名内で点の後に続く語は識別子であるため、予約語であっても引用符で囲む必要はありません。)予約語のリストはこちらを参照してください。[MySQLでの予約語の扱い](#)。特殊文字とは、英数字の組み合わせであるキャラクタセット外の`'`と`$`になります。

識別子引用符文字は(```) バッククォートです。

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

ANSI_QUOTESモードでMySQLを実行する場合は、識別子を囲む引用符として二重引用符も使用できます。

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax. (...)
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

注:**ANSI_QUOTES**モードは二重引用符を用いた文字列を識別子として解釈するため、このモードが実行されているとき文字列リテラルは一重引用符で囲まなければいけません。二重引用符は使用できません。

サーバSQLモードは**SQL モード**で表現されるように制御されています。

識別子が引用符で囲まれていれば識別子の引用符文字を識別子内に含むことができます。識別子内に含まれる文字が識別子を囲むのに使用している引用符と同じ場合、文字を二重にする必要があります。下記のステートメントは`a`b``という`c`d``カラムを含んだテーブルを作成します。

```
mysql> CREATE TABLE `a``b` (`c`d` INT);
```

識別子は1つのディジットで始まることもありますが、引用されない限りはディジットのみで構成されることはありません。

Mおよび**N**は整数ですので、**Me**や**MeN**といった形式名を使用することはお勧めできません。例: **1e**や**2e2**を識別子として使用しないでください。これは、**1e+3**といった式があいまいになるためです。コンテキストに応じて、式**1e + 3**として、または数値**1e+3**として解釈される場合があります。

Bテーブル名前名を作成するのにMD5()を使用する場合は注意が必要です。というのも、記述されたものがそのまま表現されてしまうといった、無効なフォーマットやあいまいなフォーマット名を作成する可能性があるからです。

識別子の修飾語

MySQLでは、名前に単一識別子や複合識別子を使用することができます。構成が複数のパート名からなる場合、ピリオド(.)文字で分割されなければなりません。複数パート名の頭文字は識別子として働き、最後尾の識別子が実行されているコンテキストに影響を与えます。

MySQLでは、次の形式のいずれかを使用してカラムを参照することができます。

カラム参照	意味
<code>col_name</code>	この名前のカラムが組み込まれたステートメントで使用されているテーブル内のカラム <code>col_name</code>
<code>tbl_name.col_name</code>	デフォルトデータベースのテーブル <code>tbl_name</code> 内のカラム <code>col_name</code>
<code>db_name.tbl_name.col_name</code>	データベース <code>db_name</code> のテーブル <code>tbl_name</code> 内のカラム <code>col_name</code>

複数のパート名からなる構成要素に引用符が必要な場合、名前全体を1つのものとして引用符で囲むのではなく、それぞれの名前を個別に引用符で囲んでください。例えば、``my-table.my-column``ではなく、``my-table`.`my-column``と記述してください。

対象となる参照があいまいな場合、ステートメント内のカラム参照の前に`tbl_name`や`db_name.tbl_name`をつける必要があります。例えば、テーブル`t1`と`t2`のそれぞれに同名のカラム`c`があり、`t1`と`t2`の両方を使用するSELECTステートメントで`c`を読み取るとします。この場合`c`は、ステートメントで使用されている2つのテーブル中で一意なカラムを表すものではなく、あいまいであるため、`t1.c`または`t2.c`と記述することによって、どちらのテーブルが対象が指定する必要があります。同様に、データベース`db1`のテーブル`t`とデータベース`db2`のテーブル`t`に含まれているカラムを取り出す場合は、それぞれのテーブルのカラムを`db1.t.col_name`と`db2.t.col_name`として参照します。

修飾名でピリオドの後に続く語は識別子であるため、予約語であっても引用符で囲む必要はありません。

構文`tbl_name`はデフォルトデータベースのテーブル`tbl_name`を意味します。この構文は ODBC との互換性を確保する目的で許容されています。これは、一部の ODBC プログラムでテーブル名の先頭に```文字が付けられるためです。

識別子の太文字/小文字区別

MySQL において、データベースはデータディレクトリ内のディレクトリに対応しています。データベース内の各テーブルも、データベースディレクトリ内の少なくとも1つ（記憶エンジンによってはそれ以上）のファイルに対応しています。そのため、ベースとなっているオペレーティングシステムで太文字と小文字が区別される場合、データベース名とテーブル名でも太文字と小文字が区別されます。つまり、Windows ではデータベース名とテーブル名で太文字と小文字は区別されず、ほとんどの種類の Unix では太文字と小文字が区別されることになります。ただし、重要な例外が1つあります。Mac OS X で、Unixをベースとしているがデフォルトの HFS+ ファイルシステムを使用している場合です。この場合は太文字と小文字が区別されません。しかし、Mac OS X は UFS ボリュームもサポートしています。UFS ボリュームでは Unix の場合と同じように Mac OS X でも太文字と小文字が区別されます。[SQL標準に対するMySQL拡張機能](#)を参照してください。このセクションに後述されているように、`lower_case_table_names`システム変数も、サーバがどのように識別子の太文字と小文字を区別するかということに影響を与えます。

注:いくつかのプラットフォームでは、データベース名とテーブル名で太文字と小文字は区別されませんが、同じステートメント内で異なるケースを使用して同じデータベースやテーブルを参照しないようにしてください。次のステートメントでは、同じテーブルが`my_table`および`MY_TABLE`として参照されています。したがって、このステートメントは機能しません。

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

カラム名、インデックス名、ストアされたルーチン名とカラムのエイリアスは、どのプラットフォームでも太文字と小文字が区別されません。トリガ名では太文字と小文字が区別されます。

デフォルトで、テーブルのエイリアスはUnixでは太文字と小文字が区別されますが、WindowsやMac OS Xでは区別されません。次のステートメントでは、同じエイリアスが`a`および`A`として参照されています。したがって、このステートメントはUnixでは機能しません。

```
mysql> SELECT col_name FROM tbl_name AS a
```


-> WHERE a.col_name = 1 OR A.col_name = 2;

しかし、Windowsではこの同じステートメントが機能します。このような違いがプログラムに起こらないようにするために、常に小文字を使用してデータベースとテーブルを作成および参照するなどの一貫した規則を設けてください。この規則は最大限の軽便さと使いやすさのために推奨されています。

MySQLでのテーブルとデータベース名の保存方法は`lower_case_table_names`システム変数に影響されます。これは`mysqld`起動時に設定できます。`lower_case_table_names`は以下のテーブルに示された値をとり得ます。Unixでは、`lower_case_table_names`のデフォルト値は 0で、Windowsでは1、Mac OS Xでは2となります。

値	意味
0	<code>CREATE TABLE</code> もしくは <code>CREATE DATABASE</code> ステートメントで区別された大文字/小文字を使用してテーブルとデータベース名が記憶されます。名前比較では大文字と小文字が区別されます。大文字/小文字を区別しないファイルシステム上で、 <code>--lower-case-table-names=0</code> を用いて変数値を0にし、かつ大文字/小文字を混せて <code>MyISAM</code> テーブル名にアクセスした場合、インデックスデータが破壊される恐れがあるので注意してください。
1	テーブル名はディスク上に小文字で記憶され、名前比較では大文字小文字は区別されません。MySQLでは、保管およびバックアップ時に全てのテーブル名が小文字に変換されます。このオプションはデータベース名やテーブルエイリアスにも適用されます。
2	<code>CREATE TABLE</code> または <code>CREATE DATABASE</code> ステートメントにおいて、テーブルとデータベース名は指定された大文字/小文字の形態でディスク上に記憶されますが、MySQLではバックアップ時に小文字に変換されます。名前比較では大文字と小文字が区別されません。注:これは大文字小文字が区別されないファイルシステムでのみ機能します。 <code>InnoDB</code> テーブル名は <code>lower_case_table_names=1</code> のように、小文字で記憶されます。

MySQLを単一プラットフォームでのみ使用している場合、通常は`lower_case_table_names`変数を変更する必要はありません。しかし、ファイルシステム上大文字小文字の区別方法が異なるプラットフォーム間でテーブルを移行させる場合、問題が生じる可能性があります。例えばUnix上で、`my_table`と`MY_TABLE`のように異なる2つのテーブル名を使用することはできますが、Windows上ではこれらは同一のものとして扱われます。データベースやテーブル名上の大文字/小文字に由来する移行問題を避けるには、2つのオプションがあります：

- `lower_case_table_names=1`を全システムで使用してください。この欠点は、`SHOW TABLES` または `SHOW DATABASES` を使用した場合、元の大文字/小文字で区別された名前が見られないことです。
- Unix上では`lower_case_table_names=0`を、Windows上では`lower_case_table_names=2`を使用してください。これでデータベースとテーブル名の大文字/小文字の区別が保存されます。この欠点は、Windows上で、ユーザのステートメントがデータベースとテーブル名を参照するのに大文字/小文字が正しく区別されているかどうかを常に確認しなければならないことです。ステートメントをUnixに移行する際に大文字/小文字の区別がなされる場合、大文字/小文字が正確でなければ機能しません。

例外：`InnoDB` テーブルを使用する場合、名前を強制的に小文字に変換するには、全てのプラットフォーム上で`lower_case_table_names`を 1に設定してください。

`lower_case_table_names`システム変数を 1に設定する場合、新しい変数設定で`mysqld`を再起動する前に、まず旧データベースおよびテーブル名を小文字に変換しなければなりません。

バイナリ照合順序に応じて大文字形式が同値とみなされれば、オブジェクト名は複製として扱われます。これはカーソル名、状態名、機能名、プロシージャ名、保存ポイント、ルーチンローカル変数において当てはまります。しかしカラム名、制約条件、データベース、パーティション、`PREPARE`を使用して準備されたステートメント、テーブル、トリガ、ユーザ、そしてユーザによって定義された変数においては当てはまりません。

ファイル名への識別子のマッピング

ファイルシステム上、データベース、テーブル識別子、そして名前の間には一致点があります。MySQLでは各データベースをデータディレクトリ内のディレクトリとして表現し、適切なデータベースディレクトリ内の1つかそれ以上のファイルで各テーブルを表現します。

MySQL 5.1.6以前では、ファイルシステムオブジェクトに一致するデータベースオブジェクトの識別子として使用できる文字に制限がありました。例えば、パスネーム分離文字と`'`は、テーブルファイル拡張の接頭子となるので使用できません

MySQL 5.1.6以降、ASCII NUL (0x00)を除く全ての文字は、データベースやテーブル識別子として有効です。MySQLはデータベースディレクトリやテーブルファイルを作成するとき、一致するファイルシステムオブジェクト内の問題文字を全てエンコードします。

- 基本的なローマ字(a..zA..Z)とディジット(0..9)はこのようにエンコードされます。つまり、大文字/小文字を区別するかはファイルシステムの特徴に依存します。
- 他国のアルファベットで大文字/小文字マッピングが区別される場合は下記のようにエンコードされます。

Code range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin1 Supplement + Ext A
0370..03FF	[@][5..9][g..z]	5*20= 100	88	12	Greek + Coptic
0400..052F	[@][g..z][0..6]	20*7= 140	140	137	Cyrillic
0530..058F	[@][g..z][7..8]	20*2= 40	38	2	Armenian
2160..217F	[@][g..z][9]	20*1= 20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	28*11=220	203	17	Latin Ext B + IPA
1E00..0EFF	[@][g..z][l..r]	20*7= 140	136	4	Latin Additional Extended
1F00..1FFF	[@][g..z][s..z]	20*8= 160	144	16	Greek Extended
.....	[@][a..f][g..z]	6*20= 120	0	120	RESERVED
24B6..24E9	[@][@][a..z]	26	26	0	Enclosed Alphanumerics
FF21..FF5A	[@][a..z][@]	26	26	0	Full Width forms

シーケンス内の1バイトが大文字/小文字をエンコードします。例: **LATIN CAPITAL LETTER A WITH GRAVE**は@0Gとしてエンコードされ、**LATIN SMALL LETTER A WITH GRAVE**は@0gとしてエンコードされます。ここでは、3番目のバイト(Gまたはg)は大文字/小文字を指示します。(大文字/小文字を区別しないファイルシステムでは、両文字は同義として扱われます。)

言語ブロックの中にはキリル文字のように、2番目のバイトが大文字/小文字を決定することもあります。補足ラテン1言語ブロックでは、3番目のバイトが大文字/小文字を決定します。シーケンス内の2バイトが文字の場合(拡張ギリシャ語のように)、一番左の文字が大文字/小文字を表します。他全ての文字バイトは小文字でなければなりません。

- 大文字/小文字マッピングを持たないアルファベット文字(ヘブライ語など)と同様、全ての非文字キャラクターは、16進ディジットa..fに対応する小文字を用いた16進表現でエンコードされます。

```
0x003F -> @003f
0xFFFF -> @ffff
```

16進値はucs2ダブルバイトキャラクターセット内のキャラクタ値に一致します。

ウィンドウズ上では、nulやpmやauxなどの名前はデバイス名として保存されるため、ファイル名として使えません。MySQL 5.1.10以降、これらの名前はMySQLで許可されています。サーバが一致ファイルもしくはディレクトリを作成する際に、これらは@@@を名前に付加されてエンコードされます。このようなことは全プラットフォーム上で、プラットフォーム間の一致データベースオブジェクトを移行する際に生じます。

バージョン5.1.6より前のMySQLのデータベースやテーブル内で、新エンコード利用機能がアップデートされていない特殊文字を使用している場合、**INFORMATION_SCHEMA**テーブルや**SHOW**ステートメントのアウトプット形式内では、**#mysql50#**を接頭辞として名前が表示されます。例えば、a@bのようなテーブル名があり、そのエンコーディング名がアップデートされていない場合、このように表示されます: **SHOW TABLES**。

```
mysql> SHOW TABLES;
+-----+
| Tables_in_test |
+-----+
| #mysql50#a@b |
+-----+
```

エンコーディングがアップデートされていない名前を参照するには、**#mysql50#**接頭辞を付加しなければなりません。

```
mysql> SHOW COLUMNS FROM `a@b`;
ERROR 1146 (42S02): Table 'test.a@b' doesn't exist

mysql> SHOW COLUMNS FROM `#mysql50#a@b`;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| i     | int(11) | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

特殊な接頭辞を使用する必要を無くするため、旧名をアップデートするには、[mysqlcheck](#)で再エンコードしてください。次のコマンドは全ての名前を新エンコーディングにアップデートします。

```
shell> mysqlcheck --check-upgrade --fix-db-names --fix-table-names --all-databases
```

特定のデータベースもしくはテーブルのみを確認するには、[--all-databases](#)を削除し、適切なデータベースやテーブル引数を付け加えてください。[mysqlcheck](#)起動構文に関するさらに詳しい情報は以下を参照してください。[mysqlcheck](#) — テーブル メンテナンスと修復プログラム

関数名の構文解析と名前解決

MySQL 5.1ではビルトイン（既存の）ファンクション、ユーザ定義関数（UDF）、そしてストアドファンクションがサポートされます。このセクションでは、サーバが、ビルトイン関数名を関数呼び出しもしくは識別子として認識するかどうか、また既存名によって異なる型の関数が存在する場合に、サーバがどの関数を使用するかを決定します。

ビルトイン関数名の構文解析

パーサがビルトイン関数名を解析するには、デフォルトルールにのっとり行われます。これらのルールは[IGNORE_SPACE](#) SQLモードを起動させることで変更できます。

構文解析中にビルトイン関数の名前を認識した場合、その名前が関数呼び出しを意味しているのか、テーブルやカラム名といった識別子を導く非表現であるかどうかを決定します。例えば、次のステートメントでは[count](#)に対する最初のリファレンスは関数呼び出しであるのに対し、2番目リファレンスはテーブル名です。

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

表現を解析している時にのみ、パーサはビルトイン関数名を関数呼び出しとして認識します。つまり非表現コンテキストでは、関数名は識別子として許可されます。

しかし、ビルトイン関数の中には特定の構文解析もしくは実装がなされることがあり、パーサはデフォルトで次のルールに沿って、名前が関数呼び出しか非表現コンテキストで識別子として使用されているかを区別します。

- 表現名を関数呼び出しとして使用するには、名前と次の`(` 括弧文字の間に余白があってははいけません。
- 逆に、関数名を識別子として使用するには、括弧文字をすぐ後ろに続けてはいけません。

名前と括弧文字の間に余白のない関数呼び出しの記述が要求された場合、特定認識がおこなわれるビルトイン関数にのみ適用されます。[COUNT](#)はそういった名前の1つです。後続の余白によって解釈が決定される関数名の正確なリストは、[sql/lex.h](#)ソースファイルの[sql_functions\[\]](#)配列に表示されます。MySQL 5.1以前ではそれらは多数（約200）あるため、余白のない要求を全関数呼び出しに適応させる方法が最も簡単でしょう。MySQL 5.1では、パーサが改良され、影響を受ける関数名の数が約30におさえられています。

[sql_functions\[\]](#) 配列にリストアップされていない関数には、余白は関係ありません。それらは表現コンテキスト内で使用される時のみ関数呼び出しとして解釈され、それ以外では識別子として自由に使用されることもあります。[ASCII](#)はそういった名前の1つです。しかし、こういった影響を受けない関数名に対する解釈は、表現コンテキストによって変わることがあります。[func_name \(\)](#)は単独で使用された場合、ビルトイン関数と解釈されますが、単独ではない場合、[func_name \(\)](#)がユーザによって定義された関数もしくは保存された関数と解釈されます。

[IGNORE_SPACE](#) SQL モードでは、パーサがどのように余白が区別される関数名を解釈するかを変更できます。

- 名前と後続の括弧の間に余白がない場合、パーサは 無効[IGNORE_SPACE](#)を用いることで名前を関数呼び出しと解釈します。これは関数名が非表現コンテキストで使用されているときも発生します。

```
mysql> CREATE TABLE count(i INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'count(i INT)'
```

エラーを取り除き名前を識別子として扱われるようにするには、名前の後に続く余白を使うか、引用符で囲んだ識別子として書き記してください（あるいは両方）。

```
CREATE TABLE count (i INT);
CREATE TABLE `count` (i INT);
CREATE TABLE 'count' (i INT);
```

- `IGNORE_SPACE`を有効にしているとき、パーサは関数名と後続の括弧間に余白が存在してはいけないという要求を緩和します。このことで、関数呼び出しの記述がより自由に行えるようになります。例えば、次のどちらの関数呼び出しも有効です。

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

しかし、`IGNORE_SPACE`を有効化することは、パーサが影響を受ける関数名を予約語として扱うという副作用もあります。(MySQLでの予約語の扱いを参照してください)これは名前の後に続く余白には識別子として認識されないことを意味します。後続の余白の有無を問わず、名前は関数呼び出しとして使用できますが、引用符で囲まれない場合は、非表現テキスト内での構文エラーを引き起こします。例えば、`IGNORE_SPACE`を有効化した場合、パーサが`count`を予約語として扱うため、構文エラーが生じ、両方の後続ステートメントが無効になります。

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

非表現コンテキストで関数名を使用するには、引用符で囲まれた識別子として記述してください。

```
CREATE TABLE `count` (i INT);
CREATE TABLE `count` (i INT);
```

`IGNORE_SPACE` SQL モードを有効化するには、このステートメントを使用してください。

```
SET sql_mode = 'IGNORE_SPACE';
```

`IGNORE_SPACE`はANSIのような値に含まれるコンポジットモードでも有効化されます。

```
SET sql_mode = 'ANSI';
```

どのコンポジットモードが`IGNORE_SPACE`を有効化するかを調べるにはSQL モードを参照してください。

`IGNORE_SPACE`設定におけるSQLコードの依存性を最小化するには、これらのガイドラインを使用してください。

- UDFもしくはビルトイン関数と同名のストアードファンクションの作成を避けてください。
- 非表現コンテキスト内の関数名使用を避けてください。例えば、これらのステートメントは`count` (`IGNORE_SPACE`に影響を受ける関数名のひとつ)を使用するため、`IGNORE_SPACE`が有効であれば後続名に対する余白の有無によらず、これらは無効となります。

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

非表現コンテキストで関数名を使用するには、引用符で囲まれた識別子として記述してください。

```
CREATE TABLE `count` (i INT);
CREATE TABLE `count` (i INT);
```

`IGNORE_SPACE`に影響を受ける関数名の数はMySQL 5.1.13では約200から約30に抑えられ、MySQL 5.1.13からは、下記の関数のみ`IGNORE_SPACE`設定に影響を受けます。

ADDDATE	BIT_AND	BIT_OR
BIT_XOR	CAST	COUNT
CURDATE	CURTIME	DATE_ADD
DATE_SUB	EXTRACT	GROUP_CONCAT
MAX	MID	MIN
NOW	POSITION	SESSION_USER
STD	STDDEV	STDDEV_POP
STDDEV_SAMP	SUBDATE	SUBSTR
SUBSTRING	SUM	SYSDATE
SYSTEM_USER	TRIM	VARIANCE
VAR_POP	VAR_SAMP	

MySQLの前バージョンでは、`sql/lex.h`ソースファイルの`sql_functions[]`配列の内容を確認して、どのが関数`IGNORE_SPACE`に影響を受けるかをチェックしてください。

非互換性に関する警告:MySQL 5.1.13では`IGNORE_SPACE`の影響を受ける関数名の数を抑えることでパーサオペレーションに一貫性をもたらしました。ただし、次の条件に依存する旧SQLコードの非互換性の可能性も生じます

- `IGNORE_SPACE`は無効化されています。
- 関数名に続く余白の有無は、同名を持つビルトイン関数とストアドファンクション(例: `PI()` 対 `PI ()`)を区別するのに用いられます。

MySQL 5.1.13より後で`IGNORE_SPACE`に影響を受けない関数に対しては、その方法は機能しません。前置の非互換性に対応するコードがある場合は、次のうちどちらのアプローチも使えます。

- ストアドファンクションにビルトイン関数とコンフリクトを引き起こす名前が存在する場合、余白の有無にかかわらず、修飾語付随のスキーマ名を持つストアドファンクションを参照してください。例えば、`schema_name.PI()` または `schema_name.PI ()` と書いてください。
- また、ストアド関数名を、コンフリクトを引き起こさない名前に付け替え、新しい名前を使用するために関数の起動を変更してください。

関数名の名称解決

次のルールは関数作成と起動のためにサーバがどのように関数名を参照するかについて述べられています。

- ビルトイン関数とユーザ定義関数

MySQL 5.1.14より後では、ビルトイン関数と同名のUDFを作成する際に、エラーが発生します。5.1.14前では、UDFはビルトイン関数と同名で作成はできましたが、パーサがビルトイン関数を参照する関数の起動を解除するため、UDFの起動はできませんでした。例えば、`ABS`と名づけられたUDFを作成する場合、`ABS()`を参照するとビルトイン関数が起動されます。

- ビルトイン関数とストアドファンクション

ビルトイン関数と同名のストアドファンクションを作成することは可能ですが、ストアドファンクションを起動させるにはスキーマ名で資格を与えなければいけません。例えば、`test`スキーマ内で`PI`という名のストアドファンクションを作成する場合、サーバが`PI()`をビルトイン関数参照として解釈するため、`test.PI()`として起動されます。5.1.14より、ストアド関数名がビルトイン関数名と衝突する場合、サーバから警告が発せられます。この警告は`SHOW WARNINGS`で表示できます。

- ユーザ定義関数とストアドファンクション

ユーザ定義関数とストアドファンクションは同じネームスペースを共有します。したがって、同名でUDFとストアドファンクションを作成することはできません。

前置関数名を解除するには、新しいビルトイン関数を実行するためのMySQLバージョンアップグレードを実行してください。

- すでにユーザによって定義された関数が提供名で作成され、同名の新ビルトイン関数が実行されるようにMySQLがアップグレードされた場合、UDFはアクセス不可となります。これを修正するには、`DROP FUNCTION`を使用してUDFを無効にし、次に`CREATE FUNCTION`を使ってコンフリクトを引き起こさないような異なる名前でUDFを再作成してください。
- MySQLの新バージョンで、既存のストアドファンクションと同名のビルトイン関数を起動する場合、2つの方法があります。コンフリクトを引き起こさないようにストアド関数名を変えるか、スキーマ修飾語が使用されるよう、ファンクションの呼び名を変えてください。(つまり、`schema_name`または`func_name()`構文を使用してください。)

MySQLでの予約語の扱い

`SELECT`、`DELETE`、もしくは`BIGINT`といった特定語はテーブルやカラム名といった識別子として使用するために予約されており、特別扱いを受けます。ビルトイン関数名に対しても同様のことが言える場合があります。

識別子を記述するのに予約語を引用符で囲む場合、その予約語は識別子として許可されます。


```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'
```

```
mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

例外：修飾名で点の後に続く語は識別子のため、予約語でも引用符で囲む必要はありません。

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

ビルトイン関数名は識別子として許可されますが、識別子として使用する場合は注意してください。例えば、`COUNT`はカラム名として許可されます。しかし、デフォルトでは、関数名と後続の`(`文字間の関数起動では余白は許可されていません。この条件では、関数呼び出しもしくは非ファンクションコンテキストで名前が使用されているかどうかを、パーサが区別するのを許可します。関数名の認識についてより詳しく知りたい場合は次を参照してください。[関数名の構文解析と名前解決](#)

次のテーブルに記された語はMySQL上で明確に予約されています。5.1今後バージョンアップする時のことを考慮に入れて、使用予定のある予約語を参照してみましょう。MySQLのバージョンアップ後もカバーするマニュアルではこれらを確認できます。テーブル内のたいていの語は標準SQLでカラムもしくはテーブル名としては許可されません(例：[GROUP](#))。MySQLが`yacc`パーサを使用、必要とするために、予約されている語もあります。予約語は引用符で囲まれた場合、識別子として使用できます。

ACCESSIBLE	ADD	ALL
ALTER	ANALYZE	AND
AS	ASC	ASENSITIVE
BEFORE	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BY	CALL	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	CONDITION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GRANT	GROUP	HAVING
HIGH_PRIORITY	hour_microsecond	hour_minute

HOURL_SECONDS	IF	IGNORE
IN	INDEX	INFILE
INNER	INOUT	INSENSITIVE
INSERT	INT	INT1
INT2	INT3	INT4
INT8	INTEGER	INTERVAL
INTO	IS	ITERATE
JOIN	KEY	KEYS
KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINEAR	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_SSL_VERIFY_SERVER_CERT
MATCH	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND
MINUTE_SECONDS	MOD	MODIFIES
NATURAL	NOT	NO_WRITE_TO_BINLOG
NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUT
OUTER	OUTFILE	PRECISION
PRIMARY	PROCEDURE	PURGE
RANGE	READ	READS
READ_ONLY	READ_WRITE	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SMALLINT
SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT
SSL	STARTING	STRAIGHT_JOIN
TABLE	TERMINATED	THEN
TINYBLOB	TINYINT	TINYTEXT
TO	TRAILING	TRIGGER
TRUE	UNDO	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE

USING	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING
WHEN	WHERE	WHILE
WITH	WRITE	XOR
YEAR_MONTH	ZEROFILL	

以下はMySQL 5.1で登場する新規の予約語です。

ACCESSIBLE	LINEAR	MASTER_SSL_VERIFY_SERVER_CERT
RANGE	READ_WRITE	

MySQLでは、引用符で囲まれてない識別子をキーワードとして使用できますが、これは多くのユーザが以前から使用していたからです。下記リストに例を記します。

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

ユーザによって定義された変数

ユーザによって定義された変数に値を保存し、後で参照することができます。これで1つのステートメントから次のステートメントに変数を移行させることができます。ユーザによって定義された変数はコネクションを指定しています。つまり、1クライアントによって定義されたユーザ変数は他のクライアントには確認や使用ができません。特定のクライアント接続で使用されている全ての変数は、クライアントが接続を切ったときに自動的に開放されます。

ユーザ変数は`@var_name`として記述され、このとき変数名`var_name`は現キャラクタセット、`.`、`'`そして`$`からなる英数字で構成される場合があります。デフォルトキャラクタセットは`latin1`(cp1252西ヨーロッパ言語)です。これは`--character-set-server`オプションで`mysqld`に変換されることがあります。[データおよびソート用キャラクタセット](#)を参照してください。ユーザ変数名を文字列や識別子として引用符で囲む場合、他の文字を含むことができます(例：`@'my-var'`、`@"my-var"`または`@`my-var``)。

注:MySQL5.0前ではユーザ変数名は大文字と小文字が区別され、MySQL 5.0以降は区別されません。

ユーザによって定義された変数を設定する方法の1つに、[SET](#)ステートメントを発行する方法が挙げられます。

```
SET @var_name = expr [, @var_name = expr] ...
```

[SET](#)では、`=`もしくは`:=`のどちらかが代入演算子として使用できます。各々の変数に割り当てられた`expr`は整数、実数、文字列、もしくは`NULL`値を評価できます。しかし、結果セットで変数値が選択された場合、文字列としてクライアントに返されます。

[SET](#)以外のステートメントで、ユーザ変数に値を割り当てることもできます。この場合、代入演算子は`:=`でなくてはならず、`=`ではありません。これは`=`が非[SET](#)ステートメントに置いて比較オペレータとして扱われるからです。

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+-----+-----+-----+-----+
| @t1 | @t2 | @t3 |
```

```
| @t1:=(@t2:=1)+@t3:=4 | @t1 | @t2 | @t3 |
+-----+-----+-----+
|      5 |      5 |      1 |      4 |
+-----+-----+-----+
```

ユーザ変数は表現が許可されているコンテキストで使用される可能性があります。これには現在、リテラル値を明確に要求するコンテキストは含まれません。例えばSELECTステートメントのLIMIT 節や、LOAD DATA ステートメントのIGNORE N LINES節になります。

ユーザ変数が文字値を割り当てられた場合、それは文字列と同じキャラクタセットと照合順序になります。ユーザ変数の強制力は絶対です。(これはテーブルカラム値と同等の強制力です。)

注:SELECTステートメントでは、クライアントに送信されたときのみ表現がそれぞれ評価されます。これはつまり、HAVING、GROUP BYもしくはORDER BY節において、SELECTリストに設定された変数を含む表現を参照できないことを意味します。例えば、次のステートメントは期待通りには機能しません。

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

HAVING 節でのbに対する参照は、@aaを使用したSELECTリスト内の表現に対応するエイリアスを参照します。これは期待通りには機能しません。@aaは以前選択された行のid値を含み、現在の行から選択された値は含みません。

ユーザ変数の評価順序は定義されておらず、与えられたクエリ内の要素に基づいて変更されることがあります。SELECT @a, @a := @a+1 ...では、MySQLは@aを先に評価し次に割り当てが実行されるように見えますが、クエリの変更(例えばGROUP BY、HAVINGまたはORDER BY節による変更)は評価順序を変更する可能性があります。

基本的なルールは、ステートメントの一部でユーザ変数値を割り当てないことおよび同一ステートメント内の他部分で同じ変数を使用しないことです。期待通りの結果を得られるかもしれませんが、これは確約されていません。

変数の設定および同ステートメント内での使用における他の問題点は、変数のデフォルト結果型がステートメントが開始時の変数型に基づく点です。次の例で説明します。

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

このSELECTステートメントに対して、MySQLでは、カラム 1 は文字列であり、@aから文字列への全てのアクセスを変換することがクライアントに通知されます。これは、@a が2行目では数値に設定されていても行われます。SELECTステートメント実行後、@aは次のステートメントの数値として認識されます。

この問題を避けるためには、単一ステートメント内で、同変数を設定・使用してはいけません。また、変数の使用前に型を定義するために、0、0.0もしくは"に変数を設定してもいけません。

初期化されていない変数を参照する場合、それはNULL値および文字列の型が存在します。

コメント構文

MySQL サーバでは、3つのコメントスタイルをサポートしています。

- #文字から行末まで続く。
- --シーケンスから行末まで続く。MySQLでは-- (ダッシュ2つ) のコメントスタイルでは、2つ目のダッシュの後にスペースを1つ以上挿入する必要があることに注意してください。(例えば スペース、タブ、新しい行など。) [コメントの開始記号としての--](#)で述べられたとおり、この構文は標準SQLコメント構文とは少し異なります。
- Cプログラミング言語のように/*シーケンスから続く*/シーケンスへ。開始および終了シーケンスが同行にある必要がないため、この構文は複数行にわたってコメントされることが許可されます。

次の例では3つ全てのコメントスタイルが説明されます。

```
mysql> SELECT 1+1;  # This comment continues to the end of line
mysql> SELECT 1+1;  -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
```

```
multiple-line comment
*/
1;
```

MySQL サーバではC-スタイルコメントの変形がいくつかサポートされています。これらはMySQL拡張を含むコードを記述することを可能にしますが、次のコメント形式を使用することで移行性を保持します。

```
/*! MySQL-specific code */
```

この場合、MySQLサーバは構文解析し、他のSQLステートメントのようにコメント内でのコードを実行しますが、他のSQLサーバはその拡張を認識しません。例えば、MySQL サーバは次のステートメント内で**STRAIGHT_JOIN**キーワードを認識しますが、他のサーバは認識しません。

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

'!'文字の後にバージョン番号を追加する場合、コメント内構文はMySQLバージョンが特定のバージョン番号と同等かそれ以上の場合にのみ実行されます。次のコメント内の**TEMPORARY**キーワードはMySQL 3.23.02もしくはそれ以降のサーバでのみ実行されます。

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

記述されたコメント構文はmysqlサーバによるSQLステートメントの構文解析に適用されます。mysqlクライアントプログラムはサーバに送信する前に、一部ステートメントの構文解析もします。(これは複数ステートメント入力行で、ステートメント境界表現を決定するために行われます。)

第9章 キャラクタセットサポート

目次

一般のキャラクタセットおよび照合順序	514
MySQLにおけるキャラクタセットおよび照合順序	514
デフォルトのキャラクタセットおよび照合順序の指定	516
サーバのキャラクタセットおよび照合順序	516
データベースのキャラクタセットおよび照合順序	516
テーブルのキャラクタセットおよび照合順序	517
カラムのキャラクタセットおよび照合順序	517
文字列リテラルのキャラクタセットおよび照合順序	518
各国キャラクタセット	519
キャラクタセットと照合順序の割り当ての例	520
他のDBMSとの互換性	520
接続のキャラクタセットおよび照合順序	520
照合順序に関して	522
SQLステートメントCOLLATE節を使用する	522
COLLATE節の優先順位	523
BINARY オペレータ	523
照合順序を決定するのが難しい特殊なケース	524
照合順序は適切なキャラクタセットに対応していること。	525
照合順序がもたらす結果の例	525
キャラクタセットのサポートによる影響を受ける演算	525
結果文字列	525
CONVERT() とCAST()	526
SHOW ステートメントとINFORMATION_SCHEMA	527
Unicodeのサポート	528
メタデータ用の UTF8	528
カラムキャラクタセット変換	529
MySQL でサポートされるキャラクタセットと照合順序	530
Unicode キャラクタセット	531
西ヨーロッパのキャラクタセット	533
中央ヨーロッパのキャラクタセット	535
南ヨーロッパおよび中東のキャラクタセット	535
バルト語のキャラクタセット	536
キリル語のキャラクタセット	536
アジアのキャラクタセット	537

MySQLでは、各種のキャラクタセットを使用してデータを保存したり、各種の照合順序を使用してデータを比較したりできます。サーバ、データベース、テーブルおよびカラムレベルでのキャラクタセット指定が可能です。MySQLは、MyISAM、MEMORY、NDBClusterそしてInnoDB記憶エンジンでのキャラクタセット使用をサポートしています。

この章では、以下について説明します。

- キャラクタセットと照合順序とは
- マルチレベルデフォルトシステム
- キャラクタセットと照合順序の指定構文
- 影響を受ける関数と演算
- Unicodeのサポート
- 利用可能なキャラクタセットと照合順序の注意点

キャラクタセットから生じた問題は、データ保存だけでなく、クライアントプログラムとMySQLサーバ間の通信にも影響を与えます。デフォルトと異なるキャラクタセットを使用してクライアントプログラムとサーバ間の通信を行いたい場合、どれを使用するのかわ知らせる必要があります。例えば、`utf8` Unicode キャラクタセットを使用するには、サーバ接続後にその旨を知らせてください。

```
SET NAMES 'utf8';
```

キャラクタセットに関するクライアント - サーバ間の通信問題について、さらに詳しく知りたい場合はこちらを参照してください。[接続のキャラクタセットおよび照合順序](#)。

一般のキャラクタセットおよび照合順序

キャラクタセットとは、シンボルとエンコードのセットです。照合順序とは、キャラクタセット内の文字を比較するためのルールを集めたものです。架空のキャラクタセットを例にして、キャラクタセットと照合順序の違いを見てみましょう。

次の4文字で構成されるアルファベットがあるとします：'A'、'B'、'a'、'b'。次のように、各文字に対して番号を指定します：'A' = 0、'B' = 1、'a' = 2、'b' = 3。文字'A'はシンボルであり、数字0はエンコードされた'A'です。4文字のすべてとそれぞれのエンコードを組み合わせたものをキャラクタセットと呼びます。

次に、文字列値'A'と'B'を比較してみましょう。最も簡単に比較するには、エンコードを確認します。'A'は0、'B'は1です。0は1よりも小さいので、'A'は'B'よりも小さいと表現することができます。今ここで行ったのは、キャラクタセットに対する照合順序の適用です。照合順序はルールの集まりであり、上記の場合にはルールは：「エンコードの比較」（この場合ルールは1つのみになります）。これは可能な照合順序のうちで最も単純なものであり、バイナリ照合順序と呼ばれています。

しかし、小文字と大文字を等しいと表現したい場合にはどうなるのでしょうか。その場合、少なくとも次の2つのルールが必要です。(1)小文字の'a'および'b'が大文字の'A'および'B'と同じであると見なす。(2)その後にエンコードを比較する。これは大文字と小文字を区別しない照合順序と呼ばれ、バイナリ照合順序よりも少し複雑です。

実際は、大半のキャラクタセットに多数の文字が含まれています。'A'と'B'だけではなく、アルファベットの全体から構成されています。ときには複数のアルファベットや、数千文字からなる東洋の書記体系に、多くの特殊記号や終止符が付属することもあります。また、実際には大半の照合順序に多くのルールがあります。大文字と小文字が区別されないだけではありません。アクセントが区別されない（「アクセント」はドイツ語での'Ö'のように文字に追加されるマーク）、あるいは複数文字マッピング（ドイツ語照合順序のどちらかにおける'Ö' = 'OE'のルールなど）といったルールがあります。

MySQLでは以下が可能です。

- 各種のキャラクタセットを使用して文字列を保存する。
- 各種の照合順序を使用して文字列を比較する。
- 同じサーバ、同じデータベース、あるいは同じテーブル内の異なったキャラクタセットまたは照合順序と文字列を結合する。
- 任意のレベルでキャラクタセットと照合順序を指定できるようにする。

MySQLはこれらの点において、他のDBMSに大きく差をつけています。ただし、新機能を効率的に使用するには、利用可能なキャラクタセット、各デフォルトの変更方法、各種の列演算子による処理内容を知っておかなければなりません。

MySQLにおけるキャラクタセットおよび照合順序

MySQLサーバでは複数のキャラクタセットがサポートされています。利用可能なキャラクタセットは、`SHOW CHARACTER SET`ステートメントを実行するとリストアップされます。リストの一部を以下に表示します。さらに詳しい情報が必要な場合は、次を参照してください。[MySQLでサポートされるキャラクタセットと照合順序](#)。

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
```



```

big5 | Big5 Traditional Chinese | big5_chinese_ci | 2 |
dec8 | DEC West European | dec8_swedish_ci | 1 |
cp850 | DOS West European | cp850_general_ci | 1 |
hp8 | HP West European | hp8_english_ci | 1 |
koi8r | KOI8-R Relcom Russian | koi8r_general_ci | 1 |
latin1 | cp1252 West European | latin1_swedish_ci | 1 |
latin2 | ISO 8859-2 Central European | latin2_general_ci | 1 |
swe7 | 7bit Swedish | swe7_swedish_ci | 1 |
ascii | US ASCII | ascii_general_ci | 1 |
ujis | EUC-JP Japanese | ujis_japanese_ci | 3 |
sjis | Shift-JIS Japanese | sjis_japanese_ci | 2 |
hebrew | ISO 8859-8 Hebrew | hebrew_general_ci | 1 |
tis620 | TIS620 Thai | tis620_thai_ci | 1 |
euckr | EUC-KR Korean | euckr_korean_ci | 2 |
koi8u | KOI8-U Ukrainian | koi8u_general_ci | 1 |
gb2312 | GB2312 Simplified Chinese | gb2312_chinese_ci | 2 |
greek | ISO 8859-7 Greek | greek_general_ci | 1 |
cp1250 | Windows Central European | cp1250_general_ci | 1 |
gbk | GBK Simplified Chinese | gbk_chinese_ci | 2 |
latin5 | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
...

```

キャラクタセットには少なくとも 1 つの照合順序が含まれており、複数の照合順序が含まれていることもあります。あるキャラクタセットに対して存在する照合順序をリストアップするには、[SHOW COLLATION](#) ステートメントを実行してください。例えば、キャラクタセット `latin1` (cp1252 西ヨーロッパ言語) に含まれる照合順序を調べるには、このステートメントを実行すると `latin1` で始まる照合順序が見つかります。

```

mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | | | 0 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 1 |
| latin1_danish_ci | latin1 | 15 | | | 0 |
| latin1_german2_ci | latin1 | 31 | | Yes | 2 |
| latin1_bin | latin1 | 47 | | Yes | 1 |
| latin1_general_ci | latin1 | 48 | | | 0 |
| latin1_general_cs | latin1 | 49 | | | 0 |
| latin1_spanish_ci | latin1 | 94 | | | 0 |
+-----+-----+-----+-----+-----+

```

`latin1` の照合順序には以下の意味があります：

照合順序	意味
latin1_german1_ci	ドイツ語 DIN-1
latin1_swedish_ci	スウェーデン語/フィンランド語
latin1_danish_ci	デンマーク語/ノルウェー語
latin1_german2_ci	ドイツ語 DIN-2
latin1_bin	latin1 エンコードに基づくバイナリ
latin1_general_ci	マルチリンガル(西ヨーロッパ言語)
latin1_general_cs	マルチリンガル(ISO 西ヨーロッパ言語)大文字と小文字が区別される
latin1_spanish_ci	現代スペイン語

照合順序にはこれらの一般的な特徴があります。

- 2 つの異なるキャラクタセットが同じ照合順序を共有することはできません。
- 各キャラクタセットには、デフォルト照合順序が 1 つ存在します。例えば、`latin1` のデフォルト照合順序は `latin1_swedish_ci` です。[SHOW CHARACTER SET](#) から、どの照合順序が表示されているどのキャラクタセットのデフォルトであるかがわかります。
- 照合順序名には次の規則が適用されます。関連するキャラクタセットの名前で始まる。通常は言語名が含まれており、`_ci` (大文字と小文字が区別されない)、`_cs` (大文字と小文字が区別される)、`_bin` (バイナリ) のいずれかで終わる。

デフォルトのキャラクタセットおよび照合順序の指定

サーバ、データベース、テーブル、カラムの4段階で、キャラクタセットと照合順序のデフォルト設定が用意されています。以下の説明は複雑に見えるかもしれませんが、マルチレベルのデフォルト設定では自然かつ明確な結果を得られることが実際に判明しています。

CHARACTER SETはキャラクタセットを指定する節で使用します。**CHARSET**は**CHARACTER SET**の同義語として使います。

サーバのキャラクタセットおよび照合順序

MySQLサーバにはサーバキャラクタセットとサーバ照合順序があります。サーバキャラクタセットとサーバ照合順序は、サーバ起動時に設定でき、ランタイムで変更できます。

サーバのキャラクタセットと照合順序は、**mysqld**の起動時に使用するオプションに依存します。**--character-set-server**をキャラクタセットに対して使用でき、**--collation-server**を照合順序に対して追加することもできます。キャラクタセットを指定しないのは、**--character-set-server=latin1**を指定した場合と同じです。キャラクタセット (例えば**latin1**)のみを指定して照合順序を指定しないのは、**--character-set-server=latin1 --collation-server=latin1_swedish_ci**を指定した場合と同じです。これは**latin1_swedish_ci**が**latin1**のデフォルト照合順序であるためです。したがって、以下の3つのコマンドを実行すると、いずれも同じ結果になります。

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
--collation-server=latin1_swedish_ci
```

設定を変更する手段の1つは再コンパイルです。ソースからのビルド時にデフォルトのサーバキャラクタセットと照合順序を変更するには、**--with-charset**と**--with-collation**を**configure**の引数として使用してください。例：

```
shell> ./configure --with-charset=latin1
```

または

```
shell> ./configure --with-charset=latin1 \
--with-collation=latin1_german1_ci
```

mysqldと**configure**では、キャラクタセットと照合順序の組み合わせが有効かどうかチェックされます。組み合わせが有効でない場合、各プログラムによってエラーメッセージが表示され、強制終了されます。

最新のサーバキャラクタセットと照合順序は、**character_set_server**および**collation_server**のシステム変数値で決定されます。これらの変数はランタイムで変更可能です。

データベースのキャラクタセットおよび照合順序

各データベースにはデータベースキャラクタセットとデータベース照合順序があります。**CREATE DATABASE**および**ALTER DATABASE**ステートメントには、データベースのキャラクタセットと照合順序を指定するためのオプション節があります。

```
CREATE DATABASE db_name
[[DEFAULT] CHARACTER SET charset_name]
[[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
[[DEFAULT] CHARACTER SET charset_name]
[[DEFAULT] COLLATE collation_name]
```

SCHEMAというキーワードは**DATABASE**の代わりに使用できます。

すべてのデータベースオプションは、データベースディレクトリ内の**db.opt**というテキストファイルに保存されます。

CHARACTER SETおよび**COLLATE**節で、キャラクタセットと照合順序が異なる複数のデータベースを同一のMySQLサーバ上に作成することができます。

例：

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL では、データベースキャラクタセットとデータベース照合順序が次のように選択されます。

- **CHARACTER SET X**と**COLLATE Y**の両方を指定した場合は、キャラクタセット**X**と照合順序**Y**。
- **CHARACTER SET X**を指定し、**COLLATE**を指定しなかった場合は、キャラクタセット**X**とそのデフォルト照合順序。
- **COLLATE Y**を指定し、**CHARACTER SET**を指定しなかった場合は、**Y**関連のキャラクタセットと照合順序**Y**。
- その他の場合は、サーバキャラクタセットとサーバ照合順序。

テーブルのキャラクタセットと照合順序が**CREATE TABLE**ステートメントに指定されていない場合、データベースのキャラクタセットと照合順序はデフォルト値として使用されます。これらに他の用途はありません。

デフォルトのデータベースに対するキャラクタセットと照合順序は、**character_set_database**および**collation_database**のシステム変数値によって決定されます。デフォルトのデータベースが変わるたびに、サーバはこれらの変数を設定します。デフォルトのデータベースがない場合、変数は、**character_set_server**および**collation_server**といったサーバレベルのシステム変数と同値になります。

テーブルのキャラクタセットおよび照合順序

各テーブルにはテーブルキャラクタセットとテーブル照合順序があります。**CREATE TABLE**および**ALTER TABLE**ステートメントには、テーブルのキャラクタセットと照合順序を指定するためのオプション節があります。

```
CREATE TABLE tbl_name (column_list)
  [[DEFAULT] CHARACTER SET charset_name] [COLLATE collation_name]

ALTER TABLE tbl_name
  [[DEFAULT] CHARACTER SET charset_name] [COLLATE collation_name]
```

例：

```
CREATE TABLE t1 ( ... ) CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL では、テーブルキャラクタセットとテーブル照合順序が次のように選択されます。

- **CHARACTER SET X**と**COLLATE Y**の両方を指定した場合は、キャラクタセット**X**と照合順序**Y**。
- **CHARACTER SET X**を指定し、**COLLATE**を指定しなかった場合は、キャラクタセット**X**とそのデフォルト照合順序。
- **COLLATE Y**を指定し、**CHARACTER SET**を指定しなかった場合は、**Y**関連のキャラクタセットと照合順序**Y**。
- その他の場合は、データベースキャラクタセットとデータベース照合順序。

カラムのキャラクタセットと照合順序が個別のカラム定義に指定されていない場合、テーブルのキャラクタセットと照合順序はデフォルト値として使用されます。テーブルのキャラクタセットと照合順序は MySQL 拡張であり、同等の機能は標準 SQL に存在しません。

カラムのキャラクタセットおよび照合順序

各「文字」(**CHAR**、**VARCHAR**または**TEXT**型)にはカラムキャラクタセットとカラム照合順序があります。カラム定義構文には、カラムキャラクタセットとカラム照合順序を指定するためのオプション節があります。

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
  [CHARACTER SET charset_name] [COLLATE collation_name]
```

例：

```
CREATE TABLE Table1
(
  column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci
);
```

MySQL では、カラムキャラクタセットとカラム照合順序が次のように選択されます。

- `CHARACTER SET X`と`COLLATE Y`の両方を指定した場合は、キャラクタセット`X`と照合順序`Y`。
- `CHARACTER SET X`を指定し、`COLLATE`を指定しなかった場合は、キャラクタセット`X`とそのデフォルト照合順序。
- `COLLATE Y`を指定し、`CHARACTER SET`を指定しなかった場合は、`Y`関連のキャラクタセットと照合順序`Y`。
- その他の場合は、テーブルキャラクタセットとテーブル照合順序。

`CHARACTER SET`および`COLLATE`節は標準SQLです。

文字列リテラルのキャラクタセットおよび照合順序

各文字列リテラルにはキャラクタセットと照合順序があります。

文字列リテラルでは、オプションとしてキャラクタセットイントロデューサと`COLLATE`節を指定することができます。

```
[charset_name]'string' [COLLATE collation_name]
```

例：

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

単純なステートメント`SELECT 'string'`に対して、文字列には`character_set_connection`および`collation_connection`システム変数で定義されたキャラクタセットと照合順序が存在します。

`_charset_name`は形式上イントロデューサと呼ばれています。指定すると、「キャラクタセット`X`の文字列が後続する」ことがパーサに通知されます。上記はユーザの混乱を招いていたため、ここで強調しておきますが、イントロデューサは変換の原因にはならず、文字列の値が変更されないことを示すにすぎません。標準的な16進リテラルおよび数値16進リテラルの表記 (`x'literal'`および`0xn`)の前でも、イントロデューサは有効です。

例：

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
```

MySQLでは、リテラルのキャラクタセットおよび照合順序が次のように決定されます。

- `_X`と`COLLATE Y`の両方が指定された場合、リテラルキャラクタセットは`X`、リテラル照合順序は`Y`。
- `_X`は指定されており、`COLLATE`が指定されていない場合、リテラルキャラクタセットは`X`、リテラル照合順序はそのデフォルト照合順序。
- その他の場合は、`character_set_connection`および`collation_connection`システム変数が指定するリテラルキャラクタセットとリテラル照合順序。

例：

- 文字列に`latin1`キャラクタセットと`latin1_german1_ci`照合順序が指定されている場合

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- 文字列に`latin1`キャラクタセットとそのデフォルト照合順序 (`latin1_swedish_ci`)が指定されている場合

```
SELECT _latin1'Müller';
```

- 文字列に接続デフォルトキャラクタセットおよび照合順序が指定されている場合

```
SELECT 'Müller';
```

キャラクタセットイントロデューサと`COLLATE`節は、標準SQLの指定に基づいて提供されています。

イントロデューサは後続の文字列に対してキャラクタセットを指定しますが、現在のところ、その文字列におけるパーサのエスケーププロセス内容までは変更しません。エスケープは常に、`character_set_connection`に指定されたキャラクタセットに従ってパーサが実行します。

以下の例は、イントロデューサが存在しても`character_set_connection`によるエスケーププロセスが生じる場合についてです。例では`SET NAMES(character_set_connectionを変更する接続のキャラクタセットおよび照合順序)`を用いて`HEX()` ファンクションを使い結果文字列を表示させることで、正確文字列の内容を確認することができます。

例 1 :

```
mysql> SET NAMES latin1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+-----+-----+
| HEX('à\n') | HEX(_sjis'à\n') |
+-----+-----+
| E00A      | E00A             |
+-----+-----+
1 row in set (0.00 sec)
```

ここでは '`à`' (16進E0) の後に '`\n`' ニューラインのエスケープシーケンスが続きます。エスケープシーケンスは`character_set_connectionlatin1` の値を使い、新しいリテラルニューラインを作成します。(16進 0A)。これは2番目の文字列にも起こります。つまり、`_sjis`のイントロデューサはパーサのエスケーププロセスに影響を及ぼしません。

例2 :

```
mysql> SET NAMES sjis;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+-----+-----+
| HEX('à\n') | HEX(_latin1'à\n') |
+-----+-----+
| E05C6E     | E05C6E             |
+-----+-----+
1 row in set (0.04 sec)
```

ここでは `character_set_connection` は `sjis`になります。このキャラクタセットは次のシーケンス '`à`' と '`\`' (hex values 05 and 5C)がつづく、有効なマルチバイトキャラクタです。よって、文字列の最初の2バイトは1つの`sjis` キャラクタとして実行され、 '`\`' はエスケープキャラクタとして実行されない。次の '`n`' (16進値6E) はエスケープシーケンスの一部として実行されません。これは第2文字列にとっても同様に、`_latin1`のイントロデューサはエスケーププロセッシングに影響しません。

各国キャラクタセット

標準SQL では`NCHAR`または`NATIONAL CHAR`が、事前定義キャラクタセットが`CHAR`カラムで使用されるように指定する方法のひとつとして使われています。MySQLでは5.1`utf8`事前定義キャラクタセットとして使用されます。例えば、以下のデータタイプ宣言は等価です :

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

これらと同じように :

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

`N'literal'`を使用して、各国キャラクタセットの文字列を 作成することができます。以下の二つのステートメントは等価です

```
SELECT N'some text';
SELECT _utf8'some text';
```

バージョン4.1以前のMySQLにキャラクタセットをアップグレードするには、5.1 MySQL 3.23, 4.0, 4.1 リファレンスマニュアルを参照してください。.

キャラクタセットと照合順序の割り当ての例

MySQL でデフォルトのキャラクタセットと照合順序の値がどのように決定されるかを、以下の例で示します。

例1テーブルとカラム定義

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

ここでは`latin1`キャラクタセットと`latin1_german1_ci`照合順序がカラムに指定されています。定義は明示的なので、直接的といえます。なお、`latin1`カラムの保存先が`latin2`テーブルになっていることに問題はありません。

例2テーブルとカラム定義

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

ここでは、`latin1`キャラクタセットとデフォルトしょうごうじゅんじょがカラムに指定されています。自然な指定に目増すが、デフォルト照合順序はテーブルレベルから取り込まれません。`latin1`のデフォルト照合順序は常に`latin1_swedish_ci`です。したがって、カラム`c1`には`latin1_swedish_ci`の照合順序ではなく、`latin1_danish_ci`の照合順序が設定されます。

例3テーブルとカラム定義

```
CREATE TABLE t1
(
  c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

ここでは、デフォルトキャラクタセットとデフォルト照合順序がカラムに指定されています。この場合に MySQL では、テーブルレベルまで検索してカラムのキャラクタセットと照合順序が決定されます。したがって、カラム `c1` のキャラクタセットは`latin1`、照合順序は`latin1_danish_ci`となります。.

例4データベース、テーブル + カラム定義

```
CREATE DATABASE d1
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

キャラクタセットと照合順序を指定せずにカラムを作成します。テーブルレベルのキャラクタセットと照合順序も指定しません。この場合に MySQLでは、データベースレベルまでさかのぼって処理が決定されます。(データベースの設定はテーブルの設定になり、そしてカラムの設定になります。) したがって、カラム `c1` のキャラクタセットは`latin1`、照合順序は`latin1_czech_ci`となります。

他のDBMSとの互換性

MaxDB 互換性に関し、以下の二つのステートメントは同じです。

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

接続のキャラクタセットおよび照合順序

複数のキャラクタセットとシステム変数はサーバとクライアント間の通信に関係しています。いくつかは前セクションですでに説明されています。

- サーバキャラクタセットと照合順序は、`character_set_server`および`collation_server`のシステム変数値で決定されます。
- デフォルトのデータベースに対するキャラクタセットと照合順序は、`character_set_database`および`collation_database`のシステム変数値によって決定されます。

追加キャラクタセットと照合順序システム変数がクライアント・サーバー間のコネクショントラフィックを統括する役目を担っています。どのクライアントもコネクション関係のキャラクタセットと照合順序システム変数を持っています。

「接続」とはなんでしょう。「接続」とは、サーバへの接続時に作成されるものです。クライアントは接続を介し、SQL ステートメント（クエリなど）をサーバに送信します。サーバでは接続を介し、応答（結果セットなど）をクライアントに返します。これによって、次のような、クライアントの接続についてキャラクタセットと照合順序に関する問が生じます。これら疑問に関する答えはシステム変数値によって導き出されます。

- クライアントから送信される際にステートメントはどのキャラクタセットで送られるのか。

サーバは `character_set_client` システム変数値をそのままクライアントの送るステートメントのキャラクタセットにします。

- サーバではクエリを受信した後にどのキャラクタセットに変換するのか。

これには、サーバは `character_set_connection` と `collation_connection` のシステム変数値を使用します。クライアントから送られたステートメントを `character_set_client` から `character_set_connection` に変換します(ただし `_latin1` あるいは `_utf8` のようなイントロデューサのある文字列リテラルは除く)。 `collation_connection` はリテラル文字列の比較にとって重要です。カラム値のある文字列の比較には、`collation_connection` は重要視されません。なぜならカラムには自身の照合順序があり、優先的にこれらの照合順序を参照するからです。

- サーバでは結果セットまたはエラーメッセージをクライアントに返送する前にどのキャラクタセットに変換するのか。

`character_set_results` のシステム変数値はサーバがどのキャラクタセットでクライアントにクエリ結果を返信するかを指定しています。これはカラム値やメタデータ結果に含まれるカラム名などの結果データを含みます。

これらは細かく調整することができますが、デフォルトを適用することもできます。デフォルトを適用する場合、このセクションをとおしてかまいません。

接続キャラクタセットに影響するステートメントが 2 つ存在します。

```
SET NAMES 'charset_name'
SET CHARACTER SET charset_name
```

`SET NAMES` は、クライアントから送信される SQL ステートメントのキャラクタセットを示します。たとえば、`SET NAMES 'cp1251'` は「このクライアントからの入力メッセージは今後、キャラクタセット `cp1251` になります」とサーバに通知します。加えて、クライアントに結果を返信する際サーバが使用するべきキャラクタセットも指定します。(例えば、`SELECT` ステートメントを使った場合どのカラム値に対してどのキャラクタセットを使用したいか指定します。)

`SET NAMES 'x'` ステートメントは下記の 3 ステートメントと等価です。

```
SET character_set_client = x;
SET character_set_results = x;
SET character_set_connection = x;
```

`character_set_connection` を `x` にセットすると `collation_connection` も `x` のデフォルト照合順序にセットされます。その照合順序を正確にセットする必要はありません。キャラクタセットに特定の照合順序を指定するには、オプションの `COLLATE` 節を使用してください。

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

`SET CHARACTER SET` は `SET NAMES` に似ていますが `character_set_connection` と `collation_connection` を `character_set_database` と `collation_database` にセットします。`SET CHARACTER SET x` ステートメントは下記の 3 ステートメントと等価です。

```
SET character_set_client = x;
SET character_set_results = x;
```

```
SET collation_connection = @@collation_database;
```

`collation_connection`を指定すると`character_set_connection`も照合順序に関係するキャラクタセットに指定されます(`SET character_set_connection = @@character_set_database`を実行することと同様)。`character_set_connection`を正確に指定する必要はありません。

クライアント接続時、使用したいキャラクタセット名がサーバーに送られます。サーバーはこのキャラクタセット名を使って`character_set_client`、`character_set_results`、そして`character_set_connection`のシステム変数値を指定します。結果的に、サーバーはキャラクタセット名を使用して`SET NAMES`オペレーションを実行します。

デフォルト以外のキャラクタセットを使用したい場合、`mysql` クライアントでは、起動するたびに `SET NAMES` を実行する必要はありません。`--default-character-set` オプション設定を`mysql` ステートメントラインか、オプションファイルに追加することができます。たとえば、以下のオプション設定ファイルの設定では、`mysql` を実行する度に、三つのキャラクタセット変数値を`koi8r`に変更します。

```
[mysql]
default-character-set=koi8r
```

もし`mysql`クライアントの自動再接続(推奨されていません)を使用しているのであれば、`SET NAMES`よりも`charset`を使用することをお勧めします。例：

```
mysql> charset utf8
Charset changed
```

`charset` コマンドは`SET NAMES` ステートメントを発行し、`mysql` 接続が解除され再接続された場合に使用されているデフォルトキャラクタセットも変更します。

例:例えば`column1`が`CHAR(5) CHARACTER SET latin2`として定義されていたとします。もし`SET NAMES`あるいは`SET CHARACTER SET`ではない場合、`SELECT column1 FROM t`に関してサーバーは`column1`の値を、接続時にクライアントが指定したすべての値を返信します。逆に`SET NAMES 'latin1'`あるいは`SET CHARACTER SET latin1`を`SELECT`ステートメントを発行する前に使用した場合、サーバーは結果を返信する前に`latin2`の値を`latin1`に変換します。そのような変換は低速であり、損失につながることもあります。

サーバーに結果セットの変換を実行をしてほしくない場合は、`character_set_results`を`NULL`に指定してください。

```
SET character_set_results = NULL;
```

注:現在、UCS-2 クライアントのキャラクタセットとして使用できません。これは`SET NAMES 'ucs2'`が使用できないことを意味します。

コネクションに関するキャラクタセットや照合順序システム変数値を参照するには、下記のステートメントを使用してください。

```
SHOW VARIABLES LIKE 'character_set%';
SHOW VARIABLES LIKE 'collation%';
```

照合順序に関して

下記のセクションではキャラクタセットの照合順序に関する事項を紹介します。

SQLステートメント`COLLATE`節を使用する

`COLLATE` 節では、比較に対するデフォルト照合順序が何であれ、無効にすることができます。SQLステートメントのさまざまな個所で`COLLATE`を使用することができます。以下に例を示します。

- `ORDER BY`を指定した場合

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- `AS`を指定した場合

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- **GROUP BY**を指定した場合

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- 集計関数を指定した場合

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- **DISTINCT**を指定した場合

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- **WHERE**を指定した場合

```
SELECT *
FROM t1
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
SELECT *
FROM t1
WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- **HAVING**を指定した場合

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

COLLATE節の優先順位

COLLATE節は優先順位が高いため(`||`よりも高い)、下記の2つの式は等価となります。

```
x || y COLLATE z
x || (y COLLATE z)
```

BINARY オペレータ

BINARYオペレータは2進性の文字列に続く文字列を送信します。キャラクタ毎よりも、バイト毎の比較を強制的に行う簡単な方法です。**BINARY**は後続のスペースにも重要な意味を持たせます。

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
mysql> SELECT 'a' = 'a ';
-> 1
mysql> SELECT BINARY 'a' = 'a ';
-> 0
```

BINARY strは**CAST(str AS BINARY)**の略でもあります。

キャラクタカラム定義の**BINARY**性質は別の効果があります。**BINARY**性質で定義されたキャラクタカラムはカラムのキャラクタセットの二進節を割り当てられます。全てのキャラクタセットに二進節があります。例えば、**latin1**キャラクタセットの二進節は**latin1_bin**です。よってデフォルトキャラクタセットが**latin1**の場合、下記の2カラムの定義は等価になります。

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

BINARYのカラム性質としての効果はMySQL4.1. の時のそれと効果に違いがあります。以前、**BINARY**は結果二進文字列として扱われたカラムになりました。二進文字列は、キャラクタセットや照合順序のないバイトの列で、二進照合順序のある非二審キャラクタ文字列とは違います。両文字列にとって、比較は文字列のユニットの数値をもって行われます。しかし非二審文字列にとってはユニットはキャラクタであり、キャラクタセットの中にはマルチバイトキャラクタを許容しているものもあります。**BINARY**と**VARBINARY**タイプ。

`CHARACTER SET binary`の`CHAR`、`VARCHAR`あるいは`TEXT`カラム定義での使用は二進性のデータタイプとしてカラムを扱うことになります。例えば、下記の定義は等価です

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

照合順序を決定するのが難しい特殊なケース

大多数のクエリでは、MySQL がどの照合順序により比較演算を行うかは明確になっています。たとえば以下の場合、照合順序がカラム`x`のカラム照合順序"になることは明らかです。

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

ただし、複数のオペランドが使用されていると、あいまいになることがあります。例：

```
SELECT x FROM T WHERE x = 'Y';
```

`x`の照合順序と文字列リテラル`'Y'`の照合順序のどちらがこのクエリで使用されるのでしょうか

標準 SQL では、「強制可能」ルールと呼ばれていた方法で上記の問題を解決しました。これについては、以下の発想が基本となっています。`x`と`'Y'`の両方に照合順序があるので、どちらの照合順序を優先するかを判断しなければならない。複雑な問題だが、これらのルールでほとんどの状況に対応できる。

- 明示的な `COLLATE` 節の優先順位は0です。(優先されません)
- 照合順序の異なる文字列 2 つの連結は優先順位が 1。
- カラムの照合順序、保存されたルーチンパラメータ、もしくはローカル変数値は優先順位が 2。
- 「システム定数」(`USER()` または `VERSION()` といったファンクションに返される文字列) の優先順位は3。
- リテラルの照合順序は優先順位が4。
- `NULL`もしくは`NULL`由来の表現の優先順位は5。

先行する優先順位の値は現MySQLのものである。5.1

これらのルールによって、あいまいさは次のように解消されます。

- 優先順位が最も低い照合順序を使用する。
- 双方の優先順位が一致する場合、照合順序が一致しないとエラーとなる。

例：

<code>column1 = 'A'</code>	<code>column1</code> 使用する照合順序
<code>column1 = 'A' COLLATE x</code>	<code>'A' COLLATE x</code> 使用する照合順序
<code>column1 COLLATE x = 'A' COLLATE y</code>	エラー

`COERCIBILITY()` ファンクションで文字列表現の優先順位が決定できます。

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY(VERSION());
-> 3
mysql> SELECT COERCIBILITY('A');
-> 4
```

詳しくはこちらをを参照してください。 [情報関数](#)

照合順序は適切なキャラクタセットに対応していること。

各キャラクタセットには 1 つ以上の照合順序があり、各照合順序は 1 つのキャラクタセットに関連付けられていることを思い出してください。したがって、次のステートメントはエラーになります。latin2_bin照合順序は latin1キャラクタセットに対して有効でないからです。

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

照合順序がもたらす結果の例

テーブル `T` のカラム `X` に以下の latin1 カラムの値が設定されているとします。

```
Muffler
Müller
MX Systems
MySQL
```

さらに、以下のステートメントを使用してカラムの値を取得したとします。

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

この表は、複数の異なった照合順序を `ORDER BY` 節で使用した結果の一例です。

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

例では、2 個の点の上に付いている `ü` が問題の原因です(ü)。この文字をドイツでは「ウムラウト」と呼んでいますが、私たちは分音記号付きの `U` と呼んでいます。

- 最初のカラムには、スウェーデン語/フィンランド語の照合ルールを使用した `SELECT` の結果が示されています。この照合ルールによると、分音記号付きの `U` は `Y` と一致します。
- 最初のカラムには、スウェーデン語/フィンランド語の照合ルールを使用した `SELECT` の結果が示されています。この照合ルールによると、分音記号付きの `U` は `Y` と一致します。
- 最初のカラムには、スウェーデン語/フィンランド語の照合ルールを使用した `SELECT` の結果が示されています。この照合ルールによると、分音記号付きの `U` は `Y` と一致します。

キャラクタセットのサポートによる影響を受ける演算

このセクションでは、キャラクタセット情報を計算に入れる演算について説明します。

結果文字列

MySQL には、文字列を返す多数の演算子と関数があります。このセクションでは、そのような文字列のキャラクタセットと照合順序について解説しています。

文字列の入力を取得して文字列の結果を出力として返す単純な関数では、出力のキャラクタセットおよび照合順序は主要な入力のキャラクタセットおよび照合順序と同じです。たとえば `UPPER(X)` は、キャラクタセットおよび照合が `X` と一致する文字列を返します。同じことはいかについても当てはまります。 `INSTR()`、`LCASE()`、`LOWER()`、`LTRIM()`、`MID()`、`REPEAT()`、`REPLACE()`、`REVERSE()`、`RIGHT()`、`RPAD()`、`RTRIM()`、`SOUNDEX()`、`SUBSTRING()`、`TRIM()`、`UCASE()` そして `UPPER()`。

注: `REPLACE()` 関数は他のすべての関数とは異なり、文字列入力の照合順序を無視し、大文字と小文字が区別される比較を毎回実行します。

入力文字列あるいはファンクションの結果がバイナリ文字列の場合、キャラクタセットあるいは照合順序に対する文字列はありません。これは `CHARSET()` と `COLLATION()` ファンクションを使ってチェックすることができます。両ファンクションとも、引数がバイナリ文字列であると明示するため `binary` を返します。

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+-----+-----+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+-----+-----+
| binary              | binary                  |
+-----+-----+
```

複数の文字列入力を組み合わせて単一の文字列出力を返す操作には、標準SQLの「集約ルール」が適用されます。

- 明示的な `COLLATE X` 存在する場合は `X` を使用する。
- 明示的な `COLLATE X` と `COLLATE Y` が存在する場合はエラーになる。
- 上記以外の場合で全ての照合順序が `X` であるときは `X` を使用する。
- その他の場合、結果に照合順序は含まれない。

例えば、`CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END` と指定されている場合、結果照合順序は `X` になります。同じことは以下についても当てはまります `UNION`、`||`、`CONCAT()`、`ELT()`、`GREATEST()`、`IF()`、`LEAST()`。

文字データを変換する操作のため、結果文字列のキャラクタセットと照合順序は `character_set_connection` と `collation_connection` システム変数値によって定義されています。このことは以下についてのみ当てはまります。 `CAST()`、`CONV()`、`FORMAT()`、`HEX()`、`SPACE()`。

もし文字列ファンクションに返された結果のキャラクタセットや照合順序にたいして不明の場合、確かめるために `CHARSET()` または `COLLATE()` ファンクションを使用してください。

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER() | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| test@localhost | utf8 | utf8_general_ci |
+-----+-----+-----+
```

CONVERT() と CAST()

`CONVERT()` を使用すると、異なるキャラクタセット間でデータを変換することができます。構文は以下のとおりです。

```
CONVERT(expr USING transcoding_name)
```

MySQL では、トランスコーディング名は対応するキャラクタセット名と同じです。

例：

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
  SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` は、標準SQLの仕様に基づき実装されています。

`CAST()` を使用し、文字列を別のキャラクタセットに変換することもできます。構文は以下のとおりです。

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

例：

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

`CAST()` を `CHARACTER SET` の指定なしで使用した場合、キャラクタセットと照合順序は `character_set_connection` と `collation_connection` のシステム変数で定義されます。 `CAST()` を `CHARACTER SET X` の指定ありで使用した場合、キャラクタセットは `X`、照合順序は `X` のデフォルト照合順序になります。

`COLLATE`節を`CAST()`の内部で使用することはできませんが外部では使用することができます。したがって`CAST(... COLLATE ...)`無効ですが、`CAST(...) COLLATE ...`は有効です。

例：

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

SHOW ステートメントとINFORMATION_SCHEMA

複数の`SHOW` ステートメントからキャラクタセットの追加情報が得られます。これらの中には`SHOW CHARACTER SET`、`SHOW COLLATION`、`SHOW CREATE DATABASE`、`SHOW CREATE TABLE` そして`SHOW COLUMNS`が含まれます。これらのステートメントを以下に簡単に挙げます。

さらに詳しい情報が必要な場合は、次を参照してください[SHOW 構文](#)。

`INFORMATION_SCHEMA` には`SHOW`ステートメントで表示される情報に類似した複数のテーブルが含まれます。例えば、`CHARACTER_SETS`および`COLLATIONS` テーブルは`SHOW CHARACTER SET`および`SHOW COLLATION`で表示される情報を含みます。 [21章INFORMATION_SCHEMA データベース](#)。

`SHOW CHARACTER SET` コマンドは全ての有効なキャラクタセットを示します。どのキャラクタセット名を整合させるかはオプションの`LIKE`節が使用されます。例：

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description           | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |
| latin5  | ISO 8859-9 Turkish   | latin5_turkish_ci | 1      |
| latin7  | ISO 8859-13 Baltic   | latin7_general_ci | 1      |
+-----+-----+-----+-----+
```

`SHOW COLLATION`からの出力は全ての有効なキャラクタセットを含みます。どの照合順序名を整合させるかはオプションの`LIKE`節が使用されます。例：

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | | | 0 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 0 |
| latin1_danish_ci | latin1 | 15 | | | 0 |
| latin1_german2_ci | latin1 | 31 | | Yes | 2 |
| latin1_bin | latin1 | 47 | | Yes | 0 |
| latin1_general_ci | latin1 | 48 | | | 0 |
| latin1_general_cs | latin1 | 49 | | | 0 |
| latin1_spanish_ci | latin1 | 94 | | | 0 |
+-----+-----+-----+-----+-----+-----+
```

`SHOW CREATE DATABASE`は指定されたデータベースを作成する`CREATE DATABASE`ステートメントを示します。

```
mysql> SHOW CREATE DATABASE test;
+-----+-----+
| Database | Create Database |
+-----+-----+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+-----+
```

`COLLATE`節が表示されていなければキャラクタセットのデフォルト照合順序が適用されます。

`SHOW CREATE TABLE` は類似していますが、`CREATE TABLE`既存のテーブルを作成するためのステートメントを表示します。カラム定義はキャラクタセット仕様を指定し、テーブルオプションはキャラクタセット情報を含んでいます。

`SHOW COLUMNS`ステートメントは`SHOW FULL COLUMNS`として実行された場合、テーブルカラムの照合順序を表示します。`CHAR`、`VARCHAR`、または`TEXT`データ型を含むカラムには照合順序があります。ニューメリックと他の文字列の存在しない型には照合順序がありません(`Collation`値として`NULL`で表示されます)。例：

```
mysql> SHOW FULL COLUMNS FROM person\G
***** 1. row *****
Field: id
Type: smallint(5) unsigned
Collation: NULL
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
Privileges: select,insert,update,references
Comment:
***** 2. row *****
Field: name
Type: char(60)
Collation: latin1_swedish_ci
Null: NO
Key:
Default:
Extra:
Privileges: select,insert,update,references
Comment:
```

キャラクタセットは表示の一部ではなく照合順序名で示されます。

Unicodeのサポート

MySQL 5.1 Unicode データを保存するために次の二つのキャラクタセットが用意されています

- [ucs2](#) (the UCS-2 Unicode キャラクタセット)
- [utf8](#) (Unicode キャラクタセットのUTF-8 エンコード)

UCS-2 (Unicode のバイナリ表現) では、各文字は 2 バイトの Unicode と最上位のバイトで最初に表現されます。例 : [LATIN CAPITAL LETTER A](#) はコード [0x0041](#) で、2 バイトシーケンスで保存されます。 [0x00 0x41](#)。 [CYRILLIC SMALL LETTER YERU](#) (Unicode [0x044B](#)) は2バイトシーケンスとして保存されます。 [0x00 0x41](#)。Unicode 文字および対応するコードについては、[Unicode Home Page](#)を参照してください。

現在、UCS-2はクライアントキャラクタセットとして使用ができません。これは[SET NAMES 'ucs2'](#)が使用できないことを意味します。

UTF-UTF8 キャラクタセット (Unicode 表現の変換) は、Unicode データを保存する別の方法です。RFC3629 に基づき実装されています。さまざまな Unicode 文字を長さの異なるバイトシーケンスに適合させることが、UTF8 キャラクタセットの概念です。

- 基本的なラテン文字、数字、句読点は 1 バイトを使用します。
- ヨーロッパおよび中東のスク립ト文字の多くは、2 バイトシーケンスに適合します。拡張ラテン文字 (チルダ、長音、鋭アクセント、抑音アクセントその他のアクセント付き)、キリル文字、ギリシア文字、アルメニア文字、ヘブライ文字、アラビア文字、シリア文字その他。
- 韓国語、中国語、日本語の表意文字は、3 バイトシーケンスを使用します。

RFC 3629 は 1 から4バイトまでのエンコードシーケンスを示します。現在、MySQLではUTF-8に対して4バイトシーケンスのサポートは含みません。(UTF-8 エンコードの旧標準はRFC 2279で提供されています。これは 1 から6バイトのUTF-8シーケンスを示しています。RFC 3629 はRFC 2279 を無効にします。このため、5と6バイトのシーケンスはすでに使用されていません。

ヒント : UTF8においてスペースを節約するには、[CHAR](#)ではなく[VARCHAR](#)を使用してください。そのようにしないと、MySQL では [CHAR CHARACTER SET utf8](#)で指定されたカラムに対して、一文字につき、一文字が取り得るサイズ 3バイトを常に確保しなければならないからです。たとえば、MySQLは[CHAR\(10\) CHARACTER SET utf8](#)カラムに対して30バイトを確保しなければなりません。

メタデータ用の UTF8

メタデータとは「データについてのデータです。」データベース—の内容となっているデータではなく、データベース—について説明するデータがメタデータです。したがって、カラム名、データベース名、ユーザ名、バージョン名

のほか、[SHOW](#)を実行して表示される文字列の多くがメタデータに該当します。[INFORMATION_SCHEMA](#)のテーブルコンテンツに関しても同様です。というのも、それらのテーブルにはデータベースオブジェクトに関する情報が含まれることが定義されているからです。

メタデータの表現は下記の要求を満たしていなければなりません。

- すべてのメタデータはキャラクタセットが一致している必要があります。そうっていない場合、[SHOW](#)コマンドや[INFORMATION_SCHEMA](#)のテーブル[SELECT](#)ステートメントは正確には働きません。なぜなら、これらの同じ演算結果カラムの文字列は、異なるキャラクタセットに存在するからです。
- メタデータは全ての言語の全ての文字を含んでいる必要があります。そうっていない場合、ユーザは各々の言語を使用してカラムとテーブルに名前をつけることはできません。

上記 2 つの要求を満たすために、MySQL ではメタデータが Unicode キャラクタセット (UTF8) で保存されます。これによって不具合が発生しないのは、アクセント付き文字を使用しない場合です。使用する場合、メタデータのキャラクタセットが UTF8 であることを認識する必要があります。

メタデータ要求は、[USER\(\)](#)、[CURRENT_USER\(\)](#)、[SESSION_USER\(\)](#)、[SYSTEM_USER\(\)](#)、[DATABASE\(\)](#)、そして[VERSION\(\)](#)関数では、UTF-8キャラクタセットがデフォルトで使用されることを意味します。

サーバでは[character_set_system](#)のシステム変数がメタデータキャラクタセットに名前をつけることができます。

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_system | utf8 |
+-----+-----+
```

Unicodeを用いたメタデータの保存は、サーバが、カラムのヘッダーやデフォルトの[character_set_system](#)キャラクタセット内の結果[DESCRIBE](#) 関数を返すということではありません。[SELECT column1 FROM t](#)を使用すると、[column1](#)の名前がそのままサーバから、[character_set_results](#)システム変数値によって定義されるキャラクタセット ([latin1](#)のデフォルト値) のクライアントに返されます。異なるキャラクタセットでメタデータ結果をサーバから返してほしいときは、[SET NAMES](#)ステートメントを使用してキャラクタセット変換を強制的に実行させましょう。[SET NAMES](#)は[character_set_results](#)と他の関連システム変数を設定します。(詳しくは[接続のキャラクタセットおよび照合順序](#)をご確認ください。) また、サーバから結果を受け取った後、クライアントプログラムが変換を実行することができます。クライアントが変換を実行するほうが効率的ですが、常にクライアントにオプション選択ができるわけではありません。

[character_set_results](#) がNULLに設定されている場合、変換は実行されず、サーバはオリジナルのキャラクタセットを使用してメタデータを返します。(設定は[character_set_system](#)によって指定されます)。

サーバからクライアントへのエラーメッセージは、自動的にメタデータとしてクライアントのキャラクタセットに変換されます。

たとえば[USER\(\)](#)たとえば、[USER\(\)](#) 関数を比較または割当のために単一のステートメントで使用しているとします。MySQL には自動変換機能が用意されています。

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

この機能が有効なのは、[latin1_column](#)の内容が UTF8 へと自動的に変換されてから比較が行われるからです。

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

この機能が有効なのは、[USER\(\)](#)の内容が [latin1](#)へと自動的に変換されてから割り当てが行われるからです。自動変換機能は完全には実装されていませんが、将来のバージョンでは適切に動作する予定です。

自動変換機能は SQL 標準に含まれていません。ただし、どのキャラクタセットも (サポートされている文字に関して) Unicode の「subset」であることが SQL 標準の文書に記載されています。「「スーパーセットに適用されるものはサブセットにも適用される」という有名な原則があるので、Unicode の照合順序は Unicode 以外の文字列との比較にも適用できると考えられます。

カラムキャラクタセット変換

特定のキャラクタセット使用のため、バイナリもしくはバイナリではない文字列カラムの変換には[ALTER TABLE](#)を使用してください。正確な変換のためには、以下の条件の内一つが適用されなければなりません。

- カラムがバイナリデータ型(**BINARY**、**VARBINARY**、**BLOB**)である場合、含まれる全ての値は1つのキャラクタセット (カラムを変換しようとしているキャラクタセット) を使ってエンコードされている必要があります。バイナリカラムを使って複数のキャラクタセット情報を保存する場合、MySQLはどの値がどのキャラクタセットを使用するのか特定できず、データを正確に変換できません。
- カラムにバイナリではないデータ型(**CHAR**、**VARCHAR**、**TEXT**)が存在する場合、内容は他のキャラクタセットではなく、カラムのキャラクタセットでエンコードされている必要があります。内容が他のキャラクタセットでエンコードされている場合、先にバイナリデータ型を使用するようカラムを変換して、望ましいキャラクタセットのバイナリでないカラムに変換することができます。

例えばテーブル `t` が **BINARY(50)** として定義されている `col1` というバイナリカラムだとします。カラムに含まれる情報が1つのキャラクタセットを使用してエンコードされていると想定して、キャラクタセットを持つバイナリでないカラムに変換することができます。例えば、`col1` には `greek` の文字を表すキャラクタセットのバイナリデータが含まれる場合、以下のように変換できます。

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET greek;
```

例えばそのテーブル `t` は `col1` という **CHAR(50) CHARACTER SET latin1** と定義された、バイナリでないカラムを含むとします。`utf8` を使用し多言語からの値を保存するために変換したいとします。以下のステートメントはこれを実行します。

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

カラムに、両キャラクタセットに含まれない文字がある場合、変換は正確に実行されません。

4.0もしくはそれ以前のバージョンから古いテーブルを使用する場合、特定のケースが生じます。ここでは、バイナリでないカラムは、サーバのデフォルトキャラクタセットと異なるキャラクタセットでエンコードされた値を含みます。例えば、MySQLのデフォルトキャラクタセットが `latin1` であっても、アプリケーションは `sjis` 値をカラムに保存します。適切なキャラクタセットを使用するために、カラムを変換することは可能ですが、追加ステップが必要となります。例えばサーバのデフォルトキャラクタセットが `latin1` で `col1` は **CHAR(50)** と定義されているにもかかわらず、内容は `sjis` 値とします。最初のステップは、バイナリデータ型にカラムを変換することで、既存のキャラクタセット情報を文字変換なしで取り除くことです。

```
ALTER TABLE t MODIFY col1 BINARY(50);
```

次のステップは適切なキャラクタセットを用いたバイナリでないデータ型にカラムを変換することです。:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

このプロシージャは、MySQL 4.1もしくはそれ以降にアップグレード後、テーブルが **INSERT** や **UPDATE** のようなステートメントで修正されていないことが求められます。その場合、MySQL は `latin1` を使い新しい値を保存し、カラムは `sjis` と `latin1` 値を同時に含んでおり、正確に変換することはできません。

最初にカラムを作成するときに属性を特定した場合、**ALTER TABLE** を使ってテーブルを変更しているときも属性を特定する必要があります。例えば **NOT NULL** と明示的な **DEFAULT** 値を特定した場合、**ALTER TABLE** ステートメントでも特定する必要があります。でなければ、結果のカラム定義にはそれらの属性が含まれません。

MySQL でサポートされるキャラクタセットと照合順序

MySQL では、30 を超えるキャラクタセットに対して 70 を超える照合順序がサポートされています。このセクションではMySQLがどのキャラクタセットをサポートするかを示しています。関連するキャラクタセットごとに、1つのサブセクションがあります。各キャラクタセットに対して有効な照合順序がリストアップされています。

SHOW CHARACTER SET ステートメントを用いて、有効なキャラクタセットとそのデフォルトの照合順序を常に表示することができます。

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+
| Charset | Description          | Default collation |
+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci   |
| dec8    | DEC West European      | dec8_swedish_ci   |
| cp850   | DOS West European      | cp850_general_ci  |
```

```

| hp8      | HP West European      | hp8_english_ci |
| koi8r    | KOI8-R Relcom Russian | koi8r_general_ci |
| latin1   | cp1252 West European  | latin1_swedish_ci |
| latin2   | ISO 8859-2 Central European | latin2_general_ci |
| swe7     | 7bit Swedish          | swe7_swedish_ci |
| ascii    | US ASCII               | ascii_general_ci |
| ujis     | EUC-JP Japanese       | ujis_japanese_ci |
| sjis     | Shift-JIS Japanese    | sjis_japanese_ci |
| hebrew   | ISO 8859-8 Hebrew     | hebrew_general_ci |
| tis620   | TIS620 Thai           | tis620_thai_ci |
| euckr    | EUC-KR Korean         | euckr_korean_ci |
| koi8u    | KOI8-U Ukrainian     | koi8u_general_ci |
| gb2312   | GB2312 Simplified Chinese | gb2312_chinese_ci |
| greek    | ISO 8859-7 Greek      | greek_general_ci |
| cp1250   | Windows Central European | cp1250_general_ci |
| gbk      | GBK Simplified Chinese | gbk_chinese_ci |
| latin5   | ISO 8859-9 Turkish    | latin5_turkish_ci |
| armSCII8 | ARMSCII-8 Armenian    | armSCII8_general_ci |
| utf8     | UTF-8 Unicode         | utf8_general_ci |
| ucs2     | UCS-2 Unicode         | ucs2_general_ci |
| cp866    | DOS Russian           | cp866_general_ci |
| keybcs2  | DOS Kamenicky Czech-Slovak | keybcs2_general_ci |
| macce    | Mac Central European  | macce_general_ci |
| macroman | Mac West European    | macroman_general_ci |
| cp852    | DOS Central European  | cp852_general_ci |
| latin7   | ISO 8859-13 Baltic    | latin7_general_ci |
| cp1251   | Windows Cyrillic      | cp1251_general_ci |
| cp1256   | Windows Arabic        | cp1256_general_ci |
| cp1257   | Windows Baltic        | cp1257_general_ci |
| binary   | Binary pseudo charset | binary |
| geostd8  | GEOSTD8 Georgian     | geostd8_general_ci |
| cp932    | SJIS for Windows Japanese | cp932_japanese_ci |
| eucjpms  | UJIS for Windows Japanese | eucjpms_japanese_ci |
+-----+-----+-----+

```

Unicode キャラクタセット

MySQLにはキャラクタセットが2つ存在します。これらのキャラクタセットを使用し、約650の言語でテキストを保存することができます。

- [ucs2](#) (UCS-2 Unicode) 照合順序

- [ucs2_bin](#)
- [ucs2_czech_ci](#)
- [ucs2_danish_ci](#)
- [ucs2_esperanto_ci](#)
- [ucs2_estonian_ci](#)
- [ucs2_general_ci](#) (デフォルト)
- [ucs2_hungarian_ci](#)
- [ucs2_icelandic_ci](#)
- [ucs2_latvian_ci](#)
- [ucs2_lithuanian_ci](#)
- [ucs2_persian_ci](#)
- [ucs2_polish_ci](#)
- [ucs2_roman_ci](#)
- [ucs2_romanian_ci](#)

- `ucs2_slovak_ci`
- `ucs2_slovenian_ci`
- `ucs2_spanish2_ci`
- `ucs2_spanish_ci`
- `ucs2_swedish_ci`
- `ucs2_turkish_ci`
- `ucs2_unicode_ci`
- `utf8` (UCS-8 Unicode) 照合順序
 - `utf8_bin`
 - `utf8_czech_ci`
 - `utf8_danish_ci`
 - `utf8_esperanto_ci`
 - `utf8_estonian_ci`
 - `utf8_general_ci` (デフォルト)
 - `utf8_hungarian_ci`
 - `utf8_icelandic_ci`
 - `utf8_latvian_ci`
 - `utf8_lithuanian_ci`
 - `utf8_persian_ci`
 - `utf8_polish_ci`
 - `utf8_roman_ci`
 - `utf8_romanian_ci`
 - `utf8_slovak_ci`
 - `utf8_slovenian_ci`
 - `utf8_spanish2_ci`
 - `utf8_spanish_ci`
 - `utf8_swedish_ci`
 - `utf8_turkish_ci`
 - `utf8_unicode_ci`

注 : `ucs2_roman_ci`と`utf8_roman_ci`の照合順序では、**I**と**J**および**U**と**V**は等価として比較されます。

`ucs2_hungarian_ci`と`utf8_hungarian_ci`の照合順序はMySQL 5.1.5で追加されています。

MySQL は`utf8_unicode_ci`照合順序を、以下に示される<http://www.unicode.org/reports/tr10/>Unicode 照合順序アルゴリズム(UCA) によって実行されます。照合順序ではバージョン-4.0.0 UCAのウェイトキーが使用されます。<http://>

www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt 以下では `utf8_unicode_ci` を使用しますが、`ucs2_unicode_ci` に対しても有効です。

現在、`utf8_unicode_ci` 照合順序は Unicode 照合順序アルゴリズムを部分的にのみサポートしています。中にはまだサポートされていない文字もあります。また、結合マークは完全にはサポートされていません。このことはベトナム語を中心に、ロシア内のマイノリティ言語に影響します。

`utf8_unicode_ci` 主な特徴は、拡張をサポートしていることです。それは1つの文字が他の文字のコンビネーションと等価であると比較された場合です。例えば、ドイツ語や他の言語では、`'ß'` は `'ss'` に相当します。

`utf8_general_ci` は拡張をサポートしないレガシー照合順序です。文字間で1対1の比較しかできません。つまり、`utf8_general_ci` 照合順序に対する比較の方が早いですが、`utf8_unicode_ci` に比べてわずかに正確性が劣ります。

例えば、以下の等式が `utf8_general_ci` と `utf8_unicode_ci` において証明されます。

```
Ä = A
Ö = O
Ü = U
```

照合順序間の違いは `utf8_general_ci` に対して有効です。

```
ß = s
```

`utf8_unicode_ci` にとって有効である場合：

```
ß = ss
```

MySQL が `utf8` キャラクタセットに対する特定言語の照合順序を実行するのは `utf8_unicode_ci` 内での順序が言語に対してうまく機能しないときだけです。例えば、`utf8_unicode_ci` はドイツ語とフランス語では正しく機能するので、これら2言語に対する特定の `utf8` 照合順序を作成する必要はありません。

`utf8_general_ci` もドイツ語・フランス語ともに有効ですが、ただし `'ß'` は `'s'` と等価であり、`'ss'` と等価ではありません。これが使用しているアプリケーションに対して可能な場合、`utf8_general_ci` の方が早いのでこちらをお勧めします。それ以外の場合、`utf8_unicode_ci` を使うほうがより正確です。

`utf8_swedish_ci` は他の `utf8` 特定言語の照合順序のように、追加言語ルールとともに `utf8_unicode_ci` から由来します。例えば、スウェーデン語ではドイツ語やフランス語を使用するユーザでは予期しないような以下の関係が有効です

```
Ü = Y < Ö
```

`utf8_spanish_ci` と `utf8_spanish2_ci` 照合順序は現代スペイン語および従来のスペイン語に対しても同様に対応します。両照合順序において `'ñ'` (n-tilde) は、`'n'` や `'o'` とは区別すべき文字です。さらに従来のスペイン語では `'ch'` は `'c'` と `'d'` は区別すべき文字であり、`'ll'` は `'l'` と `'m'` とともに区別すべき文字です。

西ヨーロッパのキャラクタセット

西ヨーロッパのキャラクタセットには、大部分の西ヨーロッパ言語（フランス語、スペイン語、カタロニア語、バスク語、ポルトガル語、イタリア語、アルバニア語、オランダ語、ドイツ語、デンマーク語、スウェーデン語、ノルウェー語、フィンランド語、フェロー語、アイスランド語、アイルランド語、スコットランド語、英語など）が含まれています。

- `ascii` (US ASCII) 照合順序:
 - `ascii_bin`
 - `ascii_general_ci` (デフォルト)
- `cp850` (DOS 西ヨーロッパ言語) 照合順序:
 - `cp850_bin`
 - `cp850_general_ci` (デフォルト)
- `dec8` (DEC 西ヨーロッパ言語) 照合順序:

- [dec8_bin](#)
- [dec8_swedish_ci](#) (デフォルト)
- [hp8](#) (HP 西ヨーロッパ言語) 照合順序:
 - [hp8_bin](#)
 - [hp8_english_ci](#) (デフォルト)
- [latin1](#) (cp1252 西ヨーロッパ言語) 照合順序:
 - [latin1_bin](#)
 - [latin1_danish_ci](#)
 - [latin1_general_ci](#)
 - [latin1_general_cs](#)
 - [latin1_german1_ci](#)
 - [latin1_german2_ci](#)
 - [latin1_spanish_ci](#)
 - [latin1_swedish_ci](#) (デフォルト)

[latin1](#) はデフォルトキャラクタセットです。MySQLの[latin1](#) はWindows[cp1252](#)キャラクタセットと同じです。つまり公式ISO 8859-1 あるいはIANA (アイアナ) [latin1](#)と同様ですが、アイアナ[latin1](#) は0x80 と0x9f間のコードポイントは「定義されていません」。これに対して、[cp1252](#)、つまりMySQLの[latin1](#)はそれらポジションに対して文字を指定しています。例えば 0x80 はユーロのサインです。[cp1252](#)の「定義されていない」エントリーでは、MySQLは0x81 をUnicode 0x0081、0x8d を0x008d、0x8fを0x008f、0x90を0x0090そして0x9dを0x009dに変換します。

[latin1_swedish_ci](#)照合順序は大部分のMySQLカスタマに使用されているデフォルトです。スウェーデン語/フィンランド語の照合順序ルールに基づいているとされていますが、スウェーデン人やフィンランド人の中にはこのステートメントに賛成しない人もいます。

[latin1_german1_ci](#)と[latin1_german2_ci](#)照合順序はDIN-1 およびDIN-2 標準に基づいて、DINはDeutsches Institut für Normung (アンシーのドイツ版)を表しています。DIN-1 は「辞書の照合順序」と呼ばれ、DIN-2は「電話帳の照合順序」と呼ばれています。

- [latin1_german1_ci](#) (辞書) ルール:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- [latin1_german2_ci](#) (電話帳) ルール:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

[latin1_spanish_ci](#) 照合順序では、'ñ' (n-tilde) は'n'と'o'とは区別されます。

- [macroman](#) (Mac 西ヨーロッパ言語) 照合順序:
 - [macroman_bin](#)
 - [macroman_general_ci](#) (デフォルト)
- [swe7](#) (7bit Swedish) 照合順序:

- [swe7_bin](#)
- [swe7_swedish_ci](#) (デフォルト)

中央ヨーロッパのキャラクタセット

MySQLではチェコ、スロバキア、ハンガリー、ルーマニア、スロベニア、クロアチア、ポーランドで使用するキャラクタセットも一部サポートされています。

- [cp1250](#) (Windows 中央ヨーロッパ言語) 照合順序:

- [cp1250_bin](#)
- [cp1250_croatian_ci](#)
- [cp1250_czech_cs](#)
- [cp1250_general_ci](#) (デフォルト)
- [cp1250_polish_ci](#)

- [cp852](#) (DOS 中央ヨーロッパ言語) 照合順序:

- [cp852_bin](#)
- [cp852_general_ci](#) (デフォルト)

- [keybcs2](#) (DOS Kamenicky Czech-Slovak) 照合順序 :

- [keybcs2_bin](#)
- [keybcs2_general_ci](#) (デフォルト)

- [latin2](#) (ISO 8859-2 中央ヨーロッパ言語) 照合順序:

- [latin2_bin](#)
- [latin2_croatian_ci](#)
- [latin2_czech_cs](#)
- [latin2_general_ci](#) (デフォルト)
- [latin2_hungarian_ci](#)

- [macce](#) (Mac 中央ヨーロッパ言語) 照合順序:

- [macce_bin](#)
- [macce_general_ci](#) (デフォルト)

南ヨーロッパおよび中東のキャラクタセット

MySQLでは南ヨーロッパや中東、アルメニア語、アラビア語、グルジア語、ギリシャ語、ヘブライ語、トルコ語のキャラクタセットがサポートされています。

- [armscii8](#) (ARMSII-8 Armenian) 照合順序 :

- [armscii8_bin](#)
- [armscii8_general_ci](#) (デフォルト)

- [cp1256](#) (Windows アラビア語) 照合順序:

- [cp1256_bin](#)
- [cp1256_general_ci](#) (デフォルト)
- [geostd8](#) (GEOSTD8 Georgian) 照合順序:
 - [geostd8_bin](#)
 - [geostd8_general_ci](#) (デフォルト)
- [greek](#) (ISO 8859-7 ギリシャ語) 照合順序:
 - [greek_bin](#)
 - [greek_general_ci](#) (デフォルト)
- [hebrew](#) (ISO 8859-8 ヘブライ語) 照合順序:
 - [hebrew_bin](#)
 - [hebrew_general_ci](#) (デフォルト)
- [latin5](#) (ISO 8859-9 トルコ語) 照合順序:
 - [latin5_bin](#)
 - [latin5_turkish_ci](#) (デフォルト)

バルト語のキャラクタセット

バルト語に含まれるキャラクタセットにはエストニア語、ラトビア語、そしてリトアニア言語が含まれます。

- [cp1257](#) (Windows バルト語) 照合順序:
 - [cp1257_bin](#)
 - [cp1257_general_ci](#) (デフォルト)
 - [cp1257_lithuanian_ci](#)
- [latin7](#) (ISO 8859-13 バルト語) 照合順序:
 - [latin7_bin](#)
 - [latin7_estonian_cs](#)
 - [latin7_general_ci](#) (デフォルト)
 - [latin7_general_cs](#)

キリル語のキャラクタセット

ベラルーシ語、ブルガリア語、ロシア語、ウクライナ語と共に使用するキリル語のキャラクタセットおよび照合順序を以下に示します。

- [cp1251](#) (Windows キリル語) 照合順序:
 - [cp1251_bin](#)
 - [cp1251_bulgarian_ci](#)
 - [cp1251_general_ci](#) (デフォルト)
 - [cp1251_general_cs](#)

- [cp1251_ukrainian_ci](#)
- [cp866](#) (DOS ロシア語) 照合順序:
 - [cp866_bin](#)
 - [cp866_general_ci](#) (デフォルト)
- [koi8r](#) (KOI8-R Relcom Russian) 照合順序:
 - [koi8r_bin](#)
 - [koi8r_general_ci](#) (デフォルト)
- [koi8u](#) (KOI8-U ウクライナ語) 照合順序:
 - [koi8u_bin](#)
 - [koi8u_general_ci](#) (デフォルト)

アジアのキャラクタセット

サポートされているアジアのキャラクタセットには、中国語、日本語、韓国語、タイ語が含まれています。これらは複雑な場合があります。たとえば、中国語のキャラクタセットは数千種類の文字に対応していなければなりません。[cp932](#) および [sjis](#) キャラクタセットについてのさらに詳しい情報は次を参照してください。[cp932のキャラクタセット](#)

MySQLのアジアのキャラクタセットに関する一般的な質問や問題のサポートについては、次を参照してください。[MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets](#)

- [big5](#) (Big5 Traditional Chinese) 照合順序 :
 - [big5_bin](#)
 - [big5_chinese_ci](#) (デフォルト)
- [cp932](#) (SJIS for Windows Japanese) 照合順序 :
 - [cp932_bin](#)
 - [cp932_japanese_ci](#) (デフォルト)
- [eucjpms](#) (UJIS for Windows Japanese) 照合順序 :
 - [eucjpms_bin](#)
 - [eucjpms_japanese_ci](#) (デフォルト)
- [euckr](#) (EUC-KR Korean) 照合順序:
 - [euckr_bin](#)
 - [euckr_korean_ci](#) (デフォルト)
- [gb2312](#) (GB2312 Simplified Chinese) 照合順序:
 - [gb2312_bin](#)
 - [gb2312_chinese_ci](#) (デフォルト)
- [gbk](#) (GBK Simplified Chinese) 照合順序:
 - [gbk_bin](#)
 - [gbk_chinese_ci](#) (デフォルト)

- [sjis](#) (Shift-JIS Japanese) 照合順序:
 - [sjis_bin](#)
 - [sjis_japanese_ci](#) (デフォルト)
- [tis620](#) (TIS620 Thai) 照合順序:
 - [tis620_bin](#)
 - [tis620_thai_ci](#) (デフォルト)
- [ujis](#) (EUC-JP Japanese) 照合順序:
 - [ujis_bin](#)
 - [ujis_japanese_ci](#) (デフォルト)

cp932のキャラクタセット

なぜcp932は必要なのでしょう。

MySQLでは[sjis](#) キャラクタセットはアイアナで定義される[Shift_JIS](#)キャラクタセットに対応しており、これらはJIS X0201およびJIS X0208キャラクタセットをサポートしています。(詳しくは<http://www.iana.org/assignments/character-sets>をご確認ください。)

しかしながら、記述用語として一般的に使われている「SHIFT JIS」の意味は 非常にあいまいで、しばしば各ベンダーが独自に[Shift_JIS](#)を拡張したものまで含まれることがあります。

例えば、日本語Windows環境で使用される「シフトJIS」はMicrosoftによる[Shift_JIS](#)の拡張で、正式な名称は[Microsoft Windows Codepage :932](#)もしくは[cp932](#)といいます。[cp932](#)では[Shift_JIS](#)でサポートされる文字に加え、NEC特殊文字、NEC選定IBM拡張文字—、IBM拡張文字といった 各種拡張文字がサポートされています。

多くの日本語ユーザーがこれら拡張文字等の使用にあたって 問題に直面してきました。これらの問題は以下の要因によって生じていました:

- MySQLが自動的にキャラクターセットの変換を行う。
- キャラクタセットの変換はUnicode([ucs2](#))を介して行われる。
- [sjis](#) キャラクタセットはこれら拡張文字の変換をサポートしていない。
- いわゆる「シフトJIS」と呼ばれるキャラクターセットからUnicodeへの変換には 複数の変換ルールが存在し、いくつかの文字は変換ルールによって異なるUnicode文字に 変換される。MySQLではこれらの変換ルールのうち、一つだけしかサポートされていない (詳細は後述する)。

MySQLの[cp932](#)キャラクタセットはこれらの問題を解決するよう デザインされています。

MySQLがキャラクターセットの 変換をサポートするようになった為、異なる変換ルールを持つIANAの[Shift_JIS](#)と[cp932](#)を二種類の キャラクタセットとして区別することが重要となります。

[cp932](#) は[sjis](#)とどう異なるか

[cp932](#)キャラクタセットは以下の点で[sjis](#)と異なります。

- [cp932](#)ではNEC特殊文字、NEC選定IBM拡張文字—、IBM拡張文字がサポートされる。 .
- いくつかの[cp932](#) 文字については、二つの異なる コードポイントから同一のUnicodeコードポイントに変換される。よってこれらの文字をUnicodeから[cp932](#)に戻す際には 何れか一つのコードポイントが選択されなくてはならない。この「ラウンドトリップ変換」については、Microsoftによって 推奨されるルールが使用されています。(詳しくは<http://support.microsoft.com/kb/170559/EN-US/>をご確認ください。)

この変換ルールは以下のとおりです。

- 当該文字がJIS X 0208文字とNEC特殊文字の両方に存在する場合には、 JIS X 0208のコードポイントを使用する。
- 当該文字がNEC特殊文字とIBM拡張文字の両方に存在する場合には、 NEC特殊文字のコードポイントを使用します。
- 当該文字がNEC選定—IBM拡張文字とIBM拡張文字の両方に存在する場合には、 IBM拡張文字のコードポイントを使用します。

<http://www.microsoft.com/globaldev/reference/dbcs/932.htm>で表示されるテーブルはcp932文字のUnicodeポイントに関する情報です。cp932の表に記載されている文字のうち、下に四桁の数字が表示されているものについては、その数字は対応するUnicode ucs2) コードポイントを表します。下線付きの二桁の値については これら二桁の値で始まる一連のcp932文字があることを表しています。これらの値をクリックすることで、その二桁の値で始まる cp932文字、及びそのUnicodeコードポイントが表示されます。

更に興味のある方は以下のリンクを参照してください。それぞれ、各文字の対応する Unicodeコードポイントを表示します。

- NEC 特殊文字:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm

- NEC 選定— IBM 拡張文字:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm

- IBM 選定文字:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_FA.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FB.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FC.htm

- [eucjpms](#)キャラクターセットと組み合わせて 使用することで、 cp932 でユーザー定義文字の変換がサポートされ、[sjis/ujis](#)間での変換問題にも対応しています。詳細は次を<http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html>参照してください。

いくつかの文字については、[sjis](#)とcp932.でucs2との変換ルールが異なります。以下のテーブルはこれらの違いを例示したものです。

ucs2への変換:

sjis/cp932 値	sjis -> ucs2 変換	cp932 -> ucs2 変換
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

ucs2からの変換:

ucs2 値	ucs2 -> sjis 変換	ucs2 -> cp932 変換
005C	815F	5C

007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

日本語キャラクタセットのユーザーは`--character-set-client-handshake`(もしくは `--skip-character-set-client-handshake`)を使うことで重要な効果があることに注意しなければなりません。詳しくは[コマンド オプション](#)を参照してください。

第10章 データタイプ

目次

データタイプ概要	541
数値タイプの概要	541
データと時刻タイプの概要	544
文字列タイプの概要	545
データタイプデフォルト値	547
数値タイプ	548
日付と時刻タイプ	550
DATETIME、DATE、そして TIMESTAMP タイプ	551
TIME タイプ	555
YEAR タイプ	556
2000年問題とデータタイプ	556
文字列タイプ	557
CHAR と VARCHAR タイプ	557
BINARY と VARBINARY タイプ	558
BLOBとTEXT タイプ	559
ENUM タイプ	560
SET タイプ	562
データタイプが必要とする記憶容量	563
カラムに適したタイプの選択	566
その他のデータベースエンジンのデータタイプの利用	566

MySQLは次のようないくつかのカテゴリのデータタイプをサポートします。数値タイプ、データと時刻タイプ、そして文字列(文字)タイプです。この章ではまず最初にこれらのデータタイプの概要を紹介します。そして次にそれぞれのカテゴリタイプの特性についてのより詳しい説明と、データタイプに必要な記憶容量に関する条件の要約を紹介します。最初の概要はあえて簡単な物になっています。値の指定ができる許容フォーマットのような特定のデータタイプに関する追加情報に関しては、この章の後半で紹介しているより詳しい説明を参照してください。

MySQLは、空間データの拡張子もサポートします。16章Spatial Extensionsにこれらのデータタイプの情報があります。

いくつかのデータタイプの紹介の中で、これらの規則が使用されています。

- **M** は整数タイプの最大ディスプレイ幅を示しています。浮動小数点と固定小数点タイプに関しては、**M** が格納可能な桁数の合計です。文字列タイプは **M** が最大長さです。**M** の最大許容値はデータタイプによって違います。
- **D** は浮動小数点と固定小数点タイプに適用され、小数点以下の桁数を表します。最大値は30ですが、**M**-2以上であってははいけません。
- 角かっこ (**[** と **]**) はタイプ定義のオプションを表します。

データタイプ概要

数値タイプの概要

数値データタイプの要約が次に紹介されます。追加情報については [数値タイプ](#) を参照してください。必要とする記憶容量は [データタイプが必要とする記憶容量](#) に紹介されています。

M は整数タイプの最大ディスプレイ幅を示しています。最大法定ディスプレイ幅は255です。[数値タイプ](#)で説明されているように、ディスプレイ幅はそのタイプの許容値幅とは関係ありません。浮動小数点と固定小数点タイプに関しては、**M** が格納可能な桁数の合計です。

数値コラムに対して **ZEROFILL** を指定すると、MySQLは自動的にそのコラムに **UNSIGNED** 属性を追加します。

SERIAL は **BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE** の別名です。

整数カラム定義の中の **SERIAL DEFAULT VALUE** は **NOT NULL AUTO_INCREMENT UNIQUE** の別名です。

警告: 1 つが **UNSIGNED** タイプの時に整数値間で減算を行うと、**NO_UNSIGNED_SUBTRACTION** SQLモードが有効でない限り、その結果から符号がなくなります。 **キャスト関数**と**演算子**を参照してください。

- **BIT[(M)]**

ビットフィールドタイプ**M** は1から64の、各値のビット数を表しています。**M** が削除された場合、デフォルトは1です。

- **TINYINT[(M)] [UNSIGNED] [ZEROFILL]**

大変小さい整数符号が付く範囲は **-128** から **127** です。符号が付かない範囲は **0** から **255** です。

- **BOOL**、**BOOLEAN**

これらのタイプは **TINYINT(1)** の同義語です。ゼロの値は誤りであるとみなされます。ゼロ以外の値は正確だとみなされます。

```
mysql> SELECT IF(0, 'true', 'false');
+-----+
| IF(0, 'true', 'false') |
+-----+
| false                  |
+-----+

mysql> SELECT IF(1, 'true', 'false');
+-----+
| IF(1, 'true', 'false') |
+-----+
| true                   |
+-----+

mysql> SELECT IF(2, 'true', 'false');
+-----+
| IF(2, 'true', 'false') |
+-----+
| true                   |
+-----+
```

しかしここに示されているように、**TRUE** 値と **FALSE** はそれぞれが **1** と **0** の単なる別名です。

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+-----+
| IF(0 = FALSE, 'true', 'false') |
+-----+
| true                           |
+-----+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-----+
| IF(1 = TRUE, 'true', 'false') |
+-----+
| true                           |
+-----+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-----+
| IF(2 = TRUE, 'true', 'false') |
+-----+
| false                          |
+-----+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+-----+
| IF(2 = FALSE, 'true', 'false') |
+-----+
| false                          |
+-----+
```

最後の二つのステートメントは、2 は、1 と 0 と等しくないで、表示される結果を表しています。

今後リリースされるMySQLの中で、標準SQLに基づき、ブーリアンタイプの扱いについて完全にカバーしていく予定です。

- [SMALLINT\[\(M\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

小さい整数符号が付く範囲は -32768 から 32767 です。符号が付かない範囲は 0 から 65535 です。

- [MEDIUMINT\[\(M\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

中間サイズの整数符号が付く範囲は -8388608 から 8388607 です。符号が付かない範囲は 0 から 16777215 です。

- [INT\[\(M\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

普通サイズの整数符号が付く範囲は -2147483648 から 2147483647 です。符号が付かない範囲は 0 から 4294967295 です。

- [INTEGER\[\(M\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

このタイプは INT の同義語です。

- [BIGINT\[\(M\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

大きい整数符号が付く範囲は -9223372036854775808 から 9223372036854775807 です。符号が付かない範囲は 0 から 18446744073709551615 です。

BIGINT カラムに関して注意すべき事

- 全ての演算は符号付の **BIGINT** か **DOUBLE** 値を利用しているので、ビット関数を使わない限り 9223372036854775807 (63ビット) 以上の大きい符号無し整数は利用してはいけません！もしそれをしてしまうと、**BIGINT** 値から **DOUBLE** に変換する時、丸め誤差の為に、結果の最後のいくつかの桁に誤差が出るかもしれません。

MySQLは、次のような時に **BIGINT** を扱う事ができます。

- **BIGINT** カラムに符号無し of 大きい値を格納するのに整数を使用する時
- `col_name` が **BIGINT** カラムを参照する、`MIN(col_name)` や `MAX(col_name)` の中
- 両方の演算数が整数の場合に、(+、-、*、等) 演算子を利用する時
- 文字列を利用する事で、正確な整数を **BIGINT** カラムに格納できます。この場合MySQLは、中間倍精度表現を含まない、文字列から数値への変換を行います。
- 両方の演算数が整数の場合、-、+、そして * 演算子は、**BIGINT** 演算を利用します。これは、もし二つの大きい整数を掛け合わせた場合、(または整数を戻す関数からの結果)、その結果が 9223372036854775807 以上の時には、予期しない結果になるという事を意味します。
- [FLOAT\[\(M,D\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

小さい(単精度) 浮動小数点数許容値は -3.402823466E+38 から -1.175494351E-38、0、そして 1.175494351E-38 から 3.402823466E+38 です。これらは、IEEEスタンダードに基づいた理論的な限界です。利用するハードウェアやOSによっては、実際の範囲は少し小さくなるかも知れません。

M は桁数の合計で、**D** は小数点以下の桁数の合計です。もし **M** と **D** が削除された場合、値はハードウェアに許容された限界まで格納されます。単精度小数点数は大体小数第7位まで正確です。

UNSIGNED が指定されている場合、負数は許可されません。

MySQLでは全ての計算が倍精度で行われているので、**FLOAT** を利用すると、予想外の問題が起きます。[Solving Problems with No Matching Rows](#)を参照してください。

- [DOUBLE\[\(M, D\)\] \[UNSIGNED\] \[ZEROFILL\]](#)

普通サイズ(倍精度)浮動小数点数許容値は `-1.7976931348623157E+308` から `-2.2250738585072014E-308`、`0`、そして `2.2250738585072014E-308` から `1.7976931348623157E+308` です。これらは、IEEEスタンダードに基づいた理論的な限界です。利用するハードウェアやOSによっては、実際の範囲は少し小さくなるかも知れません。

`M` は桁数の合計で、`D` は小数点以下の桁数の合計です。もし `M` と `D` が削除された場合、値はハードウェアに許容された限界まで格納されます。倍精度小数点数は大体小数第15位まで正確です。

`UNSIGNED` が指定されている場合、負数は許可されません。

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`、`REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

これらのタイプは `DOUBLE` の同義語です。例外 `:REAL_AS_FLOAT` SQLモードが無効の時は、`DOUBLE` ではなく `REAL` が `FLOAT` の同義語になります。

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

浮動小数点数です。`p` は精度をビットで表現しますが、MySQLは結果となるデータタイプに対して、`FLOAT` が `DOUBLE` のどちらを利用するかを決める為だけにこの値を利用します。`p` が0から24の時、そのデータタイプは `M` や `D` 値が無い `FLOAT` になります。`p` が25から53の時、そのデータタイプは `M` や `D` 値が無い `DOUBLE` になります。結果となるカラムの範囲は、このセクションの最初の方で説明されているように、単精度 `FLOAT` が倍精度 `DOUBLE` データタイプの物と同じです。

`FLOAT(p)` 構文はODBC互換性に添付されている。

- `DECIMAL[(M,D)] [UNSIGNED] [ZEROFILL]`

ひとかたまりの「精密」固定小数点。`M` は桁数の合計で、(精度) `D` は小数点以下の桁数の合計です。(縮尺)小数点と(負数に対する) `'-` 記号は `M` の中ではカウントされません。`D` が0の時は、小数点や端数部はありません。`DECIMAL` の最高桁数 (`M`) は65です。サポートされる最高桁数 (`D`)は30です。`D` が削除された時のデフォルトは0です。`M` が削除された時のデフォルトは10です。

`UNSIGNED` が指定されている場合、負数は許可されません。

`DECIMAL` カラムを利用した全ての基本的な計算 (`+`, `-`, `*`, `/`) は、65桁の精度で行われます。

- `DEC[(M,D)] [UNSIGNED] [ZEROFILL]`、`NUMERIC[(M,D)] [UNSIGNED] [ZEROFILL]`、`FIXED[(M,D)] [UNSIGNED] [ZEROFILL]`

これらのタイプは `DECIMAL` の同義語です。`FIXED` 同義語は他のデータベースと互換性があります。

データと時刻タイプの概要

時間データタイプの要約が次に紹介されます。追加情報については [日付と時刻タイプ](#) を参照してください。必要とする記憶容量は [データタイプが必要とする記憶容量](#) に紹介されています。

`DATETIME` と `DATE` 範囲の説明では、「サポートする」というのは、以前の値が有効であったとしても、その保障は無いという意味になります。

`SUM()` と `AVG()` 総計関数は、一時的な値とは機能しません。(それらは値を数字に変換するので、最初の数字ではない文字から後ろの部分がなくなってしまう。)この問題を防ぐ為には、数字単位に変換し総計作業を行い、そしてもう一度一時的な値に変換し直せば良いです。例：

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

- `DATE`

日付です。サポートされている範囲は `'1000-01-01'` から `'9999-12-31'` です。MySQL は、`DATE` 値を `'YYYY-MM-DD'` フォーマットで表示しますが、文字列と数字のどちらで `DATE` カラムに値を指示してもよいです。

- `DATETIME`

日付と時刻の組み合わせです。サポートされている範囲は '1000-01-01 00:00:00' から '9999-12-31 23:59:59' です。MySQL は、**DATETIME** 値を 'YYYY-MM-DD HH:MM:SS' フォーマットで表示しますが、文字列と数字のどちらで **DATETIME** カラムに値を指示してもよいです。

- **TIMESTAMP**

タイムスタンプです。範囲は '1970-01-01 00:00:01' UTC から 2037 年の途中までです。**TIMESTAMP** 値は ('1970-01-01 00:00:00' UTC)からの秒数として格納されます。**TIMESTAMP** は、'1970-01-01 00:00:00' 値を表す事はできません。なぜならば、これはその時から0秒である事に相当し、0という値は '0000-00-00 00:00:00' つまり、「ゼロ」 **TIMESTAMP** 値を表すのに用いられるからです。

TIMESTAMP カラムは **INSERT** または **UPDATE** 操作の日付と時刻を記録するのに役立ちます。自分で値を指定しない限り、テーブルの **TIMESTAMP** カラムはデフォルトで一番最近の操作の日付と時刻に自動的にセットされます。**NULL** 値を指定する事で、現在の日付と時刻を **TIMESTAMP** カラムに設定する事もできます。自動初期設定と更新の特徴については **TIMESTAMP MySQL 4.1での性質** の中で説明されています。

ディスプレイ幅が 19 文字に固定されている 'YYYY-MM-DD HH:MM:SS' フォーマットの中では、**TIMESTAMP** 値は文字列として戻されます。数字の値を得る為には +0 をタイムスタンプカラムに加える必要があります。

注:MySQL 4.1以前で使用されていた **TIMESTAMP** フォーマットはMySQL 5.1 の中ではサポートされていません。古いフォーマットに関する情報については、MySQL 3.23, 4.0, 4.1 リファレンスマニュアル を参照してください。

- **TIME**

時刻です。範囲は '-838:59:59' から '838:59:59' です。MySQL は、**TIME** 値を 'HH:MM:SS' フォーマットで表示しますが、文字列と数字のどちらで **TIME** カラムに値を指示してもよいです。

- **YEAR[(2|4)]**

2 桁、または 4 桁のフォーマットでの年です。デフォルトは 4 桁のフォーマットです。4 桁のフォーマットでは、許容値は 1901 から 2155、そして 0000 です。2 桁のフォーマットでは、許容値は1970年から2069年を表す、70 から 69 です。MySQLは **YEAR** の値を **YYYY** フォーマットで表示しますが、**YEAR** カラムには文字列と数字のどちらを使って値を指定する事もできます。

文字列タイプの概要

文字列データタイプの要約が次に紹介されています。追加情報については **文字列タイプ** を参照してください。必要とする記憶容量は **データタイプが必要とする記憶容量** に紹介されています。

MySQLは時々、文字列カラムを **CREATE TABLE** や **ALTER TABLE** ステートメントで与えられているタイプとは違う物に変更する事があります。**サイレント カラム仕様変更**を参照してください。

MySQL 4.1以降の中には、MySQL4.1以前のバージョンには無かった文字列データタイプの特徴が含まれます。

- MySQLは、文字単位の中の文字カラム定義の長さ仕様を説明します。(MySQL 4.1以前は、カラム長さはバイトで解釈されていました。)これは、**CHAR**、**VARCHAR**、そして **TEXT** タイプに適用されます。
- 多くの文字列データタイプのカラム定義に、文字セットやカラム照合を指定する属性を含む事ができます。これらの属性は **CHAR**、**VARCHAR**、**TEXT** タイプ、**ENUM**、そして **SET** データタイプに適用します。
- **CHARACTER SET** 属性は文字セットを指定し、そして **COLLATE** 属性は文字セットの照合を指定します。例:

```
CREATE TABLE t
(
  c1 VARCHAR(20) CHARACTER SET utf8,
  c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
);
```

このテーブル定義は、キャラクタセットのデフォルト照合を使った **utf8** のキャラクタセットを持つ **c1** という名前のカラムと、**latin1** のキャラクタセットと大文字と小文字を区別する照合を持つ **c2** という名前のカラムを作成します。

CHARSET は **CHARACTER SET** の同義語です。

- **ASCII** 属性は **CHARACTER SET latin1** の省略表現です。
- **UNICODE** 属性は **CHARACTER SET ucs2** の省略表現です。
- **BINARY** 属性は、カラム文字セットのバイナリ照合を指定する省略表現です。この場合、ソートと比較は数値文字値に基づきます。(MySQL 4.1以前のバージョンでは、**BINARY** はカラムにバイナリ文字列を格納させ、ソートと比較は数値バイト値に基づいていました。これは1バイト文字セットに文字値を使用するのと同じですが、複数バイト文字セットに使用するのとは違います。)
- 文字カラムのソートと比較は、カラムに割り当てられた文字セットに基づいています。(MySQL 4.1バージョン以前は、ソートと比較はサーバー文字セットの照合に基づいていました。) **CHAR**、**VARCHAR**、**TEXT**、**ENUM**、そして **SET** データタイプに対して、語彙の順番ではなく基礎となる文字コード値を利用する為に、バイナリ照合を持つカラムか、ソートと比較を行う **BINARY** 属性を宣言する事ができます。

9章キャラクタセットサポート MySQLの中での文字セットの使用についての追加情報を紹介しています。

- **[NATIONAL] CHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]**

格納時に必ずスペースを使って指定された長さに詰められる固定長文字列。**M** はカラム長さを表します。**M** の範囲は、0から255文字です。

注:**CHAR** 値が検索された時、後続スペースは削除されます。

CHAR の長さを255以上に設定しようとすると、**CREATE TABLE** か **ALTER TABLE** ステートメントが実行されたテーブル内でエラーが発生し、その作業は失敗します。

```
mysql> CREATE TABLE c1 (col1 INT, col2 CHAR(500));
ERROR 1074 (42000): Column length too big for column 'col' (max = 255);
use BLOB or TEXT instead
mysql> SHOW CREATE TABLE c1;
ERROR 1146 (42S02): Table 'test.c1' doesn't exist
```

CHAR は **CHARACTER** の省略表現です。**NATIONAL CHAR** (またはそれと同等である **NCHAR**) は、**CHAR** カラムが定義済文字セットを使用しなければならないという事を定義する為の標準のSQLの方法です。MySQL 4.1以降のバージョンでは、この定義済文字セットとして **utf8** を利用します。 **各国キャラクタセット**。

CHAR BYTE データタイプは **BINARY** データタイプの別名です。これは互換性の特徴です。

MySQLで **CHAR(0)** タイプのカラムを作成する事ができます。これは主に、カラムの存在に頼っていても、その値は実際には使用しない古いアプリケーションに対応する必要がある時に便利な物です。**CHAR(0)** はまた、二つの値だけを取り込む事ができるカラムが必要な時にも大変便利です。**CHAR(0) NULL** として定義されたカラムは1ビットだけ使用し、**NULL** と " (空の文字列) 値だけを取り込む事ができます。

- **CHAR [CHARACTER SET charset_name] [COLLATE collation_name]**

このタイプは **CHAR(1)** の同義語です。

- **[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]**

可変長文字列です。**M** はカラムの最大長さを表します。**M** の範囲は0から65,535です。(**VARCHAR** の実際の最大長さは使用する最大行サイズと文字セットによって決まります。最大有効カラム長さは65,532バイトの行サイズによります。)

注:MySQL 5.1 は、標準SQL仕様に従うので、**VARCHAR** 値から後続スペースを削除しません。

VARCHAR は **CHARACTER VARYING** の省略表現です。

VARCHAR は 1バイト、または 2バイトの長さのプリフィックスに加え、データと共に格納されています。**VARCHAR** カラムが255以上の長さであればプリフィックスの長さは 2バイトです。

- **BINARY[(N)]**

BINARY タイプは **CHAR** タイプと似ていますが、非バイナリ文字の文字列ではなく、バイナリバイト文字列を格納します。

- **VARBINARY(M)**

VARBINARY タイプは **VARCHAR** タイプと似ていますが、非バイナリ文字の文字列ではなく、バイナリバイト文字列を格納します。

- **TINYBLOB**

最長255 ($2^8 - 1$) バイトの **BLOB** カラムです。

- **TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]**

最長255 ($2^8 - 1$) 文字の **TEXT** カラムです。

- **BLOB[(M)]**

最長65,535 ($2^{16} - 1$) バイトの **BLOB** カラムです。

長さ **M** を任意で利用する事もできます。もしこれを実行すると、MySQLは **M** バイトの長さの値を保持するのに十分な最小 **BLOB** を作成します。

- **TEXT[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]**

最長65,535 ($2^{16} - 1$) 文字の **TEXT** カラムです。

長さ **M** を任意で利用する事もできます。もしこれを実行すると、MySQLは **M** 文字の長さの値を保持するのに十分な最小 **TEXT** タイプを作成します。

- **MEDIUMBLOB**

最長16,777,215 ($2^{24} - 1$) バイトの **BLOB** カラムです。

- **MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]**

最長16,777,215 ($2^{24} - 1$) 文字の **TEXT** カラムです。

- **LOBLOB**

最長4,294,967,295、または4GB ($2^{32} - 1$) バイトの **BLOB** カラムです。**LOBLOB** カラムの有効な(許可されている)最長長さは、クライアント/サーバプロトコルと使用可能メモリの中に組み込まれている最大パケットサイズにより決まります。

- **LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]**

最長4,294,967,295、または4GB ($2^{32} - 1$) バイトの **TEXT** カラムです。**LONGTEXT** カラムの有効な(許可されている)最長長さは、クライアント/サーバプロトコルと使用可能メモリの中に組み込まれている最大パケットサイズにより決まります。

- **ENUM('value1', 'value2',...)[CHARACTER SET charset_name] [COLLATE collation_name]**

一覧表です。'value1'、'value2'、...、**NULL** または特別な"エラー値"のリストから選択された、1つの値しか持つ事ができない文字列オブジェクトです。**ENUM** カラムは最高65,535 の異なる値を持つ事ができます。**ENUM** 値は、内部的には整数として表されます。

- **SET('value1', 'value2',...)[CHARACTER SET charset_name] [COLLATE collation_name]**

設定です。それぞれが、'value1'、'value2'、... 値のリストから選択されなければならない、ゼロ、またはそれ以上の値を持つ事ができる文字列オブジェクトです。**SET** カラムは最高64メンバを持つ事ができます。**SET** 値は、内部的には整数として表されます。

データタイプデフォルト値

データタイプ仕様の中の **DEFAULT value** 条項はカラムのデフォルト値を表します。例外がひとつあります。デフォルト値は一定でなければいけませんので、それは関数や式にはなり得ません。これは例えば、日付カラムの値に **NOW()** や **CURRENT_DATE** のような関数の値をデフォルトとして設定する事はできないという意味です。例外と

して、**TIMESTAMP** カラムのデフォルトとして **CURRENT_TIMESTAMP** を指定する事ができます。**TIMESTAMP** [MySQL 4.1での性質](#)を参照してください。

BLOB と **TEXT** カラムはデフォルト値として指定する事ができません。

もしカラム定義が 明示的な **DEFAULT** 値を含まない場合、MySQLはデフォルト値を次のように規定します。

もし **NULL** を値として取る事ができるなら、そのカラムは明示的な **DEFAULT NULL** 条項で定義する事ができます。

もし **NULL** を値として取る事ができなければ、MySQLは明示的な **DEFAULT** 条項でカラムを定義できません。データの入力に関しては、もし **INSERT** か **REPLACE** ステートメントがカラムの値を含んでいなければ、MySQLはその時有効なSQLモードに従ってカラムを扱います。

- もしストリクトSQLモードが有効でなければ、MySQLはカラムデータタイプに暗黙のデフォルト値を設定します。
- もしストリクトモードが有効だと、トランザクションテーブルにエラーが起き、ステートメントがロールバックされます。非トランザクションテーブルではエラーが起きますが、もしこれが複数行ステートメントの2行目が後続の行に対してのエラーだとすると、その先行する行が挿入されるでしょう。

テーブル **t** が次のように定義されたと仮定してください。

```
CREATE TABLE t (i INT NOT NULL);
```

この場合、**i** は明示的デフォルトを持ちませんので、ストリクトモードで次の各ステートメントがエラーを発生させ、行の挿入は行われません。もしストリクトモードを使用しない場合、3つ目のステートメントだけがエラーを発生させます。暗黙のデフォルトが最初の2つのステートメントに挿入されますが、**DEFAULT(i)** が値を作り出す事ができない為に、3つ目のステートメントは失敗するのです。

```
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

詳しくは [SQL モード](#) を参照してください。

与えられたテーブルに対して、どのカラムが明示的な **DEFAULT** 条項を持つかを確かめる為に、**SHOW CREATE TABLE** ステートメントを利用する事ができます。

暗黙のデフォルトは次のように定義されます。

- **AUTO_INCREMENT** 属性で宣言された整数タイプ以外の数値タイプのデフォルトは **0** です。**AUTO_INCREMENT** カラムのデフォルト値は、その配列の中の次の値です。
- **TIMESTAMP** 以外の日付と時刻タイプのデフォルトには、「ゼロ」値が適切です。テーブルの最初の **TIMESTAMP** カラムのデフォルト値は現在の日付けと時刻です。[日付と時刻タイプ](#)を参照してください。
- **ENUM** ではない文字列タイプのデフォルト値は空の文字列です。**ENUM** のデフォルトは、最初の列挙値です。

数値タイプ

MySQLは標準SQLの全ての数値データタイプをサポートします。これらのタイプは、概数値データタイプ(**FLOAT**、**REAL**、**DOUBLE PRECISION**)だけでなく、真数値データタイプ(**INTEGER**、**SMALLINT**、**DECIMAL**、**NUMERIC**)を含みます。キーワード **INT** は **INTEGER** のシノニムで、キーワード **DEC** は **DECIMAL** のシノニムです。数値タイプが必要とする記憶容量に関しては、[データタイプが必要とする記憶容量](#)を参照してください。

BIT データタイプはビットフィールド値を格納します。これは、**MyISAM**、**MEMORY**、**InnoDB** テーブルに対してサポートされています。

SQL標準への拡張として、MySQLは **TINYINT**、**MEDIUMINT**、**BIGINT** などの整数タイプもサポートします。次のテーブルには、各整数タイプが必要とする容量と値の範囲が示されています。

タイプ	バイト	最小値	最大値
		(Signed/Unsigned)	(Signed/Unsigned)

TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

MySQLがサポートするその他の拡張として、各タイプの基本キーワードに続くカッコ内に整数データタイプの表示幅を指定するオプションがあります(例 `INT(4)`)。この表示幅オプションは、コラムに指定された幅よりも小さい幅の整数値を表示する際に左側をスペースで埋めるために使用されます。

この表示幅は、コラムに格納することができる値の範囲も、コラムに指定された幅を超える値の表示される桁数も制限しません。例えば、`SMALLINT(3)`として指定されたコラムは、通常の `-32768` から `32767` の `SMALLINT` 範囲を持ち、そして、3文字で許容された範囲外の値は3文字以上の文字を使って表示されます。

オプションの拡張属性 `ZEROFILL` が続いて指定された時は、デフォルトはスペースである埋め込み文字はゼロで置き換えられます。例えば、`INT(5) ZEROFILL` として宣言されたコラムには、4の値は `00004` として表示されます。もし整数コラムの中に表示幅よりも大きい値を格納すると、MySQLが複雑なJOINに対してテンポラリテーブルを生成した時に問題が発生するという事に注意してください。これはMySQLは、データは元のコラム幅に収まると仮定するからです。

注:`ZEROFILL` 属性はコラムが式や `UNION` クエリに含まれている時は無効になります。

全ての整数タイプは、任意の(標準ではない)拡張子である `UNSIGNED` を持つ事ができます。正数だけをコラムの中で許可し、大きい上位数値範囲が必要な時には符号無しの値を利用できます。例えば、`INT` コラムが `UNSIGNED` の時、コラム範囲のサイズは同じですが、その終点は `-2147483648` と `2147483647` から、`0` と `4294967295` までシフトします。

浮動小数点と固定小数点も `UNSIGNED` になり得ます。整数タイプと同じように、この拡張子は負の値がコラムの中に格納されるのを防ぎます。しかし整数タイプとは違い、コラム値の上限は同じままです。

数値コラムに対して `ZEROFILL` を指定すると、MySQLは自動的にそのコラムに `UNSIGNED` 属性を追加します。

浮動小数点タイプでは、MySQLは単精度値には4バイトを使用し、倍精度値には8バイト使用します。

`FLOAT` と `DOUBLE` データタイプは、近似数値データ値を表す為に利用されます。`FLOAT` には、SQL基準は、括弧に囲まれたキーワード `FLOAT` に続くビットの中で精度の任意仕様(指数の範囲ではない)を許容します。MySQLはまた、この任意精度仕様もサポートしますが、その精度値は格納サイズを決定する為だけに使用されます。0から23の精度は、4バイトの単精度 `FLOAT` コラムをもたらします。24から53の精度は、8バイトの倍精度 `DOUBLE` コラムをもたらします。

MySQLは標準ではない構文を許容します。`FLOAT(M,D)`、`REAL(M,D)` または `DOUBLE PRECISION(M,D)`。ここで、「(M, D)」が意味するのは、値は合計で M 桁まで格納でき、そのうちの D 桁は小数点以下であるという事です。例えば、`FLOAT(7,4)` として定義されたコラムは、表示された時には `-999.9999` の様に見えます。MySQLは、値を格納する時に丸めを行いますので、`FLOAT(7,4)` コラムに `999.00009` を挿入すると、おおよその結果は `999.0001` になります。

MySQLは `DOUBLE` を `DOUBLE PRECISION` (標準ではない拡張子)の同義語として扱います。MySQLはまた、`REAL_AS_FLOAT` SQLモードが有効でない限り `REAL` を `DOUBLE PRECISION` (標準ではない変動)の同義語として扱います。

最大ポータビリティの為に、おおよその数値データ値のコードを必要とする格納は、精度仕様や桁数の無い `FLOAT` が `DOUBLE PRECISION` を利用する必要があります。

DECIMAL と **NUMERIC** データタイプは、真数値データ値を格納するために利用されます。MySQLでは、**NUMERIC** は **DECIMAL** として実行されます。これらのタイプは、金銭データ等の正確な精度を必要とする物を格納するために利用されます。

MySQL 5.1 は、**DECIMAL** と **NUMERIC** 値をバイナリフォーマットに格納します。MySQL 5.0.3バージョン以前では、それらは文字列として格納されていました。[22章精密計算](#)を参照してください。

DECIMAL や **NUMERIC** カラムを宣言する時は、精度とスケールを指定する事ができます。(そして実際に通常は指定しています。) 次がその例です。

```
salary DECIMAL(5,2)
```

この例の中では、**5** は精度で、**2** がスケールです。精度は、その値に格納された有効な桁数を表し、スケールは小数点以下に格納できる桁数を表しています。もしスケールが0なら、**DECIMAL** と **NUMERIC** 値は小数点と、小数点以下の数字を持ちません。

標準SQLでは、**salary** カラムは5桁で、かつ小数点以下2桁の値を格納する必要があります。それ故この場合、**salary** カラムに格納できる値の範囲は、**-999.99** から **999.99** です。

標準SQLでは、構文 **DECIMAL(M)** は、**DECIMAL(M,0)** に相当します。同じように、構文 **DECIMAL** は、**DECIMAL(M,0)** に相当し、その **M** の値はインプリメンテーションが決定する事ができます。MySQL は、この両方の **DECIMAL** と **NUMERIC** 構文の改良形をサポートします。**M** のデフォルト値は10です。

DECIMAL や **NUMERIC** の最大桁数は65ですが、**DECIMAL** や **NUMERIC** カラムの実際の範囲はカラムの精度やスケールに制約される事があります。そのようなカラムが、指定スケールで許容されている小数点以下の桁数よりも多い桁数の値を指定した場合、その値はそのスケールに変換されます。(厳密には、OS特有の機能をするのですが、通常その結果は許容桁数の切捨てになります。)

BIT データタイプは、ビットフィールド値を格納するのに利用されます。**BIT(M)** のタイプは、**M** ビット値の格納を許容します。**M** の範囲は1から64までが可能です。

ビット値を指定するには、**b'value'** 表記を利用する事ができます。**value** はゼロと1で書かれたバイナリ値です。例えば、**b'111'** と **b'1000000'** は、それぞれ7と128を表しています。[ビットフィールド値](#)を参照してください。

M ビットよりも短い **BIT(M)** カラムに値を指定すると、その値の左側にゼロが埋め込まれます。例えば、**BIT(6)** カラムに **b'101'** の値を指定すると、実際には **b'000101'** を指定した事と同じ意味になるのです。

そのデータタイプの許容範囲外の値を数値カラムに格納しようとすると、MySQLの反応はその時有効なSQLモードによって決まります。例えば、もし制限モードが有効でない場合、MySQLは値をその範囲の適切な長さに短縮し、その結果出た値を代わりに格納します。しかし、もしモードが **TRADITIONAL** に設定されていると、SQLスタンダードに従いエラーが発生し、MySQLは範囲外の値を受け付けず、挿入は失敗します。

非ストリクトモードで整数カラムに範囲外の値が指定された時、MySQLはそのカラムデータタイプの範囲に相当する終点の値を格納します。**TINYINT** か **TINYINT UNSIGNED** カラムに256を格納すると、MySQLはそれぞれに127か255を格納します。浮動小数点や固定小数点カラムが、指定された(またはデフォルトの)精度とスケールの暗黙範囲を超えた値を割り当てると、MySQLはその範囲に対応する終点を表す値を格納します。

MySQLがストリクトモードで機能していない時の短縮によって行われた変換は、**ALTER TABLE**、**LOAD DATA INFILE**、**UPDATE**、そして複数行 **INSERT** ステートメントに対しては警告としてレポートされます。MySQLがストリクトモードで機能している時は、そのテーブルがトランザクションテーブルかそれ以外の物なのかによって、これらのステートメントは失敗となり、いくつかの、または全ての値の挿入も変更も行われません。詳細に関しては [SQL モード](#) を参照して下さい。

日付と時刻タイプ

時間的な値を表す日付と時刻タイプは、**DATETIME**、**DATE**、**TIMESTAMP**、**TIME**、そして **YEAR** です。MySQLが表す事のできない不正データを指示した時に利用される「ゼロ」値と同様に、各時刻タイプは有効値範囲を持ちます。**TIMESTAMP** タイプには、後に紹介されますが、特別な更新機能があります。時刻タイプが必要とする記憶容量に関しては、[データタイプが必要とする記憶容量](#)を参照してください。

MySQLは不正データを挿入すると警告がエラーを発生させます。SQLモードを適切な値に設定する事で、MySQLがサポートする日付のタイプを指定する事ができます。(詳しくは [SQL モード](#) をご確認ください。) **ALLOW_INVALID_DATES** SQLモードを利用する事によって、**'1999-11-31'** のような確実な日付をMySQLに

指定する事ができます。これは、ユーザーが将来の処理の為にデータベースの中で指定した「間違っているかもしれない」値を格納したい時に便利です。このモードの時MySQLは、月は0から12の範囲、日付は0から31の範囲でしか確認しません。MySQLが **DATE** か **DATETIME** カラムの中で日付、または月と日付がゼロであるデータの格納を許可するので、これらの範囲はゼロを含むと定義されています。これは、誕生日など、正確な日付が不明なデータを格納する必要があるアプリケーションに大変便利です。その場合は、'1999-00-00' や '1999-01-00' 等のようにデータを格納するだけでよいのです。このようなデータを格納する時は、**DATE_SUB()** や **DATE_ADD** など、正確な日付を必要とする物のように、正確な結果を期待する事はできません。(もし日付にゼロを利用したくないのであれば、**NO_ZERO_IN_DATE** SQLモードを利用する事ができます。)

MySQLではまた、'0000-00-00' を「ダミーの日付」(もし**NO_ZERO_DATE** SQLモードを利用していないのであれば)として格納する事ができます。これは、**NULL** 値を利用するよりも便利な事が多いです。(データとインデックスの容量も少量で済みます。)

日付と時刻タイプを利用する際の注意事項があります。

- MySQLは標準アウトプットフォーマットの中で入力された日付や時刻の値を検索しますが、提供された値に対して、いくつかのフォーマットを読み取ろうとします。(例えば、日付や時刻タイプに指定される、または比較される値を指定した時。)次のセクションで紹介されているフォーマットだけがサポートされています。正当な値を提供しなければいけません。もし他のフォーマットの値を使用すると、予期しない結果が出る可能性があります。
- 2桁の年を含む日付の値は、世紀が不明な為あいまいです。MySQLは2桁の年の値を次のルールに従って解釈します。
 - 70-99 の範囲の年の値は 1970-1999 に変換されます。
 - 00-69 の範囲の年の値は 2000-2069 に変換されます。
- MySQLが値をいくつかのフォーマットで読み取るとしても、日付は一般的に頻繁に使われる、月-日-年や、日-月-年の順番ではなく、(例えば '09-04-98'、'04-09-98')、年-月-日の順番(例えば '98-09-04')で入力する必要があります。
- MySQLは、その値が数字で利用されれば、日付と時刻タイプの値を数字に変換し、またその反対も行います。
- デフォルトでは、MySQLに日付や時刻タイプの範囲外の値や、そのタイプには不正なデータ(このセクションの始めで触れたように)が入力された時は、その値を「ゼロ」に変換します。その例外は、範囲外の **TIME** 値は **TIME** 範囲の適切な終点にクリップされるという事です。

次のテーブルに、それぞれのタイプの「ゼロ」値のフォーマットが表されています。**NO_ZERO_DATE** SQLモードが有効な場合は、これらの値を利用すると警告メッセージが表示される事を覚えておいて下さい。

データタイプ	「ゼロ」 値
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000

- 「ゼロ」値は特別ですが、テーブルに表されている値を利用して格納したり、正確に参照したりする事ができます。'0' や 0 の値を利用して行う事も可能です。こちらの方が書き込むよりも簡単です。
- MyODBCで利用される「ゼロ」の日付や時刻値はODBCでは扱う事ができないので、MyODBC 2.50.12以前のバージョンでは自動的に **NULL** に変換されます。

DATETIME、DATE、そして TIMESTAMP タイプ

DATETIME、**DATE**、そして **TIMESTAMP** タイプは関連しています。このセクションでは、それらがどのように似ているのか、そしてどのような点で異なっているのかなどの、特徴について説明しています。

DATETIME タイプは、日付と時刻の両方の情報を含む値が必要な時に利用します。MySQLは、**DATETIME** 値を 'YYYY-MM-DD HH:MM:SS' のフォーマットで検索、表示します。サポートされている範囲は '1000-01-01 00:00:00' から '9999-12-31 23:59:59' です。

DATE タイプは時刻の部分は無く、日付の値だけが必要な時に利用します。MySQLは、**DATE** 値を 'YYYY-MM-DD' のフォーマットで検索、表示します。サポートされている範囲は '1000-01-01' から '9999-12-31' です。

DATETIME と **DATE** 範囲の説明では、「サポートする」というのは、以前の値が有効であったとしても、その保障は無いという意味になります。

TIMESTAMP データタイプは、MySQLのバージョンと、そのサーバーが稼動している SQL モードによって様々な性質を持っています。これらの性質は、このセクションの後のほうで説明します。

DATETIME、**DATE**、そして **TIMESTAMP** 値を、フォーマットの共通セットを利用して、指定する事ができます。

- 'YYYY-MM-DD HH:MM:SS' が 'YY-MM-DD HH:MM:SS' フォーマットの文字列として。「柔軟な」構文が許可されています。句読点文字が、日付部分と時刻部分の区切り文字として利用される事があります。例えば、'98-12-31 11:30:45'、'98.12.31 11+30+45'、'98/12/31 11*30*45'、そして '98@12@31 11^30^45' は同等です。
- 'YYYY-MM-DD' が 'YY-MM-DD' フォーマットの文字列として。「柔軟な」構文がここでも許可されています。例えば、'98-12-31'、'98.12.31'、'98/12/31'、そして '98@12@31' は同等です。
- 文字列が日付として成り立つという条件で、'YYYYMMDDHHMMSS' が 'YYMMDDHHMMSS' フォーマットの区切り文字が無い文字列として。例えば、'19970523091528' と '970523091528' は '1997-05-23 09:15:28' を表しますが、'971122129015' は不正データで、(これは意味を成さない分の部分がある為) '0000-00-00 00:00:00' となります。
- 文字列が日付として成り立つという条件で、'YYYYMMDD' が 'YYMMDD' フォーマットの区切り文字が無い文字列として。例えば、'19970523' と '970523' は '1997-05-23' を表しますが、'971332' は不正データで、(これは意味を成さない月と日の部分がある為) '0000-00-00' となります。
- 文字列が日付として成り立つという条件で、YYYYMMDDHHMMSS が YYMMDDHHMMSS フォーマットの数字として。例えば、19830905132800 と 830905132800 は '1983-09-05 13:28:00' という意味になります。
- 文字列が日付として成り立つという条件で、YYYYMMDD が YYMMDD フォーマットの数字として。例えば、19830905 と 830905 は '1983-09-05' という意味になります。
- **NOW()** や **CURRENT_DATE** のような、**DATETIME**、**DATE**、または **TIMESTAMP** コンテキストで許容可能な値を戻す関数の結果。

不正な **DATETIME**、**DATE**、または **TIMESTAMP** 値は、適切なタイプ ('0000-00-00 00:00:00' か '0000-00-00') の「ゼロ」値に変換されます。

日にち部分の区切り文字を含む文字列として指定された値には、月か日にちの値に 10 以下の2桁の値を指定する必要はありません。'1979-6-9' は '1979-06-09' と同じです。同じように、時刻部分の区切り文字を含む文字列として指定された値には、時、分、または秒の値に 10 以下の2桁の値を指定する必要はありません。'1979-10-30 1:2:3' は '1979-10-30 01:02:03' と同じです。

数字として指定された値は、6、8、12、または14桁の長さである必要があります。もし数字が8、または14桁の長さなら、それはYYYYMMDD が YYYYMMDDHHMMSS フォーマットであり、年は最初の4桁で表されていると仮定されます。もしその数字が6、または12桁であれば、YYMMDD が YYMMDDHHMMSS フォーマットであり、年は最初の2桁で表されていると仮定されます。これらの長さではない数字は後ろがゼロで詰められ、これらの中の一最近い桁数と仮定して判断されます。

区切り文字が無い文字列として指定された値はそれ自体の長さのまま判断されます。もしその文字列が8か14文字なら、年は最初の4文字で表されていると判断されます。そうでなければ、年は最初の2文字で表されていると判断されます。文字列は、左から右に順番に、年、月、日、時、分、そして秒として、その文字列に存在する限りの情報が読み取られます。これは、6文字以下の文字列は利用してはいけないという事を意味します。例えば、もし1999年3月を表そうとして '9903' と指定すると、MySQLは日付の値に「ゼロ」を挿入します。年と月の値は 99 と 03 ですが、日付の部分が全く無いのでこのような事が起こります。ですので、この値は不当な値という事になります。しかし、欠落している月や日付の部分をゼロの値を使って明確に指定する事ができます。例えば、'1999-03-00' という値を挿入する為に、'990300' を利用する事ができます。

1つの日付タイプの値を異なる日付タイプのオブジェクトに割り当てる事がある程度可能です。しかし、値の変更や情報の損失などが起こる可能性があります。

- もし **DATE** 値を **DATETIME** か **TIMESTAMP** オブジェクトに割り当てると、**DATE** 値は時刻の情報を持たないので、その結果の時刻の部分は '00:00:00' に設定されます。

- もし **DATETIME** か **TIMESTAMP** 値を **DATE** オブジェクトに割り当てると、**DATE** 値は時刻の情報を格納しないので、その結果の時刻の部分は 削除されます。
- **DATETIME**、**DATE**、そして **TIMESTAMP** 値は、全て同じフォーマットの組み合わせを利用して指定する事ができますが、それらのタイプは全て同じ範囲の値を持つわけではない事を覚えておいてください。例えば、**TIMESTAMP** 値は 1970 以前や、2037 以降にはなり得ないという事です。これは、'1968-01-01' のような日付は、**DATETIME** や **DATE** 値としては有効ですが、**TIMESTAMP** 値としては無効で、0 に変換されるという意味になります。

日付値を指定する時には、特定の落とし穴に気をつけてください。

- 文字列として指定された値に許容される柔軟なフォーマットはまぎらわしい事があります。例えば、'10:11:12' のような値は ':' が有る為に時刻値のように見えます。しかし、もし区切り文字が日付のコンテキストで利用されると、'2010-11-12' のように年として解釈されます。'10:45:15' 値は、'45' が正しい月を表す値ではないので、'0000-00-00' に変換されます。
- サーバは、それぞれが1から12、または1から31である事はもちろん、月と日の値が正しい値であることを要求します。ストリクトモードが無効の時は、'2004-04-31' のような無効な日付は '0000-00-00' に変換され、警告メッセージが表示されます。ストリクトモードが有効な時は、無効な日付はエラーを発生させます。そのような日付を許容するには、**ALLOW_INVALID_DATES** を有効にしてください。詳細については、**SQL モード** をご参照ください。
- 2桁の年を含む日付の値は、世紀が不明な為あいまいです。MySQLは2桁の年の値を次のルールに従って解釈します。
 - 00-69 の範囲の年の値は 2000-2069 に変換されます。
 - 70-99 の範囲の年の値は 1970-1999 に変換されます。

TIMESTAMP MySQL 4.1での性質

注:MySQLの古いバージョン (4.1以前)の **TIMESTAMP** データの性質は、このセクションで紹介されている物とは様々な面で明らかに違いました。古い **TIMESTAMP** データをMySQL 5.1 で利用する為に変換するには、その詳細についてMySQL 3.23, 4.0, 4.1 リファレンスマニュアル を必ず参照してください。

TIMESTAMP カラムは **DATETIME** カラムと同じフォーマットで表示されます。言い換えると、表示幅は19文字に決まられていて、フォーマットは **YYYY-MM-DD HH:MM:SS** となります。

MySQLサーバは **MAXDB** SQLモードが有効な時も実行する事ができます。このモードが有効な状態でサーバが実行された時、**TIMESTAMP** は **DATETIME** と同一です。これは、もしテーブルが作成された時にこのモードが有効だと、**TIMESTAMP** カラムは **DATETIME** カラムとして作成される、という意味になります。その結果、そのようなカラムは **DATETIME** 表示フォーマットを利用し、同じ範囲の値を持ち、自動初期化機能や、現在の日付と時刻に自動的にアップデートする機能はないという事になります。

MAXDB モードを有効にするには、起動の際に、**--sql-mode=MAXDB** サーバオプションを利用するか、ランタイム時にグローバル **sql_mode** 変数を設定して、サーバSQLのモードを **MAXDB** に設定してください。

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

クライアントは接続の為に、次のようにサーバを **MAXDB** モードで起動させる事ができます。

```
mysql> SET SESSION sql_mode=MAXDB;
```

次に紹介されている情報は、**MAXDB** モードが有効な状態で作成されなかったテーブルだけの **TIMESTAMP** カラムに適合するという事を覚えておいて下さい。なぜならば、そのようなカラムは **DATETIME** カラムとして作成されるからです。

MySQLは、日付か月のカラムにゼロを含むタイムスタンプ値や、有効でない日付値は許容しません。このルールの唯一の例外は、'0000-00-00 00:00:00' の特別値です。

いつ **TIMESTAMP** の自動初期化とアップデートが起こるのか、そしてどのカラムがそれらを行うべきなのかを決めるのに、相当な柔軟性があります。

- テーブル内の1つの **TIMESTAMP** カラムに対して、現在のタイムスタンプをデフォルト値と自動更新値として指定する事ができます。現在のタイムスタンプを、カラムを初期化するデフォルト値にする事、または自動更新のデ

フォルト値にする事、またはその両方にする事が可能です。現在のタイムスタンプを、1つのカラムを初期化するデフォルト値にし、別のカラムの自動更新のデフォルト値にする事は不可能です。

- どのTIMESTAMP カラムが現在の日付と時刻を自動的に初期化したり更新したりするのか指定する事ができます。これは、最初の TIMESTAMP カラムである必要はありません。

次のルールが TIMESTAMP カラムの初期化と更新を管理しています。

- もし DEFAULT 値がテーブルの最初の TIMESTAMP カラムに指定されたら、それは無視されません。CURRENT_TIMESTAMP、または一定の日付と時刻値がデフォルトになり得ます。
- 最初の TIMESTAMP カラムにとって、DEFAULT NULL は DEFAULT CURRENT_TIMESTAMP と同じです。それ以外の全ての TIMESTAMP カラムにとっては、DEFAULT NULL は DEFAULT 0 として扱われます。
- テーブル内の全ての1つの TIMESTAMP カラムは、現在のタイムスタンプに初期化された物か、自動的に更新された物として利用されます。
- CREATE TABLE ステートメントの中では、最初の TIMESTAMP カラムは次の方法のどれかで宣言する事ができます。
 - DEFAULT CURRENT_TIMESTAMP と ON UPDATE CURRENT_TIMESTAMP 条項の両方で、カラムはそのデフォルトに現在のタイムスタンプを持ち、それは自動的に更新されます。
 - DEFAULT と ON UPDATE 条項のどちらも、DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP とは同じではありません。
 - DEFAULT CURRENT_TIMESTAMP 条項と、ON UPDATE ではない条項で、カラムはそのデフォルトに現在のタイムスタンプを持ちますが、それは自動的に更新されません。
 - DEFAULT 条項が無い、ON UPDATE CURRENT_TIMESTAMP がある条項では、カラムのデフォルトは0で、それは自動的に更新されます。
 - 一定の DEFAULT 値の時は、カラムは一定のデフォルトを持ちます。もしカラムが ON UPDATE CURRENT_TIMESTAMP 条項を持っていればそれは自動的に更新されますが、そうでない時は更新されません。

言い換えると、現在のタイムスタンプを初期値と自動更新値の両方、またはそのどちらかに利用する事ができる、または、両方とも利用しない事も可能です。(例えば、自動初期化されたカラムを持たずに自動更新を可能にする為に ON UPDATE を指定する事が可能です。)

- CURRENT_TIMESTAMP またはその同義語(CURRENT_TIMESTAMP())、NOW()、LOCALTIME、LOCALTIME()、LOCALTIMESTAMP、または LOCALTIMESTAMP())は DEFAULT と ON UPDATE 条項の中で利用する事ができます。それらは全て「現在のタイムスタンプ」を意味します。(UTC_TIMESTAMP は許容されていません。現在のタイムゾーンが UTC でない限り、その値の範囲は TIMESTAMP カラムの値の範囲と並びません。)
- DEFAULT と ON UPDATE 属性の順番は関係ありません。もし DEFAULT と ON UPDATE の両方が TIMESTAMP カラムに指定されると、どちらかがもう片方に先行します。例えば、これらのステートメントは同等になります。

```
CREATE TABLE t (ts TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
DEFAULT CURRENT_TIMESTAMP);
```

- TIMESTAMP カラムに最初の物以外の自動デフォルトや更新を指定するには、一定の DEFAULT 値(例えば、DEFAULT 0 や DEFAULT '2003-01-01 00:00:00')を明確に指定する事によって、最初の TIMESTAMP カラムの自動初期化や更新動作を抑圧する必要があります。そして、それ以外の TIMESTAMP カラムに対しては、DEFAULT と ON UPDATE 条項の両方を削除しなければルールは最初の TIMESTAMP カラムと同じで、自動初期化や更新は行われません。

例:これらのステートメントは同等です。

```
CREATE TABLE t (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```



```
ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  DEFAULT CURRENT_TIMESTAMP);
```

MySQL サーバのタイムゾーンサポートで説明されているように、現在のタイムゾーンをそれぞれの接続ごとに設定する事ができます。TIMESTAMP 値は、現在のタイムゾーンから変換されて格納され、また検索された時に現在のタイムゾーンに再変換されながら、UTCに格納されます。タイムゾーン設定が一定である限り、格納した値と同じ値を復帰させる事ができます。もし TIMESTAMP 値を格納してから、タイムゾーンを変更して値を検索すると、検索された値は格納した値とは違ってきます。これは、同じタイムゾーンが両方向への変換に利用されなかった為に起こります。現在のタイムゾーンは、time_zone システム変数の値のように有効です。

カラムが NULL 値を含む事を許容する為に TIMESTAMP カラムの定義の中に NULL 属性を含める事ができます。例:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

もし NULL 属性が指定されていなければ、カラムを NULL に設定すると、現在のタイムスタンプに設定されます。NULL 値を許容する TIMESTAMP カラムは、下記の条件の時以外は現在のタイムスタンプを採用しないという事を覚えておいて下さい。

- そのデフォルト値は CURRENT_TIMESTAMP として定義されます。
- NOW() か CURRENT_TIMESTAMP がカラムに挿入されます。

言い換えると、NULL として定義された TIMESTAMP カラムは次のような定義を利用して作成された時だけ自動初期化します。

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

そうでなければ—ここに表されているように、もし TIMESTAMP カラムが DEFAULT TIMESTAMP を利用せずに NULL 値を許容するために定義されると ...

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT NULL);
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
```

... 現在の日付と時刻に対応した値を明確に挿入しなければいけません。例:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

TIME タイプ

MySQLは TIME 値を 'HH:MM:SS' フォーマットで検索、表示します。(または長時間値を表すには 'HHH:MM:SS' フォーマット)TIME 値の範囲は '-838:59:59' から '838:59:59' です。TIME タイプは、一日のうちの時刻を表す事ができるだけでなく(24時間以下)、経過時間や、二つの出来事の間の時間を表す事もできるので(24時間よりも長い、またはマイナスの事も有る)、時間を表す部分がとても長くなる事があります。

TIME 値は様々なフォーマットで指定する事ができます。

- 'D HH:MM:SS.fraction' フォーマットの文字列として次のうちの「柔軟な」構文の1つを利用する事もできます。'HH:MM:SS.fraction'、'HH:MM:SS'、'HH:MM'、'D HH:MM:SS'、'D HH:MM'、'D HH'、または 'SS'。ここでは、D は日を表し、0から34の値を持つ事ができます。MySQLは端数部分を格納しない事を覚えておいて下さい。
- 時刻を表す、'HHMMSS' フォーマットで区切り文字を利用しない文字列として例えば、'101112' は '10:11:12' として理解されますが、'109712' は不正データとなり(意味を成さない分の部分を持つ為)、'00:00:00' となります。
- 時刻を表す、'HHMMSS' フォーマットの数字として例えば、101112 は '10:11:12' として理解されます。以下のフォーマットもまた理解されます。SS、MMSS、HHMMSS、HHMMSS.fractionMySQLは端数部分を格納しない事を覚えておいて下さい。

- `CURRENT_TIME` のように `TIME` コンテキストの中で許容される値を返す関数の結果として

時刻部分の区切り文字を含む文字列として指定された `TIME` 値には、時、分、または秒の値に 10 以下の2桁の値を指定する必要はありません。`'8:3:2'` は `'08:03:02'` と同じです。

`TIME` カラムに省略された値を指定する際には注意してください。MySQLは、コロンが付いていない値は、その値の一番右の2桁が秒を表していると解釈します。(MySQLは `TIME` 値を、一日の内の時刻ではなく、経過時間として解釈します。)例えば、`'1112'` と `1112` は、`'11:12:00'` (11時12分過ぎ)を意味するように感じますが、MySQLはそれを `'00:11:12'` (11分12秒)と解釈します。同じように、`'12'` と `12` は `'00:00:12'` という意味になります。コロンが付いた `TIME` 値は反対に、必ず一日の内の時刻として扱われます。それは、`'11:12'` が `'11:12:00'` を表し、`'00:11:12'` では無いという事になります。

デフォルトでは、`TIME` 範囲外であるが正当である値は、その値の終点にクリップされます。例えば、`'-850:00:00'` と `'850:00:00'` は `'-838:59:59'` と `'838:59:59'` に変換されます。不正な `TIME` 値は `'00:00:00'` に変換されます。`'00:00:00'` 自体は正当な `TIME` 値なので、テーブルに格納された `'00:00:00'` の値から、元の値が `'00:00:00'` 値で指定されたのか、不正な値だったのかを知る方法は無いという事を覚えておいて下さい。

不正な `TIME` 値をもう少し厳しく扱うためには、エラーが発生するようにストリクトSQLモードを有効にしてください。[SQL モード](#)を参照してください。

YEAR タイプ

`YEAR` タイプは年を表すために利用される 1 バイトのタイプです。

MySQLは `YEAR` 値を `YYYY` フォーマットで検索、表示します。範囲は 1901 から 2155 です。

`TIME` 値は様々なフォーマットで指定する事ができます。

- `'1901'` から `'2155'` の範囲の 4 桁の文字列として
- 1901 から 2155 の範囲の 4 桁の数字として
- `'00'` から `'99'` の範囲の2桁の文字列として`'00'` から `'69'` と、`'70'` から `'99'` の範囲の値は、2000 から 2069 と、1970 から 1999 の範囲の `YEAR` 値に変換されます。
- 1 から 99 の範囲の2桁の数字として1 から 69 と、70 から 99 の範囲の値は、2001 から 2069 と、1970 から 1999 の範囲の `YEAR` 値に変換されます。ゼロを数字として直接指定して、2000 と解釈させる事ができないので、2 桁の数字の範囲は 2 桁の文字列の範囲と少しだけ違う事を覚えておいて下さい。`'0'` か `'00'` の文字列として指定すると、0000 として解釈されます。
- `NOW()` のように `YEAR` コンテキストの中で許容される値を返す関数の結果として

不正な `YEAR` 値は 0000 に変換されます。

2000年問題とデータタイプ

MySQLサーバ自体は2000年 (Y2K) コンプライアンスに対して何も問題はありません。

- MySQLサーバは、`TIMESTAMP` 値に対して、日付を 2037 年まで扱う Unix時間関数を利用しています。`DATE` と `DATETIME` 値には、9999 年までの日付が許容されています。
- 全てのMySQLの日付機能は一つのソースファイル、[sql/time.cc](#) で実行されており、2000年問題に対して安全にプログラムされています。
- MySQLでは、`YEAR` データタイプは 0 年と 1901 年から 2155 年を1バイトで格納する事ができ、2桁か4桁でそれらを表示します。全ての2桁の年は 1970 年から 2069 年の範囲だと判断されます。ですので、`YEAR` カラムに 01 を格納すると、MySQLサーバはそれを 2001 年として扱います。

MySQLサーバ自体がY2Kに対して安全だとしても、そうでないアプリケーションと共に利用すると問題が起きる可能性があります。例えば、多くの古いアプリケーションは、4桁の値よりも、曖昧である2桁の値を利用して年を格納したりコントロールしたりします。この問題は、「欠けている」値を表す為に 00 や 99 などの値を利用するアプリケーションによって作られる可能性があります。残念ながら、異なるアプリケーションは異なるプログラマによって

書かれており、それぞれが、異なる仕様や日付管理の関数を利用しているので、これらの問題を修正するのは難しいです。

それ故、MySQLサーバにY2Kの問題が無いとしても、曖昧でない値を入力する事は、アプリケーションの義務です。2桁の年を含む日付の値は、世紀が不明な為曖昧です。MySQLは内部的に4桁を利用して年を格納するので、そのような値は4桁の形に修正されなければいけません。

DATETIME、**DATE**、**TIMESTAMP**、そして **YEAR** タイプに対して、MySQLは次のルールを利用して曖昧な年の値の日付を修正します。

- 00-69 の範囲の年の値は 2000-2069 に変換されます。
- 70-99 の範囲の年の値は 1970-1999 に変換されます。

これらのルールは、データ値が何を表すかを妥当に推測する単なる経験則である事を覚えておいて下さい。もしMySQLが利用するルールが正しい値を導かなければ、4桁の年の値を含む、曖昧ではない入力が必要になります。

ORDER BY は、2桁の年を持つ **YEAR** 値を正しく分類します。

MIN() や **MAX()** 等のようないくつかの関数は、**YEAR** を数字に変換します。これは、これらの関数を利用すると、2桁の年の値は正確に機能しないという意味になります。この場合の修正は、**TIMESTAMP** や **YEAR** を4桁の年のフォーマットに変換するという事になります。

文字列タイプ

文字列タイプの種類は、**CHAR**、**VARCHAR**、**BINARY**、**VARBINARY**、**BLOB**、**TEXT**、**ENUM**、そして**SET** です。このセクションでは、これらのタイプがどのように機能するのか、クエリの中でどのように利用するのかを説明します。文字列タイプが必要とする記憶容量に関しては、[データタイプが必要とする記憶容量](#)を参照してください。

CHAR と VARCHAR タイプ

CHAR と **VARCHAR** タイプは似ていますが、格納、検索される方法が異なります。また、最大長さと、末尾のスペースが保持されるかどうかという点でも異なります。格納と検索の最中にレターケースの変換は行われません。

CHAR と **VARCHAR** タイプには、格納したい最大文字数を表す長さが宣言されています。例えば、**CHAR(30)** は最大30文字まで持つ事ができます。

CHAR カラムの長さは、テーブルを作成した時に宣言した長さに修正されます。長さは0から255までのどの長さにもなり得ます。**CHAR** 値が格納された時、指定された長さになるよう、右側が詰められます。**CHAR** 値が検索された時、後続スペースは削除されます。

VARCHAR カラム内の値は可変長の文字列です。長さは0から65,535の値で指定できます。(**VARCHAR** の最大有効長さは、最大行サイズと利用される文字サイズによって決まります。最大カラム長さは65,532バイトの行サイズによります。)

CHAR とは対照的に、**VARCHAR** 値は必要な文字数と、長さを記録する為の1バイト(255よりも長いカラムは2バイト)だけを利用して格納できます。

VARCHAR 値は格納される時に詰められません。スタンダードSQLに適合して、値が格納、検索される時に後続スペースは保持されます。

CHAR や **VARCHAR** カラムに、その最大長を超える値を指定すると、その値は切り捨てられます。切り捨てられた文字がスペースで無い場合には警告メッセージが表示されます。スペース以外の文字の切捨てに関しては、ストリクトSQLモードを利用する事で警告ではなくエラーを表示させ、その値の挿入を食い止める事ができます。[SQL モード](#)を参照してください。

次のテーブルは、**CHAR(4)** と **VARCHAR(4)** カラムに様々な文字列値を格納した結果を表示する事で、**CHAR** と **VARCHAR** の違いを表しています。

値	CHAR(4)	要求ストレージ	VARCHAR(4)	要求ストレージ
"	' '	4バイト	"	1バイト

'ab'	'ab '	4 バイト	'ab'	3バイト
'abcd'	'abcd'	4 バイト	'abcd'	5バイト
'abcdefgh'	'abcd'	4バイト	'abcd'	5バイト

テーブルの最終行に格納されたと表示されている値は、ストリクトモードを利用していない時だけ 適応される事を覚えておいて下さい。もしMySQLがストリクトモードで起動していると、コラム長を超える値は 格納されず エラーになります。

もし規定の値が **CHAR(4)** と **VARCHAR(4)** コラムに格納されると、**CHAR** コラムが検索される時に後続スペースが削除されるので、コラムから検索された値は必ずしも同じとは限りません。次の例はこれらの点を例示しています。

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab ', 'ab ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('( ', v, ')'), CONCAT('( ', c, ' ') FROM vc;
+-----+-----+
| CONCAT('( ', v, ' )' ) | CONCAT('( ', c, ' ') |
+-----+-----+
| (ab )           | (ab)           |
+-----+-----+
1 row in set (0.06 sec)
```

CHAR と **VARCHAR** コラムの中の値は、そのコラムに指定された 文字セットの照合に従って格納、比較されます。

全てのMySQL照合は **PADSPACE** タイプの物だと覚えておいてください。これは、MySQLの中の全ての **CHAR** と **VARCHAR** 値が後続スペースを無視して比較されるという事を意味します。例:

```
mysql> CREATE TABLE names (myname CHAR(10), yourname VARCHAR(10));
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO names VALUES ('Monty ', 'Monty ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Monty ', yourname = 'Monty ' FROM names;
+-----+-----+
| myname = 'Monty ' | yourname = 'Monty ' |
+-----+-----+
| 1 | 1 |
+-----+-----+
1 row in set (0.00 sec)
```

これは全てのMySQLバージョンに当てはまり、サーバのSQLモードに影響されないという事を覚えておいて下さい。

後続文字が剥ぎ取られたり、比較がそれらが無視する場合は、もしコラムが固有の値を要求するインデックスを持っていたら、後続文字数だけが異なるコラム値への挿入は重複キーエラーになります。例えば、もしテーブルが 'a' を含んでいると、'a' を格納しようとした時重複キーエラーになります。

BINARY と VARBINARY タイプ

BINARY と **VARBINARY** タイプは、非バイナリ文字列ではなく、バイナリ文字列を含んでいるところ以外で **CHAR** と **VARCHAR** に似ています。それは、それらが文字の文字列ではなく、バイトの文字列を含んでいるという事です。これは、それらが文字セットを持たず、ソートと比較は値の中の数値バイトに基づいているという意味です。

許容される最大長は、**CHAR** と **VARCHAR** と同様で、**BINARY** と **VARBINARY** の長さは文字ではなく長さのバイトであるという事以外、**BINARY** と **VARBINARY** と同じです。

BINARY と **VARBINARY** データタイプは **CHAR BINARY** と **VARCHAR BINARY** データタイプとは異なります。後者のタイプは、**BINARY** 属性によってコラムがバイナリ文字列コラムとして扱われる事はありません。代わりに、利用されるコラム文字セットのバイナリ照合を実行し、そしてそのコラム自体がバイナリバイト文字列ではなく非バイナリ文字の文字列を含みます。例えば、**CHAR(5) BINARY** は、デフォルト文字セットが **latin1** だと仮定して、**CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin** として扱われます。これは、文字セットや照合を持たない5バイトのバイナリ文字列 **BINARY(5)** とは異なります。

BINARY 値が格納される時、特定の長さまでパッド値で右側が詰められます。パッド値は **0x00** です。(ゼロバイト) 値は挿入時には右側が **0x00** で詰められ、選択時に後続バイトは削除されません。全てのバイトは、**ORDER BY** と **DISTINCT** 操作を含め、比較において重要です。**0x00** バイトとスペースは、**0x00** < スペースとなり、比較において異なります。

例:**BINARY**(3) カラムでは、挿入時 'a' は 'a\0' になります。'a\0' は挿入時 'a\0\0' になります。選択時、両方の値は変更されません。

VARBINARY では、挿入時に詰められる事も、選択時にバイトが削除される事もあります。全てのバイトは、**ORDER BY** と **DISTINCT** 操作を含め、比較において重要です。**0x00** バイトとスペースは、**0x00** < スペースとなり、比較において異なります。

後続パッドバイトが剥ぎ取られたり、比較がそれらを見捨てる場合は、もしカラムが固有の値を要求するインデックスを持っていたら、後続パッドバイトだけが異なるカラム値への挿入は重複キーエラーになります。例えば、もしテーブルが 'a' を含んでいると、'a\0' を格納しようとした時重複キーエラーになります。

もしバイナリデータの格納に **BINARY** データタイプを利用する予定で、検索した値を格納した値と同一にしたいなら、先行パッドと削除文字を注意深く検討する必要があります。次の例は、**BINARY** 値の **0x00** パッドがどのようにカラム値比較に影響するか、例を示しています。

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+-----+-----+-----+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+-----+-----+-----+
| 610000 | 0       | 1           |
+-----+-----+-----+
1 row in set (0.09 sec)
```

もし検索した値がパッドなしの指定したストレージと同じ値でなければいけないなら、**VARBINARY** か、**BLOB** データタイプの1つを代わりに利用するのが好ましいです。

BLOBとTEXT タイプ

BLOB は様々な大きさのデータを保持することができる大きいバイナリオブジェクトです。4つの **BLOB** タイプは、**TINYBLOB**、**BLOB**、**MEDIUMBLOB**、そして **LONGBLOB** です。これらは、保持することができる最大長さだけが異なっています。4つの **TEXT** タイプは、**TINYTEXT**、**TEXT**、**MEDIUMTEXT**、そして **LONGTEXT** です。これらは4つの **BLOB** タイプに対応し、最大長さとする記憶容量は同じです。[データタイプが必要とする記憶容量](#)を参照してください。**TEXT** や **BLOB** カラムのレターケースの変換は格納や検索時には行われません。

BLOB カラムはバイナリ文字列 (バイト文字列) として扱われます。**TEXT** カラムは非バイナリ文字列 (文字の文字列) として扱われます。**BLOB** カラムは文字セットを持たないので、ソートと比較は値の中の数値バイトに基づいています。**TEXT** カラムは文字セットを持つので、値は文字セットの照合に基づいてソート、比較されます。

もし **TEXT** カラムがインデックスされていたら、インデックス入力比較は最後にスペースが詰められます。これは、もしそのインデックスが固有の値を要求するなら、後続スペースだけが異なる値に対して重複キーエラーが発生するという事を意味します。例えば、もしテーブルが 'a' を含んでいると、'a' を格納しようとした時重複キーエラーになります。これは **BLOB** カラムには当てはまりません。

ストリクトモードで実行していない時に、**BLOB** や **TEXT** カラムに、そのデータタイプの最大長を超える値を指定すると、その値は切り捨てられます。切り捨てられた文字がスペースで無い場合には警告メッセージが表示されます。ストリクトモードを利用すると、警告と共に値が切り捨てられるのではなく、エラーを発生させて、その値を拒否させる事ができます。[SQL モード](#)を参照してください。

あらゆる点で、**BLOB** カラムを、好きな長さに設定できる **VARBINARY** カラムだと考える事ができます。同じように、**TEXT** カラムを **VARCHAR** カラムと考える事ができます。**BLOB** と **TEXT** は、次の点で **VARBINARY** と **VARCHAR** とは異なっています。

- **BLOB** と **TEXT** カラムのインデックスには、インデックスプリフィックス長を指定しなければいけません。**CHAR** と **VARCHAR** では、プリフィックス長は任意です。[カラムインデックス](#)を参照してください。

- BLOB と TEXT カラムは DEFAULT を持つ事ができません。

LONG と LONG VARCHAR は MEDIUMTEXT データタイプにマップします。これは互換性の特徴です。もし BINARY 属性を TEXT データタイプと一緒に利用するなら、そのカラムはカラム文字セットのバイナリ照合に指定されます。

MySQLコネクタ/ODBCは BLOB 値を LONGVARBINARY として、TEXT 値をLONGVARCHAR として定義します。

BLOB と TEXT 値は大変長くなり得るので、それらを利用する時にいくつかの制約が発生します。

- カラムの max_sort_length バイトだけがソートに利用されます。max_sort_length のデフォルト値は1024です。この値は、mysqldサーバーを開始する時に --max_sort_length=N オプションを利用して変更する事ができます。システム変数を参照してください。

max_sort_length の値をランタイムに増やす事で、ソートとグループの際により多くのバイトを有効にする事ができます。全てのクライアントがそのセッション max_sort_length 変数の値を変更する事ができます。

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
-> ORDER BY comment;
```

GROUP BY や ORDER BY を、max_sort_length バイトよりも多くのバイトを有効にしたい時の長い値を含む BLOB や TEXT カラム上で利用する別の方法は、カラム値を固定長オブジェクト変換するという方法です。これを行う標準的な方法は SUBSTRING() 関数を利用する方法です。例えば、次のステートメントによって comment カラムの2000バイトがソートの際に考慮されるようになります。

```
mysql> SELECT id, SUBSTRING(comment,1,2000) FROM t
-> ORDER BY SUBSTRING(comment,1,2000);
```

- BLOB や TEXT オブジェクトの最大サイズはそのタイプによって判断されますが、クライアントとサーバーの間で実際に送信できる最大値は、有効メモリの量とコミュニケーションバッファのサイズによって判断されます。max_allowed_packet 変数の値を変更する事でメッセージバッファサイズを変更する事ができますが、サーバとクライアントプログラムの両方に対してその作業を行う必要があります。例えば、mysql と mysqldump の両方がクライアント側の max_allowed_packet 値を変更する事を許可します。サーバパラメータのチューニング、mysql — MySQL コマンドライン ツール、mysqldump — データベースバックアッププログラムを参照して下さい。ソート中のバケットサイズとデータオブジェクトのサイズを必要とする記憶容量に基づいて比較したい場合には、データタイプが必要とする記憶容量を参照してください。

BLOB や TEXT 値はそれぞれ内部的に別々に割り当てられたオブジェクトによって表現されます。これは、テーブルが開かれた時にそれぞれのカラムに容量が一度割り当てられるという形の、その他全てのデータタイプとは異なります。

時には、メディアファイルのようなバイナリデータを BLOB や TEXT カラムに格納する事が望ましい場合もあるでしょう。MySQLの文字列操作関数がこのようなデータを利用するのに役に立つでしょう。文字列関数を参照してください。安全とその他の理由の為、これを行うには、アプリケーションユーザに FILE 特権の利用を許可するのではなく、アプリケーションコードを利用して行う方が望ましいです。詳細については、MySQLフォーラムで、様々な言語やプラットフォームで議論する事ができます。(http://forums.mysql.com/)

ENUM タイプ

ENUM は、テーブルを作成する際カラム仕様の中で明確に列挙された許容値リストから選択された値を持つ文字列オブジェクトです。

列挙値は引用された文字列直定数である必要があります。これは、式でも、文字列値を評価するものでもありません。これは、ユーザ変数を列挙値として採用するべきではないという事も意味します。

その値は、特定の条件下では()や NULL の空の文字列になる事もあります。

- もし ENUM に無効な値(許容値リストに存在しない文字列)を挿入すると、特別エラー値として空の文字列が代わりに挿入されます。この文字列は、ゼロの数値を持つという点で、「通常の」空の文字列とは区別することができます。この後でもう少し詳しく説明します。

もしストリクトSQLモードが有効なら、無効な ENUM 値を挿入しようとするエラーが発生します。

- もし **ENUM** カラムが **NULL** の許容を宣言すると、**NULL** 値はそのカラムにとって正当な値となり、デフォルト値は **NULL** になります。もし **ENUM** カラムが **NOT NULL** を宣言すると、許容値リストの最初の要素がそのデフォルト値となります。

それぞれの列挙値はインデックスを持ちます。

- カラム仕様の中の許容可能エレメントリストからの値は1から始まる番号がつけられています。
- 空の文字列エラーインデックス値は0です。これは、どの無効な **ENUM** 値に行が指定されたのかを見つける為に、次の **SELECT** ステートメントを利用する事ができるという事を意味します。

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- NULL** 値のインデックスは **NULL** です。
- 「インデックス」という言葉は、列挙値リストの中の位置だけを表しています。これは、テーブルインデックスとは全く関係がありません。

例えば、**ENUM('one', 'two', 'three')** として指定されたカラムはここに表されている値のどれでも持つ事ができます。それぞれの値のインデックスも表示されています。

値	インデックス
NULL	NULL
"	0
'one'	1
'two'	2
'three'	3

1つの列挙は最大65,535エレメントを持つ事ができます。

テーブルが作成された時に、テーブル定義の中の **ENUM** メンバー値から後続スペースが自動的に削除されます。

検索された時は、**ENUM** カラムに格納された値はカラム定義で使用されたレターケースで表示されます。**ENUM** カラムは文字セットと照合に指定できる事を覚えて置いてください。バイナリ、またはケースに敏感な照合には、カラムに値を指定する時レターケースが考慮されます。

もし **ENUM** 値を数値コンテキストで検索するなら、カラム値のインデックスは返されます。例えば、このようにして **ENUM** カラムから数値を検索する事ができます。

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

もし **ENUM** カラムに数字を格納すると、その数字は可能値のインデックスとして扱われ、格納された値はそのインデックスを持つ列挙番号となります。(しかしこれは全ての入力を文字列として扱う **LOAD DATA** とは機能しません。)もし数値が引用されると、列挙値リストの中に適合する文字列がなければ、そのままインデックスとして解釈されます。これらの理由により、**ENUM** カラムを数字のように見える列挙値で定義する事は、複雑になり得るのでお勧めできません。例えば、次のカラムは '0'、'1'、そして '2' の文字列値のある列挙番号を持ちますが、1、2、そして 3 の数値インデックス値は次のようになります。

```
numbers ENUM('0','1','2')
```

もし 2 を格納すると、それはインデックス値として解釈され、'1' となります。(インデックス2の値)もし '2' を格納すると、それは列挙値と適合するので '2' として格納されます。もし '3' を格納すると、どの列挙値とも適合しないのでインデックスとして扱われ、'2' となります。(インデックス3の値)

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
+-----+
| numbers |
+-----+
| 1       |
| 2       |
| 2       |
+-----+
```


ENUM 値は、カラム仕様にリストされた列挙番号の順番に従ってソートされます。(言い換えると、ENUM 値はそれらのインデックス番号によってソートされるという事になります。)例えば、'a' は ENUM('a', 'b') では 'b' の前にソートしますが、'b' は ENUM('b', 'a') では 'a' の前にソートします。空の文字列は、空ではない文字列の前にソートし、そして NULL 値はその他の全ての列挙値の前にソートします。予期しない結果を防ぐ為には、ENUM リストをアルファベット順に指定してください。カラムがインデックス番号ではなく、語彙的にソートされる為に、GROUP BY CAST(col AS CHAR) か GROUP BY CONCAT(col) を利用する事もできます。

ENUM カラムに有効な全ての値を究明したければ、SHOW COLUMNS FROM tbl_name LIKE enum_col を利用し、アウトプットの Type カラムの中の ENUM 定義を解析してください。

SET タイプ

SET はゼロ、またはそれ以上の値を持つことができる文字列オブジェクトであり、それらはそれぞれ、テーブルが作成された時に指定された許容値リストから選択する必要があります。複数セットメンバーによって成り立つ SET カラム値は、カンマで区切られたメンバーによって指定されます。(','この結果は、SET メンバー値自体はコンマを含むべきではないという事です。

例えば、SET('one', 'two') NOT NULL として指定されたカラムはここに表されている値のどれでも持つ事ができます。

```
"
'one'
'two'
'one,two'
```

SET は最高64の異なるメンバを持つ事ができます。

テーブルが作成された時に、テーブル定義の中の SET メンバー値から後続スペースが自動的に削除されます。

検索された時は、SET カラムに格納された値はカラム定義で使用されたレターケースで表示されます。SET カラムは文字セットと照合に指定できる事を覚えて置いてください。バイナリ、またはケースに敏感な照合には、カラムに値を指定する時レターケースが考慮されます。

MySQLは、最初のセットメンバに対応する格納値の低位ビットを利用して SET 値を数値で格納します。SET 値を数値コンテキストで検索すると、その値は、カラム値を構成するセットメンバに対応するビットセットを持ちます。例えば、このようにして SET カラムから数値を検索する事ができます。

```
mysql> SELECT set_col+0 FROM tbl_name;
```

もしメンバが SET カラムに格納されると、その数字のバイナリ表現に設定されているビットがカラム値のセットメンバを決定します。SET('a','b','c','d') として指定されたカラムには、メンバは次の少数とバイナリ値を持ちます。

SET メンバ	少数値	バイナリ値
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

もしこのカラムに、バイナリでは 1001 となる 9 を指定すると、最初と4番目の SET 値メンバである 'a' と 'd' が選択され、結果値は 'a,d' となります。

1つ以上の SET エレメントを含む値には、値を挿入する時のエレメントがどの順番でリストされるかは関係ありません。また、決められたエレメントがその値の中で何回リストされるかという事も関係ありません。値が後で検索される時には、テーブル作成時に指定された順番に従ってリストされたエレメントと一緒に、値の中のそれぞれのエレメントが一度表示されます。例えば、カラムが SET('a','b','c','d') として指定されたたと仮定します。

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

もし 'a,d', 'd,a', 'a,d,d', 'a,d,a', そして 'd,a,d' という値を挿入すると、次のようになります。

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
```


Records: 5 Duplicates: 0 Warnings: 0

すると、これらの値が検索された時、**'a,d'** と表示されます。

```
mysql> SELECT col FROM myset;
+-----+
| col |
+-----+
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
+-----+
5 rows in set (0.04 sec)
```

もしサポートされていない値に **SET** カラムを設定すると、その値は無視され警告が表示されます。

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+-----+-----+-----+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+-----+
| col |
+-----+
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
+-----+
6 rows in set (0.01 sec)
```

もしストリクトSQLモードが有効なら、無効な **SET** 値を挿入しようとするエラーが発生します。

SET 値は数値でソートされます。**NULL** 値は非 **NULL SET** 値の前にソートします。

通常は、**FIND_IN_SET()** 関数が **LIKE** オペレーターを利用して **SET** 値を検索します。

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

最初のステートメントは **set_col** が **value** セットメンバを含む行を見つけます。二つ目のステートメントも似ていますが、全く同じではありません。2つ目のステートメントは、他のセットメンバの部分列としても、**set_col** が **value** をどこかに含む行を見つけます。

次のステートメントもまた正当です。

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

これらのステートメントの最初の部分が最初のセットメンバを含む値を探します。二つ目の部分が正確に適合する値を探します。二つ目のタイプの比較に注意してください。**'val1、val2'** のセット値を比較すると、**'val2、val1'** を比較するよりも異なる結果が返されます。カラム定義の中でリストされているのと同じ順番で値を指定する必要があります。

SET カラムに有効な全ての値を究明したければ、**SHOW COLUMNS FROM tbl_name LIKE set_col** を利用し、アウトプットの **Type** カラムの中の **SET** 定義を解析してください。

データタイプが必要とする記憶容量

MySQLにサポートされているデータタイプが必要とする記憶容量がカテゴリごとにリストされています。

MyISAM テーブル内の行の最大サイズは65,534バイトです。**BLOB** と **TEXT** カラムはそれぞれ、このサイズに対してたった5から9バイトを占めています。

重要**NDBCluster** ストレージエンジンを利用しているテーブルには、必要とする記憶容量を計算する際考慮すべき4-byteアラインメントの要因があります。これは、全ての **NDB** データ格納が4バイトの倍数単位で行われるという意味になります。それ故、一テーブルの中で **NDB** 以外のストレージエンジンを利用している一カラム値は、格納に15バイトを利用し、**NDB** テーブルの中で16バイトを必要とします。この要求は、このセクションで紹介される他の全ての条件に当てはまります。例えば、**NDBCluster** テーブルの中で、**TINYINT**、**SMALLINT**、**MEDIUMINT**、そして **INTEGER (INT)**カラムタイプはそれぞれ1つのレコードにつき4バイトを必要とします。

さらに、クラスタテーブルが必要とする記憶容量を計算する時、**NDBCluster** ストレージエンジンを利用する全てのテーブルがプライマリキーを要求する事を覚えておく必要があります。もしプライマリキーがユーザによって定義されない時は、**NDB** によって「隠れ」プライマリキーが作成されます。この隠れプライマリキーは1つのテーブルレコードに付き31から35バイトを消費します。

クラスタメモリ要求を計算する時には、**MySQLForge** で有効な **ndb_size.pl** ユーティリティが便利です。このPerlスクリプトは現在のMySQL(非クラスタ)データベースに接続し、そのデータベースが **NDBCluster** ストレージエンジンを利用するとどれくらいの容量を必要とするかについてのレポートを作成します。

数値タイプが必要とする記憶容量

データタイプ	要求ストレージ
TINYINT	1バイト
SMALLINT	2バイト
MEDIUMINT	3バイト
INT, INTEGER	4バイト
BIGINT	8バイト
FLOAT(p)	4 bytes if $0 \leq p \leq 24$, 8 bytes if $25 \leq p \leq 53$
FLOAT	4バイト
DOUBLE [PRECISION], REAL	8 バイト
DECIMAL(M,D), NUMERIC(M,D)	変動; 後の説明を参照
BIT(M)	約 $(M+7)/8$ バイト

DECIMAL (と**NUMERIC**)カラムの値は、少数第9位 (10基準) の桁を4バイトにパックするバイナリフォーマットを利用して表現されます。各値の整数部と端数部の格納は別々に決定されます。9 桁の倍ごとに4バイト、「余りの」桁には4バイトの端数容量がそれぞれ必要です。余りの桁に必要なストレージ要求を以下のテーブルで紹介します。

余り桁数	バイト数
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

データと時刻タイプが必要とする記憶容量

データタイプ	記憶容量
DATE	3バイト
DATETIME	8バイト
TIMESTAMP	4バイト
TIME	3バイト
YEAR	1バイト

文字列タイプの記憶容量

データタイプ	記憶容量
CHAR(M)	M バイト、 $0 \leq M \leq 255$
VARCHAR(M)	L + 1 バイト、一方で $L \leq M$ そして $0 \leq M \leq 255$ (下のメモを参照) または L + 2 バイト、一方で $L \leq M$ そして $256 \leq M \leq 65535$ (下のメモを参照).
BINARY(M)	M バイト、 $0 \leq M \leq 255$
VARBINARY(M)	L + 1 バイト 一方で $L \leq M$ そして $0 \leq M \leq 255$ (下のメモを参照) または L + 2 バイト、 一方で $L \leq M$ そして $256 \leq M \leq 65535$ (下のメモを参照).
TINYBLOB, TINYTEXT	L+1 バイト、一方で $L < 2^8$
BLOB, TEXT	L+2 バイト 一方で $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	L+3 バイト、一方で $L < 2^{24}$
LONGBLOB, LONGTEXT	L+4 バイト、一方で $L < 2^{32}$
ENUM('value1','value2',...)	列挙値により1か2バイト(最高65,535値)
SET('value1','value2',...)	セットメンバの数により、1、2、3、4、または8バイト(最高64メンバ)

CHAR、VARCHAR、そして TEXT タイプでは、先行テーブルの中の L と M の値は文字数として解釈される必要があり、そしてカラム仕様の中のこれらのタイプの長さは文字数を表します。例えば、TINYTEXT 値を格納するには、L 文字に加え1バイトが必要です。

VARCHAR、VARBINARY、そして BLOB と TEXT タイプは可変長タイプです。それぞれが必要とする記憶容量はこれらの要因によって決まります。

- カラム値の実長さ
- カラムの可能最大長さ
- カラムに使用される文字セット

例えば、VARCHAR(10) カラムは最大長さ10の文字列を保持する事ができます。カラムが latin1 文字セットを利用すると仮定すると(一文字につき1バイト)、実際の記憶容量は文字列の長さ(L)と、その文字列の長さを記録する1バイトです。'abcd' 文字列では、L は4で、必要とする記憶容量は5バイトです。もし同じカラムが代わりに VARCHAR(500) として宣言されていたら、その文字列 'abcd' は $4 + 2 = 6$ バイトを必要とします。カラム長は255以上なので、プリフィックスには1バイトではなく2バイトが要求されます。

特定の CHAR、VARCHAR、または TEXT カラム値を格納するのに利用される バイト 数を計算するには、そのカラムに利用される文字セットを考慮に入れなければいけません。特に、utf8 ユニコード文字セットを利用する時には、全ての utf8 文字セットが同じバイト数を利用するわけではないという事を覚えておく必要があります。utf8 文字の異なるカテゴリに利用される格納についての概要は、Unicodeのサポートを参照してください。

注:VARCHAR や VARBINARY カラムの 有効 最大長は65,532です。

MySQL 5.1の NDBCLUSTER ストレージエンジンは可変幅カラムをサポートします。これは、そのような値が4バイトにそろっている場合以外は、MySQLクラスタテーブルの VARCHAR カラムが、他のストレージエンジンを利用した時と同じ容量を必要とするという意味になります。それ故、latin1 文字セットを利用する VARCHAR(50) カラムに

格納された 'abcd' 文字列は8バイトを必要とします。(MyISAM テーブルの中の同じカラム値に必要とされる6バイトではない)これは、VARCHAR(50) カラムが、格納された文字列の長さに関わらず、1つのレコードにつき52バイトを必要としていた、NDBCLUSTER の初期バージョンからの変更点を表しています。

BLOB と TEXT タイプは、そのタイプの最大長さにより、カラム値の長さを記録する為に1、2、3、またはバイトを必要とします。BLOBとTEXT タイプを参照してください。

TEXT と BLOB カラムは、TEXT カラム内のそれぞれの行が二つの部分で構成される、NDBクラスタストレージエンジンの中で異なる方法で実行されます。そのうちの1つは固定サイズ(256バイト)で、実際に元のテーブルに格納されます。それ以外の物は、隠れテーブルに格納された256バイト以上のデータで構成されます。この二つ目のテーブルの行は常に2,000バイトの長さです。もし サイズ ≤ 256 (サイズ が行のサイズを表している)時、TEXT カラムのサイズは256で、そうでない時のサイズは256 + サイズ + (2000 - (サイズ - 256) % 2000)です。

ENUM オブジェクトのサイズは異なる列挙値の数によって決定します。1バイトは、最大255の値を持つ列挙に使用されます。2バイトは、256から65,535の間の値を持つ列挙に使用されます。ENUM タイプを参照してください。

SET オブジェクトのサイズは異なるセットメンバの数によって決定します。もしセットサイズがN なら、オブジェクトは1、2、3、4、または8バイトに丸められた (N+7)/8 バイトをコピーします。SET は最高64メンバを持つ事ができます。SET タイプを参照してください。

カラムに適したタイプの選択

最適な格納の為に、毎回一番正確なタイプの利用を試みる必要があります。例えば、もし整数カラムが 1 から 99999 の範囲の値に利用されたら、MEDIUMINT UNSIGNED が最適タイプです。要求される値を全て表すタイプの中で、このタイプが使用する容量が一番少ないです。

DECIMAL カラムを利用した全ての基本的な計算 (+, -, *, /) は、65桁の精度で行われます。数値タイプの概要を参照してください。

もし精度がそれほど重要でなかったり、スピードが最優先事項でなければ、DOUBLE タイプでも十分でしょう。高精度の為に、BIGINT の中に格納されている固定小数点タイプにいつでも変換する事ができます。これで、64ビットの整数で全ての計算をし、その後必要に応じて結果を浮動小数点値に変換する事ができます。

その他のデータベースエンジンのデータタイプの利用

他のベンダーによってSQL推進の為に書かれたコードを促進する為に、次のテーブルに書かれているようにMySQLはデータタイプをマップします。これらのマッピングにより、他のデータベースシステムからテーブル定義をMySQLにインポートする事が容易になります。

他のベンダータイプ	MySQLタイプ
BOOL,	TINYINT
BOOLEAN	TINYINT
CHAR VARYING(M)	VARCHAR(M)
DEC	DECIMAL
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT

LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

データタイプのマッピングは、元のタイプの仕様が廃棄された後、テーブル作成時に行われます。もし他のベンダーが利用したタイプでテーブルを作成して、`DESCRIBE tbl_name` ステートメントを発行すると、MySQLは、同等のMySQLタイプを利用してテーブル構成をレポートします。例:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DESCRIBE t;
```

```
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| a     | tinyint(1) | YES  |     | NULL    |      |
| b     | double     | YES  |     | NULL    |      |
| c     | mediumtext | YES  |     | NULL    |      |
| d     | decimal(10,0) | YES  |     | NULL    |      |
+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```


第11章 関数と演算子

目次

演算子	570
演算子の優先順位	570
式評価でのタイプ変換	570
比較関数と演算子	572
論理演算子	575
制御フロー関数	577
文字列関数	578
文字列比較関数	587
正規表現	589
数字関数	592
算術演算子	592
数学関数	593
日付時刻関数	599
MySQL が使用するカレンダーは ?	613
全文検索関数	613
ブール全文検索	617
クエリ拡張を伴う全文検索	619
全文ストップワード	619
全文制限	622
微調整 MySQL 全文検索	622
キャスト関数と演算子	624
XML 関数	626
その他の関数	629
ビット関数	629
暗号化関数と圧縮関数	630
情報関数	634
その他の関数	640
GROUP BY 句との関数および修飾子の使用	642
GROUP BY (集約) 関数	642
GROUP BY 修飾子	645
非常時フィールドとの GROUP BY および HAVING	647

式は SQL 文のいくつかのポイントで使用することができます。例えば、**SELECT** 文の **ORDER BY** 句や **HAVING** 句、**SELECT** 文、**DELETE** 文、**UPDATE** 文の **WHERE** 句、または **SET** 文で使用することができます。式は、リテラル値やカラム値、**NULL**、組み込み関数、ストアド ファンクション、ユーザ定義の関数、そして演算子で書くことができます。この章は、MySQL で式を書くことができる関数と演算子を説明します。ストアド ファンクションおよびユーザ定義の関数の書き方は、[17章ストアードプロシージャとファンクション](#)と [Adding New Functions to MySQL](#) にあります。サーバが、異なる関数の引用をどう解釈するかについてのルールは、[関数名の構文解析と名前解決](#)を参照してください。

NULL を含む式は、その関数または演算子の資料で特別に説明されていない限り、常に **NULL** 値を生成します。

注記 :デフォルトでは、関数名とそれに続く丸括弧 (()) の間にはスペースを入れないことになっています。これは、MySQL パーサが、関数呼び出しと、関数と同じ名前を持つテーブルまたはカラムの参照を区別するのに役立ちます。しかし、関数インスウの周りにスペースを入れることは許可されています。

MySQL サーバが関数名の後のスペースを受け入れることは、`--sql-mode=IGNORE_SPACE` オプションで開始することで分かります。([SQL モード](#) 参照) 各クライアント プログラムは、`mysql_real_connect()` に `CLIENT_IGNORE_SPACE` オプションを使用することによって、この動作を指定することができます。どちらの場合でも、すべての関数名は予約語になります。

簡略化のため、この章で挙げられるほとんどの例は、省略形で `mysql` プログラムからの出力を記載しています。例は以下のように表示されず：

```
mysql> SELECT MOD(29,9);
```

```
+-----+
| mod(29,9) |
+-----+
|      2 |
+-----+
1 rows in set (0.00 sec)
```

このようなフォーマットで記されます：

```
mysql> SELECT MOD(29,9);
-> 2
```

注記

この章には多くの情報が含まれているため、特定の関数や演算子の情報を探すのは容易ではありません。情報の検索をより簡単にするため、各関数および演算子へのアンカーがこのマニュアルには加えられています。この資料の HTML バージョンでは、目的の関数などの HTML ページに掲載されているかが分かれば、直接その関数へナビゲートすることが出来ます。これは、[#function_function-name](#) を URL に追加することで可能になります。例えば、この資料のオンライン バージョンで [DATE_FORMAT](#) 関数の情報を探す場合は、日付時刻関数についての説明があるページに行き (<http://dev.mysql.com/doc/refman/5.1/en/date-and-time-functions.html>)、ウェブブラウザのアドレスバーのアドレスに [#function_date-format](#) を加えます。これで、[DATE_FORMAT](#) 関数に直接飛ぶことが出来ます。資料を単一ページバージョンでダウンロードした場合は、単に適切なアンカーの引用を追加してください。これと同様の方法で、適切な URL に [#operator_operator-name](#) を加えることによって、特定の演算子に飛ぶことも可能です。

演算子

演算子の優先順位

演算子の優先順位は、次の表で優先順位の低いものから高いものへと示されています。同じ行に並んで記載されている演算子は、優先順位が同じものです。

```
:=
||, OR, XOR
&&, AND
NOT
BETWEEN, CASE, WHEN, THEN, ELSE
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
<<, >>
-, +
*, /, DIV, %, MOD
^
- (unary minus), ~ (unary bit inversion)
!
BINARY, COLLATE
```

注記 [:HIGH_NOT_PRECEDENCE](#) SQL モードが有効になっていると、[NOT](#) の優先順位は [!](#) 演算子と同じになります。 [SQL モード](#) をご参照ください。

演算子の優先順位は、式の項の評価の順序を決定します。この順位とグループを明示的に上書きするには、丸括弧 (()) を使用します。例：

```
mysql> SELECT 1+2*3;
-> 7
mysql> SELECT (1+2)*3;
-> 9
```

式評価でのタイプ変換

演算子が異なるタイプのオペランドと共に使用される場合、オペランドを適合理化するため、タイプ変換が起こります。変換のあるものは、暗黙的に発生します。例えば MySQL は、必要に応じて数字を自動的にストリングに変換、またはその逆を行います。

```
mysql> SELECT 1+'1';
-> 2
mysql> SELECT CONCAT(2,'test');
-> '2 test'
```

また、明示的な変換を行うことも可能です。数字を明示的にストリングに変換したい場合は、`CAST()` または `CONCAT()` 関数を使用してください (`CAST()` を推奨) :

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
-> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
-> 38.8, '38.8'
```

次のルールは、比較の演算に対してどのように変換が行われるかを示しています :

- 一方か両方の引数が `NULL` の場合、比較の結果は、`NULL-safe <=>` 等値比較演算子以外は、`NULL` になります。 `NULL <=> NULL` の場合、結果は `true` です。
- 比較の演算の両方の引数がストリングの場合、それらはストリングとして比較されます。
- 両方の引数が整数の場合、それらは整数として比較されます。
- 16 進値が数字として比較されない場合は、バイナリ ストリングとして扱われます。
- 引数の一方が `TIMESTAMP` または `DATETIME` カラムで、他の引数が定数の場合、定数は比較が行われる前に、タイムスタンプに変換されます。これは、ODBC により適合させるためです。これは `IN()` への引数には適用されませんのでご注意ください！ 念のため、比較の際は常に完全な日付時刻、日付、または時刻ストリングを使用してください。
- 他のすべてのケースでは、引数は浮動小数点 (実) 数として比較されます。

次の例は、比較の演算の、ストリングから数字への変換を表したものです :

```
mysql> SELECT 1 > '6x';
-> 0
mysql> SELECT 7 > '6x';
-> 1
mysql> SELECT 0 > 'x6';
-> 0
mysql> SELECT 0 = 'x6';
-> 1
```

ストリング カラムを数字と比較する際、MySQL は、カラムのインデックスを使用して値を迅速に検索することができませんので注意してください。 `str_col` がインデックスの付いたストリング カラムである場合は、そのインデックスを、次のステートメントで検索を行う時に使用することはできません :

```
SELECT * FROM tbl_name WHERE str_col=1;
```

その理由は、`'1'`、`' 1'`、または `'1a'` のように、値 `1` に変換されうるストリングが数多くあるためです。

浮動小数点数 (または浮動小数点数に変換される値) を使用する比較は、それらの数字は不正確であるため、概算になります。そのため、一貫性のない結果が導き出される場合があります :

```
mysql> SELECT '18015376320243458' = 18015376320243458;
-> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
-> 0
```

そのような結果が発生しうるのは、53 ビットの精度しか持たない浮動小数点単に値が変換され、丸めの対象になるためです :

```
mysql> SELECT '18015376320243459'+0.0;
-> 1.8015376320243e+16
```

そのうえ、ストリングから浮動小数点への変換と、整数から浮動小数点への変換は、同様に起こらない場合もあります。整数は CPU によって浮動小数点に変換される可能性があり、対してストリングは、浮動小数点の掛け算を含む比較中の数字によって変換された数字であるためです。

表記されている結果はシステムによって異なり、コンピュータアーキテクチャやコンパイラのバージョン、または最適化のレベルなどの要素に影響される場合があります。それらの問題を避ける方法のひとつとして、`CAST()` を使用すると、値が暗黙的に浮動小数点数に変換されなくなります：

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
-> 1
```

浮動小数点の比較についての詳細は、[Problems with Floating-Point Comparisons](#) をご覧ください。

比較関数と演算子

比較の演算の結果は、`1` (`TRUE`)、`0` (`FALSE`)、または `NULL` の値になります。これらの演算は、数字とストリングの両方に適応します。必要に応じて、ストリングは数字に、数字はストリングに自動的に変換されます。

このセクションの関数のうちには、`1` (`TRUE`)、`0` (`FALSE`)、または `NULL` 以外の値を戻すものもあります。`LEAST()` および `GREATEST()` などがその例です。しかし、それらが戻す値は、[式評価でのタイプ変換](#) で説明されているルールによって行われた比較の演算に基づいています。

比較のために値を特定のタイプに変換するには、`CAST()` 関数を使用することができます。ストリング値は、`CONVERT()` を使用して、異なる文字セットに変換することが可能です。[キャスト関数と演算子](#) を参照してください。

デフォルトによって、文字比較は大文字小文字の区別の必要はなく、現在の文字セットを使用します。デフォルトは `latin1` (cp1252 West European) で、English でも正常に作用します。

- `=`

等しい：

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
-> 0
mysql> SELECT '0.01' = 0.01;
-> 1
```

- `<=>`

`NULL` - 安全等価。この演算は、`=` 演算子のように、等価比較を行います。両方のオペランドが `NULL` であれば、`NULL` でなく `1` を返し、一方のオペランドが `NULL` の場合は、`NULL` でなく `0` を返します。

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
-> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
-> 1, NULL, NULL
```

- `<>`, `!=`

等しくない：

```
mysql> SELECT '0.01' <> '0.01';
-> 1
mysql> SELECT .01 <> '0.01';
-> 0
mysql> SELECT 'zapp' <> 'zapp';
-> 1
```

- `<=`

より少ないか等しい：

```
mysql> SELECT 0.1 <= 2;
-> 1
```

- <

より少ない :

```
mysql> SELECT 2 < 2;
-> 0
```

- >=

より多いか等しい :

```
mysql> SELECT 2 >= 2;
-> 1
```

- >

より多い :

```
mysql> SELECT 2 > 2;
-> 0
```

- IS boolean_value, IS NOT boolean_value

boolean_value が TRUE か FALSE 、または UNKNOWN になり得る、ブーリアン値に対して値をテストします。

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
-> 1, 1, 1
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
-> 1, 1, 0
```

- IS NULL 、 IS NOT NULL

値が NULL であるか否かをテストします。

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1, 1, 0
```

ODBC プログラムとうまく適合させるため、IS NULL を使用する場合、MySQL は次の追加機能をサポートします :

- 一番最近の AUTO_INCREMENT 値を含む行を、その値を生成した直後に、次のフォームのステートメントを発行することによって検索することができます :

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

この動作は、SQL_AUTO_IS_NULL=0 を設定すると無効になります。SET 構文 を参照してください。

- NOT NULL として宣言された DATE および DATETIME カラムでは、次のようなステートメントを使用することで、特殊な日付 '0000-00-00' を検索することができます :

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

ODBC は '0000-00-00' をサポートしていないため、ODBC アプリケーションのあるものの作動にこれが必要になります。

- expr BETWEEN min AND max

expr が min より多いか等しく、expr が max より少ないか等しい場合、BETWEEN は 1 を返し、それ以外では 0 を返します。すべての引数が同じタイプの場合は、これは式 (min <= expr AND expr <= max) と等価になります。もしくは、式評価でのタイプ変換にあるルールによってタイプ変換が実施されますが、3 つすべての引数に適用されます。

```
mysql> SELECT 1 BETWEEN 2 AND 3;
-> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
-> 1
```

```
mysql> SELECT 2 BETWEEN 2 AND '3';
-> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
-> 0
```

BETWEEN を日付値または時刻値と使用する場合に最善の結果を得るには、値を所望のデータタイプに明示的に変換するため、**CAST()** を使用します。例：**DATETIME** をふたつの **DATE** 値と比較する場合、**DATE** 値を **DATETIME** 値に変換します。**'2001-1-1'** のような文字列定数を、**DATE** との比較で使用する場合、文字列を **DATE** にキャストします。

- **expr NOT BETWEEN min AND max**

これは、**NOT (expr BETWEEN min AND max)** と同様です。

- **COALESCE(value,...)**

リストの最初の非 **NULL** 値を戻すか、非 **NULL** 値がない場合は **NULL** を戻します。

```
mysql> SELECT COALESCE(NULL,1);
-> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
-> NULL
```

- **GREATEST(value1,value2,...)**

引数ふたつ以上では、最大の (最大値の) 引数を戻します。それらの引数は、**LEAST()** に対するものと同じルールで比較されます。

```
mysql> SELECT GREATEST(2,0);
-> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
-> 767.0
mysql> SELECT GREATEST('B','A','C');
-> 'C'
```

引数のどれかが **NULL** である場合、**GREATEST()** は **NULL** を戻します。

- **expr IN (value,...)**

expr が、**IN** リストのどれかの値と等しい場合は **1** を戻し、それ以外では **0** を戻します。すべての値が定数の場合、**expr** のタイプに基づいて評価し、分類します。その際の項目の検索は、バイナリ検索を使って行われます。これはつまり、**IN** は、**IN** 値のリストがすべて定数で構成されている場合、非常に速いということです。もしくは、**式評価でのタイプ変換** にあるルールによってタイプ変換が実施されますが、すべての引数に適用されます。

```
mysql> SELECT 2 IN (0,3,5,7);
-> 0
mysql> SELECT 'wefw' IN ('wee','wefwf','weg');
-> 1
```

引用符で括られた値 (文字列など) と括られていない値 (数字など) の比較ルールは異なるため、**IN** リストの引用符で括られた値と、括られていない値を決して混同しないでください。タイプの混同は、上記の理由により、結果の矛盾の原因になることがあります。例えば、**IN** 式を次のようには書かないでください：

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

正しい書き方はこのようになります：

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

IN リストの値の数は、**max_allowed_packet** 値によってのみ制限されます。

SQL の標準に準拠するため、**IN** は、左側の式が **NULL** である場合だけでなく、リストに一致するものがない場合、また、リストの式のひとつが **NULL** である場合にも、**NULL** を戻します。

IN() 構文は、ある種の副問い合わせを書くのにも使用できます。**ANY**、**IN**、そして **SOME** を持つサブクエリを参照してください。

- **expr NOT IN (value,...)**

これは、NOT (expr IN (value,...)) と同様です。

- ISNULL(expr)

expr が NULL の場合、ISNULL() は 1 を返し、それ以外では 0 を返します。

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

= の代わりに、ISNULL() を使って、値が NULL であるかテストすることができます。(= で値を NULL と比較すると、常に false が生じます。)

ISNULL() 関数は IS NULL 比較演算子と、いくつかの特殊な動作を共有します。IS NULL の解説を参照してください。

- INTERVAL(N,N1,N2,N3,...)

$N < N1$ の場合は 0 を、 $N < N2$ の場合は 1 を返し、というように続き、また N が NULL の場合は -1 を返します。すべての引数は整数として扱われます。この関数の $N1 < N2 < N3 < \dots < Nn$ が正しく作動することは必須条件です。これは、バイナリ検索が使用されていることが理由です (高速)。

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

- LEAST(value1,value2,...)

引数ふたつ以上では、最小の (最小値の) 引数を返します。引数は次のルールを使用して比較されます：

- 戻り値が INTEGER 文脈で使用されている場合、またはすべての引数が整数値である場合、それらは整数として比較されます。
- 戻り値が REAL 文脈で使用されている場合、またはすべての引数が実数値である場合、それらは実数として比較されます。
- 引数のいずれかが大文字小文字の区別のあるストリングである場合、引数は大文字小文字の区別のあるストリングとして比較されます。
- 他のすべてのケースでは、引数は大文字小文字の区別のあるストリングとして比較されます。

引数のどれかが NULL である場合、LEAST() は NULL を返します。

```
mysql> SELECT LEAST(2,0);
-> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
-> 3.0
mysql> SELECT LEAST('B','A','C');
-> 'A'
```

上記の変換ルールは、いくつかの境界例では異常な結果を引き起こす場合がありますのでご注意ください：

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
-> -9223372036854775808
```

これは、MySQL が 9223372036854775808.09223372036854775808.0 を整数の文脈で読み取ることが原因で起こります。整数表記は値を保持するのに十分ではないので、符号付整数にラップします。

論理演算子

SQL では、すべての論理演算子は **TRUE**、**FALSE**、または **NULL** に評価されます (**UNKNOWN**)。MySQL では、これらは 1 (**TRUE**)、0 (**FALSE**)、そして **NULL** として実行されます。サーバのあるものは **TRUE** にゼロ以外のすべての値を戻す場合があるものの、このほとんどは各種の SQL データベース サーバにとって通常のことです。

- **NOT, !**

NOT 演算。オペランドが 0 の場合は 1 に、オペランドがゼロ以外の場合は 0 に評価し、そして **NOT NULL** は **NULL** を戻します。

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
-> 1
mysql> SELECT NOT NULL;
-> NULL
mysql> SELECT !(1+1);
-> 0
mysql> SELECT ! 1+1;
-> 1
```

最後の例は、式の評価を **!(1)+1** と同様に行うため、1 を生成します。

- **AND, &&**

AND 演算。すべてオペランドがゼロ以外で非 **NULL** の場合は 1 に、ひとつ以上のオペランドが 0 の場合は 0 に評価し、それ以外は **NULL** を戻します。

```
mysql> SELECT 1 && 1;
-> 1
mysql> SELECT 1 && 0;
-> 0
mysql> SELECT 1 && NULL;
-> NULL
mysql> SELECT 0 && NULL;
-> 0
mysql> SELECT NULL && 0;
-> 0
```

- **OR, ||**

OR 演算。両方のオペランドが非 **NULL** である時、オペランドのどれかがゼロ以外である場合は結果は 1、その他は 0 になります。ひとつが **NULL** オペランドであれば、他のオペランドがゼロ以外である場合の結果は 1、その他は **NULL** になります。両方のオペランドが **NULL** であれば、結果は **NULL** になります。

```
mysql> SELECT 1 || 1;
-> 1
mysql> SELECT 1 || 0;
-> 1
mysql> SELECT 0 || 0;
-> 0
mysql> SELECT 0 || NULL;
-> NULL
mysql> SELECT 1 || NULL;
-> 1
```

- **XOR**

XOR 演算。オペランドのどちらかが **NULL** である場合は **NULL** を戻します。非 **NULL** のオペランドの場合は、ゼロ以外のオペランドの数が奇数であれば 1 に評価し、それ以外は 0 を戻します。

```
mysql> SELECT 1 XOR 1;
-> 0
mysql> SELECT 1 XOR 0;
-> 1
mysql> SELECT 1 XOR NULL;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1;
-> 1
```

a XOR b は、数学的に **(a AND (NOT b)) OR ((NOT a) and b)** に等価です。

制御フロー関数

- `CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END`

`CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END`

最初の例は、`value=compare_value` に `result` を戻します。2 番目は `true` である最初の条件に結果を戻します。一致する結果値がない場合は、`ELSE` のあとの結果が戻され、`ELSE` パートがない場合は、`NULL` が戻されます。

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
-> WHEN 2 THEN 'two' ELSE 'more' END;
-> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'
mysql> SELECT CASE BINARY 'B'
-> WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
-> NULL
```

`CASE` 式のデフォルトの戻り値タイプは、すべての戻り値の適合集合体タイプですが、使用される文脈にもよります。ストリング文脈で使用される場合は、結果はストリングとして戻されます。数値文脈で使用される場合は、結果は 10 進値、実数値、または整数値として戻されます。

注記:ここで示されている `CASE` 式の構文は、ストアードルーチン内で使用する場合、`CASEステートメント` で説明されている、`CASE` 文とはやや異なります。`CASE` 文は `ELSE NULL` 句を持つことができず、`END` でなく、`END CASE` で終了します。

- `IF(expr1,expr2,expr3)`

`expr1` が `TRUE` である場合は (`expr1 <> 0` および `expr1 <> NULL`)、`IF()` は `expr2` を戻します。それ以外では、`expr3` を戻します。`IF()` は、使用されている文脈によって、数値値もしくはストリング値を戻します。

```
mysql> SELECT IF(1>2,2,3);
-> 3
mysql> SELECT IF(1<2,'yes','no');
-> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
-> 'no'
```

`expr2` または `expr3` のうちのひとつが、明示的に `NULL` である場合は、`IF()` 関数の結果タイプは、非 `NULL` 式のタイプになります。

`expr1` は整数値として評価されます。つまり、浮動小数点値、またはストリング値をテストしている場合は、比較演算を使用して行うべきということになります。

```
mysql> SELECT IF(0.1,1,0);
-> 0
mysql> SELECT IF(0.1<>0,1,0);
-> 1
```

この最初の例では、`IF(0.1)` は、`IF(0)` のテストの結果として `0.1` が整数値に変換されるため、`0` を戻します。これは不測の結果であるかもしれません。2 番目の例では、比較テストは、オリジナルの浮動小数点値がゼロ以外であるかテストします。比較の結果は整数として使用されます。

デフォルトの `IF()` の戻り値タイプは (一時テーブルに保管される時に重要な場合あり)、次のように計算されます：

式	戻り値
<code>expr2</code> または <code>expr3</code> はストリングを戻す	ストリング
<code>expr2</code> または <code>expr3</code> は浮動小数点値を戻す	浮動小数点
<code>expr2</code> または <code>expr3</code> は整数を戻す	整数

`expr2` と `expr3` の両方がストリングで、どちらかのストリングが大文字小文字の区別をする場合、結果は大文字子目の区別があります。

注記 :IF 文 もあり、それはここで説明されている IF() 関数 とは異なります。IF ステートメント を参照してください。

- IFNULL(expr1,expr2)

expr1 が NULL でない場合、IFNULL() は expr1 を返し、それ以外では expr2 を返します。IFNULL() は、使用されている文脈によって、数値値もしくは文字列値を返します。

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

IFNULL(expr1,expr2) のデフォルトの結果値は、STRING、REAL、または INTEGER の順に、ふたつの式のより「一般的」なものです。式や MySQL が一時テーブルの IFNULL() によって戻された値を内部に蓄積しなければならない場所に基づくテーブルの大文字小文字を考慮してください：

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| test  | varbinary(4) |      |      |          |       |
+-----+-----+-----+-----+-----+
```

この例では、test カラムのタイプは VARBINARY(4) です。

- NULLIF(expr1,expr2)

expr1 = expr2 が true の場合は NULL を返し、それ以外では expr1 を返します。これは、CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END と同様です。

```
mysql> SELECT NULLIF(1,1);
-> NULL
mysql> SELECT NULLIF(1,2);
-> 1
```

MySQL は、引数が等しくない場合、expr1 を 2 度評価しますのでご注意ください。

文字列関数

文字列値の関数は、結果の長さが max_allowed_packet システム環境変数より長くなると、NULL を返します。サーバパラメータのチューニング を参照してください。

文字列の位置を演算する関数では、最初の位置は 1 と数値付けられます。

- ASCII(str)

文字列 str の左側の文字の数値を返します。str が空の文字列である場合は、0 を返します。str が NULL である場合は NULL を返します。ASCII() は、0 から 255 の数値を持つ文字に使用できます。

```
mysql> SELECT ASCII('2');
-> 50
mysql> SELECT ASCII(2);
-> 50
mysql> SELECT ASCII('dx');
-> 100
```

ORD() 関数も併せてご参照ください。

- BIN(N)

N のバイナリ値の文字列表現を戻します。**N** は longlong (**BIGINT**) 数字です。これは、**CONV(N,10,2)** に等価になります。**N** が **NULL** である場合は **NULL** を戻します。

```
mysql> SELECT BIN(12);
-> '1100'
```

- **BIT_LENGTH(str)**

ストリング **str** の長さをビットで戻します。

```
mysql> SELECT BIT_LENGTH('text');
-> 32
```

- **CHAR(N,... [USING charset_name])**

CHAR() 各引数 **N** を整数として解釈し、それらの整数のコード値によって与えられた文字を構成するストリングを戻します。**NULL** 値はとばされます。

```
mysql> SELECT CHAR(77,121,83,81,'76');
-> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
-> 'MMM'
```

255 より大きい **CHAR()** 引数は複数結果バイトに変換されます。例えば、**CHAR(256)** は **CHAR(1,0)** に等しく、**CHAR(256*256)** は **CHAR(1,0,0)** に等しいことになります：

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+-----+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+-----+
| 0100          | 0100          |
+-----+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+-----+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+-----+
| 010000          | 010000          |
+-----+
```

デフォルトにより、**CHAR()** はバイナリ ストリングを戻します。与えられた文字セットでストリングを生成するには、オプションの **USING** 句を使用します：

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+-----+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+-----+
| binary              | utf8                            |
+-----+
```

USING が与えられ、結果ストリングが与えられた文字セットにとって不当になる場合は、警告が発行されます。また、厳密な SQL モードが有効にされた場合は、**CHAR()** からの結果は **NULL** になります。

- **CHAR_LENGTH(str)**

文字で測られたストリング **str** の長さを戻します。マルチバイト文字は、1 文字として数えられます。つまり、2 バイトの文字を 5 つ含むストリングには、**CHAR_LENGTH()** は 5 を戻すところを、**LENGTH()** は 10 を戻します。

- **CHARACTER_LENGTH(str)**

CHARACTER_LENGTH() is a synonym for **CHAR_LENGTH()**.

- **CONCAT(str1,str2,...)**

引数を連結した結果であるストリングを戻します。ひとつ以上の引数を持つ場合があります。すべての引数が非バイナリ ストリングである場合、結果は非バイナリ ストリングになります。引数がひとつでもバイナリ ストリングを含む場合は、結果はバイナリ ストリングになります。数値の引数はそれに等しいバイナリ ストリング形態に変換されます。それを避けたい場合は、次の例のように、明示的なタイプ キャストを使用することができます：

```
SELECT CONCAT(CAST(int_col AS CHAR), char_col);
```

引数のどれかが `NULL` である場合、`CONCAT()` は `NULL` を返します。

```
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
-> NULL
mysql> SELECT CONCAT(14.3);
-> '14.3'
```

- `CONCAT_WS(separator, str1, str2, ...)`

`CONCAT_WS()` は Concatenate With Separator (セパレータと連結) を意味しており、`CONCAT()` の特殊な形態です。最初の引数が、残りの引数のセパレータになります。セパレータは、連結されるストリングの間に追加されます。セパレータは、あとの引数と同じく、ストリングである場合があります。セパレータが `NULL` の場合は、結果は `NULL` になります。

```
mysql> SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');
-> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',', 'First name', NULL, 'Last Name');
-> 'First name,Last Name'
```

`CONCAT_WS()` は空のストリングをとばしません。しかし、セパレータ引数のあとの `NULL` 値はすべてとばします。

- `CONV(N, from_base, to_base)`

異なる基数間の数値を変換します。基数 `from_base` から基数 `to_base` に変換された、数値 `N` の文字列表現を返します。引数のどれかが `NULL` である場合は `NULL` を返します。引数 `N` は整数として解釈されますが、整数またはストリングとして特定される場合があります。最小限の基数は 2 で、最大の基数は 36 です。`to_base` が負数である場合は、`N` は符号付き数として登録されます。それ以外では、`N` は符号なしとして扱われます。`CONV()` は 64 ビット精度で動作します。

```
mysql> SELECT CONV('a', 16, 2);
-> '1010'
mysql> SELECT CONV('6E', 18, 8);
-> '172'
mysql> SELECT CONV(-17, 10, -18);
-> '-H'
mysql> SELECT CONV(10+'10'+10+0xa, 10, 10);
-> '40'
```

- `ELT(N, str1, str2, str3, ...)`

`N = 1` の場合は `str1` を返し、`N = 2` の場合は `str2` を返し、というふうに続きます。`N` が 1 以下か、引数の数より大きければ、`NULL` を返します。`ELT()` は `FIELD()` の補数です。

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
-> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
-> 'foo'
```

- `EXPORT_SET(bits, on, off[, separator[, number_of_bits]])`

値 `bits` の各ビットセットには `on` ストリングが返され、各再生ビットには `off` が返されます。`bits` のビットは右から左に検査されます (下位ビットから上位ビット)。ストリングは、`separator` ストリング (デフォルトはコンマ ',') で区切られた結果の左から右意へ追加されます。検査されたビットの数は `number_of_bits` によって与えられます (デフォルトでは 64)。

```
mysql> SELECT EXPORT_SET(5, 'Y', 'N', '', 4);
-> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6, '1', '0', '', 10);
-> '0,1,1,0,0,0,0,0,0,0'
```

- `FIELD(str, str1, str2, str3, ...)`

`str1`、`str2`、`str3`、... リストの `str` の開始位置 (ポジション) を戻します。`str` が見つからない場合は、`0` を戻します。

`FIELD()` へのすべての引数が文字列の場合、すべての引数は文字列として比較されます。すべての引数が数値の場合、それらは数値として比較されます。それ以外は、引数は `double` として比較されます。

`str` が `NULL` である場合、`NULL` はいかなる値との比較でも等価にならないため、戻り値は `0` になります。`FIELD()` は `ELT()` の補数です。

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 0
```

- `FIND_IN_SET(str, strlist)`

文字列 `str` が `N` サブ文字列で構成される文字列 リスト `strlist` 内にある場合は、`1` から `N` の範囲の値を戻します。文字列 リストは、`'` 文字で区切られたサブ文字列で構成された文字列です。最初の引数が定数列で、`2` 番目がタイプ `SET` のカラムの場合、`FIND_IN_SET()` 関数はビット演算を使用するために最適化されます。`str` が `strlist` 内にない場合、または `strlist` が空の文字列の場合は、`0` を戻します。引数のどちらかが `NULL` である場合は `NULL` を戻します。この関数は、最初の引数がコンマ (`,`) 文字を含む場合は正常に作動しません。

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
-> 2
```

- `FORMAT(X,D)`

数字 `X` を `'#,###,###.###'` のようにフォーマットし、`D` 少数位まで丸め、その結果を文字列として戻します。`D` が `0` の場合、結果は小数点または小数部を持ちません。

```
mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
-> '12,332'
```

- `HEX(N_or_S)`

`N_or_S` が数字の場合、`N` の `16` 進値の文字列表現を戻します。`N` は `longlong (BIGINT)` 数です。これは、`CONV(N,10,16)` に等価になります。

`N_or_S` が文字列の場合は、`N_or_S` の各文字が二桁の `16` 進数に変換される、`N_or_S` の `16` 進数字列表現を戻します。

```
mysql> SELECT HEX(255);
-> 'FF'
mysql> SELECT HEX(0x616263);
-> 'abc'
mysql> SELECT HEX('abc');
-> 616263
```

- `INSERT(str,pos,len,newstr)`

文字列 `str` を、位置 `pos` で始まるサブ文字列と、文字列 `newstr` に置換された `len` 文字長と共に戻します。`pos` が文字列の長さに収まらない場合は、元の文字列を返します。`len` が残りの文字列の長さに収まらない場合は、位置 `pos` からの残りの文字列を置換します。引数のどれかが `NULL` である場合は `NULL` を戻します。

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
-> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
-> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
-> 'QuWhat'
```

この関数はマルチバイトでも安全です。

- **INSTR(str,substr)**

文字列 **str** 内のサブ文字列 **substr** の最初の発生位置を返します。これは、**LOCATE()** の引数がふたつの形式の、引数の順番が逆になったものと同じですが、

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

この関数はマルチバイトでも安全で、少なくともひとつの引数がバイナリ文字列である場合は、大文字小文字の区別が必須です。

- **LCASE(str)**

LCASE() は **LOWER()** の別名です。

- **LEFT(str,len)**

文字列 **str** からの左側の **len** 文字を返し、引数が **NULL** である場合は **NULL** を返します。

```
mysql> SELECT LEFT('foobarbar', 5);
-> 'fooba'
```

- **LENGTH(str)**

バイトで測られた文字列 **str** の長さを返します。マルチバイト文字は、複数バイトとして数えられます。つまり、2 バイトの文字を 5 つ含む文字列には、**CHAR_LENGTH()** は 5 を返すところを、**LENGTH()** は 10 を返します。

```
mysql> SELECT LENGTH('text');
-> 4
```

- **LOAD_FILE(file_name)**

ファイルを読み取り、その内容を文字列として返します。この関数を使用するには、ファイルがサーバホストに置かれていなければならないのと、ファイルへのフルパス名を特定し、**FILE** 権限を持つ必要があります。ファイルはあらゆる点で読取可能でなければならないと、**max_allowed_packet** バイトより小さなサイズである必要があります。

ファイルが存在しない場合、または、上記の条件が満たされておらず、読取が不可能な場合、この関数は **NULL** を返します。

MySQL 5.1.6 からは、**character_set_filesystem** システム環境変数が、リテラル文字列として与えられたファイル名の解釈をコントロールします。

```
mysql> UPDATE t
SET blob_col=LOAD_FILE('/tmp/picture')
WHERE id=1;
```

- **LOCATE(substr,str), LOCATE(substr,str,pos)**

最初の構文は、文字列 **str** 内のサブ文字列 **substr** の最初の発生位置を返します。2 番目の構文は、位置 **pos** で始まる文字列 **str** 内のサブ文字列 **substr** の最初の発生位置を返します。**str** 内に **substr** がない場合は 0 を返します。

```
mysql> SELECT LOCATE('bar', 'foobarbar');
-> 4
mysql> SELECT LOCATE('xbar', 'foobar');
-> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
-> 7
```

この関数はマルチバイトでも安全で、少なくともひとつの引数がバイナリ文字列である場合は、大文字小文字の区別が必須です。

- **LOWER(str)**

現在の文字セットのマッピングに基づいてすべての文字が小文字に変更された文字列 `str` を返します。デフォルトは `latin1` (cp1252 West European) です。

```
mysql> SELECT LOWER('QUADRATICALLY');
-> 'quadratically'
```

この関数はマルチバイトでも安全です。

- `LPAD(str,len,padstr)`

`len` 文字の長さへ、文字列 `padstr` で左にパッドされた文字列 `str` を返します。`str` が `len` より長い場合は、戻り値は `len` 文字に縮められます。

```
mysql> SELECT LPAD('hi',4,'?');
-> '??hi'
mysql> SELECT LPAD('hi',1,'?');
-> 'h'
```

- `LTRIM(str)`

頭のスペース文字を除いた文字列 `str` を返します。

```
mysql> SELECT LTRIM(' barbar');
-> 'barbar'
```

この関数はマルチバイトでも安全です。

- `MAKE_SET(bits,str1,str2,...)`

`bits` セット内の対応するビットを持つ文字列で構成されるセット値 (',' 文字によって区切られたサブ文字列を含む) を返します。`str1` はビット 0 に対応し、`str2` はビット 1 に対応する、というようになります。`str1`、`str2`、... 内の `NULL` 値は結果に追加されません。

```
mysql> SELECT MAKE_SET(1,'a','b','c');
-> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
-> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
-> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
-> ''
```

- `MID(str,pos,len)`

`MID(str,pos,len)` は、`SUBSTRING(str,pos,len)` のシノニムです。

- `OCT(N)`

`N` の 8 進数の文字列表現を返します。`N` は `longlong` (`BIGINT`) 数字です。これは、`CONV(N,100.8)` に等価になります。`N` が `NULL` である場合は `NULL` を返します。

```
mysql> SELECT OCT(12);
-> '14'
```

- `OCTET_LENGTH(str)`

`OCTET_LENGTH()` は `LENGTH()` のシノニムです。

- `ORD(str)`

文字列 `str` の左端の文字がマルチバイト文字の場合は、次の公式を使ってその構成バイトの数値から計算された、その文字のコードを返します：

```
(1st byte code)
+ (2nd byte code × 256)
+ (3rd byte code × 2562) ...
```

左端の文字がマルチバイト文字でない場合は、`ORD()` は `ASCII()` 関数と同じ値を返します。

```
mysql> SELECT ORD('2');  
-> 50
```

- **POSITION(substr IN str)**

POSITION(substr IN str) は **LOCATE(substr,str)** のシノニムです。

- **QUOTE(str)**

SQL 文で、適切にエスケープされたデータ値として使用できる結果を生成する文字列を引用します。文字列は単一引用符で囲まれ、単一引用符 (`'`)、バックスラッシュ (`\`)、ASCII **NUL**、そして バックスラッシュによって先行された Control-Z の各インスタンスと共に戻されます。引数が **NULL** の場合は、戻り値は単一引用符の囲みなしの語句「**NULL**」になります。

```
mysql> SELECT QUOTE('Don\'t!');  
-> 'Don\'t!'  
mysql> SELECT QUOTE(NULL);  
-> NULL
```

- **REPEAT(str,count)**

count 回繰り返された文字列 **str** で構成される文字列を戻します。**count** が 1 より小さい場合は、空の文字列を戻します。**str** もしくは **count** が **NULL** である場合は **NULL** を戻します。

```
mysql> SELECT REPEAT('MySQL', 3);  
-> 'MySQLMySQLMySQL'
```

- **REPLACE(str,from_str,to_str)**

文字列 **to_str** によって置換された文字列 **from_str** のすべての発生と共に、文字列 **str** を戻します。**REPLACE()** は、**from_str** を検索する際、大文字小文字を区別した検出を行います。

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');  
-> 'WwWwWw.mysql.com'
```

この関数はマルチバイトでも安全です。

- **REVERSE(str)**

文字の順序が逆になった文字列 **str** を戻します。

```
mysql> SELECT REVERSE('abc');  
-> 'cba'
```

この関数はマルチバイトでも安全です。

- **RIGHT(str,len)**

文字列 **str** からの右側の **len** 文字を戻し、引数が **NULL** である場合は **NULL** を戻します。

```
mysql> SELECT RIGHT('foobarbar', 4);  
-> 'rbar'
```

この関数はマルチバイトでも安全です。

- **RPAD(str,len,padstr)**

len 文字の長さへ、文字列 **padstr** で右にパッドされた文字列 **str** を戻します。**str** が **len** より長い場合は、戻り値は **len** 文字に縮められます。

```
mysql> SELECT RPAD('hi',5,'?');  
-> 'hi???'  
mysql> SELECT RPAD('hi',1,'?');  
-> 'h'
```

この関数はマルチバイトでも安全です。

- **RTRIM(str)**

最後のスペース文字を除いた文字列 `str` を返します。

```
mysql> SELECT RTRIM('barbar ');
-> 'barbar'
```

この関数はマルチバイトでも安全です。

- **SOUNDEX(str)**

`str` から soundex スtring を返します。サウンドがほぼ同じなふたつの String は、同等の soundex String を持っています。標準の soundex String は長さ 4 文字ですが、**SOUNDEX()** 関数は任意の長さの String を返します。標準の soundex String を得るには、結果に **SUBSTRING()** を使用することができます。`str` 内のすべての非アルファベット文字は無視されます。A から Z 以外のすべての国際アルファベット文字は、母音として扱われます。

重要点: **SOUNDEX()** を使用する場合は、次の制限に留意してください:

- 現在実装されているこの関数は、英語言語のみとの作動が意図されています。多言語での String は、正確な結果を生成できない場合があります。
- この関数は、**utf-8** を含むマルチバイト文字セットを使用する String では、一貫性のある結果を提供する保証はありません。

今後のリリースでは、これらの制限をなくせるよう努力しています。詳細は Bug #22638 をご覧ください。

```
mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
```

注記: この関数は、もっと一般的な拡張版ではなく、元来の Soundex アルゴリズムを実装しています。その相違点としては、元来のバージョンは最初に母音を破棄してから、複製を捨てますが、拡張版ではまず複製を破棄し、それから母音を捨てます。

- **expr1 SOUNDS LIKE expr2**

これは、**SOUNDEX(expr1) = SOUNDEX(expr2)** と同様です。

- **SPACE(N)**

N スペース文字で構成される String を返します。

```
mysql> SELECT SPACE(6);
-> '      '
```

- **SUBSTRING(str,pos), SUBSTRING(str FROM pos), SUBSTRING(str,pos,len), SUBSTRING(str FROM pos FOR len)**

len 引数なしのフォームは、位置 **pos** ではじまる、String `str` からのサブ String を返します。**len** 引数を持つフォームは、位置 **pos** ではじまる、String `str` からのサブ String **len** 文字長を返します。**FROM** を使用するフォームは標準の SQL 構文です。また、**pos** にマイナス値を使用することも可能です。その場合、サブクエリの頭は、String の最初でなく、String の最後からの **pos** 文字です。マイナス値は、この関数のあらゆるフォームで、**pos** に使用することもできます。

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
-> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
-> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'
```

この関数はマルチバイトでも安全です。

`len` が 1 以下の場合、結果は空のストリングになります。

`SUBSTR()` は `SUBSTRING()` のシノニムです。

- `SUBSTRING_INDEX(str,delim,count)`

デリミッタ `delim` の `count` 発生前に、ストリング `str` を戻します。`count` がプラスの場合、最後のデリミッタ (左から数えて) の左側のすべてを戻します。`count` がマイナスの場合、最後のデリミッタ (右から数えて) の右側のすべてを戻します。`SUBSTRING_INDEX()` は、`delim` を検索する際、大文字小文字を区別した検出を行います。

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
-> 'mysql.com'
```

この関数はマルチバイトでも安全です。

- `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)`, `TRIM([remstr FROM] str)`

すべての `remstr` プレフィックスまたはサフィックスを除いたストリング `str` を戻します。拡張子 `BOTH`、`LEADING`、または `TRAILING` のうちいずれも与えられていない場合は、`BOTH` が仮定されます。`remstr` はオプションで、指定されていない限り、スペースは除かれます。

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
-> 'barx'
```

この関数はマルチバイトでも安全です。

- `UCASE(str)`

`UCASE()` は `UPPER()` のシノニムです。

- `UNHEX(str)`

`HEX(str)` の逆演算を行います。引数内の 16 進数のそれぞれのペアを数字として解釈し、それを数字で表される文字に変換します。結果としての文字はバイナリストリングとして戻されます。

```
mysql> SELECT UNHEX('4D7953514C');
-> 'MySQL'
mysql> SELECT 0x4D7953514C;
-> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
-> 'string'
mysql> SELECT HEX(UNHEX('1267'));
-> '1267'
```

引数ストリング内の文字は、正当な 16 進数である必要があります: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. `UNHEX()` が引数内で非 16 進数に遭遇する場合は `NULL` を戻します:

```
mysql> SELECT UNHEX('GG');
+-----+
| UNHEX('GG') |
+-----+
| NULL       |
+-----+
```

`NULL` という結果は、`UNHEX()` への引数が `BINARY` カラムである場合、値が保存される時に 0x00 バイトでパッドされるために起こりますが、これらのバイトは検索でストリップされません。例えば、`'aa'` は `'aa '` として `CHAR(3)` カラムに保存され、`'aa'` (トレーリング パッドスペースがストリップされた状態) として検索されます。それによ

り、カラム値の `UNHEX()` は `A` を戻します。それに対し、`'aa'` は `BINARY(3)` カラムに `'aa\0'` として保存され、`'aa\0'` (トレーリング パッド `0x00` バイトがストリップされていない常態で) として検索されます。`'\0'` は正当な 16 進数ではないので、カラム値の `UNHEX()` は `NULL` を戻します。

- `UPPER(str)`

現在の文字セットのマッピングに基づいてすべての文字が大文字に変更されたストリング `str` を戻します。デフォルトは `latin1` (cp1252 West European) です。

```
mysql> SELECT UPPER('Hej');
-> 'HEJ'
```

この関数はマルチバイトでも安全です。

文字列比較関数

文字列関数がバイナリ ストリングを引数として与えられている場合、結果ストリングもバイナリ ストリングとなります。ストリングに変換された数字は、バイナリ ストリングとして扱われます。これは比較にのみ影響を及ぼします。

通常、文字列比較の式に大文字小文字の区別のあるものがある場合、その比較は大文字小文字の区別のある様式で行われます。

- `expr LIKE pat [ESCAPE 'escape_char']`

SQL の簡単な正規の比較式を使用してのパターン マッチング。1 (`TRUE`) または 0 (`FALSE`) を戻します。`expr` もしくは `pat` のどちらかが `NULL` である場合、結果は `NULL` になります。

パターンはリテラル ストリングである必要があります。例えば、文字列式、またはテーブル カラムとして指定するのでもかまいません。

SQL 標準に当たり、`LIKE` は文字ごとにマッチングを行うので、`=` 比較演算子とは異なる結果を生成することができます。

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----+
|                                0 |
+-----+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-----+
|                                1 |
+-----+
```

`LIKE` では、次のふたつのワイルドカード文字をパターンで使用することができます：

文字	説明
<code>%</code>	0 からあらゆる数の文字でもマッチする。
<code>_</code>	ひとつの文字を明確にマッチする。

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
-> 1
```

ワイルドカード文字のリテラル インスタンスをテストするには、エスケープ文字で優先させます。`ESCAPE` 文字を指定しない場合は、`'\'` が仮定されます。

ストリング	説明
<code>\%</code>	<code>'%'</code> 文字をひとつマッチする。
<code>_</code>	<code>'_'</code> 文字をひとつマッチする。

```
mysql> SELECT 'David!' LIKE 'David\_';
-> 0
mysql> SELECT 'David\_ ' LIKE 'David\_';
-> 1
```

異なるエスケープ文字を指定するには、`ESCAPE` 句を使用します：

```
mysql> SELECT 'David\_ ' LIKE 'David\_ ' ESCAPE '\\';
-> 1
```

エスケープシーケンスは空か、1文字長である必要があります。MySQL 5.1.2 から
は、`NO_BACKSLASH_ESCAPES` SQL モードを有効にすると、シーケンスを空にすることはできません。

次のふたつのステートメントは、オペランドのひとつがバイナリストリングでない限り、文字列比較は大文字小文字の区別をしないことを示しています：

```
mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0
```

MySQL では、`LIKE` を数値式で使用することができます。(標準の SQL `LIKE` のエクステンションです)。

```
mysql> SELECT 10 LIKE '1%';
-> 1
```

注記 :MySQL は C エスケープ構文をストリングで使用するため (例えば、`'\n'` で改行文字を表現)、`LIKE` ストリングで使用する `'\\'` はすべて二重にする必要があります。例えば、`'\\n'` を検索するには、`'\\\\n'` と指定します。`'\\'` の検索には、`'\\\\\\\\'` と指定します。これは、バックスラッシュがパーサによってストリップされ、そしてパターンのマッチが実行された時にもストリップされるため、ひとつのバックスラッシュを残してマッチさせるためです。(例外 :パターンストリングの最後では、バックスラッシュは `'\\\\'` と指定できます。ストリングの末尾では、エスケープの後に連なるものがないため、バックスラッシュはそのもので独立することができます)。

- `expr NOT LIKE pat [ESCAPE 'escape_char']`

これは、`NOT (expr LIKE pat [ESCAPE 'escape_char'])` と同様です。

注記

`NULL` を含むカラムとの `NOT LIKE` 比較を伴う Aggregate クエリは、予想外の結果を生成します。例として、次の表とデータを参考にしてください：

```
CREATE TABLE foo (bar VARCHAR(10));
INSERT INTO foo VALUES (NULL), (NULL);
```

クエリ `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` は `0` を返しますが、`SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` は `2` を返すと思われます。しかし、この場合は異なります：2 番目のクエリは `0` を返します。これは、`NULL NOT LIKE expr` が、`expr` の値に関わりなく、常に `NULL` を返すためです。`NULL` を伴う aggregate クエリと、`NOT RLIKE` または `NOT REGEXP` を使用する比較でも同様です。このような場合、次のように、`OR (AND ではなく)` を使用して、`NOT NULL` を明示的にテストする必要があります：

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `expr NOT REGEXP pat, expr NOT RLIKE pat`

これは、`NOT (expr REGEXP pat)` と同様です。

- `expr REGEXP pat, expr RLIKE pat`

パターン `pat` に対して、ストリングの式 `expr` のパターン照合を行います。このパターンは拡張正規表現にもなりえます。正規表現の構文については、[正規表現](#) で説明されています。`expr` が `pat` と一致する場合は `1` を返し、そ

れ以外では 0 を返します。expr もしくは pat のどちらかが NULL である場合、結果は NULL になります。RLIKE は、mSQL との互換性のために用意された、REGEXP のシノニムです。

パターンはリテラル スtring である必要があります。例えば、文字列式、またはテーブル カラムとして指定するのでもかまいません。

注記 :MySQL は C エスケープ構文を String で使用するため (例えば、'\n' で改行文字を表現)、REGEXP String で使用する '\ ' はすべて二重にする必要があります。

REGEXP は、バイナリ String と使用する場合以外では、大文字小文字の区別をしません。

```
mysql> SELECT 'Monty!' REGEXP 'm%y%';
-> 0
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*\\.\\*line';
-> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
-> 1 0
mysql> SELECT 'a' REGEXP '[a-d]';
-> 1
```

REGEXP および RLIKE は、文字のタイプを決定する際に、現行の文字セットを使用します。デフォルトは latin1 (cp1252 West European) です。注意 : これらの演算子はマルチバイトでは安全ではありません。

- STRCMP(expr1,expr2)

STRCMP() は、String が同じであれば 0 を返し、現行のソート順において最初の引数が 2 番目のものより小さい場合は -1、そしてそれ以外では 1 を返します。

```
mysql> SELECT STRCMP('text', 'text2');
-> -1
mysql> SELECT STRCMP('text2', 'text');
-> 1
mysql> SELECT STRCMP('text', 'text');
-> 0
```

STRCMP() は、比較が行われる際、現行の文字セットを使用します。これによって、デフォルトの比較挙動では、ひとつか双方のオペランドがバイナリ String でない限り、大文字小文字の区別がなくなります。

正規表現

正規表現は、複雑な検索でパターンを特定する協力的な方法です。

MySQL はヘンリー・スペンサーの正規表現の実装を使用します。これは、POSIX 1003.2 との適合性を目指したものです。付録 E Credits をご覧ください。MySQL は、SQL 文での、REGEXP 演算子とのパターン照会演算をサポートするため、拡張バージョンを使用します。Pattern Matching および 文字列比較関数 を参照してください。

このセクションでは、MySQL で REGEXP 演算に使用される特殊な文字や構文の概要や例を記載します。ヘンリー・スペンサーの regex(7) マニュアル ページにあるような詳細は網羅していません。このマニュアル ページは MySQL のソース配布物の、regex ディレクトリ下の regex.7 ファイルに含まれています。

正規表現は String のセットを示します。最も簡単な正規表現は、特殊な文字を使用していないものです。例えば、正規表現 hello は hello のみにマッチします。

重要な正規表現は特定の特殊構文を使用し、ひとつ以上の String とマッチするようにします。例えば、正規表現 hello|word は、String hello または String word とマッチします。

さらに複雑な例としては、正規表現 B[an]*s は、String Bananas、Baaaaas、Bs のいずれとでもマッチし、また、他の B で始まる String、s で終わる String、ひとつでも a または n 文字を間に含む String と一致します。

REGEXP 演算子の正規表現は、次の特殊文字および構文のいずれかを使用する場合があります :

- ^

ストリングの頭にマッチ。

```
mysql> SELECT 'fo\info' REGEXP '^fo$';      -> 0
mysql> SELECT 'fofo' REGEXP '^fo$';        -> 1
```

- **\$**

ストリングの最後にマッチ。

```
mysql> SELECT 'fo\no' REGEXP 'fo\no$';      -> 1
mysql> SELECT 'fo\no' REGEXP 'fo$';         -> 0
```

- **.**

あらゆる文字とマッチ (改行復帰および通常改行を含む)。

```
mysql> SELECT 'fofo' REGEXP 'f.f.$';        -> 1
mysql> SELECT 'fo\r\info' REGEXP 'f.f.$';   -> 1
```

- **a***

ゼロ以上の **a** 文字のあらゆるシークエンスにマッチ。

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';         -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';      -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';         -> 1
```

- **a+**

1 以上の **a** 文字のあらゆるシークエンスにマッチ。

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';         -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';         -> 0
```

- **a?**

ゼロ、または 1 以上の **a** 文字とマッチ。

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';         -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';        -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';       -> 0
```

- **de|abc**

シークエンス **de** または **abc** のどちらかをマッチ。

```
mysql> SELECT 'pi' REGEXP 'pi|apa';        -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';       -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';       -> 1
mysql> SELECT 'apa' REGEXP '^(pi|apa)$';   -> 1
mysql> SELECT 'pi' REGEXP '^(pi|apa)$';    -> 1
mysql> SELECT 'pix' REGEXP '^(pi|apa)$';   -> 0
```

- **(abc)***

シークエンス **abc** のゼロ以上のインスタンスをマッチ。

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';       -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';      -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';     -> 1
```

- **{1}, {2,3}**

{n} または **{m,n}** 表記は、パターンの中のアトム (または 「piece」) の発生の多くにマッチする正規表現の、より一般的な書き方を提供します。 **m** および **n** は整数です。

- **a***

a{0,} として書くことができます。

- `a+`

`a{1,}` として書くことができます。

- `a?`

`a{0,1}` として書くことができます。

より正確を期するため、`a{n}` は `a` の `n` インスタンスに完全にマッチします。`a{n,}` は `n` が、`a` のより多くのインスタンスにマッチします。`a{m,n}` は `a` の `n` インスタンスを介して `m` に包括的にマッチします。

`m` および `n` は、0 から `RE_DUP_MAX` (デフォルトは 255) の範囲に包括的に含まれなければなりません。`m` および `n` の両方が与えられてる場合は、`m` は、`n` と均等か、それより少なくなければなりません。

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';      -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';      -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';    -> 1
```

- `[a-dX]`, `[^a-dX]`

`a`、`b`、`c`、`d`、または `X` である (`^` が使用されている場合はそれ以外の) 文字とはすべてマッチします。ふたつの文字の間の `-` 文字は、最初の文字からふたつ目の文字までのすべての文字とマッチする範囲を形成します。例えば、`[0-9]` はすべての 10 進数とマッチします。リテラル `]` 文字を含むには、左大括弧 `[` のすぐ後に続ける必要があります。リテラル `-` 文字を含むには、最初または最後に書き込んでください。`[]` 組の内側の、定義された特殊な意味を持たない文字はすべて、それ自体としかマッチしません。

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';        -> 1
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]$';        -> 0
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]+$';       -> 1
mysql> SELECT 'aXbc' REGEXP '[^a-dXYZ]+$';      -> 0
mysql> SELECT 'gheis' REGEXP '[^a-dXYZ]+$';     -> 1
mysql> SELECT 'gheisa' REGEXP '[^a-dXYZ]+$';    -> 0
```

- `[.characters]`

括弧式 (`[` と `]` で書かれたもの) に囲まれた中で、照合要素である文字のシーケンスをマッチします。`characters` は単一の文字、または `newline` のような文字の名称です。文字の名称の完全なリストは、`regexp/cname.h` ファイルに含まれています。

```
mysql> SELECT '~' REGEXP '[[-.]]';              -> 1
mysql> SELECT '~' REGEXP '[[:tilde]]';          -> 1
```

- `[=character_class=]`

括弧式 (`[` と `]` で書かれたもの) に囲まれた中で、`[=character_class=]` は等価クラスを表します。これは、それ自体を含む、同じ照合値を持つすべての文字にマッチします。例えば、`o` および `(+)` が等価クラスのメンバーである場合は、`[o=]`、`[=(+)=]`、そして `[o(+=)]` はすべて同義です。等価クラスを範囲の週末点として使用できない場合があります。

- `[[:character_class:]]`

括弧式 (`[` と `]` で書かれたもの) に囲まれた中で、`[[:character_class:]]` は、そのクラスに属するすべての文字とマッチする文字クラスを表します。次のテーブルは標準のクラス名のリストです。これらの名称は、`ctype(3)` マニュアル ページで定義されている文字クラスを参照しています。特定のロケールが他のクラス名を提供する場合もあります。文字クラスを範囲の週末点として使用できないこともあります。

<code>alnum</code>	英数文字
<code>alpha</code>	アルファベット文字
<code>blank</code>	空白文字
<code>cntrl</code>	制御文字
<code>digit</code>	数字文字
<code>graph</code>	図形文字

<code>lower</code>	小文字アルファベット文字
<code>print</code>	図形またはスペース文字
<code>punct</code>	句読点文字
<code>space</code>	スペース、タブ、改行、および改行復帰
<code>upper</code>	大文字アルファベット文字
<code>xdigit</code>	16 進数文字

```
mysql> SELECT 'justalnums' REGEXP '[:alnum:]+';    -> 1
mysql> SELECT '!' REGEXP '[:alnum:]+';            -> 0
```

- `[:<:]`, `[:>:]`

これらのマーカは語境界を参考にしています。これらは語の最初と最後それぞれにマッチします。単語とはその前後に別の単語文字が存在しない、単語文字のシーケンスと定義されています。単語文字とは、`alnum` クラス、またはアンダースコア (`_`) の英数文字のことです。

```
mysql> SELECT 'a word a' REGEXP '[:<:]word[:>:]'; -> 1
mysql> SELECT 'a xword a' REGEXP '[:<:]word[:>:]'; -> 0
```

正規表現の特殊文字のリテラル インスタンスを使用するには、ふたつのバックスラッシュ (`\`) 文字を頭につけます。MySQL パーサはふたつのバックスラッシュのうちのひとつを解釈し、正規表現ライブラリがもう一方を解釈します。例えば、特殊 `+` 文字を含むストリング `1+2` とマッチするには、以下の正規表現のうち、最後のものだけが正解になります：

```
mysql> SELECT '1+2' REGEXP '1+2';                -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';               -> 0
mysql> SELECT '1+2' REGEXP '1\\\\+2';              -> 1
```

数字関数

算術演算子

通常の算術演算子を利用することができます。結果の精度は次のルールに従って判断されます：

- `-`、`+`、および `*` の場合、両方の引数が整数であれば、結果は `BIGINT` (64 ビット) の精度で計算されますのでご注意ください。
- 引数のひとつが符号のない整数であり、もう一方の引数も整数である場合は、結果は符号なしの整数になります。
- `+`、`-`、`/`、`*`、`%` オペランドのいずれかが実数またはストリング値であれば、結果の精度は最大精度を持つ引数の精度になります。
- 乗算および除算では、ふたつの高精度値を使用する場合の結果の精度は、最初の引数の精度と、`div_precision_increment` グローバル変数の値を足したものになります。例えば、式 `5.05 / 0.0014` は小数点以下 6 桁の精度 (`3607.142857`) を持ちます。

これらのルールは各演算に適用され、入れ子算は各コンポーネントの精度を示唆します。したがって、`(14620 / 9432456) / (24250 / 9432456)` はまず `(0.0014) / (0.0026)` に解析され、最終的に結果は小数点以下 8 桁 (`0.57692308`) になります。

これらの適用ルールと方法のため、計算のコンポーネントとサブコンポーネントが適切なレベルの精度を用いるよう注意してください。詳細は [キャスト関数と演算子](#) を参照してください。

- `+`

加算：

```
mysql> SELECT 3+5;
      -> 8
```

- `-`

減算：


```
mysql> SELECT 3-5;
-> -2
```

- -

単項マイナス。この演算子は引数の符号を変更します。

```
mysql> SELECT - 2;
-> -2
```

注記 : この演算子が **BIGINT** と使用される場合は、戻り値も **BIGINT** になります。そのため、 -2^{63} の値を持つ可能性のある整数に **-** を使用するのとは避けてください。

- *

乗算 :

```
mysql> SELECT 3*5;
-> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
-> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
-> 0
```

整数の乗算の結果は **BIGINT** 計算の 64 ビット範囲を越えるため、最後の式の結果は正しくありません。([数値タイプ参照](#))

- /

除算 :

```
mysql> SELECT 3/5;
-> 0.60
```

ゼロによる除算は **NULL** の結果を生成します :

```
mysql> SELECT 102/(1-1);
-> NULL
```

結果が整数に返還される状況下では、除算は **BIGINT** 算術でのみ計算されます。

- DIV

整数除算。**FLOOR()** に類似していますが、**BIGINT** 値でも安全です。

```
mysql> SELECT 5 DIV 2;
-> 2
```

- N % M

モジュロ演算。**M** によって除算された **N** の余りを返します。詳細は、[数学関数](#) の **MOD()** に関する説明をご覧ください。

数学関数

すべての数学関数は、エラーのイベントで **NULL** を返します。

- **ABS(X)**

X の絶対値を返します。

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

この関数は、**BIGINT** 値とも安全に使用できます。

- **ACOS(X)**

X のアーク コサインを戻します。これは、コサインが X であるものの値です。 X が -1 から 1 の範囲にない場合は **NULL** を戻します。

```
mysql> SELECT ACOS(1);
-> 0
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.5707963267949
```

- **ASIN(X)**

X のアーク サインを戻します。これは、サインが X であるものの値です。 X が -1 から 1 の範囲にない場合は **NULL** を戻します。

```
mysql> SELECT ASIN(0.2);
-> 0.20135792079033
mysql> SELECT ASIN('foo');

+-----+
| ASIN('foo') |
+-----+
| 0 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+-----+-----+-----+
```

- **ATAN(X)**

X のアーク タンジェントを戻します。これは、タンジェントが X であるものの値です。

```
mysql> SELECT ATAN(2);
-> 1.1071487177941
mysql> SELECT ATAN(-2);
-> -1.1071487177941
```

- **ATAN(Y,X), ATAN2(Y,X)**

ふたつの変数 X および Y のアーク タンジェントを戻します。これは、両方の引数の符号が結果の象限の判定に使用される以外は、 Y/X のアーク タンジェントの計算に類似しています。

```
mysql> SELECT ATAN(-2,2);
-> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
-> 1.5707963267949
```

- **CEILING(X), CEIL(X)**

X より大きな整数値のうち、最小のものを戻します。

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEIL(-1.23);
-> -1
```

これらのふたつの関数は同義です。戻り値は **BIGINT** に変換されますのでご注意ください。

- **COS(X)**

X のコサインを戻します。 X はラジアンで与えられています。

```
mysql> SELECT COS(PI());
-> -1
```

- **COT(X)**

X のコタンジェントを返します。

```
mysql> SELECT COT(12);
-> -1.5726734063977
mysql> SELECT COT(0);
-> NULL
```

- **CRC32(expr)**

巡回符号検査値を算定し、32 ビットの符号のない値を返します。引数が **NULL** である場合、結果は **NULL** になります。引数は文字列になると想定され、そして文字列でない場合でも、(可能であれば)文字列として扱われます。

```
mysql> SELECT CRC32('MySQL');
-> 3259397556
mysql> SELECT CRC32('mysql');
-> 2501908538
```

- **DEGREES(X)**

ラジアンからディグリーに変換された引数 X を返します。

```
mysql> SELECT DEGREES(PI());
-> 180
mysql> SELECT DEGREES(PI() / 2);
-> 90
```

- **EXP(X)**

e (自然対数の底) の X 乗の値を返します。

```
mysql> SELECT EXP(2);
-> 7.3890560989307
mysql> SELECT EXP(-2);
-> 0.13533528323661
mysql> SELECT EXP(0);
-> 1
```

- **FLOOR(X)**

X より小さな整数値のうち、最大のものを返します。

```
mysql> SELECT FLOOR(1.23);
-> 1
mysql> SELECT FLOOR(-1.23);
-> -2
```

戻り値は **BIGINT** に変換されますのでご注意ください。

- **FORMAT(X,D)**

数字 X を '#,###,###.##' のようにフォーマットし、 D 少数位まで丸め、その結果を文字列として返します。詳細は、[文字列関数](#) をご覧ください。

- **LN(X)**

X の自然対数を返します。これは、 X の底 e の対数です。

```
mysql> SELECT LN(2);
-> 0.69314718055995
mysql> SELECT LN(-2);
-> NULL
```

この関数は **LOG(X)** と同義です。

- **LOG(X), LOG(B,X)**

ひとつのパラメータで呼び出される場合、この関数は X の自然対数を返します。

```
mysql> SELECT LOG(2);
-> 0.69314718055995
mysql> SELECT LOG(-2);
-> NULL
```

ふたつのパラメータで呼び出される場合、この関数は任意のベース B に対して X の自然対数を返します。

```
mysql> SELECT LOG(2,65536);
-> 16
mysql> SELECT LOG(10,100);
-> 2
```

$\text{LOG}(B,X)$ は $\text{LOG}(X) / \text{LOG}(B)$ に等価です。

- $\text{LOG2}(X)$

X のベース 2 の対数を返します。

```
mysql> SELECT LOG2(65536);
-> 16
mysql> SELECT LOG2(-100);
-> NULL
```

$\text{LOG2}()$ は、保存のために数字が何ビットを必要とするか調べるのに便利です。この関数は式 $\text{LOG}(X) / \text{LOG}(2)$ と同義です。

- $\text{LOG10}(X)$

X のベース 10 の対数を返します。

```
mysql> SELECT LOG10(2);
-> 0.30102999566398
mysql> SELECT LOG10(100);
-> 2
mysql> SELECT LOG10(-100);
-> NULL
```

$\text{LOG10}(X)$ は $\text{LOG}(10,X)$ と等価です。

- $\text{MOD}(N,M)$, $N \% M$, $N \text{ MOD } M$

モジュロ演算。 M によって除算された N の余りを返します。

```
mysql> SELECT MOD(234, 10);
-> 4
mysql> SELECT 253 \% 7;
-> 1
mysql> SELECT MOD(29,9);
-> 2
mysql> SELECT 29 MOD 9;
-> 2
```

この関数は、**BIGINT** 値とも安全に使用できます。

$\text{MOD}()$ はまた、小数部を持つ値にも利用でき、除算の後に正確な余りを返します。

```
mysql> SELECT MOD(34.5,3);
-> 1.5
```

$\text{MOD}(N,0)$ は **NULL** を返します。

- $\text{PI}()$

π (pi) の値を返します。表示されるデフォルトの少数点以下の桁数は 7 ですが、MySQL は内部的に全倍精度値を使用します。

```
mysql> SELECT PI();
-> 3.141593
mysql> SELECT PI()+0.000000000000000000;
```

```
-> 3.141592653589793116
```

- **POW(X,Y), POWER(X,Y)**

X の Y 乗の値を返します。

```
mysql> SELECT POW(2,2);
-> 4
mysql> SELECT POW(2,-2);
-> 0.25
```

- **RADIANS(X)**

ディグリーからラジアンに変換された引数 X を返します。(π ラジアンは 100 ディグリーと等価です)。

```
mysql> SELECT RADIANS(90);
-> 1.5707963267949
```

- **RAND(), RAND(N)**

$0 \leq v < 1.0$ の範囲にあるランダムな浮動小数点値 v を返します。定数整数引数 N が指定されている場合は、カラム値の反復可能なシーケンスを生成するシード値として使用されます。

```
mysql> SELECT RAND();
-> 0.9233482386203
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND();
-> 0.63553050033332
mysql> SELECT RAND();
-> 0.70100469486881
mysql> SELECT RAND(20);
-> 0.15888261251047
```

定数イニシャライザを使用すれば、シードは実行の前の、ステートメントがコンパイルされる際に一度初期化されます。MySQL 5.1.16 からは、非定数イニシャライザ (カラム名など) が引数として使用される場合は、シードは **RAND()** の各呼び出しの値で初期化されます。(これは、等価の引数値に対しては、**RAND()** は毎回同じ値を返すということを示しています)。MySQL 5.1.3 から 5.1.15 では、非定数引数は許可されていません。それ以前では、非定数引数の使用の効果は未定義になっています。

$i \leq R < j$ の範囲のランダムな整数 R を取得するには、式 $\text{FLOOR}(i + \text{RAND()} * (j - i))$ を使用します。例えば、 $7 \leq R < 12$ の範囲にあるランダムな整数を得るには、次のステートメントを使うことができます：

```
SELECT FLOOR(7 + (RAND() * 5));
```

ORDER BY はカラムを複数回評価するため、**ORDER BY** 句内で **RAND()** 値を持つカラムを使用することはできません。しかし、次のように行を順不同に抽出することは可能です：

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

LIMIT と結合された **ORDER BY RAND()** は、行のセットからランダムなサンプルを選ぶ場合に便利です：

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d -> ORDER BY RAND() LIMIT 1000;
```

WHERE 句内の **RAND()** は、**WHERE** が実行されるたびに再評価されますのでご注意ください。

RAND() は完璧なランダム発生器というわけではありませんが、同じ MySQL バージョンのプラットフォーム間においてポータブルな ad hoc ランダム数を生成する最も速い方法です。

- **ROUND(X), ROUND(X,D)**

引数 X を D 小数点に丸めます。丸めアルゴリズムは X のデータタイプに基づきます。 D は特別に指定されない限り、デフォルトにより 0 になります。 D は時に負数で、値 X の小数点左側の D 桁がゼロになる原因になります。

```
mysql> SELECT ROUND(-1.23);
```

```

-> -1
mysql> SELECT ROUND(-1.58);
-> -2
mysql> SELECT ROUND(1.58);
-> 2
mysql> SELECT ROUND(1.298, 1);
-> 1.3
mysql> SELECT ROUND(1.298, 0);
-> 1
mysql> SELECT ROUND(23.298, -1);
-> 20

```

出力型は最初の引数 (整数、重複、または 10 進数と想定) と同じタイプです。つまり、整数引数では、結果は整数 (小数点なし) になるということになります。

ROUND() は、最初の引数が 10 進値である時、高精度値引数に対して精度算数ライブラリを使用します：

- 高精度値数に対して、**ROUND()** は「四捨五入」ルールを行使します：.5 以上の小数部を持つ値は、正数である場合は次の整数に切り上げられ、負数である場合は切り下げられます。(つまりゼロから切り遠ざけられる)。0.5 未満の小数部を持つ値は、正数である場合は次の整数に切り下げられ、負数である場合は切り上げられます。
- 近似数値では、結果は C ライブラリによります。多くのシステムはで、これはつまり **ROUND()** は " 最も近い偶数に丸める " ルールを使用しているということになります：なんらかの小数部を持つ値は最も近い偶数の整数に丸められます。

次の例は高精度値と近似値で、丸め方がどう異なるかを示しています：

```

mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3          | 2            |
+-----+-----+

```

詳細は [22章精密計算](#) をご覧ください。

• SIGN(X)

X が負数か、ゼロか、または正数かによって、引数の符号を **-1**、**0**、もしくは **1** として戻します。

```

mysql> SELECT SIGN(-32);
-> -1
mysql> SELECT SIGN(0);
-> 0
mysql> SELECT SIGN(234);
-> 1

```

• SIN(X)

X のサインを戻します。**X** はラジアンで与えられています。

```

mysql> SELECT SIN(PI());
-> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
-> 0

```

• SQRT(X)

非負数 **X** の平方根を戻します。

```

mysql> SELECT SQRT(4);
-> 2
mysql> SELECT SQRT(20);
-> 4.4721359549996
mysql> SELECT SQRT(-16);
-> NULL

```

• TAN(X)

X のタンジェントを戻します。 X はラジアンで与えられています。

```
mysql> SELECT TAN(PI());
-> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
-> 1.5574077246549
```

- **TRUNCATE(X,D)**

D 小数点を切り捨てて、数字 X を戻します。 D が 0 の場合、結果は小数点または小数部を持ちません。 D は時に負数で、値 X の小数点左側の D 桁がゼロになる原因になる場合があります。

```
mysql> SELECT TRUNCATE(1.223,1);
-> 1.2
mysql> SELECT TRUNCATE(1.999,1);
-> 1.9
mysql> SELECT TRUNCATE(1.999,0);
-> 1
mysql> SELECT TRUNCATE(-1.999,1);
-> -1.9
mysql> SELECT TRUNCATE(122,-2);
-> 100
mysql> SELECT TRUNCATE(10.28*100,0);
-> 1028
```

すべての数字はゼロに向かって丸められます。

日付時刻関数

このセクションでは、時間値の処理に使用できる関数について説明します。各日付日時タイプが持つ値の範囲の説明と、値が指定されている場合の有効なフォーマットの説明は [日付と時刻タイプ](#) をご覧ください。

日付関数の使用例です。次のクエリはすべての行を、過去 30 日以内の `date_col` で選択します：

```
mysql> SELECT something FROM tbl_name
-> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

またこのクエリは、将来欺く日付で行を選択しますのでご注意ください。

日付値を受け入れる関数は通常、日付時刻値を受け入れ、時刻の部分を無視します。そして時刻値を受け入れる関数は通常、日付時刻値を受け入れ、日付の部分を無視します。

現在の日付または時刻をそれぞれ戻す関数は、クエリ実行の開始時点で、各クエリにつき一度だけ評価されます。つまり、単一クエリ内での、[NOW\(\)](#) などの関数の複数の参照は、常に同じ結果を生成します（我々の目的に関しては、単一クエリはストアド ルーチンまたはトリガ、およびそのルーチン/トリガによって呼び出されたサブルーチンへの呼び出しも含みます）。またこの法則は、[CURDATE\(\)](#)、[CURTIME\(\)](#)、[UTC_DATE\(\)](#)、[UTC_TIME\(\)](#)、[UTC_TIMESTAMP\(\)](#)、およびそれらのシノニムにも適合します。

[CURRENT_TIMESTAMP\(\)](#)、[CURRENT_TIME\(\)](#)、[CURRENT_DATE\(\)](#)、そして [FROM_UNIXTIME\(\)](#) 関数は、`time_zone` 接続の現行時間帯での値を戻し、それらはシステム環境変数の値として利用できます。また、[UNIX_TIMESTAMP\(\)](#) は、その引数が現行時間帯での日付時刻値であると想定します。詳細は [MySQL サーバのタイムゾーン サポート](#) を参照してください。

日付関数のあるものは、その他とは異なり、「zero」日付、または '2001-11-00' のような不完全な日付とも使用できます。日付の一部を抽出する関数は通常、不完全な日付でも問題ありません。例：

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
-> 0, 0
```

他の関数は完全な日付を必要とし、日付が不完全な場合は `NULL` を戻します。これらには日付演算を行う関数、または日付の一部をマップし名前にする関数が含まれます。例：

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
-> NULL
mysql> SELECT DAYNAME('2006-05-00');
-> NULL
```

- [ADDDATE\(date,INTERVAL expr unit\)](#), [ADDDATE\(expr,days\)](#)

2 番目の引数の [INTERVAL](#) フォームで呼び出される際、[ADDDATE\(\)](#) は [DATE_ADD\(\)](#) のシノニムになります。関連する関数 [SUBDATE\(\)](#) は [DATE_SUB\(\)](#) のシノニムです。[INTERVAL unit](#) 引数の詳細については、[DATE_ADD\(\)](#) のディスカッションをご覧ください。

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
```

2 番目の引数の [days](#) フォームで呼び出される場合、MySQL はそれを [expr](#) に加えるために、整数の日数として扱います。

```
mysql> SELECT ADDDATE('1998-01-02', 31);
-> '1998-02-02'
```

- [ADDTIME\(expr1,expr2\)](#)

[ADDTIME\(\)](#) は、[expr2](#) を [expr1](#) に加え、その結果を戻します。[expr1](#) は時刻式、または日付時刻式で、[expr2](#) は時刻式です。

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999',
-> '1 1:1:1.000002');
-> '1998-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
-> '03:00:01.999997'
```

- [CONVERT_TZ\(dt,from_tz,to_tz\)](#)

[CONVERT_TZ\(\)](#) は、日付時刻値 [dt](#) を、[from_tz](#) が指定する時間帯から、[to_tz](#) が指定する時間帯に変換し、結果の値を戻します。時間帯は、[MySQL サーバのタイムゾーンサポート](#) で説明されているように指定されています。引数が無効な場合、この関数は [NULL](#) を戻します。

値が、[from_tz](#) から UTC に変換される際に [TIMESTAMP](#) でサポートされている範囲から外れた場合、変換は行われません。[TIMESTAMP](#) の範囲は [データと時刻タイプの概要](#) に記載されています。

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
-> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
-> '2004-01-01 22:00:00'
```

注記 ['MET'](#) または ['Europe/Moscow'](#) などの、名前付きの時間帯を使用するには、時間帯テーブルが正確に設定されている必要があります。手順については [MySQL サーバのタイムゾーンサポート](#) をご覧ください。

他のテーブルが [LOCK TABLES](#) でロックされている間に [CONVERT_TZ\(\)](#) を使用したい場合は、[mysql.time_zone_name](#) テーブルもロックする必要があります。

- [CURDATE\(\)](#)

関数がストリングで使用されているか、もしくは数値コンテキストで使用されているかによって、現在の日付を ['YYYY-MM-DD'](#) または [YYYYMMDD](#) フォーマットの値で戻します。

```
mysql> SELECT CURDATE();
-> '1997-12-15'
mysql> SELECT CURDATE() + 0;
-> 19971215
```

- [CURRENT_DATE](#), [CURRENT_DATE\(\)](#)

[CURRENT_DATE](#) および [CURRENT_DATE\(\)](#) は [CURDATE\(\)](#) のシノニムです。

- [CURTIME\(\)](#)

関数がストリングで使用されているか、もしくは数値コンテキストで使用されているかによって、現在の時刻を ['HH:MM:SS'](#) または [HHMMSS](#) フォーマットの値で戻します。値は現在の時間帯で表現されています。

```
mysql> SELECT CURTIME();
```

```

-> '23:50:26'
mysql> SELECT CURTIME() + 0;
-> 235026

```

- `CURRENT_TIME`, `CURRENT_TIME()`

`CURRENT_TIME` および `CURRENT_TIME()` は `CURTIME()` のシノニムです。

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP()`

`CURRENT_TIMESTAMP` および `CURRENT_TIMESTAMP()` は `NOW()` のシノニムです。

- `DATE(expr)`

日付、または日付時刻式 `expr` の日付部分を抽出します。

```

mysql> SELECT DATE('2003-12-31 01:02:03');
-> '2003-12-31'

```

- `DATEDIFF(expr1,expr2)`

`DATEDIFF()` は、ひとつの日付から他の日付への日数の値として表現された `expr1 - expr2` を戻します。`expr1` および `expr2` は日付または日付と時刻の表現です。値の日付部分のみが計算に使用されます。

```

mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
-> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
-> -31

```

- `DATE_ADD(date,INTERVAL expr unit)`, `DATE_SUB(date,INTERVAL expr unit)`

これらの関数は日付演算を行います。`date` は、開始日を指定する `DATETIME` または `DATE` 値です。`expr` は開始日に追加、または開始日から引かれる区間値を指定する表現です。`expr` はストリングで、負のインターバルの場合は '-' で始まることがあります。`unit` は、表現が解釈されるべきユニットを示すキーワードです。

`INTERVAL` キーワードおよび `unit` 指定子は、大文字小文字の区別をしません。

次の表は、各 `unit` 値に対して予想される `expr` 引数のフォームを示したものです。

unit 値	予想される <code>expr</code> フォーマット
<code>MICROSECOND</code>	<code>MICROSECONDS</code>
<code>SECOND</code>	<code>SECONDS</code>
<code>MINUTE</code>	<code>MINUTES</code>
<code>HOUR</code>	<code>HOURS</code>
<code>DAY</code>	<code>DAYS</code>
<code>WEEK</code>	<code>WEEKS</code>
<code>MONTH</code>	<code>MONTHS</code>
<code>QUARTER</code>	<code>QUARTERS</code>
<code>YEAR</code>	<code>YEARS</code>
<code>SECOND_MICROSECOND</code>	<code>'SECONDS.MICROSECONDS'</code>
<code>MINUTE_MICROSECOND</code>	<code>'MINUTES.MICROSECONDS'</code>
<code>MINUTE_SECOND</code>	<code>'MINUTES:SECONDS'</code>
<code>HOUR_MICROSECOND</code>	<code>'HOURS.MICROSECONDS'</code>
<code>HOUR_SECOND</code>	<code>'HOURS:MINUTES:SECONDS'</code>
<code>HOUR_MINUTE</code>	<code>'HOURS:MINUTES'</code>
<code>DAY_MICROSECOND</code>	<code>'DAYS.MICROSECONDS'</code>
<code>DAY_SECOND</code>	<code>'DAYS HOURS:MINUTES:SECONDS'</code>

DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

MySQL は、`expr` フォーマットにおいてはいかなる句読区切り記号の使用も許容します。上記の表の区切り記号は提案にすぎません。`date` 引数が `DATE` 値であり、行う計算が `YEAR`、`MONTH`、および `DAY` 部のみ (時刻部分はなし) を含む場合は、結果は `DATE` 値になります。他の場合は、結果は `DATETIME` 値になります。

また、日付演算は、`INTERVAL` を `+` または `-` 演算子と共に使用しても行うことができます：

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

`INTERVAL expr unit` は、一方の表現が日付か日付時刻値であれば、どちら側の `+` 演算子でも使用できます。`-` 演算子に関しては、`INTERVAL expr unit` は、インターバルから日付や日付日時値を抽出しても意味がないため、右側でのみ使用できます。

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
-> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
-> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
-> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 SECOND);
-> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 DAY);
-> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
-> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
-> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
-> '1997-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND);
-> '1993-01-01 00:00:01.000001'
```

短すぎる区間値を指定した場合 (`unit` キーワードから予想されるすべての区間部分は含みません)、MySQL は区間値の一番左の部分が放置されているものと想定します。例えば、`DAY_SECOND` の `unit` を指定した場合、`expr` の値は日にち、時間、分、秒の部分を持つものと想定されます。`'1:10'` のような値を指定すると、MySQL は日にちと時間の部分が抜けており、値は分と秒を示しているものと想定します。つまり、`'1:10' DAY_SECOND` は、`'1:10' MINUTE_SECOND` と同等に解釈されます。これは、MySQL が `TIME` 値を、時刻ではなく経過時間として解釈するやり方に相似しています。

時間部分を含むなにかを日付値に追加、または日付値から抽出する場合、結果は自動的に日付時刻値に変換されます：

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
-> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
-> '1999-01-01 01:00:00'
```

`MONTH`、`YEAR_MONTH`、または `YEAR` を加え、結果の日付が新しい月の最大日数より大きな日を持つ場合、その日は新しい月の最大日数に調整されます。

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
-> '1998-02-28'
```

日付算術演算には完全な日付が必須であり、`'2006-07-00'` のような不完全な日付や、誤った形の日付では正常に作動しません：

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
-> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
-> NULL
```

- **DATE_FORMAT(date,format)**

date 値を **format** ストリングに基づいてフォーマットします。

次の指定子は **format** ストリングで使用されていることもあります。**'%'** 文字は、書式指定子の前に必要なものです。

指定子	解説
%a	簡略曜日名 (Sun..Sat)
%b	簡略月名 (Jan..Dec)
%c	月、数字 (0..12)
%D	英語の接尾辞を持つ日にち (0th, 1st, 2nd, 3rd, ...)
%d	日にち、数字 (00..31)
%e	日にち、数字 (0..31)
%f	マイクロ秒 (000000..999999)
%H	時間 (00..23)
%h	時間 (01..12)
%I	時間 (01..12)
%i	分、数字 (00..59)
%j	通日 (001..366)
%k	時間 (0..23)
%l	時間 (1..12)
%M	月名 (January..December)
%m	月、数字 (00..12)
%p	AM または PM
%r	時間、12 時間単位 (hh:mm:ss に AM または PM が続く)
%S	秒 (00..59)
%s	秒 (00..59)
%T	時間、24 時間単位 (hh:mm:ss)
%U	週 (00..53)、週の開始は日曜日
%u	週 (00..53)、週の開始は月曜日
%V	週 (01..53)、週の開始は日曜日で、%X と使用
%v	週 (01..53)、週の開始は月曜日で、%x と使用
%W	曜日名 (Sunday..Saturday)
%w	曜日 (0=Sunday..6=Saturday)
%X	年間の週、週の始まりは日曜日、週、数字、4 桁 ; %V と使用
%x	年間の週、週の始まりは月曜日、数字、4 桁、%v と使用
%Y	年、数字、4 桁
%y	年、数字 (2 桁)
%%	リテラル '%' 文字
%x	x、上記にないすべての 'x'

MySQL は '2004-00-00' のような不完全な日付の格納を許可するため、月と日にちの指定子の範囲は 0 から始まります。

MySQL 5.1.12 から、日にちおよび月の名称に使用される言語と、省略後は、`lc_time_names` システム環境変数 ([MySQL サーバのローケル サポート](#)) の値によって管理されます。

MySQL 5.1.15 からは、`DATE_FORMAT()` は文字セットを持つストリングと、`character_set_connection` および `collation_connection` によって提示された照合を戻し、非 ASCII 文字を含む月と曜日の名前を戻せるようになりました。5.1.15 の前は、戻り値はバイナリ ストリングでした。

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
        '%D %y %a %d %m %b %j');
-> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
        '%H %k %l %r %T %S %w');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
-> '00'
```

- `DATE_SUB(date,INTERVAL expr unit)`

`DATE_ADD()` 参照。

- `DAY(date)`

`DAY()` は `DAYOFMONTH()` のシノニムです。

- `DAYNAME(date)`

`date` に対して曜日の名前を戻します。MySQL 5.1.12 からは、名前に使用される言語は、`lc_time_names` システム環境変数 ([MySQL サーバのローケル サポート](#)) の値によって管理されます。

```
mysql> SELECT DAYNAME('1998-02-05');
-> 'Thursday'
```

- `DAYOFMONTH(date)`

0 から 31 の範囲内の日にちを、`date` に対して戻します。

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

- `DAYOFWEEK(date)`

`date` (1 = Sunday 、 2 = Monday 、 ... 、 7 = Saturday) に対する曜日のインデックスを戻します。これらのインデックス値は、ODBC 標準に対応しています。

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
```

- `DAYOFYEAR(date)`

1 から 366 の範囲内の通日を、`date` に対して戻します。

```
mysql> SELECT DAYOFYEAR('1998-02-03');
-> 34
```

- `EXTRACT(unit FROM date)`

`EXTRACT()` 関数は、`DATE_ADD()` または `DATE_SUB()` と同様の装置指定子を使用しますが、データ演算を行うのではなく、データから一部を抽出します。


```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND
-> FROM '2003-01-02 10:30:00.000123');
-> 123
```

- **FROM_DAYS(N)**

日数 **N** を得て、**DATE** 値を戻します。

```
mysql> SELECT FROM_DAYS(729669);
-> '1997-10-07'
```

FROM_DAYS() を古い日付で注意深く使用します。グレゴリオ暦 (1582) の出現を優先する値と共に使用することが目的ではありません。詳細は [MySQL が使用するカレンダーは？](#) を参照してください。

- **FROM_UNIXTIME(unix_timestamp)**, **FROM_UNIXTIME(unix_timestamp,format)**

unix_timestamp 引数の表現を、関数がストリングで使用されたか、または数字のコンテキストで使用されたかによって、**'YYYY-MM-DD HH:MM:SS'** または **YYYYMMDDHHMMSS** のフォーマットで値として戻します。値は現在の時間帯で表現されます。**unix_timestamp** は、**UNIX_TIMESTAMP()** 関数によって生成されるような内部タイムスタンプ値です。

format が与えられていれば、**DATE_FORMAT()** 関数のエントリで挙げられているのと同じ方法で使用する **format** ストリングに基づいて、結果はフォーマットされます。

```
mysql> SELECT FROM_UNIXTIME(875996580);
-> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
-> 19971004222300
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
-> '%Y %D %M %h:%i:%s %x');
-> '2003 6th August 06:22:58 2003'
```

注記 **UNIX_TIMESTAMP()** および **FROM_UNIXTIME()** を使って **TIMESTAMP** 値と Unix タイムスタンプ値間を変換する場合、マッピングは双方向に対して 1 対 1 ではないので、変換は高損失になります。詳細は **UNIX_TIMESTAMP()** 関数の説明をご覧ください。

- **GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')**

フォーマット ストリングを戻します。この関数は、**DATE_FORMAT()** と **STR_TO_DATE()** 関数の組み合わせで使用する と便利です。

1 番目と 2 番目の引数に対する有効な値は、複数の可能なフォーマット ストリングで結果が生じます (使用される指定子については、**DATE_FORMAT()** 関数の説明にあるテーブルをご覧ください)。ISO フォーマットは ISO 8601 ではなく、ISO 9075 を参照しています。

関数呼び出し	結果
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'

GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

TIMESTAMP は、**GET_FORMAT()** への最初の引数としても使用でき、その場合、関数は **DATETIME** に対してと同じ値を返します。

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
-> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
-> '2003-10-31'
```

- **HOUR(time)**

time の正時 (hour) を返します。戻り値の範囲は、時刻値の 0 から 23 です。ただし、**TIME** 値の範囲は実際にはもっと大きいため、**HOUR** は 23 以上の値を返すことができます。

```
mysql> SELECT HOUR('10:05:03');
-> 10
mysql> SELECT HOUR('272:59:59');
-> 272
```

- **LAST_DAY(date)**

日付または日付時刻を取り、月の最後の日の換算値を返します。引数が無効である場合は **NULL** を返します。

```
mysql> SELECT LAST_DAY('2003-02-05');
-> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
-> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
-> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
-> NULL
```

- **LOCALTIME, LOCALTIME()**

LOCALTIME および **LOCALTIME()** は **NOW()** のシノニムです。

- **LOCALTIMESTAMP, LOCALTIMESTAMP()**

LOCALTIMESTAMP および **LOCALTIMESTAMP()** は **NOW()** のシノニムです。

- **MAKEDATE(year,dayofyear)**

日付、提示された年、そして通日の値を返します。**dayofyear** は 0 より大きくなければならず、さもなければ結果は **NULL** になります。

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
-> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
-> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
-> NULL
```

- **MAKETIME(hour,minute,second)**

hour、**minute**、および **second** 引数から計算された時間値を返します。

```
mysql> SELECT MAKETIME(12,15,30);
-> '12:15:30'
```

- **MICROSECOND(expr)**

時間または日付時刻式 `expr` からのマイクロ秒を、0 から 999999 までの範囲の数値として戻します。

```
mysql> SELECT MICROSECOND('12:00:00.123456');
-> 123456
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');
-> 10
```

- **MINUTE(time)**

0 から 59 の範囲内で、`time` の分数を戻します。

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
-> 5
```

- **MONTH(date)**

0 から 12 の範囲内で、`date` の月を戻します。

```
mysql> SELECT MONTH('1998-02-03');
-> 2
```

- **MONTHNAME(date)**

`date` の月の完全名を戻します。MySQL 5.1.12 からは、名前に使用される言語は、`lc_time_names` システム環境変数 ([MySQL サーバのローケル サポート](#)) の値によって管理されます。

```
mysql> SELECT MONTHNAME('1998-02-05');
-> 'February'
```

- **NOW()**

関数がストリングで使用されているか、もしくは数値コンテキストで使用されているかによって、現在の日付を 'YYYY-MM-DD HH:MM:SS' または YYYYMMDDHHMMSS フォーマットの値で戻します。値は現在の時間帯で表現されています。

```
mysql> SELECT NOW();
-> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 19971215235026
```

NOW() は、ステートメントが実行を開始する時間を示す定数時間を戻します。(ストアド ルーチンまたはトリガ内で、**NOW()** はルーチンまたはトリガ文が実行を開始する時間を戻します。)これは、正確な実行時間を戻す **SYSDATE()** の動作によって異なります。

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 | 0        | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 | 0        | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

ふたつの関数の違いに関する詳細は、**SYSDATE()** の説明をご覧ください。

- **PERIOD_ADD(P,N)**

N 月を、期間 P に加えます (フォーマットは YYYYMM または YYYYMM)。フォーマット YYYYMM で値を戻します。期間引数 P は日付値ではありません のご注意ください。

```
mysql> SELECT PERIOD_ADD(9801,2);
-> 199803
```

- **PERIOD_DIFF(P1,P2)**

期間 **P1** と **P2** 間の月の数を返します。**P1** および **P2** は、**YYMM** または **YYYYMM** のフォーマットになります。期間引数 **P1** および **P2** は日付値ではありませんのでご注意ください。

```
mysql> SELECT PERIOD_DIFF(9802,199703);
-> 11
```

- **QUARTER(date)**

date の四半期を 1 から 4 の範囲内で返します。

```
mysql> SELECT QUARTER('98-04-01');
-> 2
```

- **SECOND(time)**

0 から 59 の範囲内で、**time** の秒数を返します。

```
mysql> SELECT SECOND('10:05:03');
-> 3
```

- **SEC_TO_TIME(seconds)**

関数がストリングで使用されているか、もしくは数値コンテキストで使用されているかによって、正時、分、秒に変換された **seconds** 引数を、'**HH:MM:SS**' または **HHMMSS** のフォーマットの値で返します。

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

- **STR_TO_DATE(str,format)**

これは **DATE_FORMAT()** 関数の反転です。ストリング **str** と フォーマット ストリング **format** を受取ります。**STR_TO_DATE()** は、フォーマット ストリングが日付と時間の両方の部分を含む場合は **DATETIME** 値を返し、ストリングが日付または時間の部分の一方のみを含む場合は **DATE** もしくは **TIME** 値を返します。

str に含まれる日付、時刻、または日付時刻値は、**format** で示されるフォーマットで提供してください。**format** で使用できる指定子については、**DATE_FORMAT()** 関数の説明を参照してください。**str** が不当な日付、時刻、または日付時刻値を含む場合は、**STR_TO_DATE()** は **NULL** を返します。また、不当な値は警告を生成します。

日付値の部分を確認する範囲は、**DATETIME**、**DATE**、そして **TIMESTAMP タイプ** で説明されている通りです。つまり、例えば、「zero」日付、または部分の値が 0 の日付は、SQL モードが特にそれらを禁止する設定になっていない限り、使用が許可されます。

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
-> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
-> '2004-04-31'
```

注記 :年と週のコンビネーションは、週が月の境界を越えた場合、年と月を一意的に識別できないため、フォーマット **"%X%V"** を使用して、年 - 週ストリングを日付に変換することはできません。年 - 週を日付に変換するには、曜日も同じく指定する必要があります :

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
-> '2004-10-18'
```

- **SUBDATE(date,INTERVAL expr unit), SUBDATE(expr,days)**

2 番目の引数の **INTERVAL** フォームで呼び出される際、**SUBDATE()** は **DATE_SUB()** のシノニムになります。**INTERVAL unit** 引数の詳細については、**DATE_ADD()** のディスカッションをご覧ください。

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
```

2 番目のフォームは、`days` に整数値を使用することを許可します。そのような場合は、日付または日付時刻式 `expr` から日数が減算されると解釈されます。

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
-> '1997-12-02 12:00:00'
```

- `SUBTIME(expr1,expr2)`

`SUBTIME()` は、`expr1` と同じフォーマットで値として表現された `expr1 - expr2` を戻します。`expr1` は時刻または日付時刻式であり、`expr2` 時刻表現です。

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
-> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
-> '-00:59:59.999999'
```

- `SYSDATE()`

関数がストリングで使用されているか、もしくは数値コンテキストで使用されているかによって、現在の日付を 'YYYY-MM-DD HH:MM:SS' または 'YYYYMMDDHHMMSS' フォーマットの値で戻します。

`SYSDATE()` は、それが実行された時間を戻します。これは `NOW()` の動作によって異なり、ステートメントが実行を開始する時間を示す定数時間を戻します。(ストアド ルーチンまたはトリガ内で、`NOW()` はルーチンまたはトリガ文が実行を開始する時間を戻します。)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 | 0        | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 | 0        | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

そのほか、`SET TIMESTAMP` 文は `NOW()` によって戻された値に影響を及ぼしますが、`SYSDATE()` によって戻された値には影響しません。つまり、バイナリ ログのタイムスタンプ設定は、`SYSDATE()` の呼び出しには効果をもたらさないということになります。

`SYSDATE()` は同じステートメントの中でも、異なる値を戻すことができ、また `SET TIMESTAMP` に影響を受けないため、これは非決定性であり、従ってステートメントに基づくバイナリ ロギングが使用されている場合、複製は安全でないということになります。これが問題になる場合は、行ベースのロギングを使用するか、または `--sysdate-is-now` オプションでサーバを起動して、`SYSDATE()` が `NOW()` のエイリアスになるようにしてください。

- `TIME(expr)`

時刻、または日付時刻式 `expr` の時刻部分を抽出し、ストリングとして戻します。

```
mysql> SELECT TIME('2003-12-31 01:02:03');
-> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
-> '01:02:03.000123'
```

- `TIMEDIFF(expr1,expr2)`

`TIMEDIFF()` は時刻値として表現された `expr1 - expr2` を戻します。`expr1` および `expr2` は時刻、または日付時刻式ですが、双方とも同じタイプであることが重要です。

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
-> '2000:01:01 00:00:00.000001');
-> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
-> '1997-12-30 01:01:01.000002');
-> '46:58:57.999999'
```

- `TIMESTAMP(expr)`, `TIMESTAMP(expr1,expr2)`

単一引数では、この関数は日付または日付時刻式 `expr` を日付時刻値として戻します。ふたつの引数では、時刻式 `expr2` を日付、または日付時刻式 `expr1` に加え、結果を日付時刻値として戻します。

```
mysql> SELECT TIMESTAMP('2003-12-31');
-> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
-> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(unit,interval,datetime_expr)`

整数式 `interval` を、日付または日付時刻式 `datetime_expr` に加えます。`interval` のユニットは、次の値のひとつである `unit` 引数によって提示されます: `FRAC_SECOND`、`SECOND`、`MINUTE`、`HOURL`、`DAY`、`WEEK`、`MONTH`、`QUARTER`、または `YEAR`。

`unit` 値は、記載されているキーワードのどれかを使用するか、または `SQL_TSI_` のプリフィックスでの指定が可能です。例えば、`DAY` と `SQL_TSI_DAY` は両方とも正当です。

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'
```

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

日付または日付時刻式 `datetime_expr1` および `datetime_expr2` 間の整数の差を戻します。結果のユニットは、`unit` 引数によって提示されます。`unit` の正当な値は、`TIMESTAMPADD()` 関数の説明で挙げられているものと同じです。

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1
```

- `TIME_FORMAT(time,format)`

これは `DATE_FORMAT()` 関数のように使用されますが、`format` スtring は時間、分、秒のみのための書式指定子を含む場合があります。他の指定子は `NULL` 値か `0` を生成します。

`time` 値が 23 より大きな時間部を含む場合、`%H` および `%k` 時間書式指定子が 0..23 の通常の範囲より大きな値を生成します。他の時間書式指定子は、時間値モジュロ 12 を作成します。

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %l %I');
-> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

秒に変換された `time` 引数を戻します。

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

- `TO_DAYS(date)`

日付 `date` をもって、日数 (0 年からの日数) を戻します。

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('1997-10-07');
-> 729669
```

`TO_DAYS()` は、カレンダーが変更された際に失われた日を考慮しないので、グレゴリオ暦 (1582) の出現を優先される値と使用する目的はありません。1582 より前の日付 (または他のロケールでの後の年) に関しては、この関数からの結果は信頼できません。詳細は [MySQL が使用するカレンダーは？](#) をご覧ください。

MySQL は **日付と時刻タイプ** のルールを使用して、日付の 2 桁の年の値を 4 桁のフォームに変換することに留意してください。例えば、'1997-10-07' と '97-10-07' は同一の日付と考えられます：

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');
-> 729669, 729669
```

- **UNIX_TIMESTAMP()**, **UNIX_TIMESTAMP(date)**

引数なしで呼び出された場合、Unix タイムスタンプ ('1970-01-01 00:00:00' UTC 以来の秒数) を符号なしの整数として戻します。**UNIX_TIMESTAMP()** が **date** 引数で呼び出された場合は、'1970-01-01 00:00:00' UTC 以後の秒として引数の値が戻されます。**date** は、**DATE** スtring、**DATETIME** スtring、**TIMESTAMP**、またはフォーマット **YYMMDD** もしくは **YYYYMMDD** 内のナンバーである場合があります。サーバは **date** を現在の時間帯の値として解釈し、UTC の内部値に変換します。クライアントは、**MySQL サーバのタイムゾーンサポート** で説明されているように、独自の時間帯を設定することができます。

```
mysql> SELECT UNIX_TIMESTAMP();
-> 882226357
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
-> 875996580
```

UNIX_TIMESTAMP が **TIMESTAMP** カラムに使用される際、関数は明示的な「string-to-Unix-timestamp」の変換なしに、内部タイムスタンプ値を直接戻します。**UNIX_TIMESTAMP()** に範囲外の日付を渡すと、**0** が戻されます。

注記 :**UNIX_TIMESTAMP()** および **FROM_UNIXTIME()** を使って **TIMESTAMP** 値と Unix タイムスタンプ値間を変換する場合、マッピングは双方向に対して 1 対 1 ではないので、変換は高損失になります。例えば、現地時間帯の変更に対する変換のため、ふたつの **UNIX_TIMESTAMP()** がふたつの **TIMESTAMP** 値を、同じ Unix タイムスタンプ値にマップすることが考えられます。**FROM_UNIXTIME()** はその値を、本来の **TIMESTAMP** 値のひとつのみにマップをして返します。次が **CET** 時間帯で **TIMESTAMP** 値を使用した例です：

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT FROM_UNIXTIME(1111885200);
+-----+
| FROM_UNIXTIME(1111885200) |
+-----+
| 2005-03-27 03:00:00 |
+-----+
```

UNIX_TIMESTAMP() カラムを減算するには、結果を符号付きの整数にキャストする方法もあります。詳細は **キャスト関数と演算子** を参照してください。

- **UTC_DATE**, **UTC_DATE()**

関数が **String** で使用されているか、もしくは数値コンテキストで使用されているかによって、現在の UTC 日付を 'YYYY-MM-DD' または **YYYYMMDD** フォーマットの値で戻します。

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

- **UTC_TIME**, **UTC_TIME()**

関数が **String** で使用されているか、もしくは数値コンテキストで使用されているかによって、現在の UTC 時刻を 'HH:MM:SS' または **HHMMSS** フォーマットの値で戻します。

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753
```

- [UTC_TIMESTAMP](#), [UTC_TIMESTAMP\(\)](#)

関数がストリングで使われているか、もしくは数値コンテキストで使われているかによって、現在の UTC 日付を 'YYYY-MM-DD HH:MM:SS' または YYYYMMDDHHMMSS フォーマットの値で戻します。

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
-> '2003-08-14 18:08:04', 20030814180804
```

- [WEEK\(date\[,mode\]\)](#)

この関数は [date](#) に週の数を戻します。[WEEK\(\)](#) の、引数がふたつのフォームによって、週が日曜で始まるか、月曜で始まるか、また、戻り値の範囲は 0 から 53 か、1 から 53 かを指定することが可能です。[mode](#) 引数が省略された場合は、[default_week_format](#) システム環境変数の値が使用されます。詳細は [システム変数](#) を参照してください。

次のテーブルは、[mode](#) 引数がどのように作用するかを示したものです。

	開始日		
モード	曜日	範囲	Week 1 は下記の最初の週...
0	日曜日	0-53	この年の日曜日
1	月曜日	0-53	この年は 3 日以上
2	日曜日	1-53	この年は日曜日
3	月曜日	1-53	この年は 3 日以上
4	日曜日	0-53	この年は 3 日以上
5	月曜日	0-53	この年は月曜日
6	日曜日	1-53	この年は 3 日以上
7	月曜日	1-53	この年は月曜日

```
mysql> SELECT WEEK('1998-02-20');
-> 7
mysql> SELECT WEEK('1998-02-20',0);
-> 7
mysql> SELECT WEEK('1998-02-20',1);
-> 8
mysql> SELECT WEEK('1998-12-31',1);
-> 53
```

日付が先年の最後の週に該当する場合、[2](#)、[3](#)、[6](#)、または [7](#) をオプションの [mode](#) 引数として使用しなければ、MySQL は 0 を戻すので注意してください：

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
-> 2000, 0
```

与えられた日付が 1999 年の 52 週目に発生するため、MySQL は [WEEK\(\)](#) 関数に 52 を戻すべきだという意見もあります。しかし当社では、関数が「与えられた年の週の」を戻すべきだと考え、0 を戻しています。これにより、日付から日にち部分を抽出する他の関数と併用する際に、[WEEK\(\)](#) 関数をより信頼して使用できるようになっています。

結果において、与えられた日付の週の最初の日を含む年の評価をしたい場合は、[0](#)、[2](#)、[5](#)、または [7](#) を、オプションの [mode](#) 引数として使用してください。

```
mysql> SELECT WEEK('2000-01-01',2);
-> 52
```

その替わりとして、[YEARWEEK\(\)](#) 関数を使用することもできます：

```
mysql> SELECT YEARWEEK('2000-01-01');
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
-> '52'
```

- [WEEKDAY\(date\)](#)

`date` (0 = Monday 、 1 = Tuesday 、 ... 、 ... 6 = Sunday) の曜日のインデックスを返します。

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
-> 1
mysql> SELECT WEEKDAY('1997-11-05');
-> 2
```

- `WEEKOFYEAR(date)`

1 から 53 の範囲で、日付の暦週を返します。`WEEKOFYEAR()` は `WEEK(date,3)` に等価な互換性の関数です。

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
-> 8
```

- `YEAR(date)`

0 から 9999 の範囲、または「zero」日付には 0 で、`date` の年を返します。

```
mysql> SELECT YEAR('98-02-03');
-> 1998
```

- `YEARWEEK(date)`, `YEARWEEK(date,mode)`

日付の年と週を返します。`mode` 引数は、`WEEK()` への `mode` 引数とまったく同様に作用します。結果の年は、日付引数の年によって、年の最初の週と、最後の週で異なる場合があります。

```
mysql> SELECT YEARWEEK('1987-01-01');
-> 198653
```

週の数、`WEEK()` が提示された年のコンテキストの週を返す場合、`WEEK()` 関数がオプションの引数 0 または 1 に戻すもの (0) によって異なります。

MySQL が使用するカレンダーは？

MySQL は、proleptic Gregorian calendar として知られる暦を使用しています。

ユリウス暦からグレゴリオ暦に改めたすべての国では、その変移の際に少なくとも 10 日の日数を減らさなければなりませんでした。この仕組みを理解するには、初めてユリウス暦からグレゴリオ暦への変更が行われた 1582 年の 10 月を考慮に入れてください：

月曜日	火曜日	水曜日	木曜日	金曜日	土曜日	日曜日
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

10 月 4 日から 10 月 15 日の間には日付がありません。この不連続性を カットオーバー と呼びます。カットオーバーの前がユリウス暦で、カットオーバーに続く日付はすべてグレゴリオ暦です。カットオーバーの途中の日付は存在しません。

まだ実際には使用されていなかった間のカレンダーは proleptic と呼ばれています。従って、最初から常にグレゴリオ暦が使用されており、カットオーバーが起こることもなかったと仮定した暦が proleptic Gregorian calendar ということになります。これが MySQL の使用する暦であり、標準 SQL の必須となっています。このため、MySQL `DATE` または `DATETIME` 値として格納されたカットオーバー前の日付は、その違いを補正する調整が必要です。カットオーバーが起こった時期が国によって異なるのも重要な点で、その時期が後であるほど、失われる日数は多いことになります。例えば、イギリスでは 1752 年にカットオーバーが起こり、9 月 2 日の水曜日の翌日が、9 月 14 日の木曜日でした。ロシアは 1918 年までユリウス暦を使用し、変更の際に 13 日を失いました。世に言う「十月革命」は、グレゴリオ暦では 11 月に起こったものです。

全文検索関数

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])

search_modifier:
{
```

```

    IN BOOLEAN MODE
  | IN NATURAL LANGUAGE MODE
  | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
  | WITH QUERY EXPANSION
}

```

MySQL は全文インデックスおよび検索をサポートします：

- MySQL の全文インデックスは、タイプ **FULLTEXT** のインデックスです。
- 全文インデックスは **MyISAM** テーブルとのみ使用されており、**CHAR**、**VARCHAR**、または **TEXT** カラムのためにだけ作成されます。
- **FULLTEXT** インデックスの定義は、テーブルを作成する時に、**CREATE TABLE** 文で提示することができるほか、**ALTER TABLE** または **CREATE INDEX** を使用して後で付け加えることも可能です。
- 大きなデータセットに関しては、**FULLTEXT** インデックスを持たないテーブルにロードし、その後でインデックスを作成するほうが、すでに **FULLTEXT** インデックスを持つテーブルにロードするよりも断然速く読み込めます。

全文検索は **MATCH() ... AGAINST** シンタックスを用いて行われます。**MATCH()** は、検索用にカラムに名称をつける、カンマで区切られたリストを使用します。**AGAINST** は検索するストリングと、実行する検索のタイプを示すオプションの修飾子を利用します。検索ストリングは、変数やカラム名ではなく、リテラル ストリングでなければなりません。全文検索には 3 種類あります：

- ブール検索は、特別なクエリ言語のルールを使用した検索ストリングを解釈します。ストリングは検索の対象になる言葉を含みます。また、単語は整合行で提示または不提示にされなければならない、もしくは、通常より高く、または低く加重するべき、等の条件を指定する演算子も含むことができます。「some」や「then」のような一般的な単語はストップワードで、検索ストリングにあってもマッチしません。**IN BOOLEAN MODE** 修飾子はブール検索を特定します。詳細は [ブール全文検索](#) をご覧ください。
- 自然言語の検索は、検索ストリングを人間の自然な言語でのフレーズ (フリーテキストのフレーズ) として解釈します。これには特別な演算子はありません。ストップワード リストは適用されます。また、行の 50% 以上にある言葉は常用語と判断され、検出はされません。全文検索は、**IN NATURAL LANGUAGE MODE** 修飾子が与えられている、または修飾子がまったくない場合は、自然言語検索になります。
- クエリ拡張検索は、自然言語検索が改変されたものです。自然言語検索を行うには、検索ストリングが使用されます。そして、検索によって返された最も関連性の強い行からの言葉が検索ストリングに加えられ、再度検索されます。クエリは 2 度目の検索からの行を戻します。**IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION** または **WITH QUERY EXPANSION** 修飾子は、クエリ拡張検索を特定します。詳細は [クエリ拡張を伴う全文検索](#) をご覧ください。

IN NATURAL LANGUAGE MODE および **IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION** 修飾子は、MySQL 5.1.7 から追加されました。

全文検索の制約は、[全文制限](#) に挙げられています。

```

mysql> CREATE TABLE articles (
->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
->   title VARCHAR(200),
->   body TEXT,
->   FULLTEXT (title,body)
-> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
-> ('MySQL Tutorial','DBMS stands for DataBase ...'),
-> ('How To Use MySQL Well','After you went through a ...'),
-> ('Optimizing MySQL','In this tutorial we will show ...'),
-> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
-> ('MySQL vs. YourSQL','In the following database comparison ...'),
-> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+-----+-----+-----+-----+

```

```

| id | title | body |
+----+-----+-----+
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
+----+-----+-----+
2 rows in set (0.00 sec)

```

MATCH() 関数は、テキスト コレクション に対する文字列を自然言語検索します。コレクションは、**FULLTEXT** インデックスを含む、ひとつ以上のカラムのセットです。検索文字列は、**AGAINST()** への引数として与えられます。テーブルの各行に対し、検索文字列と、**MATCH()** リストで名付けられたカラムの行内のテキスト間の類似性を測り、**MATCH()** が関連性のある値を返します。

デフォルトでは、検索は大文字小文字の区別のある方法で行われます。しかし、バイナリ照合を用いて、インデックスのつけられたカラムに対し、大文字小文字の区別のある古テキスト検索を行うことができます。例えば、**latin1** 文字セットを使用するカラムに、全文検索のために大文字小文字の区別をするよう、**latin1_bin** の照合を割り当てることができます。

以前に挙げた例のように、**MATCH()** が **WHERE** 句で使用されるとき、返された行はまず、最高レベルの関連性があるとした上で自動的に保管されます。関連値は、負でない浮動小数点数です。ゼロ レリバンスとは、類似性がまったくないという意味です。レリバンスは、行の単語の数、行の一意性のある単語の数、コレクション内の単語の合計数、そして特定の単語を含む資料 (行) の数に基づいて計算されます。

単に検出を数えるには、次のクエリを使用してください：

```

mysql> SELECT COUNT(*) FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+
| COUNT(*) |
+-----+
| 2 |
+-----+
1 row in set (0.00 sec)

```

しかし、次のようにクエリを書き換えたほうが手軽な場合もあります：

```

mysql> SELECT
-> COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
-> AS count
-> FROM articles;
+-----+
| count |
+-----+
| 2 |
+-----+
1 row in set (0.03 sec)

```

最初のクエリは関連性の大きさによって結果をソートし、2 番目のクエリではそれを行いません。しかし、2 番目のクエリは 1 番目が行わない、テーブル全体のスキャンをします。1 番目は数行のマッチしか検出されなければ時間はかかりませんが、そうでなくとも 2 番目は、どちらにしても多くの行を読むので素早く終わります。

自然言語の全文検索では、**MATCH()** 関数で名付けられたカラムが、テーブルの **FULLTEXT** インデックスのどれかに含まれるカラムと同じでなければなりません。先行のクエリに関しては、**MATCH()** 関数で名付けられたカラム (**title** and **body**) は、**article** テーブルの **FULLTEXT** インデックスの定義で名付けられたものと同じです。**title** と **body** を別々に検索したい場合は、各カラムに別々の **FULLTEXT** インデックスを作成する必要があります。

また、ブール検索もしくはクエリ拡張との検索を行うことも可能です。これらの検索タイプは **ブール全文検索** と **クエリ拡張を伴う全文検索** で説明されています。

インデックスを用いた全文検索は、インデックスが複数のテーブルをまたぐことはできないため、**MATCH()** 句の単一テーブルからのカラムにしか名前が付けられません。ブール検索はインデックスがなくても行えます (ただしスピードは落ちる)。その場合、複数のテーブルからのカラムを名付けることは可能です。

先行の例は、関連性が減少する順序に行が戻される **MATCH()** 関数の使い方を簡単に説明したものでした。次の例は関連値を明示的に引き出す方法です。**SELECT** が **WHERE** 句も **ORDER BY** 句も含んでいないため、行は順序付けられていません：

```

mysql> SELECT id, MATCH (title,body)
-> AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score

```

```
-> FROM articles;
+-----+
| id | score |
+-----+
| 1 | 0.65545833110809 |
| 2 | 0 |
| 3 | 0.66266459226608 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
+-----+
6 rows in set (0.00 sec)
```

次の例はさらに複雑なものです。クエリは関連値を戻し、また、関連性が減少する順序に行をソートします。この結果を得るため、[MATCH\(\)](#) を 2 度指定してください：一度は [SELECT](#) リスト、そしてもう一度は [WHERE](#) 句で指定します。MySQL の最適化プログラムが、ふたつの [MATCH\(\)](#) 呼び出しがまったく同じもので、全文検索コードを一度だけ実行されることに気付くため、これによって追加のオーバーヘッドが起こることはありません。

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root'
-> IN NATURAL LANGUAGE MODE) AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root'
-> IN NATURAL LANGUAGE MODE);
+-----+
| id | body | score |
+-----+
| 4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
| 6 | When configured properly, MySQL ... | 1.3114095926285 |
+-----+
2 rows in set (0.00 sec)
```

MySQL [FULLTEXT](#) の実装は、true 語文字 (文字、数字、および線文字) のすべてのシークエンスを言葉みなします。そのシークエンスはまた、アポストロフィ (') も含むことができますが、1 行にひとつのみです。つまり、`aaa'bbb` は一語とみなされますが、`aaa"bbb` は二語の扱いです。単語の頭または終わりのアポストロフィは、[FULLTEXT](#) パーサが取ってしまうので、`'aaa'bbb'` ならば `aaa'bbb` になります。

[FULLTEXT](#) パーサは特定の区切り文字を見て、語の頭と最後を定義します。その例には、' ' (スペース), ',' (カンマ), そして '.' (ピリオド) があります。単語が非区切り文字 (例えば中国語) で区切られている場合は、[FULLTEXT](#) パーサは単語の最初と最後を定義することができません。単語や、インデックスのついた他の表現をそのような言語で [FULLTEXT](#) インデックスに加えるには、事前に処理して " " などの任意の区切り文字で区切る必要があります。

MySQL 5.1 では、組み込まれた全文パーサを置き換えるプラグインを書くことができます。詳細は、[The MySQL Plugin Interface](#) を参照してください。例えば、パーサ プラグインのソースコードについては、MySQL のソース配布物の [plugin/fulltext](#) ディレクトリをご覧ください。

単語のあるものは、全文検索では無視されます：

- 短すぎる単語は無視されます。全文検索で検出される言葉で最も短いものは 4 文字です。
- ストップワード リストにある言葉は無視されます。ストップワードは「the」や「some」などの常用語で、語義の値はゼロとされています。すでに組み込まれているストップワードのリストがありますが、ユーザ定義リストで書き換えることができます。

デフォルトのストップワード リストは [全文ストップワード](#) で挙げられています。デフォルトの最短の単語の長さとストップワード リストは、[微調整 MySQL 全文検索](#) で説明されているように変更することができます。

コレクションとクエリの中のすべての正しい言葉は、コレクションまたはクエリでの重要性によって加重されています。従って、多くの資料に登場する言葉は、このコレクションでは語義の値が低いので、比重が低く (あるものはゼロ) なっています。逆に、稀な言葉は高く重みづけがされます。言葉の比重は、行の関連性を計算するために組み合わせて応用されます。

このような技術は、コレクションが大きいほど効果的に作用します (実際、そうなるように綿密に調整されています)。ごく小さなテーブルでは、言葉の分配が語義の値を適切に反映しないため、この形式においては時に不可解な結果が出る場合があります。例えば、「MySQL」という言葉は既出の [articles](#) テーブルのすべての行に含まれていますが、この単語で検索しても結果は出ません：

```
mysql> SELECT * FROM articles
```



```
-> WHERE MATCH (title,body)
-> AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

「MySQL」という言葉は少なくとも 50 % の行で提示されているため、検索結果は空になります。このように、この言葉は効果的にストップワードとして扱われます。大きなデータセットでは、これは最も望ましい動作です：自然言語のクエリは、1GB テーブルの毎 2 行目は戻さなくなっています。小さなデータセットにとっては、これはあまり望ましい動作ではありません。

テーブルの行の半分にマッチする言葉は、関連のある資料を見つけるのに適しません。事実、関連のないものも大量に検出されるでしょう。これはインターネットのサーチエンジンでの検索と同じ論理です。このため、この言葉を含む行は、この特定のデータセットにおいて 語義の値が低く定められています。あるデータセットでは、提示された単語が 50% の境界値を越えても、他のデータセットではまた異なります。

50% の境界値は、全文検索を行うとその重要性が明らかになります：テーブルを作成し、テキストの 1 行または 2 行のみを挿入してみると、テキストのすべての単語は少なくとも 50% の行に存在することが分かります。そのため、検出結果は検出されません。少なくとも 3 行以上を挿入するようにしてください。50% の制限を避ける必要がある場合は、ブール検索をお試しください。詳細は [ブール全文検索](#) をご覧ください。

ブール全文検索

MySQL は **IN BOOLEAN MODE** 修飾子を使用して、ブール全文検索を行うことができます。

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
-> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+-----+-----+
| id | title                | body                |
+-----+-----+
| 1 | MySQL Tutorial       | DBMS stands for DataBase ... |
| 2 | How To Use MySQL Well | After you went through a ... |
| 3 | Optimizing MySQL     | In this tutorial we will show ... |
| 4 | 1001 MySQL Tricks    | 1. Never run mysql as root. 2. ... |
| 6 | MySQL Security       | When configured properly, MySQL ... |
+-----+-----+
```

+ および - 演算子は、その言葉が含まれるものを検索するか、含まれないものを検索するかを示します。従って、このクエリは「MySQL」という単語を含むすべての行を引き出しますが、「YourSQL」という単語は含まれません。

ブール全文検索は以下の特徴を持っています：

- 50% の境界値を用いません。
- 行を自動的に関連性の降順にソートすることはありません。先行のクエリの結果からもこれが分かります：最高の関連性を持つ行は、「MySQL」を 2 度含んでいるものですが、最初でなく最後に挙げられています。
- **FULLTEXT** インデックスなしでも実行が可能ですが、その方法での検索は速度が極めて遅くなります。
- 全文パラメータの最小および最大の単語の長さの適用。
- ストップワード リストは適用されます。

ブール全文検索の機能は次の演算子をサポートします：

- +

頭にプラス記号が付くのは、その言葉が戻される各行に必ず含まれていなければならないことを示します。

- -

頭にマイナス記号が付くのは、その言葉が戻される行のいずれにも絶対に含まれるべきでないことを示します。

注記：- 演算子は、本来なら他の検索語によって検出される行を除外するためだけのものです。従って、- によって優先された検索語のみを含むブール モードの検索は、空の結果を返します。「除外された検索語を含むものをのぞいたすべての行」が返されるわけではありません。

- (演算子なし)

デフォルトにより (+ も - も指定されていない場合)、その単語は任意になりますが、その語を含む行は上位に順位づけられます。これは、[IN BOOLEAN MODE](#) 修飾子なしの [MATCH\(\) ... AGAINST\(\)](#) の動作を模倣しています。

- [> <](#)

このふたつの演算子は、行にかけられた関連値への、単語の寄与度を変更します。> 演算子は寄与度を高め、< は低めます。以下のリストに続く例を参照してください。

- [\(\)](#)

丸括弧は単語を部分式にグループ分けします。丸括弧でまとめられたグループは入れ子になります。

- [~](#)

頭につくチルダ (波型記号) は否定演算子になり、行の関連性への単語の貢献が否定的になります。これは「noise」単語をマークするのに便利です。そのような単語を含む行は、他よりも低く順位づけられますが、- 演算子のように除外されることはありません。

- [*](#)

アスタリスク (星印) は前方一致 (またはワイルドカード) 演算子として機能します。他の演算子とは異なり、単語に付加して影響をあたえます。* の演算子を単語の前につければマッチします。

- ["](#)

二重引用符 ("") でフレーズを囲むと、そのフレーズそのものを持つ行のみにマッチします。フレーズを単語に分ける全文エンジンは、[FULLTEXT](#) インデックスで、その単語を検索します。非言語文字は正確にマッチする必要があります: フレーズ検索は、そのフレーズとまったく同じ単語を同じ並びで含むマッチのみを必要とします。例えば、["test phrase"](#) は ["test, phrase"](#) とマッチします。

フレーズの単語がインデックスにある単語とマッチしない場合は、結果は空になります。例として、すべての単語がストップワードであったり、インデックスつき単語の必須の文字数に満たない場合などは、結果が空になります。

次の例はブール全文演算子を使用する検索ストリングを、いくつかデモンストレートしたものです:

- ['apple banana'](#)

ふたつの単語のうち、すくなくともひとつを含む行を検出。

- ['+apple +juice'](#)

両方の語を含む行を検出。

- ['+apple macintosh'](#)

単語「apple」を含む行を検出し、さらに「macintosh」を含んでいる場合は行を高く順位づける。

- ['+apple -macintosh'](#)

単語「apple」を含み、「macintosh」を含まない行を検出。

- ['+apple ~macintosh'](#)

単語「apple」を含む行を検出するが、行が単語「macintosh」も含む場合は、含まないものよりも行を低く順位づける。これは、「macintosh」が含まれると完全に除外される ['+apple -macintosh'](#) の検索より「柔らかい」。

- ['+apple +\(>turnover <strudel\)'](#)

「apple」と「turnover」、もしくは「apple」と「strudel」(順序は不問)を含む行を検出するが、「apple turnover」を「apple strudel」より高く順序づける。

- ['apple*'](#)

単語「apple」、「apples」、「applesauce」、または「applet」を含む行を検出。

- "some words"

「some words」とまったく同じフレーズを含む行を検出（例えば、「some words of wisdom」を含む行は該当するが、「some noise words」は該当しない）。フレーズを囲む「"」文字は、フレーズを区切る演算子であることに注意。それらは検索ストリングそのものを囲む引用符ではない。

クエリ拡張を伴う全文検索

全文検索はクエリの拡張をサポートします（特にその変異型の「ブラインド クエリ拡張」）。これは一般的に、検索フレーズが短すぎる時に役に立ちます。フレーズが短いのは主に、ユーザに曖昧な知識しかなく、全文検索エンジンの暗示検索能力に頼る場合ですが、全文検索エンジンではその能力が不十分です。例えばユーザが「database」で検索する場合は、「MySQL」、「Oracle」、「DB2」、そして「RDBMS」を指していると考えられ、これらのフレーズはすべて「databases」とマッチし戻されます。これが暗示検索能力です。

ブラインド クエリ拡張（自動関連フィードバックとも言う）は、検索フレーズの後に **WITH QUERY EXPANSION** または **IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION** を加えることによって有効になります。これは検索を2度行うことで作動し、2度目の検索には、最初の検索で検出された資料から、最も関連性の強い単語を抜き出してつなぎ合わせた、独自の検索フレーズを使用します。従って、資料のどれかに単語「databases」および「MySQL」が含まれている場合、2度目の検索では「database」を含んでいなくても、「MySQL」を含む資料が検出されます。次の例はその相違点を示しています：

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+
| id | title | body |
+-----+-----+
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' WITH QUERY EXPANSION);
+-----+-----+
| id | title | body |
+-----+-----+
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
| 3 | Optimizing MySQL | In this tutorial we will show ... |
+-----+-----+
3 rows in set (0.00 sec)
```

他の例では、Georges Simenon 著の Maigret についての書籍を検索する時に、ユーザが「Maigret」のスペルを知らないと仮定します。クエリ拡張なしで、「Megre and the reluctant witnesses」で検索した場合、「Maigret and the Reluctant Witnesses」の単語群でしか検出されません。クエリ拡張を使用すれば、2度目の検索で、「Maigret」を含むすべての書籍が検出されます。

注記：ブラインド クエリ拡張は関連性のない雑多な資料も戻しがちなため、検索フレーズが短い時にだけ使用することをお勧めします。

全文ストップワード

次のテーブルは、フルテキストのストップワードのデフォルトのリストです。

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway

anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	knows	known	last	lately
later	latter	latterly	least	less

lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly

try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
would	wouldn't	yes	yet	you
you'd	you'll	you're	you've	your
yours	yourself	yourselves	zero	

全文制限

- 全文検索は [MyISAM](#) テーブルでのみサポートされています。
- 全文検索は、ほとんどのマルチバイト文字セットと使用できます。例外は Unicode で、[utf8](#) 文字セットは使用可能ですが、[ucs2](#) 文字セットは使用できません。
- 中国語や日本語のような表意文字を用いる言語は区切り符号を持ちません。従って、[FULLTEXT](#) パーサはその種の言語では単語の始めと終わりを区別することができません。この含意と問題の回避については [全文検索関数](#) で説明されています。
- 単一テーブル内での複数の文字セットの使用はサポートされているものの、[FULLTEXT](#) インデックスのすべてのカラムは、同じ文字セットと照合を使用する必要があります。
- [MATCH\(\)](#) カラム リストは、[MATCH\(\)](#) が [IN BOOLEAN MODE](#) でない限り、[FULLTEXT](#) インデックスのテーブルのための定義のカラム リストと正確に一致していなければなりません。ブール モードの検索はインデックス付きでないカラムでも行えますが、スピードは遅くなるでしょう。
- [AGAINST\(\)](#) への引数は定数ストリングでなければなりません。

微調整 MySQL 全文検索

MySQL の全文検索の機能は、ユーザが調整できるパラメータをほとんど持っていません。全文検索の動作をある程度コントロールすることは可能ですが、変更にはソースコードの改変が必要になる場合があるので、MySQL ソース配布物が必要です。詳細は [ソースのディストリビューションを使用した MySQL のインストール](#) を参照してください。

全文検索は最大の効果を発揮するよう、慎重に調整されています。デフォルトの動作を改変すると、多くの場合、その効果を低めることになります。特に知識がない限り、MySQL のソースを変更しないでください。

このセクションで説明されている全文変数のほとんどは、サーバの起動時に設定する必要があります。変更にはサーバの再起動が必要です。サーバが作動している間は手を加えることはできません。

変数のあるものは、変更するとテーブルの [FULLTEXT](#) インデックスを再構築しなければなりません。この手順は、このセクションの最後で説明されています。

- インデックスを付けるにあたっての単語の最小および最大の文字数は、[ft_min_word_len](#) および [ft_max_word_len](#) システム環境変数によって定義されています。([システム変数](#) 参照) デフォルトの最小値は 4 文字で、最大値はバージョンによって異なります。これらの値を変更する場合は、[FULLTEXT](#) インデックスを再構築する必要があります。例えば、3 文字でも検索を可能にしたい場合、次のラインをオプション ファイルに入力することで、[ft_min_word_len](#) 変数を設定できます：

```
[mysqld]
ft_min_word_len=3
```

その後、サーバを再起動し、[FULLTEXT](#) インデックスを再構築します。このリストの後にある説明の、[myisamchk](#) についての記述は特に注意してお読みください。

- デフォルトのストップワードリストを書き換えるには、[ft_stopword_file](#) システム環境を設定してください。([システム変数](#) 参照) 変数値は、ストップワードリストを含むファイルのパス名か、ストップワードのフィルタ処理を無効にする空のストリングになります。この変数の値が、ストップワード ファイルの内容を変更した後、サーバを再起動し、[FULLTEXT](#) インデックスを再構築してください。

ストップワードリストはフリー形態です。つまり、改行、スペース、またはコンマなどの非英数文字を使用して、ストップワードを区切ることができます。例外は、単語の一部として扱われる、下線文字 ('_') と単一引用符 ('') です。ストップワードリストの文字セットは、サーバのデフォルトの文字セットです。[サーバのキャラクタセットおよび照合順序](#) 参照。

- 自然言語検索の 50% の境界値は、選択された特定の加重スキームによって定義されています。これを無効にするには、[storage/myisam/ftdefs.h](#) で次のラインを探してください：

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

その語、MySQL を再コンパイルします。この場合は、インデックスを再構築する必要はありません。注記 : この変更を行うことで、[MATCH\(\)](#) 関数に対して適切な関連値を提供する MySQL の能力は大幅に低下します。一般的な単語をどうしても検索する必要があるなら、50% の境界値を変更しなくても済む、[IN BOOLEAN MODE](#) を使用して検索するほうが賢明です。

- ブール全文検索に使用した演算子を変更するには、[ft_boolean_syntax](#) システム環境変数を設定します。この変数はサーバの使用中でも変更することができますが、実行するには [SUPER](#) 権限が必須です。この場合は、インデックスを再構築する必要はありません。この変数の設定をつかさどるルールの説明を、[システム変数](#) でご覧ください。
- 言語文字とされる文字のセットを変更したい場合、方法はふたつあります。ハイフン文字 ('-') を言語文字として扱いたいと仮定します。下記のどちらかの方法を使用してください：
 - MySQL ソースを改変する : [storage/myisam/ftdefs.h](#) で、[true_word_char\(\)](#) および [misc_word_char\(\)](#) マクロをご覧ください。そのマクロのどちらかに ' ' を加え、MySQL を再コンパイルします。
 - 文字セット ファイルを改変する : これには再コンパイルは不要です。[true_word_char\(\)](#) マクロは、「character type」テーブルを使用して、他の文字と、アルファベットおよび数字を区別します。文字セットの XML ファイルのひとつで、[<ctype><map>](#) の内容を編集し、' ' を「letter」に指定します。その後、[FULLTEXT](#) インデックスに、提示された文字セットを使用します。

改変の後で、[FULLTEXT](#) インデックスを含む各テーブルのインデックスを再構築します。

インデックスに影響を及ぼす全文変数 ([ft_min_word_len](#)、[ft_max_word_len](#)、または [ft_stopword_file](#)) を改変する場合、もしくはストップワードファイルそのものを変更する場合、変更を行った後に [FULLTEXT](#) インデックスを再構築し、サーバを再起動させてください。この場合にインデックスを再構築するには、[QUICK](#) 修復オペレーションを行えば十分です；

```
mysql> REPAIR TABLE tbl_name QUICK;
```

[FULLTEXT](#) インデックスをひとつでも含むテーブルはそれぞれ、上記のように修復が必要です。さもないと、テーブルのクエリが誤った結果を生産し、テーブルの変更によって、サーバはテーブルを修復が必要な破損があるものとみなします。

`myisamchk` を使用してテーブルのインデックスを改変する操作（修復や分析）を行った場合、特に指定しない限り、`FULLTEXT` インデックスは、最小文字数、最大文字数、そしてストップワード ファイルに対するデフォルトの全文パラメータ値を使用して再構築されます。これはクエリの失敗につながります。

問題の原因は、これらのパラメータがサーバにしか認識されていないことです。それらは `MyISAM` インデックス ファイルには保存されていません。この問題を避けるには、サーバによって使用される最小または最大文字数、もしくはストップワード ファイル値を改変した場合、`mysqld` に使用する `myisamchk` と同じ `ft_min_word_len`、`ft_max_word_len`、および `ft_stopword_file` 値を指定してください。例えば、最小文字数を 3 に設定した場合、次のように `myisamchk` をもってテーブルを修復することができます：

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

`myisamchk` とサーバが、間違いなく全文パラメータに同じ値を使用するよう、それぞれをオプション ファイルの `[mysqld]` と `[myisamchk]` のセクションに置いてください：

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

`myisamchk` の使用に替わる方法は、`REPAIR TABLE`、`ANALYZE TABLE`、`OPTIMIZE TABLE`、もしくは `ALTER TABLE` 文の使用です。これらのステートメントは、適切な全文パラメータ値を選ぶことのできるサーバによって実行されます。

キャスト関数と演算子

- `BINARY`

`BINARY` 演算子はそれに続いて、バイナリ スtringにStringをキャストします。これはカラムの比較を強制的に、文字ごとでなくバイトごとに行わせる簡易な方法です。カラムが `BINARY` または `BLOB` と定義されていない場合でも、大文字小文字を区別した比較になります。`BINARY` もまた、後続のスペースを重要なものにします。

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
mysql> SELECT 'a' = 'a ';
-> 1
mysql> SELECT BINARY 'a' = 'a ';
-> 0
```

比較において、`BINARY` は演算全体に影響を与えます。同じ結果を持つどちらのオペランドの前にでも、与えることができます。

`BINARY str` は、`CAST(str AS BINARY)` の省略表記です。

コンテキストのあるものでは、インデックス付きのカラムを `BINARY` にキャストした場合、MySQL はそのインデックスを有効に使うことができません。

- `CAST(expr AS type)`, `CONVERT(expr,type)`, `CONVERT(expr USING transcoding_name)`

`CAST()` および `CONVERT()` 関数はひとつのタイプの値をもって、他のタイプの値を生成します。

その `type` は次の値のどれかになりえます：

- `BINARY[(N)]`
- `CHAR[(N)]`
- `DATE`
- `DATETIME`
- `DECIMAL`

- [SIGNED \[INTEGER\]](#)
- [TIME](#)
- [UNSIGNED \[INTEGER\]](#)

[BINARY](#) は、[BINARY](#) データ タイプを持つストリングを生成します。これが比較に及ぼす影響については [BINARY と VARBINARY タイプ](#) をご覧ください。任意の長さ [N](#) が与えられた場合、[BINARY\(N\)](#) は、キャストが [N](#) バイト以下の引数を使用する原因となります。[N](#) バイトより短い値は、[0x00](#) バイトで [N](#) の長さまでパッドされます。

[CHAR\(N\)](#) 句は、キャストが [N](#) 文字以下の引数を使用する原因になります。

[CAST\(\)](#) および [CONVERT\(... USING ...\)](#) は標準の SQL シンタックスです。[CONVERT\(\)](#) の非 [USING](#) フォームは ODBC シンタックスです。

[USING](#) を持つ [CONVERT\(\)](#) は、異なる文字セット間のデータを変換するのに使用されます。MySQL では、符号化名は対応文字セット名と同じものです。例えば、このステートメントは、デフォルトの文字セットのストリング ['abc'](#) を、[utf8](#) 文字セットの対応ストリングに変換します。

```
SELECT CONVERT('abc' USING utf8);
```

バイナリストリングは文字セットを持たないため、通常は [BLOB](#) 値、またはバイナリストリングを、大文字小文字の区別のない方法で比較はできず、従って大文字小文字という概念はありません。大文字小文字を区別しない比較を行うには、[CONVERT\(\)](#) 関数を使用して、値を非バイナリストリングに変換します。結果の文字セットが大文字小文字を区別しない照合を得た場合、[LIKE](#) 演算は大文字小文字の区別をしません：

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

異なる文字セットを使用するには、先行するステートメントで、その名前を [latin1](#) の代わりにします。大文字小文字の区別のない照合を確実に使用するには、[CONVERT\(\)](#) 呼び出しの後に [COLLATE](#) 句を特定します。

[CONVERT\(\)](#) を、異なる文字セットで示されているストリングの比較に、より一般的に使用することができます。

キャスト関数は、[CREATE ... SELECT](#) 文で、特定のタイプのカラム作成したい場合に役立ちます：

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

この関数はまた、[ENUM](#) カラムを語彙順にソートしたい場合にも利用できます。通常は、[ENUM](#) カラムのソートは内部数値を使用して行います。値を [CHAR](#) にキャストすると、結果は語彙順になります：

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

[CAST\(str AS BINARY\)](#) は [BINARY str](#) と同じものです。[CAST\(expr AS CHAR\)](#) は式を、デフォルトの文字セットを持つストリングとして扱います。

[CONCAT\('Date: ',CAST\(NOW\(\) AS DATE\)\)](#) のようなより複雑な式の一部として使用する場合、[CAST\(\)](#) もまた結果を変えます。

データを異なるフォーマットに抽出するには、[CAST\(\)](#) ではなく、[LEFT\(\)](#) または [EXTRACT\(\)](#) のようなストリング関数を使用します。詳細は [日付時刻関数](#) を参照してください。

ストリングを数値コンテキストの数値にキャストするには、通常はストリング値を数字のように使用するだけで済みます。

```
mysql> SELECT 1+'1';
-> 2
```

ストリング コンテキストで数字を使用する場合、その数字は自動的に [BINARY](#) ストリングに変換されます。

```
mysql> SELECT CONCAT('hello you ',2);
-> 'hello you 2'
```

MySQL は、符号付きでも、符号無しでも、64 バイト値での演算をサポートします。算術演算子 ([+](#) または [-](#) など) を使用しており、演算のひとつは符号のない整数である場合、結果は符合なしになります。[SIGNED](#) および [UNSIGNED](#)

キャスト演算子を使用して、演算を符号付き、もしくは符号なしの 64 ビットの整数にキャストすることで、これをそれぞれオーバーライドすることができます。

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
-> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
-> -1
```

オペランドのどちらかが浮動小数点値で有る場合、結果は浮動小数点値になり、前のルールには影響を受けません。(このコンテキストでは、DECIMAL カラム値は浮動小数点値とみなされます。)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
-> -1.0
```

算術演算でストリングを使用している場合、これは浮動小数点数に変換されます。

「zero」日付ストリングを日付に変換する場合、CONVERT() と CAST() は NULL を返し、NO_ZERO_DATE SQL モードが有効になれば警告を発行します。

XML 関数

このセクションでは MySQL での XML と関連する機能について説明します。

--xml オプションで呼び出して、mysql および mysqldump クライアントの XML フォーマットの出力を MySQL から得ることは可能です。mysql — MySQL コマンドライン ツール および mysqldump — データベースバックアッププログラムを参照してください。

MySQL 5.1.5 からは、基礎的な XPath (XML Path Language) 機能を提供するふたつの関数を利用することができます。

これらの関数は現在もまだ開発中ですのでご注意ください。MySQL 5.1 と今後のためにも、これらの関数や XML および XPath の機能を改良し続けていきます。これらについてご意見や質問のある方、また他のユーザからのアドバイスを得たい方は、MySQL XML User Forum をご覧ください。

- ExtractValue(xml_frag, xpath_expr)

ExtractValue() はふたつのストリング引数、XML マークアップのフラグメント xml_frag、そして XPath 式 xpath_expr (locator と呼ばれる) を取り、XPath 式によってマッチされたエレメントの子である、最初のテキストノードのテキスト (CDATA) を返します。これは、/text() を付加した後に、xpath_expr を使用してマッチを行うのと同様です。つまり、ExtractValue('<a>Sakila', '/a/b') と ExtractValue('<a>Sakila', '/a/b/text()') は同じ結果を生成します。

複数のマッチが検出される場合、各マッチングエレメントの、最初の子のテキストノードの内容は、単一の、スペースで区切られたストリングとして (マッチした順で) 戻されます。

(拡大された) 式 — に対して、マッチするテキストノードが検出されない場合 — どういう理由であれ、xpath_expr が有効で、xml_frag が適切に成型されていれば — 空のストリングが戻されます。空のエレメントでの整合と、整合するものがないのとは、区別はされません。これはデザインによるものです。

xml_frag でマッチするエレメントが見つからなかったのか、またはマッチするエレメントはあったものの、非子テキストノードを含んでいたのかを判断する必要がある場合は、XPath count() 関数を使用する式の結果をテストしてください。例えば、次のように、これらのステートメントの両方が空のストリングを返す場合：

```
mysql> SELECT ExtractValue('<a><b></b></a>', '/a/b');
+-----+
| ExtractValue('<a><b></b></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c></c></a>', '/a/b');
+-----+
| ExtractValue('<a><c></c></a>', '/a/b') |
+-----+
```

```
+-----+
|               |
+-----+
1 row in set (0.00 sec)
```

しかし、次のように、実際にマッチするエレメントがあったのかを確認することはできます：

```
mysql> SELECT ExtractValue('<a><b></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><b></a>', 'count(/a/b)') |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><c></a>', 'count(/a/b)') |
+-----+
| 0 |
+-----+
1 row in set (0.01 sec)
```

`ExtractValue()` は `CDATA` のみを返し、マッチング タグに含まれるタグや、それらのコンテンツは戻されません (次の例の、`val1` として戻された結果を参照)。

```
mysql> SELECT
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val3,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
-> ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '/b') AS val5;
+-----+-----+-----+-----+
| val1 | val2 | val3 | val4 | val5 |
+-----+-----+-----+-----+
| ccc  | ddd  | ddd  |      | ddd eee |
+-----+-----+-----+-----+
```

MySQL 5.1.8 からは、この関数は、`contains()` との比較に現行の SQL の照合を使用します。(以前は、バイナリ — 大文字小文字の区別あり — 比較が常に使用されていました。)

- `UpdateXML(xml_target, xpath_expr, new_xml)`

この関数は、XML マークアップ `xml_target` の提示されたフラグメントの単一部を、新しい XML フラグメント `new_xml` に置き換え、その後チャージされた XML を返します。置換された `xml_target` の一部は、ユーザから提供された XPath 式 `xpath_expr` にマッチします。`xpath_expr` にマッチする式が検出されない場合、または複数のマッチが見つかった場合、この関数は独自の `xml_target` XML フラグメントを返します。3 つすべての引数はストリングでなければなりません

```
mysql> SELECT
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val3,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
-> UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
-> \G

***** 1. row *****
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```

次は、いくつかの基本的な XPath 式の説明と例です：

- `/tag`

`<tag/>` がルートのエレメントである場合にのみ、`<tag/>` にマッチします。

例 `/a` は、最外部の (ルート) タグとマッチするため、`<a><b/></a>` に一致を持ちます。このインスタンスでは他のエレメントの子になるため、`<b><a/></b>` の内側 `a` エレメントとは一致しません。

- `/tag1/tag2`

`<tag1/>` の子と、`<tag1/>` がルートのエレメントである場合にのみ、`<tag2/>` にマッチします。

例 : ルートのエレメント `a` の子であるため、`/a/b` は XML フラグメント `<a><b/></a>` 内の `b` エレメントとマッチします。このケースでは `b` はルートのエレメント (従って他のどのエレメントの子でもない) ため、`<b><a/></b>` でマッチするものではありません。XPath 式もまた、`<a><c><b/></c></a>` でマッチするものではありません。従って、`b` は `a` の子孫ですが、`a` の子ではありません。

この構築は 3 つ以上のエレメントに拡張可能です。例えば、XPath 式 `/a/b/c` は、フラグメント `<a><b><c/></b></a>` の `c` エレメントに一致します。

- `//tag`

`tag` のすべてのインスタンスと一致します。

例 : `//a` は、次のうちのどの `a` エレメントとも一致します : `<a><b><c/></b></a>`; `<c><a><b/></a></b>`; `<c><b><a/></b></c>`

`//` は `/` との結合が可能です。例えば、`//a/b` は、フラグメント `<a><b/></a>` または `<a><b><c/></b></a>` のどれの `b` エレメントともマッチします。

- * 演算子は、どのエレメントともマッチする「wildcard」のように作用します。例えば、式 `*/b` は、XML フラグメント `<a><b/></a>` または `<c><b/></c>` のどの `b` エレメントともマッチします。しかし、`b` は他のどれかのエレメントの子であるため、この式はフラグメント `<b><a/></b>` ではマッチを生産しません。ワイルドカードはどのポジションでも使用することができます : 式 `*/b/*` は、それ自身がルートのエレメントでない `b` エレメントの、どの子ともマッチします。

- 複数のクエータが、`|` (論理和 OR) 演算子を用いてマッチすることができます。例えば、XPath 式 `//b|/c` は、XML ターゲットのすべての `b` および `c` エレメントにマッチします。

- その特性のひとつ以上の値に基づいたエレメントにマッチすることも可能です。これは、シンタックス `tag[@attribute="value"]` を用いて行います。例えば、XPath 式 `//b[@id="idB"]` は、フラグメント `<a><b id="idA"/><c/><b id="idB"/></a>` の 2 番目の `b` エレメントに一致します。`attribute="value"` を持ついかなるに対しても一致するには、XPath 式 `//*[attribute="value"]` を使用します。

複数の属性値をフィルターにかけるには、単に複数の属性比較句を継続的に使用します。例えば、XPath 式 `//b[@c="x"][@d="y"]` は、与えられた XML フラグメントの各所で起こっている `<b c="x" d="y"/>` エレメントに一致します。

同じ特性が複数の値のうちのひとつとマッチするエレメントを見つけるには、`|` 演算子によってつながれた複数のクエータを使う必要があります。例えば、`c` 特性が値 23 もしくは 17 を持つ、すべての `b` エレメントをマッチするには、式 `//b[@c="23"]|b[@c="17"]` を使用します。

XPath シンタックスのさらに詳しい説明や使用法は、このマニュアルの対象範囲ではありません。決定的な情報については [XML Path Language \(XPath\) 1.0 standard](#) をご覧ください。XPath をご存知ない方、基本を復習したい方は [Zvon.org XPath Tutorial](#) を参照してください。複数の言語でご覧いただけます。

これらの関数にサポートされている XPath シンタックスは、現在、以下の制限の対象となっています :

- ノード セット間比較 (`/a/b[@c=@d]` など) はサポートされていません。`const` が定数値のフォーム `[@attribute="const"]` の唯一の比較は、現在可能です。サポートされている比較演算子は、同等と不等 (`=` と `!=`) のみです。
- 相対クエータ式はサポートされていません。XPath 式は、`/` または `//` で始まります。
- `::` 演算子はサポートされていません。
- 「Up-and-down」ナビゲーションは、パスがルート エレメントの「上」をリードする場合はサポートされていません。つまり、現在のエレメントのひとつ以上の祖先が同時にルート エレメントの祖先であり、与えられたエレメントの祖先の継承上でマッチする式を使用することができません (Bug #16321 参照)。

- 次の XPath 関数はサポートされていません：

- `id()`
- `lang()`
- MySQL 5.1.8 より前は、`last()` 関数はサポートされていません (Bug #16318 参照)。
- `local-name()`
- `name()`
- `namespace-uri()`
- `normalize-space()`
- `starts-with()`
- `string()`
- `substring-after()`
- `substring-before()`
- `translate()`

- 次の軸はサポートされていません：

- `following-sibling`
- `following`
- `preceding-sibling`
- `preceding`

MySQL 5.1.10 からは、XPath 式は引数として `ExtractValue()` に渡され、`UpdateXML()` が、XML ネームスペース記号を採用してマークアップとの使用を有効にするエレメント セレクタに、コロン文字 (「:」) を含むこともあります。例：

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//e:f');
+-----+
| ExtractValue(@xml, '//e:f') |
+-----+
| 444                          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
+-----+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+-----+
| <a>111<g:h>555</g:h></a>                  |
+-----+
1 row in set (0.00 sec)
```

これは [Apache Xalan](#) と他のいくつかのパーサによって利用できるものに似ており、また、ネームスペース宣言または `namespace-uri()` や、`local-name()` 関数を要求したりするよりより単純です。

その他の関数

ビット関数

MySQL はビット演算に `BIGINT` (64 ビット) 演算を使用し、演算子が 64 ビットの最大範囲を持つようにします。

- |

ビット単位の論理積：

```
mysql> SELECT 29 | 15;
-> 31
```

結果は符合なしの 64 ビット整数です。

- &

ビット単位の論理積：

```
mysql> SELECT 29 & 15;
-> 13
```

結果は符合なしの 64 ビット整数です。

- ^

ビット単位の排他的論理和：

```
mysql> SELECT 1 ^ 1;
-> 0
mysql> SELECT 1 ^ 0;
-> 1
mysql> SELECT 11 ^ 3;
-> 8
```

結果は符合なしの 64 ビット整数です。

- <<

longlong (BIGINT) ナンバーを左にシフトします。

```
mysql> SELECT 1 << 2;
-> 4
```

結果は符合なしの 64 ビット整数です。

- >>

longlong (BIGINT) ナンバーを右にシフトします。

```
mysql> SELECT 4 >> 2;
-> 1
```

結果は符合なしの 64 ビット整数です。

- ~

すべてのビットを反転します。

```
mysql> SELECT 5 & ~1;
-> 4
```

結果は符合なしの 64 ビット整数です。

- BIT_COUNT(N)

引数 N で設定されているビットの数を返します。

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
-> 4, 3
```

暗号化関数と圧縮関数

このセクションの関数は暗号化と復号化、そして圧縮と非圧縮を行います。

暗号化または圧縮	復号化または解凍
AES_ENCRYPT() [631]	AES_DECRYPT() [631]
COMPRESS() [631]	UNCOMPRESS() [634]
ENCODE() [632]	DECODE() [632]
DES_ENCRYPT() [632]	DES_DECRYPT() [632]
ENCRYPT() [633]	使用不可
MD5() [633]	使用不可
OLD_PASSWORD() [633]	使用不可
PASSWORD() [633]	使用不可
SHA() or SHA1() [634]	使用不可
使用不可	UNCOMPRESSED_LENGTH() [634]

注記 :暗号化および圧縮関数はバイナリ スtringを戻します。これらの関数の多くは、結果が任意のバイト値を含む場合があります。これらの結果を保存したい場合は、[CHAR](#) や [VARCHAR](#) カラムでなく、[BLOB](#) を使用して、後続のスペースの削除でデータ値が変更される可能性を避けてください。

注記 :MD5 および SHA-1 アルゴリズムの利用についてはすでに知られています。開発者は、このセクションで紹介されている他の暗号化関数の使用も考慮してください。

- [AES_ENCRYPT\(str,key_str\)](#), [AES_DECRYPT\(crypt_str,key_str\)](#)

これらの関数では、以前は「Rijndael」として知られていた公式の AES (Advanced Encryption Standard) アルゴリズムを使用した、データの暗号化と復号化が可能です。128 ビットのキーの長さを使用したエンコードを行いますが、ソースを改変することで 256 ビットまで延長することができます。当社では、より速く、ほとんどの使用では十分に安全なため、128 ビットを採用しています。

[AES_ENCRYPT\(\)](#) は Stringを暗号化し、バイナリ Stringを戻します。[AES_DECRYPT\(\)](#) は Stringを暗号化された Stringを復号化し、本来の Stringを戻します。入力引数の長さは自由です。どちらかの引数が [NULL](#) の場合は、この関数の結果も [NULL](#) になります。

AES ブロックレベル アルゴリズムであるため、長さが不揃いな Stringのエンコードにはパッドを使用し、次の方式を使って結果 Stringの長さが計算されるようにします。

```
16 * (trunc(string_length / 16) + 1)
```

[AES_DECRYPT\(\)](#) が無効な日付または不正なパッドを検出した場合は、[NULL](#) が戻されます。しかし、入力データまたはキーが無効になっている場合は、[AES_DECRYPT\(\)](#) が非 [NULL](#) 値 (不要データの可能性あり) を戻すことも考えられます。

AES 関数を使用して、暗号化されたフォームのデータを、クエリを改変することによって格納することができます :

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

[AES_ENCRYPT\(\)](#) および [AES_DECRYPT\(\)](#) は、現在 MySQL で使用可能なものの中で、暗号的に最も安全な暗号化関数だと考えられています。

- [COMPRESS\(string_to_compress\)](#)

Stringを圧縮し、結果をバイナリ Stringとして戻します。この関数では、MySQL が [zlib](#) のような圧縮ライブラリとコンパイルされている必要があります。その条件が満たされない場合、その戻り値は常に 0 になります。圧縮された Stringは、[UNCOMPRESS\(\)](#) によって非圧縮することができます。

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
-> 21
mysql> SELECT LENGTH(COMPRESS(''));
-> 0
mysql> SELECT LENGTH(COMPRESS('a'));
-> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
```

-> 15

圧縮されたストリングの内容は次の方法で格納されます：

- 空のストリングは空のストリングとして格納。
- 空でないストリングは、圧縮されたストリングの後に、4 バイト長の非圧縮ストリングとして (下位バイトから) 格納されます。ストリングの最後にスペースがある場合は、最後のスペースが除かれることがないよう、' ' 文字が追加されます。結果は [CHAR](#) または [VARCHAR](#) カラムに格納されます。([CHAR](#) または [VARCHAR](#) を使用して、圧縮されたストリングを保存するのはお薦めできません。 [BLOB](#) カラムをご使用ください。)
- [DECODE\(crypt_str,pass_str\)](#)

暗号化されたストリング [crypt_str](#) を、[pass_str](#) を使用し、パスワードとして復号化します。[crypt_str](#) は、[ENCODE\(\)](#) から戻されたストリングであるべきです。

- [ENCODE\(str,pass_str\)](#)

[pass_str](#) を使用し、[str](#) をパスワードとして暗号化します。結果を復号化するには [DECODE\(\)](#) を用います。

結果は、[str](#) と同じ長さのバイナリ ストリングになります。

暗号化の強度は、ランダム発生器の質によります。短いストリングでも十分です。

- [DES_DECRYPT\(crypt_str\[,key_str\]\)](#)

[DES_ENCRYPT\(\)](#) によって暗号化されたストリングを復号化します。エラーが起きた場合、この関数は [NULL](#) を戻します。

この関数は、MySQL が SSL サポートで設定されている場合のみ作動しますのでご注意ください。詳細は [接続安全](#) を参照してください。

[key_str](#) 引数を与えられていない場合、[DES_DECRYPT\(\)](#) は暗号化されたストリングの最初のバイトを調査して、本来のストリングの暗号化に使用した DES キー ナンバーを特定し、DES キー ファイルからキーを読み取って、メッセージを復号化します。これを正しく行うには、ユーザは [SUPER](#) 権限を持っている必要があります。キー ファイルは [--des-key-file](#) サーバ オプションで特定できます。

この関数を [key_str](#) 引数に渡した場合、そのストリングはメッセージの復号化のキーとして使用されます。

[crypt_str](#) 引数が暗号化されたストリングでない場合は、MySQL は与えられた [crypt_str](#) を戻します。

- [DES_ENCRYPT\(str\[,key_num|key_str\]\)](#)

Triple-DES アルゴリズムを使用して、与えられたキーでストリングを暗号化します。

この関数は、MySQL が SSL サポートで設定されている場合のみ作動しますのでご注意ください。詳細は [接続安全](#) を参照してください。

使用する暗号化キーは、与えられていれば、[DES_ENCRYPT\(\)](#) への 2 番目の引数に基づいて選択されます：

引数	解説
引数なし	DES キー ファイルの最初のキーが使用される。
key_num	DES キー ファイルからの与えられたキー番号 (0-9) が使用される。
key_str	与えられたキー ストリングが str の暗号化に使用される。

キー ファイルは [--des-key-file](#) サーバ オプションで特定できます。

戻されるストリングは、最初の文字が [CHAR\(128 | key_num\)](#) であるバイナリ ストリングです。エラーが起きた場合、[DES_ENCRYPT\(\)](#) は [NULL](#) を戻します。

暗号化されたキーが分かりやすいように、128 が加えられます。ストリング キーを使用する場合は、[key_num](#) は 127 です。

結果のストリングの長さは次の方式によって提示されます：

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

DES キー ファイルの各ラインは次のフォーマットを含みます：

```
key_num des_key_str
```

各 `key_num` 値は、0 から 9 の範囲の数字でなければなりません。ファイル内のラインの順は特定されていません。`des_key_str` はメッセージの暗号化に使用されるストリングです。数字とキーの間には、少なくともひとつはスペースが入ります。最初のキーは、`DES_ENCRYPT()` へのキー引数を指定しなかった場合に使用されるデフォルトのキーです。

MySQL に、キー ファイルからの新しいキー値を、`FLUSH DES_KEY_FILE` 文で読み込むよう指示することができます。これには `RELOAD` 権限が必須です。

デフォルト キーのセットを持つことの利点のひとつは、エンドユーザにこれらの値を復号化する権利を与えることなく、既存の暗号化されたカラム値を確認する方法をアプリケーションに与えられることです。

```
mysql> SELECT customer_address FROM customer_table
> WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCRYPT(str[,salt])`

Unix `crypt()` システム呼び出しを使って `str` を暗号化し、バイナリ ストリングを戻します。`salt` 引数は少なくとも 2 文字のストリングでなければいけません。`salt` が与えられていない場合は、ランダム値が使用されます。

```
mysql> SELECT ENCRYPT('hello');
-> 'VxuFAJXVAROC'
```

`ENCRYPT()` は、少なくともいくつかのシステムでは、`str` の最初の 8 文字以外のすべてを無視します。この動作は、`crypt()` システム呼び出しを基本とした実装によって定められています。

`utf8` 以外のマルチバイト文字セットとの `ENCRYPT()` の使用は、システム呼び出しが、ストリングがゼロ バイトによって終了させられると想定するため、お薦めできません。

`crypt()` が使用しているシステムで利用できない場合 (Windows のケースなど)、`ENCRYPT()` は常に `NULL` を戻します。

- `MD5(str)`

MD5 128 ビットのチェックサムを、ストリング用に計算します。その値は 32 16進数のバイナリ ストリングとして戻され、または引数が `NULL` の場合は `NULL` が戻されます。例として、戻り値をハッシュ キーとして使用することができます。

```
mysql> SELECT MD5('testing');
-> 'ae2b1fca515949e5d54fb22b8ed95575'
```

これは、「RSA Data Security, Inc. MD5 Message-Digest Algorithm.」です。

値を大文字に変換したい場合は、[キャスト関数と演算子](#) の `BINARY` 演算子のエントリで挙げられているバイナリ ストリングの変換に関する説明をご覧ください。

このセクション始めの MD5 アルゴリズムに関する注記をご覧ください。

- `OLD_PASSWORD(str)`

セキュリティ向上のため、`PASSWORD()` の実装が変更された際に、`OLD_PASSWORD()` が MySQL に追加されました。`OLD_PASSWORD()` は `PASSWORD()` のバイナリ ストリングの旧 (4.1 の前) 実装の値を戻し、使用しているバージョンの 5.1 MySQL サーバに接続する必要のある 4.1 より前のクライアントが、自らを締め出すことなく、パスワードをリセットすることを許可することが目的です。詳細は [MySQL 4.1 のパスワードハッシュ](#) を参照してください。

- `PASSWORD(str)`

平文のパスワード `str` からパスワード スtring を計算して戻し、バイナリ スtring か、引数が `NULL` の場合は `NULL` を戻します。この関数を使用して、`user` 権限テーブルの `Password` カラムの格納の MySQL パスワードを暗号化します。

```
mysql> SELECT PASSWORD('badpwd');
-> '*AAB3E285149C0135D51A520E1940DD3263DC008C'
```

`PASSWORD()` の暗号化は一方的なものです (可逆性はない)。

`PASSWORD()` の行うパスワードの暗号化は、Unix パスワードの暗号化とは異なります。`ENCRYPT()` を参照してください。

注記 :`PASSWORD()` 関数は、MySQL サーバの認証システムによって使用されます。独自にアプリケーションでは使用しないでください。その代わりには、`MD5()` または `SHA1()` をお勧めします。また、[RFC 2195, section 2 \(Challenge-Response Authentication Mechanism \(CRAM\)\)](#) で、パスワードの扱いとアプリケーションの認証セキュリティについての詳細をご覧ください。

- `SHA1(str), SHA(str)`

String の SHA-1 160 ビットのチェックサムを、RFC 3174 (Secure Hash Algorithm) で説明されているように計算します。その値は 40 16進数のバイナリ スtring として戻され、または引数が `NULL` の場合は `NULL` が戻されます。この関数の使用例のひとつとして、ハッシュ キーとしての使用が考えられます。また、パスワードの保管のための暗号化関数としても使用できます。`SHA()` は `SHA1()` と同義です。

```
mysql> SELECT SHA1('abc');
-> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

`SHA1()` は、`MD5()` と同等に、暗号化に関してはさらに安全であると考えられています。ただし、このセクション始めの MD5 と SHA-1 アルゴリズムに関する注記をご参照ください。

- `UNCOMPRESS(string_to_uncompress)`

`COMPRESS()` 関数によって圧縮された String を非圧縮します。引数が圧縮された値でない場合は、結果は `NULL` になります。この関数では、MySQL が `zlib` のような圧縮ライブラリとコンパイルされている必要があります。その条件が満たされない場合、その戻り値は常に `NULL` になります。

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
-> 'any string'
mysql> SELECT UNCOMPRESS('any string');
-> NULL
```

- `UNCOMPRESSED_LENGTH(compressed_string)`

圧縮された String の、圧縮前の長さを戻します。

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
-> 30
```

情報関数

- `BENCHMARK(count,expr)`

`BENCHMARK()` 関数は、式 `expr` を `count` の回数だけ繰り返し実行します。MySQL がどれだけ素早く式を処理するかをこれで計ることも可能です。この結果値は常に 0 になります。この使用目的は、クエリの実行時間を報告する `mysql` クライアント内からです：

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+-----+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+-----+
| 0 |
+-----+
1 row in set (4.74 sec)
```


報告された時間は、クライアント側の経過時間であり、サーバ側の CPU 時間ではありません。[BENCHMARK\(\)](#) を複数回実行し、サーバ コンピュータにどれだけ負担がかかっているかについて、結果を解釈することをお勧めします。

- [CHARSET\(str\)](#)

ストリング引数の文字セットを戻します。

```
mysql> SELECT CHARSET('abc');
-> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
-> 'utf8'
mysql> SELECT CHARSET(USER());
-> 'utf8'
```

- [COERCIBILITY\(str\)](#)

ストリング引数の照合型変換値を戻します。

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY(USER());
-> 3
mysql> SELECT COERCIBILITY('abc');
-> 4
```

戻り値は下の表にあるような意味を持ちます。値が低いほど、優先順位は高くなります。

型変換属性	意味	例
0	明示的な照合	COLLATE 句との値
1	照合なし	異なる照合とのストリングの結合
2	暗示的な照合	カラム値、ストアド ルーチン パラメータ、またはローカル変数
3	系統定数	USER() 戻り値
4	型変換可能	リテラル ストリング
5	無視可能	NULL または NULL から引き出された式

- [COLLATION\(str\)](#)

ストリング引数の照合を戻します。

```
mysql> SELECT COLLATION('abc');
-> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
-> 'utf8_general_ci'
```

- [CONNECTION_ID\(\)](#)

接続のコネクション ID (スレッド ID) を戻します。すべての接続は、接続しているクライアントのセットの中で一意となる ID を持っています。

```
mysql> SELECT CONNECTION_ID();
-> 23786
```

- [CURRENT_USER, CURRENT_USER\(\)](#)

現在のクライアントの認証にサーバが使用した MySQL アカウントの、ユーザ名とホスト名のコンビネーションを戻します。このアカウントは、開発者のアクセス特権を確認します。[SQL SECURITY DEFINER](#) 特徴で定義されたストアド ルーチン内で、[CURRENT_USER\(\)](#) はルーチンのクリエイターを戻します。戻り値は [utf8](#) 文字セット内のストリングです。

[CURRENT_USER\(\)](#) の値は、[USER\(\)](#) の値によって異なる場合があります。

```
mysql> SELECT USER();
-> 'davida@localhost'
```

```
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user '@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
-> '@localhost'
```

この例は、クライアントが `davida` のユーザ名を指定 (`USER()` の値で示されるように) した場合でも、サーバは匿名のユーザ アカウント (`CURRENT_USER()` 値の空のユーザ名部分に見られるように) を使用してクライアントを認証するということを示しています。これが起こるひとつの原因として、`davida` の権限テーブルにアカウント リストがないことが挙げられます。

- **DATABASE()**

デフォルト (現行の) データベース名を、`utf8` 文字セット内のストリングとして戻します。デフォルトのデータベースがない場合は、**DATABASE()** は `NULL` を戻します。ストアド ルーチン内で、デフォルトのデータベースはルーチンが関連するデータベースですが、コーリング コンテキストのデフォルトのデータベースと同様である必要はありません。

```
mysql> SELECT DATABASE();
-> 'test'
```

デフォルトのデータベースがない場合は、**DATABASE()** は `NULL` を戻します。

- **FOUND_ROWS()**

SELECT 文は、サーバがクライアントに戻す行の数を制限するために、`LIMIT` 句を含んでいる場合があります。場合によっては、`LIMIT` なしでステートメントが返す行の数を知らることが望ましいですが、ステートメントを再度実行しないでください。この行のカウントを得るには、**SELECT** 文に `SQL_CALC_FOUND_ROWS` オプションを含み、その後に **FOUND_ROWS()** を実行します：

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
-> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

2 番目の **SELECT** は、最初の **SELECT** が返した、`LIMIT` 句なしで書かれた行数を示す数字を戻します。

最も最近の **SELECT** 文に `SQL_CALC_FOUND_ROWS` オプションがない場合、**FOUND_ROWS()** はその文によって戻された結果セットの行の数を戻します。

FOUND_ROWS() によって得られる行数は一過性のもので、**SELECT SQL_CALC_FOUND_ROWS** 文に続くステートメントを過ぎると取得できなくなるようになっています。この値を後で参照する必要がある場合は保存してください：

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

SELECT SQL_CALC_FOUND_ROWS を使用している場合、MySQL は完全な結果セットにいくつ行があるか計算する必要があります。しかし、結果セットをクライアントに送る必要がないため、`LIMIT` なしでクエリを再度実行するより速く行えます。

`SQL_CALC_FOUND_ROWS` および **FOUND_ROWS()** は、クエリが戻す行の数を制限する際に便利ですが、クエリを再度実行することなく完全な結果セットの行の数を決定するためにも利用できます。検索結果の他のセクションを表示するページへのリンクを含む、ページ表示を提示するウェブ スクリプトが例に挙げられます。**FOUND_ROWS()** を使用することで、残りの結果がさらに何ページを必要とするかを決定することができます。

`SQL_CALC_FOUND_ROWS` および **FOUND_ROWS()** の使用は、**UNION** の複数箇所 `LIMIT` が起こる場合があるため、簡単な **SELECT** 文よりも、**UNION** 文に対してのほうがより複雑になります。これは、**UNION** の個々の **SELECT** 文に用いられるか、または **UNION** 結果全体にグローバルに適用されます。

UNION に対する `SQL_CALC_FOUND_ROWS` の目的は、グローバルな `LIMIT` なしで返される行数を戻すことです。**UNION** との `SQL_CALC_FOUND_ROWS` の使用の条件は以下：

- `SQL_CALC_FOUND_ROWS` キーワードが、**UNION** の最初の **SELECT** に表示されている。

- **UNION ALL** が使用されている場合のみ、**FOUND_ROWS()** の値は正確。ALL なしで **UNION** が使用される場合は、複製が除去され、**FOUND_ROWS()** の値は近似のみになる。
- **UNION** で **LIMIT** が提示されない場合、**SQL_CALC_FOUND_ROWS** は無視され、**UNION** を処理するために作成された一時テーブルの行の数を返す。
- **LAST_INSERT_ID()**, **LAST_INSERT_ID(expr)**

MySQL 5.1.12 以降では、**LAST_INSERT_ID()** (引数なし) は、最も最近に実行された **INSERT** 文の結果として **AUTO_INCREMENT** カラムに正常に インサートされた、自動的に生成された最初の値を返します。**LAST_INSERT_ID()** の値は、正常にインサートされた行がない場合は、未変更のままになります。

例えば、**AUTO_INCREMENT** 値を生成する行をインサートした後は、次のようにして値を得ることができます：

```
mysql> SELECT LAST_INSERT_ID();
-> 195
```

MySQL 5.1.11 以前では、**LAST_INSERT_ID()** (引数なし) は、行が正常にインサート、または更新された場合、自動低に生成された最初の値を返します。つまり、戻された値は、テーブルに正常にインサートされなかった値である可能性があります。正常にインサートされた行がなければ、**LAST_INSERT_ID()** は 0 を返します。

LAST_INSERT_ID() の値は、**INSERT** または **UPDATE** 文のすべての行が正常である場合、全バージョンにわたって一貫するでしょう。

実行中のステートメントが、**LAST_INSERT_ID()** の値に影響をおよぼすことはありません。ひとつのステートメントで **AUTO_INCREMENT** 値を生成し、その後、独自の **AUTO_INCREMENT** カラムで行をテーブルにインサートする複数行の **INSERT** 文で、**LAST_INSERT_ID()** を照会すると仮定します。**LAST_INSERT_ID()** の値は 2 番目のステートメントに安定したまま残ります。2 番目以降の行でのその値は、以前の行の挿入に影響されません。(しかし、**LAST_INSERT_ID()** と **LAST_INSERT_ID(expr)** への参照を混ぜると、その効果は未定義になります)。

以前のステートメントがエラーを戻した場合、**LAST_INSERT_ID()** は未定義になります。トランザクション テーブルでは、ステートメントがエラーによってロールバックされる場合、**LAST_INSERT_ID()** は未定義のまま残されます。手動の **ROLLBACK** では、**LAST_INSERT_ID()** の値はトランザクションの前に復元されず、**ROLLBACK** 時点と同じまま残ります。

ストアド ルーチン (プロシージャまたは関数) もしくはトリガのボディ内で、**LAST_INSERT_ID()** の値は、これらの種類のオブジェクトの外で実行されたステートメントと同様に変化します。後に続くステートメントに参照される **LAST_INSERT_ID()** の値に基づくストアド ルーチンもしくはトリガの効果は、ルーチンの種類によって異なります：

- ストアド プロシージャが **LAST_INSERT_ID()** の値を変えるステートメントを実行する場合、変更された値はプロシージャ呼び出しに従うステートメントによって参照されます。
- 値を変更するストアド ファンクションやトリガでは、値は関数やトリガが終了した時に復元され、続くステートメントは変更された値を参照しません。

生成された ID は、接続ベースで サーバ内で保持されます。つまり、関数によって指定のクライアントに戻された値は、そのクライアントによって **AUTO_INCREMENT** カラムに影響を及ぼす最も最近のステートメントのために生成された、最初の **AUTO_INCREMENT** 値です。この値は、他のクライアントが独自の **AUTO_INCREMENT** 値を生成した場合でも、他のクライアントによって影響を受けることはありません。この動作は、各クライアントが他のクライアントの動向を気にせず、ロックやトランザクションなしで、独自の ID を呼び出せるようにします。

行の **AUTO_INCREMENT** カラムを 非「magic」値 (**NULL** でも 0 でもない値) に設定する場合、**LAST_INSERT_ID()** の値は変更されません。

重要点：単一の **INSERT** 文を使用して複数の行をインサートする場合、**LAST_INSERT_ID()** は、最初の インサートされた行のみに対して生成された値を返します。これは、他のサーバに対して同じ **INSERT** 文を簡単に再現できるようにするためです。

例：

```
mysql> USE test;
Database changed
```

```
mysql> CREATE TABLE t (
  -> id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  -> name VARCHAR(10) NOT NULL
  -> );
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> INSERT INTO t VALUES (NULL, 'Bob');
```

Query OK, 1 row affected (0.01 sec)

```
mysql> SELECT * FROM t;
```

```
+----+-----+
| id | name |
+----+-----+
| 1 | Bob |
+----+-----+
```

1 row in set (0.01 sec)

```
mysql> SELECT LAST_INSERT_ID();
```

```
+-----+
| LAST_INSERT_ID() |
+-----+
| 1 |
+-----+
```

1 row in set (0.00 sec)

```
mysql> INSERT INTO t VALUES
```

```
-> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
```

Query OK, 3 rows affected (0.00 sec)

Records: 3 Duplicates: 0 Warnings: 0

```
mysql> SELECT * FROM t;
```

```
+----+-----+
| id | name |
+----+-----+
| 1 | Bob |
| 2 | Mary |
| 3 | Jane |
| 4 | Lisa |
+----+-----+
```

4 rows in set (0.01 sec)

```
mysql> SELECT LAST_INSERT_ID();
```

```
+-----+
| LAST_INSERT_ID() |
+-----+
| 2 |
+-----+
```

1 row in set (0.00 sec)

2 番目の `INSERT` 文が 3 つの新しい行を `t` にインサートしても、これらの行の 1 番目に生成された ID は 2 であり、次の `SELECT` 文に対して `LAST_INSERT_ID()` が返す値も同じです。

`INSERT IGNORE` を使用して行を無視する場合は、`AUTO_INCREMENT` カウンタは増分されず、行がインサートされなかったことを反映して、`LAST_INSERT_ID()` は 0 を返します。

`expr` が `LAST_INSERT_ID()` への引数として与えられる場合、その引数の値は関数によって戻され、`LAST_INSERT_ID()` によって戻される次の値として記憶されます。これによってシークエンスのシミュレーションをすることも可能です：

1. テーブルを作成してシークエンス カウンタを保留にし、初期化：

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

2. テーブルを使用して、次のようにシークエンス番号を生成：

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

UPDATE 文はシーケンス カウンタを増分し、**LAST_INSERT_ID()** への次の呼び出しが更新された値を戻すようにします。**SELECT** 文はその値を引き出します。**mysql_insert_id()** C API 関数は、値の入手に使用することもできます。詳細は **mysql_insert_id()** を参照してください。

LAST_INSERT_ID() を呼び出さずにシーケンスを生成することはできますが、このように関数を使用することの利点は、ID 値が自動的に生成された最後の値として保持されることです。独自のシーケンス値を生成する他のクライアントと互いに影響しあうことなく、複数のクライアントが **UPDATE** 文を発行し、**UPDATE** 文 (または **mysql_insert_id()**) でそれぞれのシーケンス値を取得することができるため、マルチユーザでも安全です。

mysql_insert_id() は **INSERT** および **UPDATE** 文の後にのみ更新され、**SELECT** もしくは **SET** のような他の SQL 文を実行した後に、C API 関数を使用して **LAST_INSERT_ID(expr)** の値を引き出すことはできないのでご注意ください。

- **ROW_COUNT()**

ROW_COUNT() は、先行するステートメントによって更新、インサート、または削除された行の数を戻します。これは **mysql** クライアントが表示する行のカウンタおよび、**mysql_affected_rows()** C API 関数からの値と同じです。

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          3 |
+-----+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```

- **SCHEMA()**

この関数は **DATABASE()** のシノニムです。

- **SESSION_USER()**

SESSION_USER() は **USER()** のシノニムです。

- **SYSTEM_USER()**

SYSTEM_USER() は **USER()** のシノニムです。

- **USER()**

現行の MySQL ユーザ名とホスト名を、**utf8** 文字セット内のストリングとして戻します。

```
mysql> SELECT USER();
-> 'davida@localhost'
```

その値はサーバへの接続時に指定したユーザ名と、接続したホストからのクライアントを示します。値は **CURRENT_USER()** によって異なる場合があります。

次のように、ユーザ名の部分だけを抽出することができます：

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);
-> 'davida'
```

- `VERSION()`

MySQL サーバのバージョンを示す文字列を返します。その文字列は、`utf8` 文字セットを使用します。

```
mysql> SELECT VERSION();
-> '5.1.15-beta-standard'
```

`-log` で終わるバージョン文字列は、ロギングが有効になっていることを表しています。

その他の関数

- `DEFAULT(col_name)`

テーブル カラムにデフォルト値を返します。カラムがデフォルト値を持たない場合はエラーが発生します。

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)`

数字 `X` を `'#,###,###.##'` のようにフォーマットし、`D` 少数位まで丸め、その結果を文字列として返します。詳細は、[文字列関数](#) をご覧ください。

- `GET_LOCK(str,timeout)`

`timeout` 秒の待機時間を使用して、文字列 `str` によって与えられた名前でのロックの獲得を試みます。ロックの獲得が成功した場合は `1` を返し、試行が時間切れになった場合 (例えば、他のクライアントがすでにその名前をロックしている場合) は `0` を、または、エラーが発生 (メモリの不足、または `mysqldadmin kill` によるスレッドの停止) した場合は `NULL` を返します。`GET_LOCK()` でロックを獲得した場合、`RELEASE_LOCK()` を実行した時、新しい `GET_LOCK()` を実行した時、または接続が切断された時 (正常または異常な終了を問わず) にリリースされます。`GET_LOCK()` でロックを獲得した場合は、トランザクションと対話しないようにしてください。これは、トランザクションをコミットしても、トランザクション中に獲得されたそれらのロックはリリースされないためです。

この関数は、アプリケーションロックの実装、またはレコードロックのシミュレートに使用することができます。名前はサーバ全体に渡ってロックされます。ひとつのクライアントが名前をロックすると、`GET_LOCK()` が他のクライアントからの同じ名前の使用要求をブロックします。これによって、与えられたロック名を承認したクライアントが、名前を使用して協調任意型のロックを行うことができます。ただし同時に、協調するクライアントのセットにないクライアントも、過失にせよ故意にせよ、名前をロックすることができることになり、協調するクライアントがその名前を使用できなくなりますので注意してください。それを防ぐひとつの方法は、データベース固有、またはアプリケーション固有のロック名を使用することです。フォーム `db_name.str` または `app_name.str` のロック名を使用するのもその一例です。

```
mysql> SELECT GET_LOCK('lock1',10);
-> 1
mysql> SELECT IS_FREE_LOCK('lock2');
-> 1
mysql> SELECT GET_LOCK('lock2',10);
-> 1
mysql> SELECT RELEASE_LOCK('lock2');
-> 1
mysql> SELECT RELEASE_LOCK('lock1');
-> NULL
```

2 番目の `RELEASE_LOCK()` 呼び出しは、ロック `'lock1'` が 2 番目の `GET_LOCK()` 呼び出しによって自動的にリリースされるため、`NULL` を返します。

注記: クライアントが、他のクライアントによってすでに確保されたロックの獲得を試みると、`timeout` 引数によってそのクライアントはブロックされます。ブロックされたクライアントが停止する場合、そのスレッドはロックがタイムアウトを要求するまで停止しません。これは既知のバグです。

- `INET_ATON(expr)`

ネットワークアドレスのドット形式のクワッド表示が文字列として与えられ、アドレスの数値を示す整数を返します。アドレスは 4 または 8 バイトのアドレスである可能性があります。

```
mysql> SELECT INET_ATON('209.207.224.40');
-> 3520061480
```


生成される数字は常にネットワークバイト順になります。例えばこの例のように、数字は $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$ として計算されます。

また `INET_ATON()` は、短縮形式の IP アドレスを理解します：

```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
-> 2130706433, 2130706433
```

注記 :`INET_ATON()` によって記憶数値が生成される場合は、`INT UNSIGNED` カラムの使用を推奨します。(符号付の) `INT` カラムを使用すると、最初のオクテットが 127 以上である IP アドレスに対応する値は正しく保存されません。詳細は [数値タイプ](#) を参照してください。

- `INET_NTOA(expr)`

数字のネットワークアドレス (4 または 8 バイト) を与えられ、アドレスのドット形式のクワッド表示をストリングとして戻します。

```
mysql> SELECT INET_NTOA(3520061480);
-> '209.207.224.40'
```

- `IS_FREE_LOCK(str)`

`str` と名付けられたロックが使用可能か (ロックされていないか) 調べます。ロックが使用可能 (まだ使用されていない) 場合は `1` を、すでに使用されている場合は `0` を返し、エラーが発生した場合 (引数が不正確、など) は `NULL` を戻します。

- `IS_USED_LOCK(str)`

`str` と名付けられたロックが使用されているか (ロックされているか) 調べます。ロックされている場合は、ロックを持っているクライアントの接続識別子を戻します。ロックされていない時は `NULL` を戻します。

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

この関数は、マスター / スレーブの同期化のコントロールに役立ちます。スレーブがマスター ログで指定された位置まで読み取り、すべてのアップデートを適用するまでブロックします。戻り値は、指定の位置まで進むまでスレーブが待たなければいけないログ イベントの数です。この関数は、スレーブ SQL スレッドが開始されていない、スレーブのマスター情報が初期化されていない、引数が正しくない、またはエラーが発生、という場合は `NULL` を戻します。タイムアウトの時間を越えると `-1` が戻されます。`MASTER_POS_WAIT()` の待機中にスレーブ SQL スレッドが停止すると、関数は `NULL` を戻します。スレーブが指定の位置を過ぎたら、関数はただちに戻しを行います。

`timeout` 値が指定された場合、`timeout` の秒数を経過したのち `MASTER_POS_WAIT()` は待機をやめます。`timeout` は 0 より大きい数字でなければなりません。0 または負数の `timeout` では待機時間なしになります。

- `NAME_CONST(name,value)`

与えられた値を戻します。結果セットのカラムの生成に使用された場合、`NAME_CONST()` が、カラムが与えられた名前を持つ原因になります。

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|    14 |
+-----+
```

この関数は MySQL 5.0.12 から、内部使用のみの目的で追加されました。[ストアドルーチンとトリガのバイナリログ](#) で説明されているように、ローカル ルーチン変数への参考を含むストアドルーチンからのステートメントを書く時にサーバが使用します。`mysqlbinlog` からの出力にこの関数が含まれる場合があります。

- `RELEASE_LOCK(str)`

`GET_LOCK()` で獲得されたストリング `str` によって名付けられたロックをリリースします。ロックがリリースされた場合は `1` を、ロックがこのスレッドによって確立されていない場合 (その場合ロックはリリースされません) は

0 を、そして、名前付きのロックが存在しない場合は `NULL` を返します。`GET_LOCK()` への呼び出しで獲得、またはすでにリリースされていない限り、ロックは存在しません。

`DO` 文は `RELEASE_LOCK()` との使用に便利です。詳細は [DO 構文](#) を参照してください。

- `SLEEP(duration)`

`duration` 引数で指定され秒数間だけ休止 (一時停止) し、その後 0 を返します。`SLEEP()` が妨げられた場合は 1 を返します。継続時間はマイクロ秒で指定された少数部を持つ場合があります。

- `UUID()`

1977 年 10 月に、The Open Group が発行した「DCE 1.1:Remote Procedure Call」(Appendix A) CAE (Common Applications Environment) Specifications (Document Number C706、<http://www.opengroup.org/public/pubs/catalog/c706.htm>) に基づいて生成された Universal Unique Identifier (UUID) を返します。

UUID は、スペースおよび時間においてグローバルに一意の数字としてデザインされています。`UUID()` へのふたつの呼び出しは、互いに接続されていない別々のコンピュータ上で行った場合でも、それぞれ異なるふたつの値を生成することが想定されます。

A UUID is a 128-bit number represented by a string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

- 最初の 3 桁はタイムスタンプから生成されます。
- 4 番目の数字は、タイムスタンプ値が単調整を失う場合 (例えば、夏時間の影響などで) に備えて、一時的な一意性を保ちます。
- 5 番目の数字は、スペースの一意性を提供する IEEE 802 ノード番号です。後者が利用できない場合 (例えば、ホスト コンピュータが Ethernet カードを持たない、または使用のオペレーション システムでインターフェイスのハードウェア アドレスを見つける方法が分からない、など)、ランダムな数字で代替されます。その場合、スペースの一意性は保証されません。しかしそれでも、不調和が起こる可能性は非常に低いと思われます。

現在、インターフェイスの MAC アドレスは、FreeBSD と Linux でのみ考慮されています。他のオペレーション システムでは、MySQL はランダムに生成された 48 ビットの数字を使用します。

```
mysql> SELECT UUID();
-> '6ccd780c-baba-1026-9564-0040f4311e29'
```

`UUID()` はまだ複製との作業は不可能ですのでご注意ください。

- `VALUES(col_name)`

`INSERT ... ON DUPLICATE KEY UPDATE` 文では、`UPDATE` 句の `VALUES(col_name)` 関数を使用して、ステートメントの `INSERT` 部分からのカラム値を参照することができます。つまり、`UPDATE` 句内の `VALUES(col_name)` は、複製キーとの衝突もなく、インサートされる `col_name` 値を参照するということです。この関数は複数行のインサートにおいて特に便利です。`VALUES()` 関数は、`INSERT ... ON DUPLICATE KEY UPDATE` 文においてのみ有用で、その他では `NULL` を返します。`INSERT ... ON DUPLICATE KEY UPDATE` [構文](#) 参照。

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
-> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

GROUP BY 句との関数および修飾子の使用

GROUP BY (集約) 関数

このセクションでは、値のセットを演算するグループ (集約) 関数について説明します。特別に説明されていない限り、グループ関数は `NULL` 値を無視します。

`GROUP BY` 句を含まないステートメントでグループ関数を使用する場合、すべての行をグループ分けするのと同様の効果になります。

数値引数では、分散値と標準偏差関数が **DOUBLE** 値を返します。**SUM()** および **AVG()** 関数は、高精度値引数 (整数または **DECIMAL**) に対して **DECIMAL** 値を返し、近似値引数 (**FLOAT** または **DOUBLE**) に対して **DOUBLE** 値を返します。

SUM() および **AVG()** 集約関数は、一時値とはうまく作動しません。(値を数字に変換し、最初の非数値文字の後のパートを失います)。この問題を回避するには、数値ユニットを変換し、集約演算を行い、一時値に変換しなおすという方法があります。例：

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

- **AVG([DISTINCT] expr)**

expr の平均値を返します。**expr** の固有値の平均を返すのに、**DISTINCT** オプションを使用することができます。

一致する行がない場合、**AVG()** は **NULL** を返します。

```
mysql> SELECT student_name, AVG(test_score)
-> FROM student
-> GROUP BY student_name;
```

- **BIT_AND(expr)**

expr 内のすべてのビットの、ビット単位の **AND** を返します。計算は 64 ビット (**BIGINT**) の精度で行われます。

この関数は、一致する行がない場合は、**18446744073709551615** を返します。(これは、すべてのビットが 1 に設定された、符号なしの **BIGINT** 値の値です。)

- **BIT_OR(expr)**

expr 内のすべてのビットの、ビット単位の **OR** を返します。計算は 64 ビット (**BIGINT**) の精度で行われます。

この関数は、一致する行がない場合は、**0** を返します。

- **BIT_XOR(expr)**

expr 内のすべてのビットの、ビット単位の **XOR** を返します。計算は 64 ビット (**BIGINT**) の精度で行われます。

この関数は、一致する行がない場合は、**0** を返します。

- **COUNT(expr)**

SELECT 分によって引き出された行の、非 **NULL** 値の数を返します。結果は **BIGINT** 値になります。

一致する行がない場合、**COUNT()** は **0** を返します。

```
mysql> SELECT student.student_name,COUNT(*)
-> FROM student,course
-> WHERE student.student_id=course.student_id
-> GROUP BY student_name;
```

COUNT(*) は、**NULL** 値を含む含まざるに関わらず、引き出された行の数を返すという点でやや異なります。

COUNT(*) は、**SELECT** がひとつのテーブルから検索し、他のカラムは引き出されず、また **WHERE** カラムがない場合、きわめて素早く戻すよう最適化されています。例：

```
mysql> SELECT COUNT(*) FROM student;
```

この最適化は、正確な行の数がこの保存エンジンに保管されており、素早いアクセスが可能のため、**MyISAM** テーブルにのみ適用します。**InnoDB** をはじめとするトランザクション保存エンジンに関しては、正確な行の数を保存するのは、複数のトランザクションが起こって、それぞれが行の係数に影響をおよぼす場合があるため、より困難になります。

- **COUNT(DISTINCT expr,[expr...])**

異なる非 **NULL** 値の数を返します。

一致する行がない場合、`COUNT(DISTINCT)` は 0 を返します。

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

MySQL では、式のリストを提供することにより、`NULL` を含まない、異なる式のコンビネーションの数を得ることができます。標準 SQL では、`COUNT(DISTINCT ...)` 内で、すべての式の連結を行わなければなりません。

- `GROUP_CONCAT(expr)`

この関数は、グループからの連結された非 `NULL` 値を伴うストリング結果を返します。非 `NULL` 値がない場合は `NULL` を返します。全構文は次の通りです：

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
             [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
-> GROUP_CONCAT(test_score)
-> FROM student
-> GROUP BY student_name;
```

または：

```
mysql> SELECT student_name,
-> GROUP_CONCAT(DISTINCT test_score
-> ORDER BY test_score DESC SEPARATOR '')
-> FROM student
-> GROUP BY student_name;
```

MySQL では、式のコンビネーションの連結された値を得ることができます。`DISTINCT` を使用することで、重複した値を除くことが可能です。結果の値をソートしたい場合は、`ORDER BY` 句を使用してください。逆順でソートするには、`DESC` (降順) キーワードを、`ORDER BY` 句のソートするカラムの名前に加えてください。デフォルトでは昇順になっています。これは、`ASC` を使うことで明示的に指定することができます。`SEPARATOR` の後には、結果の値の間に挿入されるべきストリング値が続きます。デフォルトはコンマ (',') です。`SEPARATOR "` を使用すると、セパレータを一挙に取り除くことができます。

`group_concat_max_len` システム環境変数は、許可された最大の長さに設定することができます。(デフォルト値は 1024)。ランタイムでこれを行う構文は次です。`val` は符号なしの整数になります：

```
SET [SESSION | GLOBAL] group_concat_max_len = val;
```

最大の長さが設定された場合、結果はその最大の長さに切り詰められます。

`GROUP_CONCAT()` によって戻されるタイプは、`group_concat_max_len` が 512 より大きい場合意外は常に `VARCHAR` になります。512 を越える場合は `BLOB` になります。

`CONCAT()` および `CONCAT_WS()` も併せてご覧ください：[文字列関数](#)。

- `MIN([DISTINCT] expr), MAX([DISTINCT] expr)`

`expr` の最小または最大値を返します。`MIN()` および `MAX()` はストリングの引数を取る場合があります。その場合、それらは最小または最大のストリング値を返します。[MySQLにおけるインデックスの使用](#) をご覧ください。`DISTINCT` キーワードで `expr` の固有の値の最小または最大を検出できますが、その場合、`DISTINCT` を省略した場合と同じ結果を生成します。

一致する行がない場合、`MIN()` および `MAX()` は `NULL` を返します。

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
-> FROM student
-> GROUP BY student_name;
```

`MIN()`、`MAX()`、および他の集約関数に関しては、MySQL は現在、`ENUM` と `SET` カラムを、セット内でのそれらのストリングの相対位置によってではなく、ストリング値によって比較しています。これは、`ORDER BY` がそれらをどう比較するかによって異なります。この点は、将来の MySQL リリースに反映される予定です。

- **STD(expr) STDDEV(expr)**

expr の母標準偏差を戻します。これは標準 SQL へのエクステンションです。この関数の **STDDEV()** フォームは、Oracle との相互性のために提供されています。標準 SQL 関数 **STDDEV_POP()** を代わりに使用することも可能です。

これらの関数は、一致する行がない場合は、**NULL** を戻します。

- **STDDEV_POP(expr)**

expr の母標準偏差 (**VAR_POP()** の平方根) を戻します。**STD()** または **STDDEV()** を使用することもできます。これらは同等ですが標準 SQL ではありません。

一致する行がない場合、**STDDEV_POP()** は **NULL** を戻します。

- **STDDEV_SAMP(expr)**

expr の試料標準偏差 (**VAR_SAMP()** の平方根) を戻します。

一致する行がない場合、**STDDEV_SAMP()** は **NULL** を戻します。

- **SUM([DISTINCT] expr)**

expr の集計を戻します。返しセットが行を持たない場合、**SUM()** は **NULL** を戻します。MySQL 5.1 で **DISTINCT** を使用して、**expr** の重複しない値のみを集計することができます。

一致する行がない場合、**SUM()** は **NULL** を戻します。

- **VAR_POP(expr)**

expr の母標準分散を戻します。行をサンプルではなく全母集団としてとらえ、行の数を分母として得ます。また、**VARIANCE()** を使用することもできます。これは同等ですが標準 SQL ではありません。

一致する行がない場合、**VAR_POP()** は **NULL** を戻します。

- **VAR_SAMP(expr)**

expr のサンプル分散を戻します。この分母は行の数から 1 をひいたものです。

一致する行がない場合、**VAR_SAMP()** は **NULL** を戻します。

- **VARIANCE(expr)**

expr の母標準分散を戻します。これは標準 SQL へのエクステンションです。標準 SQL 関数 **VAR_POP()** を代わりに使用することも可能です。

一致する行がない場合、**VARIANCE()** は **NULL** を戻します。

GROUP BY 修飾子

GROUP BY 句は、要約出力に行を追加する **WITH ROLLUP** 修飾子を許可します。これらの行は、高レベル (または超集約) の要約演算を表します。したがって **ROLLUP** は、複数レベルでの解析で質問に単一エリで答えることを可能にします。これは、例えば、OLAP (Online Analytical Processing) 演算へのサポートに使用することができます。

Sales と名付けられたテーブルが、売り上げの収益性を記録するために、**year**、**country**、**product**、そして **profit** のカラムを持つ場合：

```
CREATE TABLE sales
(
  year   INT NOT NULL,
  country VARCHAR(20) NOT NULL,
  product VARCHAR(32) NOT NULL,
  profit INT
);
```

テーブルのコンテンツを、次のように簡単な **GROUP BY** で年ごとに要約することができます：

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
```

```
+-----+
| year | SUM(profit) |
+-----+
| 2000 |      4525 |
| 2001 |      3010 |
+-----+
```

この出力は各年の収益合計を表示しますが、すべての年にわたる収益合計を確認したい場合は、各値を自分で合計するか、別のクエリを実行する必要があります。

または、単一クエリで両方のレベルの解析を提供する **ROLLUP** を使用することもできます。**GROUP BY** 句に **WITH ROLLUP** 修飾子を加えると、クエリがすべての年にわたる総合計の値を示す行を生成します：

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
```

```
+-----+
| year | SUM(profit) |
+-----+
| 2000 |      4525 |
| 2001 |      3010 |
| NULL |      7535 |
+-----+
```

総合計の超集約ラインは、**year** カラムの値 **NULL** によって特定されます。

ROLLUP は、複数の **GROUP BY** カラムがある場合に、さらに複雑な効果をあらわします。この場合、「break」（値の変更）が最後のグループ分けのカラムにある度に、クエリは追加の超集約要約行を生成します。

例えば、**ROLLUP** なしの場合、**year**、**country**、そして **product** を基にした **sales** テーブルの要約はこのようになる場合があります：

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product;
```

```
+-----+
| year | country | product | SUM(profit) |
+-----+
| 2000 | Finland | Computer |      1500 |
| 2000 | Finland | Phone   |       100 |
| 2000 | India   | Calculator |      150 |
| 2000 | India   | Computer |     1200 |
| 2000 | USA     | Calculator |       75 |
| 2000 | USA     | Computer |     1500 |
| 2001 | Finland | Phone   |        10 |
| 2001 | USA     | Calculator |       50 |
| 2001 | USA     | Computer |     2700 |
| 2001 | USA     | TV       |       250 |
+-----+
```

この出力は year/country/product レベルでのみの解析での要約値を示します。**ROLLUP** が加えられる時、クエリは複数の追加行を生成します：

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP;
```

```
+-----+
| year | country | product | SUM(profit) |
+-----+
| 2000 | Finland | Computer |      1500 |
| 2000 | Finland | Phone   |       100 |
| 2000 | Finland | NULL    |     1600 |
| 2000 | India   | Calculator |      150 |
| 2000 | India   | Computer |     1200 |
| 2000 | India   | NULL    |     1350 |
| 2000 | USA     | Calculator |       75 |
| 2000 | USA     | Computer |     1500 |
| 2000 | USA     | NULL    |     1575 |
| 2000 | NULL    | NULL    |      4525 |
| 2001 | Finland | Phone   |        10 |
| 2001 | Finland | NULL    |         10 |
| 2001 | USA     | Calculator |       50 |
| 2001 | USA     | Computer |     2700 |
+-----+
```



```

| 2001 | USA   | TV       | 250 |
| 2001 | USA   | NULL    | 3000 |
| 2001 | NULL  | NULL    | 3010 |
| NULL | NULL  | NULL    | 7535 |
+-----+-----+-----+

```

このクエリでは、**ROLLUP** 句を加えると、ひとつでなく、よっつの解析のレベルでの要約情報が出力に含まれます。以下が **ROLLUP** 出力の解釈方法です：

- 指定の year と country に対する product 行の各セットに続き、追加の要約行がすべての product の合計を示して生成されます。これらの行は **NULL** に対して **product** カラム セットを備えています。
- 指定の year に対する product 行の各セットに続き、追加の要約行がすべての country と product の合計を示して生成されます。これらの行は **NULL** に対して **country** および **products** カラム セットを備えています。
- そして最後に、他のすべての行に続き、追加の要約行がすべての year 、 country 、 および product の総合系を示して生成されます。この行は **NULL** に対して **year** 、 **country** および **products** カラム セットを備えています。

ROLLUP を使用する際のその他の注意

次の項目は、**ROLLUP** の MySQL 実装特定の動作をリストしたものです：

ROLLUP を使用する場合、**ORDER BY** 句を同時に使用して結果をソートすることはできません。つまり、**ROLLUP** と **ORDER BY** は互いに排し合うということになります。しかし、ソートの順番をいくらかコントロールすることは可能です。MySQL の **GROUP BY** が結果をソートし、そして明示的な **ASC** および **DESC** キーワードを **GROUP BY** 内で名付けられたカラムと使用し、各カラムのソート順を指定することができます。(しかし、**ROLLUP** によって加えられた高レベルな要約行は、ソート順に関わらず、それらが計算された行の後に現れます。)

LIMIT はクライアントに戻される行の数を限定するのに使用できます。**LIMIT** は **ROLLUP** の後に適用され、それによって **ROLLUP** によって追加された行に対しての制限が適用されます。例：

```

mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP
-> LIMIT 5;
+-----+-----+-----+
| year | country | product | SUM(profit) |
+-----+-----+-----+
| 2000 | Finland | Computer | 1500 |
| 2000 | Finland | Phone   | 100 |
| 2000 | Finland | NULL    | 1600 |
| 2000 | India   | Calculator | 150 |
| 2000 | India   | Computer | 1200 |
+-----+-----+-----+

```

LIMIT を **ROLLUP** と使用すると、超集約行を理解するにはコンテキストが少ないため、より解釈が難しい結果を生成する場合があります。

各超集約行の **NULL** 指示子は、行がクライアントに送られた時に生成されます。サーバは、**GROUP BY** 句で名付けられたカラムを、変更値を持つ左側のものに続いて調査します。それらの名前に語彙がマッチした名称を持つ、結果セット内のすべてのカラムには、その値が **NULL** に設定されます。(カラム番号によってグループ分けのカラムを指定する場合、サーバは番号によってどのカラムを **NULL** に設定するかを確認します。)

超集約行の **NULL** 値は、クエリの処理の非常に遅い時点で結果セットに配置されるため、それらをクエリそのものの中で **NULL** 値としてテストすることはできません。例えば、クエリに **HAVING product IS NULL** を追加して、超集約行以外のすべての出力から除くことはできません。

一方、**NULL** 値はクライアント側には **NULL** として表れ、MySQL クライアント プログラミング インターフェイスのいずれかを使用してテストすることができます。

非常時フィールドとの GROUP BY および HAVING

MySQL は **GROUP BY** の使用を拡張し、**GROUP BY** 句には現れない **SELECT** リストでの、超集約カラムまたは計算の使用を可能にします。この機能を利用して、不要なカラムのソートやグループ分けを避けることで、性能を改善することができます。例えば、次のクエリでは、**customer.name** のグループ分けをする必要がありません：

```
SELECT order.custid, customer.name, MAX(payments)
FROM order, customer
WHERE order.custid = customer.custid
GROUP BY order.custid;
```

標準 SQL では、**GROUP BY** 句に `customer.name` を加える必要があります。MySQL では、この名前は二重化しています。

GROUP BY 部から省略したカラムがグループ内で一定していない場合は、この機能を使用しないでください。サーバはいかなる値もグループから自由に戻すことができ、すべての値が同じでない限り、結果は不確定です。

同様の MySQL 拡張機能が **HAVING** 句に適用されます。SQL の基準では、**GROUP BY** 句で検出されないカラムで、集約関数で囲まれていないものを、**HAVING** 句で名付けることはできません。MySQL では、それらのカラムで計算を簡易化することができます。この拡張は、グループ分けされていないカラムが、同じグループに関する値を持っていることを前提としています。それ以外では、結果は不確定になります。

ONLY_FULL_GROUP_BY SQL モードが有効になっている場合、**GROUP BY** への MySQL 拡張は適用されません。これは、**GROUP BY** 句で名付けられていないカラムは、集約関数で使われていない場合、**SELECT** リスト、または **HAVING** 句で利用することができません。

選択リストの拡張も、**ORDER BY** に適用できます。つまり、**GROUP BY** 句に表れない **ORDER BY** 句の非集約カラムまたは計算を使用することができます。この拡張は、**ONLY_FULL_GROUP_BY** SQL モードが有効になっている時は適用しません。

いくつかのケースでは、**MIN()** および **MAX()** を使用して、ユニークなもの意外でも特定のカラム値を取得することができます。次の例は、`sort` カラムでの最小値を含む行からの `column` の値を与えます：

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,''),column)),7)
```

[The Rows Holding the Group-wise Maximum of a Certain Column](#) 参照。

標準 SQL に準じる場合は、**GROUP BY** 句で式を使用することはできないのでご注意ください。式にエイリアスを使用することで、この制限を回避することが可能です：

```
SELECT id,FLOOR(value/100) AS val
FROM tbl_name
GROUP BY id, val;
```

MySQL は **GROUP BY** 句での式の使用を許可していません。例：

```
SELECT id,FLOOR(value/100)
FROM tbl_name
GROUP BY id, FLOOR(value/100);
```

第12章 SQL ステートメント構文

目次

データ定義ステートメント	650
ALTER DATABASE 構文	650
ALTER TABLE 構文	650
ALTER LOGFILE GROUP 構文	657
ALTER TABLESPACE 構文	658
ALTER SERVER 構文	659
CREATE DATABASE 構文	659
CREATE INDEX 構文	660
CREATE TABLE 構文	661
CREATE LOGFILE GROUP 構文	675
CREATE TABLESPACE 構文	676
CREATE SERVER 構文	677
DROP DATABASE 構文	677
DROP INDEX 構文	678
DROP TABLE 構文	678
DROP LOGFILE GROUP 構文	679
DROP TABLESPACE 構文	679
DROP SERVER 構文	679
RENAME DATABASE 構文	680
RENAME TABLE 構文	680
データ取り扱いステートメント	681
DELETE 構文	681
DO 構文	683
HANDLER 構文	683
INSERT 構文	684
LOAD DATA INFILE 構文	690
REPLACE 構文	697
SELECT 構文	698
サブクエリ構文	710
TRUNCATE 構文	719
UPDATE 構文	719
MySQL ユーティリティ ステートメント	721
DESCRIBE 構文	721
HELP 構文	721
USE 構文	723
MySQL トランザクションとロッキング関連のステートメント	724
START TRANSACTION、COMMIT、そして ROLLBACK 構文	724
ロールバックできないステートメント	726
暗黙のコミットを引き起こすステートメント	726
SAVEPOINT と ROLLBACK TO SAVEPOINT 構文	726
LOCK TABLES と UNLOCK TABLES 構文	727
SET TRANSACTION 構文	729
XA トランザクション	730
データベース管理ステートメント	733
アカウント管理ステートメント	733
テーブル メンテナンス ステートメント	742
SET 構文	747
SHOW 構文	752
その他の管理ステートメント	776
複製ステートメント	780
マスタ サーバをコントロールする SQL ステートメント	780
スレーブ サーバをコントロールする SQL ステートメント	783
プリペアド ステートメントの為の SQL 構文	790

この章では、MySQL にサポートされている SQL ステートメントの構文について説明します。後半の章に、ステートメントについての追加説明があります。

- [EXPLAIN](#) ステートメントについては [6章最適化](#) で紹介されています。
- ストアド ルーチンの書き込みのステートメントについては [17章ストアドプロシージャとファンクション](#) で紹介されています。
- トリガの書き込みのステートメントについては [18章トリガ](#) で紹介されています。
- ビューに関連するステートメントについては [20章ビュー](#) で紹介されています。
- イベント スケジュールのステートメントについては [19章Event Scheduler](#) で紹介されています。

データ定義ステートメント

ALTER DATABASE 構文

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification [alter_specification] ...

alter_specification:
    [DEFAULT] CHARACTER SET charset_name
    | [DEFAULT] COLLATE collation_name
```

[ALTER DATABASE](#) でデータベースの全体的な特徴を変更する事ができます。これらの特徴は、データベースディレクトリの `db.opt` ファイルに格納されています。[ALTER DATABASE](#) を利用する為には、データベース上の [ALTER](#) 権限が必要です。[ALTER SCHEMA](#) は [ALTER DATABASE](#) の同義語です。

[CHARACTER SET](#) 条項はデフォルト データベースの文字セットを変更します。[COLLATE](#) 条項はデフォルト データベースの照合を変更します。[9章キャラクタセットサポート](#) は文字セットと照合名を検討します。

ステートメントがデフォルト データベースに適応する場合、データベース名は省略する事ができます。

MySQL Enterprise

```
#####MySQL #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

ALTER TABLE 構文

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    table_option ...
    | ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
    | ADD [COLUMN] (column_definition,...)
    | ADD {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
    | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
    | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX|KEY] [index_name] [index_type] (index_col_name,...)
    | ADD FULLTEXT [INDEX|KEY] [index_name] (index_col_name,...)
        [WITH PARSER parser_name]
    | ADD SPATIAL [INDEX|KEY] [index_name] (index_col_name,...)
    | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
    | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
    | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
    | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
    | DROP [COLUMN] col_name
    | DROP PRIMARY KEY
    | DROP {INDEX|KEY} index_name
    | DROP FOREIGN KEY fk_symbol
    | DISABLE KEYS
```

```

| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name [, col_name] ...
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| PARTITION BY partition_options
| ADD PARTITION (partition_definition)
| DROP PARTITION partition_names
| COALESCE PARTITION number
| REORGANIZE PARTITION partition_names INTO (partition_definitions)
| ANALYZE PARTITION partition_names
| CHECK PARTITION partition_names
| OPTIMIZE PARTITION partition_names
| REBUILD PARTITION partition_names
| REPAIR PARTITION partition_names
| REMOVE PARTITIONING

index_col_name:
  col_name [(length)] [ASC | DESC]

index_type:
  USING {BTREE | HASH}

```

ALTER TABLE で既存テーブルの構造を変更する事ができます。例えば、カラムの追加や削除、インデックスの作成や破壊、既存カラム タイプの変更、またはカラムやテーブル自体の名前の変更をする事ができます。テーブルや、テーブル タイプのコメントを変更する事もできます。

多くの許容される変更の構文は、**CREATE TABLE** ステートメントの条項に似ています。詳細については、**CREATE TABLE 構文** をご参照ください。

ストレージ エンジンがその操作をサポートしていないテーブルに対しては、いくつかの操作の結果は警告になってしまいかもしれません。これらの警告は **SHOW WARNINGS** で表示する事ができます。詳しくは **SHOW WARNINGS 構文** を参照してください。

ほとんどの場合、**ALTER TABLE** は元テーブルのテンポラリ コピーを作成する事で起動します。そのコピー上で変更が行われ、その後元テーブルが削除されて新しいテーブルがリネームされます。**ALTER TABLE** が実行している間、他のクライアントが元テーブルを読む事ができます。新しいテーブルの準備ができるまで更新と書き込みは止められ、その後更新に失敗する事なく新しいテーブルに自動的にリダイレクトされます。

テンポラリ テーブルが必要ない場合がいくつかあります。

- もし、他のオプション無しで **ALTER TABLE tbl_name RENAME TO new_tbl_name** を利用すると、MySQLはテーブル **tbl_name** に対応する全てのファイルをリネームします。(テーブルをリネームする為に **RENAME TABLE** ステートメントを利用する事もできます。詳しくは **RENAME TABLE 構文** を参照してください。)リネームされたテーブルに与えられた権限は、新しい名前に移動しません。それらは手動で変更しなければいけません。
- ALTER TABLE ...ADD PARTITION** はMySQLクラスタ以外にテンポラリテーブルを作成しません。**RANGE** や **LIST** パーティションの **ADD** や **DROP** 操作は直接の操作、またはそれに近い操作です。**HASH** や **KEY** パーティションの **ADD** や **COALESCE** 操作は変更されたパーティション間でデータをコピーします。**LINEAR HASH/KEY** が利用されていない限り、これは新しいテーブルを作成するのとはほとんど変わりません。(操作はパーティションごとに行われますが。) **REORGANIZE** 操作は変更されたパーティションだけをコピーし、変更されていない物には関係しません。

そうでない場合は、そのデータが必ずコピーされる必要がある訳ではなくても、MySQL がテンポラリ テーブルを作成します。(カラム名を変更した時と同じように) **MyISAM** テーブルは、高い値に **mysam_sort_buffer_size** システム変数を設定する事で、インデックスの再作成操作のスピードを上げる事ができます。(これは変更プロセスの中で一番遅い操作です。)

- ALTER TABLE** を利用するには、テーブルに **ALTER**、**INSERT**、そして **CREATE** 権限が必要です。
- IGNORE** はスタンダード SQL の MySQL 拡張子です。これは、新しいテーブルのユニーク キーに複製があったり、ストリクト モードが有効時に警告が出たりした時に **ALTER TABLE** がどのように機能するかコントロールします。もし **IGNORE** が指定されなければ、複製キー エラーが起きた時、コピーは異常終了し、元に戻されます。もし **IGNORE** が指定されると、ユニーク キーに複製された行の、最初の行だけが使用され、それ以外の相反する行は削除されます。不正な値は、適合する許容値に一番近い値まで切り捨てられます。

- `table_option` は `CREATE TABLE` ステートメントで利用する事ができる種類のテーブル オプションを意味します。(CREATE TABLE 構文 に全てのテーブル オプションがリストされています。)これは、`ENGINE`、`AUTO_INCREMENT`、そして `AVG_ROW_LENGTH` 等のオプションを含んでいます。しかし、`ALTER TABLE` は `DATA DIRECTORY` と `INDEX DIRECTORY` テーブル オプションを無視します。

例えば、テーブルが `InnoDB` テーブルになるように変換するには、このステートメントを利用します。

```
ALTER TABLE t1 ENGINE = InnoDB;
```

MySQL 5.1.11 以降のバージョンでは、不注意なデータ ロスを防ぐ為に、テーブルのストレージ エンジンを `MERGE` や `BLACKHOLE` 変更する時に `ALTER TABLE` を利用する事はできません。

`AUTO_INCREMENT` カウンターを新しい行で使用するように値を変える為には、これを行って下さい。

```
ALTER TABLE t2 AUTO_INCREMENT = value;
```

既に使用されている値と同じ、またはそれ以下の値にカウンターをリセットする事はできません。`MyISAM` では、`AUTO_INCREMENT` カラム内の値が現在の最高値と同じかそれ以下の値なら、その値は現在の最高値プラス1にリセットされます。`InnoDB`では、もしその値がカラム内の現在の最高値以下であれば、エラー メッセージは表示されず、現在のシーケンス値は変更されません。

- 単一 `ALTER TABLE` ステートメントの中で、カンマで区切られた複数の `ADD`、`ALTER`、`DROP`、そして `CHANGE` 条項を発行する事ができます。これは、1つの `ALTER TABLE` ステートメントに対して1つの条項しか許可しない、スタンダード SQL の MySQL 拡張子です。例えば、単一ステートメントに複数のカラムをドロップするには、これを実行してください。

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- `CHANGE col_name`、`DROP col_name`、そして `DROP INDEX` はスタンダード SQL の MySQL 拡張子です。
- `MODIFY` は `ALTER TABLE` のオラクル拡張子です。
- `COLUMN` という言葉は任意であり、省く事ができます。
- `column_definition` 条項は `CREATE TABLE` と同じように、`ADD` と `CHANGE` に同じ構文を利用します。この構文は、データタイプだけでなく、カラム名も含むという事を覚えておいて下さい。詳しくは `CREATE TABLE 構文` を参照してください。
- `CHANGE old_col_name column_definition` 条項を利用してカラムをリネームする事ができます。それを行うには、カラムの古い名前と新しい名前、そして現在そのカラムが持つタイプを指定してください。例えば、`INTEGER` カラムを `a` から `b` にリネームするには、次の物を実行する事ができます。

```
ALTER TABLE t1 CHANGE a b INTEGER;
```

もしカラム名ではなくカラム タイプを変更したければ、`CHANGE` 構文には、両方同じだとしても古いカラム名と新しいカラム名が必要です。例:

```
ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

リネームせずにカラム タイプを変更するには、`MODIFY` を利用する事もできます。

```
ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- インデックスを持つカラムを短くする為に `CHANGE` や `MODIFY` を利用し、その結果カラム長がインデックス長よりも短くなるなら、MySQL はインデックスを自動的に短くします。
- `CHANGE` や `MODIFY` を利用してデータタイプを変更する時、MySQL は既存カラムを新しいタイプに可能な限り変換しようと試みます。
- テーブル行の中の指定した場所にカラムを追加するには、`FIRST` か `AFTER col_name` を利用してください。デフォルトはカラムを最後に追加します。`CHANGE` か `MODIFY` 操作の中で `FIRST` と `AFTER` を利用する事もできます。
- `ALTER ...SET DEFAULT` または `ALTER ...DROP DEFAULT` はそれぞれ、カラムに新しいデフォルト値を指定したり、古いデフォルト値を削除したりします。もし古いデフォルトが削除されて、カラムが `NULL` になり得るなら、新しいデフォルトは `NULL` です。もしカラムが `NULL` になり得ないなら、`データタイプデフォルト値` に説明されているように、MySQLはデフォルト値を割り当てます。

- **DROP INDEX** はインデックスを削除します。これはスタンダード SQL の MySQL 拡張子です。詳しくは [DROP INDEX 構文](#) を参照してください。
- もしカラムがテーブルからドロップされると、そのカラムが関わっている全てのインデックスからも削除されます。もし1つのインデックスを構成している全てのカラムがドロップされると、そのインデックスもドロップされます。
- もしテーブルがカラムを1つだけ含んでいると、そのカラムはドロップできません。もしテーブルを削除したいのであれば、代わりに **DROP TABLE** を利用してください。
- **DROP PRIMARY KEY** はプライマリ インデックスをドロップします。注意:MySQL の古いバージョンは、もしプライマリ インデックスが存在しなければ、**DROP PRIMARY KEY** はテーブルの中の最初の **UNIQUE** インデックスをドロップします。これは、主キーがエラーにならないテーブルに **DROP PRIMARY KEY** を利用しようとしている MySQL 5.1 では起こりません。

もし **UNIQUE INDEX** が **PRIMARY KEY** がテーブルに追加されると、MySQL が重複キーをなるべく早く見つけられるように、非ユニーク インデックスの前に格納されます。

- いくつかのストレージ エンジンでは、インデックスを作成する時にタイプを指定する事ができます。**index_type** 指定子の構文は **USING type_name** です。MySQL 5.1.10 以前では、**USING** はインデックス カラム リストの前だけに与える事ができました。5.1.10 以降のバージョンでの望ましい位置は、カラム リストの後ろです。MySQL 5.3 以降は、カラム リストの前でのオプションの使用は見られません。
- **ORDER BY** で指定した順番の行で新しいテーブルを作成する事ができます。挿入と削除の後ではテーブルの順番が変わってしまう事を覚えておいて下さい。このオプションは、ほとんど毎回同じ順番で行のクエリを行う場合に便利です。テーブルに大きい変更を行った後にこのオプションを利用すると、高い性能を得る事が可能でしょう。テーブルが、後でカラムをオーダーしたい順番になっていれば、MySQL のソートは簡単になる場合があります。

ORDER BY 構文は、ソートの昇順や降順を指示する為に、それぞれが任意で **ASC** または **DESC** が付随する1つか2つのカラム名を指定できます。デフォルトは昇順です。カラム名だけがソート基準として許されていて、任意の式は許されていません。

- **MyISAM** テーブル上で **ALTER TABLE** を利用すると、別のバッチに全ての非ユニークインデックスが作成されます。(**REPAIR TABLE** に関して)多くのインデックスがある時は、この方法で **ALTER TABLE** が大変早くなります。

この特徴は明確に起動する事ができます。**ALTER TABLE ... DISABLE KEYS** は MySQL に **MyISAM** テーブルの非ユニーク インデックスの更新を停めるよう命令します。**ALTER TABLE ...**その時は、欠けているインデックスを再作成する為に **ALTER TABLE ...ENABLE KEYS** を利用する必要があります。MySQL は、キーを1つ1つ挿入するよりも大変早い特別なアルゴリズムを利用してこの作業を行いますので、大量挿入を行う前にキーを無効化しておく事でかなりのスピードアップを実現する事ができます。**ALTER TABLE ...**の利用**ALTER TABLE ...DISABLE KEYS**の利用は、先に触れられた権限に加え、**INDEX** 権限を必要とします。

ENABLE KEYS と **DISABLE KEYS** は、MySQL 5.1.11 以前では分割されたテーブルに対してはサポートされていませんでした。(バグ #19502)

- **FOREIGN KEY** と **REFERENCES** 条項は、**ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...)** を施行する **InnoDB** ストレージ エンジンにサポートされています。詳しくは [FOREIGN KEY 制約](#) を参照してください。その他のストレージ エンジンでは、条項は解析されますが、無視されます。**CHECK** 条項は、全てのストレージ エンジンに解析されますが、無視されます。詳しくは [CREATE TABLE 構文](#) を参照してください。構文を受け入れながらも無視するのは、別のSQLサーバからコードをポートし易くし、参照を利用してテーブルを作成するアプリケーションを起動させるという、互換性の為です。詳しくは [MySQLと標準SQLとの違い](#) を参照してください。

単一 **ALTER TABLE** ステートメントの別々の条項の中に外部キーを追加したりドロップしたりはできません。別々のステートメントを利用しなければいけません。

- **InnoDB** は外部キーをドロップする為の **ALTER TABLE** の利用をサポートします。

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

単一 **ALTER TABLE** ステートメントの別々の条項の中に外部キーを追加したりドロップしたりはできません。別々のステートメントを利用しなければいけません。

追加情報については [を参照してください](#)。 [FOREIGN KEY 制約](#)

- テーブルが書き込みロックされ、[ALTER TABLE](#) がテーブル構造を変更するのに利用されると、保留中の [INSERT DELAYED](#) ステートメントは失われてしまいます。
- もしテーブルのデフォルト文字セットと全ての文字カラム([CHAR](#)、[VARCHAR](#)、[TEXT](#))を新しい文字セットに変更したければ、このようなステートメントを利用してください。

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

警告: 先行操作が文字セット間でカラム値の変換を行います。もし1つの文字セット内にカラムを持つが([latin1](#)等)、格納値は実際には別の互換性の無い文字セット([utf8](#) 等) を利用しているという場合には、これは 必要ではありません。このような場合は、それぞれのカラムに対して次のような作業が必要になります。

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

[BLOB](#) カラムへの、またはそれからの変換時には変換が起きない為、これが有効なのです。

[CONVERT TO CHARACTER SET binary](#) を指定すると、[CHAR](#)、[VARCHAR](#)、そして [TEXT](#) カラムはそれぞれが対応するバイナリ文字列タイプに変換されます。([BINARY](#)、[VARBINARY](#)、[BLOB](#))これは、そのカラムはそれ以上文字セットを持たなくなり、それに続く [CONVERT TO](#) 操作が適応されないという事を意味します。

テーブルの default 文字セットだけを変更するには、このステートメントを利用してください。

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

[DEFAULT](#) という言葉は任意です。テーブルに追加する新しいカラムの文字セットを指定しなければ、デフォルトの文字セットがその時に利用される文字セットです。(例えば、[ALTER TABLE ... ADD column](#) と共に)

- [.ibd](#) ファイルの中のテーブルスペースで作成された [InnoDB](#) テーブルに対して、そのファイルは廃棄、またインポートする事ができます。[.ibd](#) ファイルを廃棄する為には、このステートメントを利用してください。

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

これは現在の [.ibd](#) ファイル削除しますので、まずバックアップがある事を確認してください。テーブルスペースファイルが廃棄されている最中にテーブルにアクセスしようとすると、エラーが発生します。

[.ibd](#) のバックアップ ファイルをインポートする為には、まずそれをデータベース ディレクトリにコピーし、そしてこのステートメントを発行してください。

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

[SELECT 構文](#)を参照してください。

- MySQL 5.1.5では [ALTER TABLE](#) の領域確保に関する拡張子がいくつか追加されました。これらは、領域確保されたテーブルと共に、再領域確保、追加、ドロップ、マージ、そしてパーティションの分割、また、領域確保のメンテナンスを行う為に利用する事ができます。

そのまま [partition_options](#) 条項を領域確保されたテーブル上で [ALTER TABLE](#) と共に利用すると、[partition_options](#) によって定義された領域確保スキームに従って、そのテーブルを再領域確保します。この条項は必ず [PARTITION BY](#) で始まり、[CREATE TABLE](#) の [partition_options](#) 条項に適応するのと同様で、同じ構文と別のルールが後に続きます。(さらに詳細な説明については [CREATE TABLE 構文](#) を参照してください。) また、まだ分割されていない既存テーブルの分割に利用する事もできます。例えば、この様に定義された (領域確保されていない) テーブルを見てください。

```
CREATE TABLE t1 (
  id INT,
  year_col INT
);
```

このテーブルは [id](#) カラムを領域確保キーとして利用し、[HASH](#)によって、このステートメントを用いて、8区画に分割する事ができます。

```
ALTER TABLE t1
```

```
PARTITION BY HASH(id)
PARTITIONS 8;
```

ALTER TABLE ... PARTITION BY ステートメントを利用して作成されたテーブルは、CREATE TABLE ... PARTITION BY を利用して作成された物と同じルールに従わなければいけません。これは、[領域確保制限でも説明されているように](#)、テーブルが持っているであろう全てのユニークキー間(主キーを含む)の関係を管理するルール、そして分割式の中で利用されるカラムを含みます。領域確保キーとユニークキー[]分割数を指定する為の CREATE TABLE ... PARTITION BY ルールは、ALTER TABLE ... PARTITION BY にも適応します。

ALTER TABLE ... PARTITION BY は MySQL 5.1.6 から利用可能になりました。

ALTER TABLE ADD PARTITION の [partition_definition](#) 条項は、同名の CREATE TABLE ステートメント条項に対して同名の条項がするのと同じように、同じオプションをサポートします。(構文と説明に関しては [CREATE TABLE 構文](#) を参照してください。)ここに表示されているように、分割済のテーブルが作成されていると仮定します。

```
CREATE TABLE t1 (
  id INT,
  year_col INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999)
);
```

次のように、2002 以下の数値を格納する為にこのテーブルに新しいパーティション p3 を追加する事ができます。

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

1つ、または複数の RANGE や LIST 分割をドロップする為に DROP PARTITION を利用する事ができます。このステートメントは HASH や KEY パーティションと一緒に利用する事はできませんが、代わりに COALESCE PARTITION を利用してください。(下記参照)[partition_names](#) リストの中で名前が付けられた、ドロップされたパーティションに格納されたデータは全て廃棄されます。例えば、あらかじめ定義されたテーブル t1 を利用し、ここに表されているように p0 と p1 という名前が付いたパーティションをドロップする事ができます。

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```

DROP PARTITION は NDB Cluster ストレージ エンジンを利用するテーブルと一緒に機能しない事を覚えて置いてください。RANGE と LIST [パーティションの管理](#) と MySQL Cluster の既知の制限 [を参照して下さい](#)。

ADD PARTITION と DROP PARTITION は現在 IF [NOT] EXISTS をサポートしていません。パーティションや分割されたテーブルをリネームする事も不可能です。その代わりに、もしパーティションをリネームしたいのであれば、パーティションをドロップして再作成する必要があります。そしてもし分割済テーブルをリネームしたければ、全てのパーティションをドロップし、テーブルをリネームし、そしてドロップされたパーティションをもう一度追加する必要があります。

数字 でパーティション数を減らす為に HASH か KEY で分割されたテーブルと一緒に COALESCE PARTITION を利用する事ができます。次の定義を利用して t2 テーブルを作成したと仮定してください。

```
CREATE TABLE t2 (
  name VARCHAR (30),
  started DATE
)
PARTITION BY HASH( YEAR(started) )
PARTITIONS 6;
```

次のステートメントを利用して t2 で利用されたパーティション数を6から4に減らす事ができます。

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

最後の [number](#) パーティションに含まれているデータは、残りのパーティションにマージする事ができます。この場合、パーティション4と5は、最初の4つのパーティションにマージされます。(パーティション0、1、2、そして3)

分割されたテーブルで利用されたパーティションのいくつかを変更するには、REORGANIZE PARTITION を利用する事ができます。このステートメントの利用方法はいくつかあります:

- いくつかのセットになったパーティションを単一パーティションにマージする。この方法は、いくつかのパーティションに `partition_names` リストの中で名前をつけ、`partition_definition`に1つの定義を付ける事で行う事ができます。
- 1つの既存パーティションをいくつかのパーティションに分割する。これは、単一パーティションに `partition_names` で名前をつけ、いくつかの `partition_definitions` を与える事で実行できます。
- `VALUES LESS THAN` を利用して定義されたパーティションのサブセットの範囲、または `VALUES IN` を利用して定義されたパーティションのサブセットの値リストを変更する。

注意:明確に名前が付けられていないパーティションに対しては、MySQL は自動的に `p0`、`p1`、`p2`、などのようなデフォルト名を付けます。MySQL 5.1.7 以降のバージョンでは、サブパーティションに関しても同じ事が言えます。

`ALTER TABLE ... REORGANIZE PARTITION` ステートメントに関する詳細情報と例に関しては、[パーティショニング管理](#) を参照してください。

- いくつかの追加条項は、パーティションのメンテナンスと修復機能に似た物を、`CHECK TABLE` や `REPAIR TABLE` (分割されたテーブルにはサポートされていない物) などの様なステートメントを利用して、分割されていないテーブルに提供します。これらには、`ANALYZE PARTITION`、`CHECK PARTITION`、`OPTIMIZE PARTITION`、`REBUILD PARTITION`、そして `REPAIR PARTITION` が含まれます。これらのオプションのそれぞれは、カンマで区切られた1つか複数のパーティション名で構成される `partition_names` 条項を利用します。パーティションは変更されるテーブルの中に既に存在していなければいけません。これらについての更なる情報や例については、[パーティションのメンテナンス](#) を参照してください。
- テーブルや、そのデータに影響を与える事無くテーブルの分割を除去できるようにする為、MySQL 5.1.8 で `REMOVE PARTITIONING` が導入されました。(以前は `ENGINE` オプションを利用して行われていました。)このオプションは、追加、ドロップ、ドロップ カラムやインデックスをリネームする為に利用されるような、`ALTER TABLE` オプションと組み合わせる事ができます。

MySQL 5.1.7 以前のバージョンでは、`ENGINE` オプションを `ALTER TABLE` と一緒に利用すると、テーブルが削除されてしまう可能性がある領域確保の原因になりました。MySQL 5.1.8 からは、このオプションによる、テーブルに利用されるストレージ エンジンの変更はほとんど無く、領域確保にも全く影響を与えません。

`mysql_info()` C API 機能を利用すると、いくつかの行がコピーされ、(`IGNORE` が利用された時) いくつかの行がユニークキー値の複製の為に削除されたのかを確認する事ができます。詳しくは `mysql_info()` を参照してください。

`ALTER TABLE` の利用方法を表すいくつかの例があります。ここに表されているように作成された、テーブル `t1` から始めましょう。

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

`t1` から `t2` のテーブルの名前を付けるには

```
ALTER TABLE t1 RENAME t2;
```

カラム `a` を `INTEGER` から `TINYINT NOT NULL` (同じ名前のまま)に変更する為に、そしてカラム `b` を、`b` から `c` に変更するのと同じように、`CHAR(10)` から `CHAR(20)` に変更する為には:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

`d` と名づけられた新しい `TIMESTAMP` カラムを追加する為には:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

カラム `d` とカラム `a` にインデックスを追加する為には:

```
ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

カラム `c` を取り除く為には:

```
ALTER TABLE t2 DROP COLUMN c;
```

c と名づけられた新しい **AUTO_INCREMENT** カラムを追加する為には:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (c);
```

AUTO_INCREMENT カラムはインデックスされなければいけない為、**c** (**PRIMARY KEY** として)をインデックスした、そして、主キー カラムは **NULL** になり得ない為 **c** を **NOT NULL** として宣言したという事を覚えておいて下さい。

AUTO_INCREMENT カラムを追加する時、カラム値はシーケンス番号で自動的にファイルされます。**MyISAM** テーブルには、**ALTER TABLE** の前に **SET INSERT_ID=value** を実行する事、または**AUTO_INCREMENT=value** テーブルオプションを利用する事により最初のシーケンス番号を設定する事ができます。詳しくは **SET 構文** を参照してください。

MyISAM テーブルを利用すると、**AUTO_INCREMENT** カラムを変更しなければ、シーケンス番号は影響を受けません。もし **AUTO_INCREMENT** カラムをドロップし、そして別の **AUTO_INCREMENT** カラムを追加すると、番号は1から始まる順番に並べ直されます。

レプリケーションが利用された時、テーブルに **AUTO_INCREMENT** カラムを追加してもスレーブとマスターの行は同じ順番にはならないでしょう。これは、行の番号付けがテーブルに利用される特別なストレージ エンジンと、行が挿入される順番によって決まる為に起こります。もしマスターとスレーブ上で同じ順番である事が重要であれば、**AUTO_INCREMENT** 番号を割り当てる前に行の順番を整えておく必要があります。テーブル **t1** に **AUTO_INCREMENT** カラムを追加したいと仮定して、次のステートメントは **t1** と同一の **AUTO_INCREMENT** カラムを持つ新しいテーブル **t2** を作成します。

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

これは、テーブル **t1** はカラム **col1** と **col2** を持つと仮定します。

このステートメント セットはまた、**t1** と同一の **AUTO_INCREMENT** カラムを追加した

新しいテーブル **t2** を作成します。

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE T2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

重要:マスターとスレーブ上で同じ順番である事を保障する為に、**t1** の全てのカラムは **ORDER BY** 条項内で参照を付けられる必要があります。

AUTO_INCREMENT カラムを持つコピーを作成、実装する為に利用した方法に関わらず、最後のステップは元テーブルをドロップし、コピーをリネームする事です。

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

Problems with ALTER TABLE も参照してください。

ALTER LOGFILE GROUP 構文

```
ALTER LOGFILE GROUP logfile_group
ADD UNDOFILE 'file'
INITIAL_SIZE [=] size
ENGINE [=] engine
```

このステートメントは、**'file'** と名付けられた **UNDO** ファイルを、既存ログファイルグループ **logfile_group** に追加します。**ALTER LOGFILE GROUP** ステートメントはたった一つの **ADD UNDOFILE** 条項を持ちます。**DROP UNDOFILE** 条項はサポートされていません。

INITIAL_SIZE パラメータは **UNDO** ファイルの冒頭のサイズをバイトで設定します。**my.cnf** で利用されている物と同様、大きさによって一文字の省略形を持つ **size** に従う事もできます。通常これは **M** (メガ バイト) が **G** (ギガ バイト) のどちらかの文字です。

ENGINE パラメータ(要求された)が、このログ ファイル グループによって利用されるストレージエンジンを決め、その名前は **engine** となります。MySQL 5.1では、**engine** に受け入れられる値は **NDB** と **NDBCLUSTER** だけです。

ここに、ログ ファイル グループ **lg_3** が既に **CREATE LOGFILE GROUP** を利用して作成されていると仮定した例があります。(**CREATE LOGFILE GROUP** 構文 を参照してください。)

```
ALTER LOGFILE GROUP lg_3
ADD UNDOFILE 'undo_10.dat'
INITIAL_SIZE=32M
ENGINE=NDB;
```

ALTER LOGFILE GROUP が **ENGINE = NDB** と共に利用された時、**UNDO** ログ ファイルがそれぞれのクラスタ データ ノード上に作成されます。**INFORMATION_SCHEMA.FILES** テーブルに問い合わせる事によって **UNDO** ファイルが作成され、それらの情報を得た事を証明する事ができます。例:

```
mysql> SELECT FILE_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg_3';
+-----+-----+-----+
| FILE_NAME | LOGFILE_GROUP_NUMBER | EXTRA |
+-----+-----+-----+
| newdata.dat | 0 | CLUSTER_NODE=3 |
| newdata.dat | 0 | CLUSTER_NODE=4 |
| undo_10.dat | 11 | CLUSTER_NODE=3 |
| undo_10.dat | 11 | CLUSTER_NODE=4 |
+-----+-----+-----+
4 rows in set (0.01 sec)
```

(詳しくは **INFORMATION_SCHEMA.FILES** テーブル を参照してください。)

MySQL 5.1.6 では **ALTER LOGFILE GROUP** が追加されました。MySQL 5.1 では MySQL クラスタのディスク データ ストレージと一緒にの時のみ利用する事ができます。詳しくは **MySQL Cluster ディスク データ ストレージ** を参照してください。

ALTER TABLESPACE 構文

```
ALTER TABLESPACE tablespace
ADD DATAFILE 'file'
INITIAL_SIZE [=] size
ENGINE [=] engine

ALTER TABLESPACE tablespace
DROP DATAFILE 'file'
ENGINE [=] engine
```

このステートメントは新しいデータ ファイルを追加する時かテーブルスペースからデータ ファイルをドロップする時に利用する事ができます。

ADD DATAFILE 異形では、**size** がバイトで計算され、**INITIAL_SIZE** 構文を利用して初期のサイズを指定する事が要求されます。**my.cnf** で利用されている物と同様、大きさによって一文字の省略形を持つ整数値に従う事もできます。通常これは **M** (メガ バイト) か **G** (ギガ バイト)のどちらかの文字です。

一度データ ファイルが作成されると、そのサイズは変更できませんが、追加の **ALTER TABLESPACE ... ADD DATAFILE** ステートメントを利用する事によりテーブルスペースにより多くのデータ ファイルを追加する事ができます。

DROP DATAFILE を **ALTER TABLESPACE** と共に利用する事で、テーブルスペースから '**file**' をドロップする事ができます。このファイルは **CREATE TABLESPACE** か **ALTER TABLESPACE** を利用してテーブルスペースに既に追加されていなければいけません。そうでなければエラーが発生します。

ALTER TABLESPACE ... ADD DATAFILE と **ALTER TABLESPACE ... DROP DATAFILE** の2つは、テーブルスペースに利用されるストレージ エンジンを指定する **ENGINE** 条項を必要とします。MySQL 5.1では、**engine** に受け入れられる値は **NDB** と **NDBCLUSTER** だけです。

ALTER TABLESPACE ... ADD DATAFILE が **ENGINE = NDB** と共に利用された時、データ ファイルがそれぞれのクラスタ データ ノード上に作成されます。**INFORMATION_SCHEMA.FILES** テーブルに問い合わせる事によってデー

タ ファイルが作成され、それらの情報を得た事を証明する事ができます。例えば、次のクエリは `newts` 名付けられたテーブルスペースに属する全てのデータ ファイルを表しています。

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
```

LOGFILE_GROUP_NAME	FILE_NAME	EXTRA
lg_3	newdata.dat	CLUSTER_NODE=3
lg_3	newdata.dat	CLUSTER_NODE=4
lg_3	newdata2.dat	CLUSTER_NODE=3
lg_3	newdata2.dat	CLUSTER_NODE=4

2 rows in set (0.03 sec)

詳しくはこちらを参照してください。 [INFORMATION_SCHEMA FILES テーブル](#)

MySQL 5.1.6 では `ALTER TABLESPACE` が追加されました。MySQL 5.1 では MySQL クラスタの ディスク データ ストレージと一緒に時のみ利用する事ができます。詳しくは [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

ALTER SERVER 構文

```
ALTER SERVER server_name
OPTIONS (option ...)
```

`server_name` のサーバ情報を変更し、`CREATE SERVER` コマンドごとの指定オプションを調整します。詳しくは [CREATE SERVER 構文](#) を参照してください。 `mysql.servers` テーブル内の対応するフィールドは適宜更新されます。

例えば、`USER` オプションを更新する為には:

```
ALTER SERVER s OPTIONS (USER 'sally');
```

`ALTER SERVER` を利用する為に特別な権限は必要ありません。

`ALTER SERVER` は自動コミットを引き起こしません。

CREATE DATABASE 構文

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_specification [create_specification] ...]
```

`create_specification`:

```
[DEFAULT] CHARACTER SET charset_name
|[DEFAULT] COLLATE collation_name
```

`CREATE DATABASE` は、与えられた名前データベースを作成します。このステートメントを利用する為には、データベースに `CREATE` 権限が必要です。 `CREATE SCHEMA` は `CREATE DATABASE` の同義語です。

もしデータベースが存在し、`IF NOT EXISTS` を指定しなかった場合、エラーが発生します。

`create_specification` オプションはデータベースの特徴を指定します。データベースの特徴は、データベース ディレクトリの `db.opt` ファイルに格納されています。 `CHARACTER SET` 条項はデフォルト データベースの文字セットを指定します。 `COLLATE` 条項はデフォルト データベースの照合を指定します。 [9章 キャラクタセットサポート](#) は文字セットと照合名を検討します。

MySQL 内のデータベースは、その中のテーブルに対応するディレクトリとして実行されます。最初にデータベースが作成された時にはその中にはテーブルが無いので、`CREATE DATABASE` ステートメントが、MySQL データディレクトリ下にディレクトリと `db.opt` ファイルだけを作成します。 [識別子](#) に許可されたデータベース名のルールが紹介されています。もしデータベース名が特別な文字を含んでいる場合、その名前は [ファイル名への識別子のマッピング](#) に表されているようにそれらの文字が暗号化された形を含んだ物になります。

もしデータ ディレクトリ下にディレクトリを手動で作成したら(例えば `mkdir` を利用して)、サーバはそれをデータベース ディレクトリとみなし、`SHOW DATABASES` のアウトプット内に現れます。

データベースを作成する為に [mysqladmin](#) プログラムを利用する事もできます。詳しくは [mysqladmin — MySQL サーバの管理を行うクライアント](#) を参照してください。

CREATE INDEX 構文

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
[index_type]
ON tbl_name (index_col_name,...)
[index_option ...]

index_col_name:
col_name [(length)] [ASC | DESC]

index_type:
USING {BTREE | HASH}

index_option:
KEY_BLOCK_SIZE value
| index_type
| WITH PARSER parser_name
```

CREATE INDEX は、インデックスを作成する為に **ALTER TABLE** ステートメントにマップされています。詳しくは [ALTER TABLE 構文](#) を参照してください。インデックスに関する更なる情報については、[MySQLにおけるインデックスの使用](#) を参照してください。

通常、テーブルが **CREATE TABLE** で作成される時にテーブル上に全てのインデックスを作成します。詳しくは [CREATE TABLE 構文](#) を参照してください。**CREATE INDEX** で既存テーブルにインデックスを追加する事ができます。

(col1,col2,...) のカラム リストは複合カラム インデックスを作成します。インデックス値は与えられたカラムの値を結合する事によって形作られます。

CHAR、**VARCHAR**、**BINARY**、そして **VARBINARY** カラムには、インデックス プリフィックス長を指定する為に **col_name(length)** 構文を利用して、カラム値の最初に部分だけを利用するインデックスを作成する事ができます。**BLOB** と **TEXT** カラムもまたインデックスする事ができますが、プリフィックス長を与える 必要があります。。プリフィックス長は非バイナリ文字列タイプには文字で指定され、バイナリ文字列タイプにはバイトで指定されます。これは、インデックス エントリは**CHAR**、**VARCHAR**、そして **TEXT** カラムのそれぞれのカラム値の最初の **length** 文字で、そして **BINARY**、**VARBINARY**、そして **BLOB** カラムのそれぞれのカラム値の最初の **length** バイトで成り立っているという事です。

ここに表示されているステートメントは、**name** カラムの最初の10文字を利用してインデックスを作成します。

```
CREATE INDEX part_of_name ON customer (name(10));
```

もしカラム内の名前の最初の10文字が違っていれば、このインデックスは **name** カラム全体から作成されたインデックスよりも遅くは無いはずですが、また、部分的なカラムをインデックスに利用する事でインデックス ファイルを小さくする事ができるので、ディスクのスペースを節約し、**INSERT** 操作を早くする事ができます。

プリフィックスは最高で1000バイトの長さまで可能です。(InnoDB テーブルは767バイト)非バイナリ データ タイプ(**CHAR**、**VARCHAR**、**TEXT**)では **CREATE INDEX** ステートメントのプリフィックス長は文字数で解釈される一方、プリフィックス リミットはバイトで計算されるという事を覚えておいて下さい。マルチバイトの文字セットを利用するカラムのプリフィックス長を指定する時にはこれを考慮に入れておいて下さい。

UNIQUE インデックスは、インデックス内の全ての値は明確でなければいけないというような制限を作成します。既存行とマッチするキー値の新しい行を追加しようとするとエラーが発生します。全てのエンジンに対して、**UNIQUE** インデックスは **NULL** を含む事ができるカラムの複数 **NULL** 値を許容します。

FULLTEXT インデックスは **MyISAM** テーブルにだけサポートされており、**CHAR**、**VARCHAR**、そして **TEXT** カラムだけを含む事ができます。インデックスする作業は必ずカラム全体に対して行われますので、部分的インデックスはサポートされておらず、プリフィックス長を指定しても無視されます。操作に関しての詳細は [全文検索関数](#) を参照してください。

SPATIAL インデックスは **MyISAM** テーブルに対してだけサポートされており、**NOT NULL** として定義された空間カラムだけを含む事ができます。[16章Spatial Extensions](#)で空間データタイプについて説明されています。

MySQL 5.1 内では

- もし [MyISAM](#)、[InnoDB](#)、または [MEMORY](#) ストレージエンジンを利用していれば、その時だけ [NULL](#) 値を持つ事ができるカラム上にインデックスを追加する事ができます。
- もし [MyISAM](#) か [InnoDB](#) ストレージエンジンを利用していれば、その時だけ [BLOB](#) か [TEXT](#) カラム上にインデックスを追加する事ができます。

[index_col_name](#) 仕様は [ASC](#) か [DESC](#) で終わる事ができます。これらのキーワードは昇順や降順インデックス値ストレージを指定する為の将来の拡張子として許容されます。現在は、それらは解析されますが無視されます。インデックス値は毎回昇順で格納されます。

インデックス カラム リストに続き、インデックス オプションが与えられます。[index_option](#) 値は次のうちのどれかになり得ます。

- [KEY_BLOCK_SIZE value](#)

このオプションはインデックス キー ブロックに利用するサイズについてストレージ エンジンにヒントを提供します。このエンジンは必要に応じて値を変更する事が可能です。0という値は、デフォルト値を利用しなければいけないという事を表しています。[KEY_BLOCK_SIZE](#) は MySQL 5.1.10 で追加されました。

- [index_type](#)

いくつかのストレージ エンジンでは、インデックスを作成する時にタイプを指定する事ができます。別々のストレージ エンジンにサポートされた許容インデックス タイプは次のテーブルに表されています。複数インデックスタイプがリストされている部分に関しては、インデックス指定子が指示されていなければ最初の物がデフォルトです。

ストレージ エンジン	許容インデックス タイプ
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH 、 BTREE

もし規定のストレージ エンジンに対して正当ではないインデックス タイプを指定し、しかしクエリに影響を与えずにそのエンジンが利用できる別の有効なインデックス タイプが存在すれば、エンジンはその有効なタイプを利用します。

例：

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index USING BTREE ON lookup (id);
```

[TYPE type_name](#) は [USING type_name](#) の同義語として解釈されます。しかし、[USING](#) が好ましい形です。

注意:MySQL 5.1.10 以前のバージョンでは、オプションは [ON tbl_name](#) 条項の前にだけ指示する事ができました。このオプションのこの位置での利用は 5.1.10 以降は廃止予定で、MySQL 5.3 以降はサポートされなくなります。

- [WITH PARSER parser_name](#)

このオプションは [FULLTEXT](#) インデックスとだけ利用する事ができます。もしフル テキスト インデックスと検索操作が特別対応を必要とするなら、これはインデックスを利用してパーサー プラグインと提携します。プラグインの作成に関しての詳細は[The MySQL Plugin Interface](#) を参照してください。

CREATE TABLE 構文

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
  (create_definition,...)
  [table_option ...]
  [partition_options]
```

または:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
  [(create_definition,...)]
  [table_option ...]
```

```
[partition_options]
select_statement
```

または:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
{ LIKE old_tbl_name | (LIKE old_tbl_name) }
```

```
create_definition:
column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  [index_option ...]
| {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
  [index_option ...]
| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
  [index_name] [index_type] (index_col_name,...)
  [index_option ...]
| {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
  [index_option ...]
| [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (index_col_name,...) [reference_definition]
| CHECK (expr)
```

```
column_definition:
col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string'] [reference_definition]
```

```
data_type:
BIT[(length)]
| TINYINT[(length)] [UNSIGNED] [ZEROFILL]
| SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
| MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
| INT[(length)] [UNSIGNED] [ZEROFILL]
| INTEGER[(length)] [UNSIGNED] [ZEROFILL]
| BIGINT[(length)] [UNSIGNED] [ZEROFILL]
| REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
| DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
| FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
| DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
| NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
| DATE
| TIME
| TIMESTAMP
| DATETIME
| YEAR
| CHAR(length)
  [CHARACTER SET charset_name] [COLLATE collation_name]
| VARCHAR(length)
  [CHARACTER SET charset_name] [COLLATE collation_name]
| BINARY(length)
| VARBINARY(length)
| TINYBLOB
| BLOB
| MEDIUMBLOB
| LONGBLOB
| TINYTEXT [BINARY]
  [CHARACTER SET charset_name] [COLLATE collation_name]
| TEXT [BINARY]
  [CHARACTER SET charset_name] [COLLATE collation_name]
| MEDIUMTEXT [BINARY]
  [CHARACTER SET charset_name] [COLLATE collation_name]
| LONGTEXT [BINARY]
  [CHARACTER SET charset_name] [COLLATE collation_name]
| ENUM(value1,value2,value3,...)
  [CHARACTER SET charset_name] [COLLATE collation_name]
| SET(value1,value2,value3,...)
  [CHARACTER SET charset_name] [COLLATE collation_name]
| spatial_type
```

```
index_col_name:
col_name [(length)] [ASC | DESC]
```

```
index_type:
USING {BTREE | HASH}
```

```

index_option:
    KEY_BLOCK_SIZE value
| index_type
| WITH PARSER parser_name

reference_definition:
    REFERENCES tbl_name [(index_col_name,...)]
    [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_option:
    [TABLESPACE tablespace_name STORAGE DISK]
    ENGINE [=] engine_name
| AUTO_INCREMENT [=] value
| AVG_ROW_LENGTH [=] value
| [DEFAULT] CHARACTER SET charset_name
| CHECKSUM [=] {0 | 1}
| COLLATE collation_name
| COMMENT [=] 'string'
| CONNECTION [=] 'connect_string'
| DATA DIRECTORY [=] 'absolute path to directory'
| DELAY_KEY_WRITE [=] {0 | 1}
| INDEX DIRECTORY [=] 'absolute path to directory'
| INSERT_METHOD [=] { NO | FIRST | LAST }
| KEY_BLOCK_SIZE [=] value
| MAX_ROWS [=] value
| MIN_ROWS [=] value
| PACK_KEYS [=] {0 | 1 | DEFAULT}
| PASSWORD [=] 'string'
| ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
| UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    PARTITION BY
        [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list)
        | RANGE(expr)
        | LIST(expr)
    [PARTITIONS num]
    [SUBPARTITION BY
        [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list)
    [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
    [VALUES {LESS THAN (expr) | MAXVALUE | IN (value_list)}]
    [[STORAGE] ENGINE [=] engine_name]
    [COMMENT [=] 'comment_text' ]
    [DATA DIRECTORY [=] 'data_dir']
    [INDEX DIRECTORY [=] 'index_dir']
    [MAX_ROWS [=] max_number_of_rows]
    [MIN_ROWS [=] min_number_of_rows]
    [TABLESPACE [=] (tablespace_name)]
    [NODEGROUP [=] node_group_id]
    [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
    [[STORAGE] ENGINE [=] engine_name]
    [COMMENT [=] 'comment_text' ]
    [DATA DIRECTORY [=] 'data_dir']
    [INDEX DIRECTORY [=] 'index_dir']
    [MAX_ROWS [=] max_number_of_rows]
    [MIN_ROWS [=] min_number_of_rows]
    [TABLESPACE [=] (tablespace_name)]
    [NODEGROUP [=] node_group_id]

```

```
select_statement:
[IGNORE | REPLACE] [AS] SELECT ... (Some legal select statement)
```

CREATE TABLE は、与えられた名前でデータ ベースを作成します。テーブルに対して **CREATE** 権限を持つ必要があります。

識別子 に許容テーブル名のルールが紹介されています。デフォルトによって、デフォルト データベースの中にテーブルが作成されます。テーブルが既に存在したり、デフォルト データベースが無かったり、データベースが存在しなかったりするとエラーが発生します。

指定データベース内にテーブルを作成するには、テーブル名を `db_name.tbl_name` と指定する事ができます。この作業は、デフォルト データベースが無くても、あると仮定して行われます。もし引用識別子を利用するなら、データベースとテーブル名は別々に引用してください。例えば、`'mydb.mytbl'` ではなく `'mydb'.'mytbl'` と書いてください。

テーブルを作成する時、**TEMPORARY** キーワードを利用する事ができます。**TEMPORARY** テーブルは現在の接続でのみ現れ、接続が終了すると自動的にドロップされます。これは、2つの異なる接続同士、または、既存の同名の非**TEMPORARY** テーブルとお互いに対立する事無く、同じテンポラリ テーブル名を利用する事ができるという意味になります。(テンポラリ テーブルがドロップされるまで、既存テーブルは隠されています。)テンポラリ テーブルを作成する為には **CREATE TEMPORARY TABLES** 特権を持つ必要があります。

注意:**CREATE TABLE** は、もし **TEMPORARY** キーワードを利用すると自動的に現在のアクティブなトランザクションを行いません。

もしテーブルが存在すると **IF NOT EXISTS** キーワードはエラーが起こるのを防ぎます。しかし、**CREATE TABLE** ステートメントに指示されたテーブルと既存テーブルが同一の構造である事の照合は行われません。注意:もし **IF NOT EXISTS** を **CREATE TABLE ... SELECT** ステートメント内で利用すると、**SELECT** 部分によって選択された全ての行はテーブルが既に存在するかどうかに関わらず挿入されます。

MySQL はそれぞれのテーブルをデータベース ディレクトリ内に `.frm` テーブル フォーマットで表します。テーブルのストレージ エンジン別のファイルを作成する事もあります。**MyISAM** テーブルの場合、ストレージ エンジンはデータとインデックス ファイルを作成します。従って、各 **MyISAM** テーブル `tbl_name` は3つのディスク ファイルを持ちます。

ファイル	目的
<code>tbl_name.frm</code>	テーブル フォーマット (定義) ファイル
<code>tbl_name.MYD</code>	データ ファイル
<code>tbl_name.MYI</code>	インデックス ファイル

13章ストレージエンジンとテーブルタイプ でテーブルを表す為にそれぞれのストレージ エンジンがどのファイルを作成するのか説明されています。もしテーブル名が特別な文字を含んでいる場合、そのテーブル ファイルの名前は**ファイル名への識別子のマッピング**に表されているようにそれらの文字が暗号化された形を含んだ物になります。

data_type は、データ タイプはカラム定義だという事を意味します。**spatial_type** は空間データ タイプを意味します。表示されるデータ タイプ構文はただの見本です。各タイプの性質についての情報だけでなく、カラム データ タイプを指定する事ができる構文の完全な説明については、**10章データタイプ** と **16章Spatial Extensions** を参照してください。

いくつかの属性は全てのデータ タイプには対応しません。**AUTO_INCREMENT** は整数タイプにのみ対応します。**DEFAULT** は **BLOB** や **TEXT** タイプには対応しません。

- もし **NULL** か **NOT NULL** のどちらも指定されなければ、そのカラムは **NULL** が指定されたという形で扱われます。
- 整数カラムは追加属性 **AUTO_INCREMENT** を持つ事ができます。インデックスされた **AUTO_INCREMENT** カラムに **NULL** (推奨) か **0** の値を挿入すると、カラムは次のシーケンス値に設定されます。通常これは、**value** が現在テーブルの中にあるカラムの最大値である、**value+1** です。**AUTO_INCREMENT** シーケンスは **1** で始まります。

行の挿入後に **AUTO_INCREMENT** 値を検索するには、**LAST_INSERT_ID()** SQL 機能が **mysql_insert_id()** C API 関数を利用してください。**情報関数**、**mysql_insert_id()** を参照して下さい。

もし **NO_AUTO_VALUE_ON_ZERO** SQL モードが有効であれば、新しいシーケンス値を発生させずに **0** を **AUTO_INCREMENT** カラムに **0** として格納する事ができます。詳しくは **SQL モード** を参照してください。

注意:各テーブルに [AUTO_INCREMENT](#) カラムは1つだけ存在する事ができ、それはインデックスされる必要があります。 [DEFAULT](#) 値を持つ事はできません。 [AUTO_INCREMENT](#) カラムは正数のみを含んでいる時だけ正しく機能します。負数を挿入すると、とても大きな正数を挿入したと解釈されます。これは、数字が正数から負数に「ラップ」される時の精度の問題を避ける為に、また 0 を含む [AUTO_INCREMENT](#) カラムを誤って採用してしまわない為に行われます。

[MyISAM](#) テーブルには、複合カラム キー内の [AUTO_INCREMENT](#) セカンダリ カラムを指定する事ができます。詳しくは [Using AUTO_INCREMENT](#) を参照してください。

MySQL 互換性がいくつかの ODBC アプリケーション持つ為に、次のクエリを利用して、最後に挿入された行に [AUTO_INCREMENT](#) 値を見つける事ができます。

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

[InnoDB](#) と [AUTO_INCREMENT](#) の更なる情報に関しては、[AUTO_INCREMENT カラムが InnoDB 内でどのように機能するか](#) を参照してください。

- [SERIAL](#) 属性は [BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE](#) のエイリアスです。
- 文字データタイプ([CHAR](#)、[VARCHAR](#)、[TEXT](#))は、文字セットとカラムの照合を指定する為に [CHARACTER SET](#) と [COLLATE](#) 属性を含む事ができます。詳細に関しては [9章キャラクタセットサポート](#) を参照して下さい。 [CHARSET](#) は [CHARACTER SET](#) の同義語です。例:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.1 は、文字の中の文字カラム定義の長さ仕様を説明します。(MySQL 4.1 以前のバージョンでは、それらはバイトで解釈されます。) [BINARY](#) と [VARBINARY](#) の長さはバイトで表されています。

- [DEFAULT](#) 条項はカラムのデフォルト値を指定します。例外がひとつあります。デフォルト値は一定でなければいけませんので、それは関数や式にはなり得ません。これは例えば、日付カラムの値に [NOW\(\)](#) や [CURRENT_DATE](#) のような関数の値をデフォルトとして設定する事はできないという意味です。例外として、[TIMESTAMP](#) カラムのデフォルトとして [CURRENT_TIMESTAMP](#) を指定する事ができます。詳しくは [TIMESTAMP MySQL 4.1での性質](#) を参照してください。

もしカラム定義が明示的な [DEFAULT](#) 値を含まない場合、MySQL はデフォルト値を [データタイプデフォルト値](#) のように規定します。

[BLOB](#) と [TEXT](#) カラムはデフォルト値として割り当てる事ができません。

- カラムのコメントは、255文字の長さまでで [COMMENT](#) オプションで指定できます。コメントは [SHOW CREATE TABLE](#) と [SHOW FULL COLUMNS](#) ステートメントによって表示されます。
- [KEY](#) は通常 [INDEX](#) の同義語です。キー属性 [PRIMARY KEY](#) はまた、カラム定義の中では単に [KEY](#) として指定できます。これは、互換性の為に他のデータベースと共に利用されます。
- [UNIQUE](#) インデックスは、インデックス内の全ての値は明確でなければいけないというような制限を作成します。既存行とマッチするキー値の新しい行を追加しようとするとエラーが発生します。全てのエンジンに対して、[UNIQUE](#) インデックスは [NULL](#) を含む事ができるカラムの複数 [NULL](#) 値を許容します。
- [PRIMARY KEY](#) は、全てのキー カラムが [NOT NULL](#) として定義されなければいけないユニーク インデックスです。もしそれらが [NOT NULL](#) として明示的に宣言されなければ、MySQL はそれらを暗示的に(そして静かに)宣言します。1つのテーブルは1つの [PRIMARY KEY](#) しか持つ事ができません。もし [PRIMARY KEY](#) が無いのにアプリケーションがテーブル内で [PRIMARY KEY](#) を要求したら、MySQL は [PRIMARY KEY](#) として [NULL](#) カラムを持たない最初の [UNIQUE](#) インデックスを返します。

[InnoDB](#) テーブル内で長い [PRIMARY KEY](#) を持つとスペースを無駄に利用します。(詳しくは [InnoDB テーブルとインデックス構造](#) を参照してください。)

- 作成されたテーブル内では、[PRIMARY KEY](#) が最初に置かれ、次に [UNIQUE](#) インデックスが続き、そして次に非ユニーク インデックスが続きます。このおかげで MySQL オプティマイザがどのインデックスを優先して利用するのか、また複製 [UNIQUE](#) キーをより早く検索する為に役立ちます。

- **PRIMARY KEY** は複合カラム インデックスになり得ます。しかし、カラム仕様内で **PRIMARY KEY** キー属性を利用して複合カラム インデックスを作成する事はできません。それをして、単一カラムが最初に来るという印が付けられるだけです。別々の **PRIMARY KEY(index_col_name, ...)** 条項を利用しなければいけません。
- もし **PRIMARY KEY** か **UNIQUE** インデックスが整数タイプを持つ1つだけののカラムで構成されていたら、そのカラムを **SELECT** ステートメントの中で **_rowid** として参照する事もできます。
- MySQL 内では、**PRIMARY KEY** の名前は **PRIMARY** です。他のインデックスに関しては、もし名前を割り当てなければ、それを固有の物にする為に任意のサフィックス(**_2**、**_3**、**_...**)を利用して、最初にインデックスされたカラムと同じ名前に指定されます。**SHOW INDEX FROM tbl_name** を利用してテーブルのインデックス名を調べる事ができます。詳しくは **SHOW INDEX 構文** を参照してください。
- いくつかのストレージ エンジンでは、インデックスを作成する時にタイプを指定する事ができます。**index_type** 指定子の構文は **USING type_name** です。

例:

```
CREATE TABLE lookup
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

MySQL 5.1.10 以前では、**USING** はインデックス カラム リストの前だけに与える事ができました。5.1.10 以降のバージョンでの望ましい位置は、カラム リストの後ろです。MySQL 5.3 以降は、カラム リストの前でのオプションの使用は見られません。

index_option 値はインデックスの追加オプションを指定します。**USING** はそのようなオプションの1つです。許容 **index_option** 値の詳細に関しては **CREATE INDEX 構文** を参照してください。

インデックスに関する更なる情報については、**MySQLにおけるインデックスの使用** を参照してください。

- MySQL 5.1 では、**MyISAM**、**InnoDB**、そして **MEMORY** ストレージ エンジンだけが **NULL** 値を持つ事ができるカラムをインデックス上でサポートしています。それ以外の場合、インデックスされたカラムを **NOT NULL** として宣言しなければエラーが発生します。
- **CHAR**、**VARCHAR**、**BINARY**、そして **VARBINARY** カラムには、インデックス プリフィックス長を指定する為に **col_name(length)** 構文を利用して、カラム値の最初に部分だけを利用するインデックスを作成する事ができます。**BLOB** と **TEXT** カラムもまたインデックスする事ができますが、プリフィックス長を与える 必要がありません。プリフィックス長は非バイナリ文字列タイプには文字で指定され、バイナリ文字列タイプにはバイトで指定されます。これは、インデックス エントリは **CHAR**、**VARCHAR**、そして **TEXT** カラムのそれぞれのカラム値の最初の **length** 文字で、そして **BINARY**、**VARBINARY**、そして **BLOB** カラムのそれぞれのカラム値の最初の **length** バイトで成り立っているという事です。このようにカラム値のプリフィックスだけをインデックスする事で、インデックス ファイルをとて小さくする事ができます。詳しくは **カラムインデックス** を参照してください。

MyISAM と **InnoDB** ストレージ エンジンだけが **BLOB** と **TEXT** カラムのインデックスをサポートします。例:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

プリフィックスは最高で1000バイトの長さまで可能です。(InnoDB テーブルは767バイト)非バイナリ データ タイプ (**CHAR**、**VARCHAR**、**TEXT**)では **CREATE TABLE** ステートメントのプリフィックス長は文字数で解釈される一方、プリフィックス リミットはバイトで計算されるという事を覚えておいて下さい。マルチバイトの文字セットを利用するカラムのプリフィックス長を指定する時にはこれを考慮に入れておいて下さい。

- **index_col_name** 仕様は **ASC** か **DESC** で終わる事ができます。これらのキーワードは昇順や降順インデックス値ストレージを指定する為の将来の拡張子として許容されます。現在は、それらは解析されますが無視されます。インデックス値は毎回昇順で格納されます。
- **SELECT** 内の **TEXT** か **BLOB** カラムに対して **ORDER BY** か **GROUP BY** を利用する時、サーバは **max_sort_length** システム変数によって指示された初期バイト数だけを利用して値をソートします。詳しくは **BLOBとTEXT タイプ** を参照してください。
- フル テキスト検索に利用される特別な **FULLTEXT** インデックスを作成する事ができます。**MyISAM** ストレージ エンジンだけが **FULLTEXT** インデックスをサポートします。それらは **CHAR**、**VARCHAR**、そして **TEXT** カラムからのみ作成する事ができます。インデックスする作業は必ずカラム全体に対して行われますので、部分的インデックスはサポートされておらず、プリフィックス長を指定しても無視されます。操作に関する詳細は **全文検索関数**

を参照してください。[WITH PARSER](#) 条項は、もしフル テキスト インデックスと検索操作が特別対応を必要とするなら、インデックスと共にパーサー プラグインと提携する為に [index_option](#) 値として指定する事ができます。この条項は [FULLTEXT](#) インデックスだけに対して正当です。プラグインの作成に関しての詳細は[The MySQL Plugin Interface](#) を参照してください。

- 空間データ タイプ上に [SPATIAL](#) インデックスを作成する事ができます。空間タイプは [MyISAM](#) テーブルに対してだけサポートされており、インデックスされたカラムは [NOT NULL](#) として宣言されなければいけません。詳しくは [16章Spatial Extensions](#) を参照してください。
- [InnoDB](#) テーブルは外部キー制約のチェックをサポートします。詳しくは [InnoDB ストレージ エンジン](#) を参照してください。[InnoDB](#) 内の [FOREIGN KEY](#) 構文は、このセクションの最初で[CREATE TABLE](#) ステートメントの為に紹介された構文よりも制限的である事を覚えておいてください。参照テーブルのカラムは必ず明確に名前を付けられる必要があります。[InnoDB](#) は、外部キーへの [ON DELETE](#) と [ON UPDATE](#) 作用の両方をサポートします。正確な構文に関しては、[FOREIGN KEY 制約](#) を参照してください。

その他のストレージ エンジンに関しては、MySQL サーバは [CREATE TABLE](#) ステートメント内の [FOREIGN KEY](#) と [REFERENCES](#) 構文を検索し無視します。[CHECK](#) 条項は、全てのストレージ エンジンに解析されますが、無視されます。詳しくは [外部キー](#) を参照してください。

- [MyISAM](#) テーブルに対しては、各 [NULL](#) カラムは1ビット余分に利用し、一番近いバイト数に丸められます。バイトでの最大行長は次のように計算されます。

```
row length = 1
+ (sum of column lengths)
+ (number of NULL columns + delete_flag + 7)/8
+ (number of variable-length columns)
```

[delete_flag](#) は静的行フォーマットのテーブルに対しては1です。静的テーブルは、行が削除されたかどうか指示するフラッグに対して行レコード内でビットを利用します。[delete_flag](#) は、フラッグがダイナミック行ヘッダー内で格納される為、ダイナミック テーブルに対しては0です。

これらの計算は、[NULL](#) カラムと [NOT NULL](#) カラムの格納サイズが同じである [InnoDB](#) テーブルには適応しません。

[TABLESPACE ... STORAGE DISK](#) テーブルオプションは [NDB Cluster](#) テーブルとのみ使用されます。これはテーブルをクラスタ ディスク データ テーブルスペースに割り当てます。[tablespace_name](#) と名づけられたテーブルスペースは [CREATE TABLESPACE](#) を利用してあらかじめ作成されている必要があります。このテーブルオプションは MySQL 5.1.6 で紹介されています。詳しくは[MySQL Cluster ディスク データ ストレージ](#) を参照してください。

[ENGINE](#) テーブル オプションはテーブルにストレージ エンジン指定します。

[ENGINE](#) テーブル オプションは、次のテーブルに表されているストレージ エンジン名を利用します。

ストレージ エンジン	説明
ARCHIVE	アーカイブ ストレージ エンジン詳しくは ARCHIVE ストレージエンジン を参照してください。
CSV	カンマで区切られた値のフォーマットで行を格納するテーブル詳しくは CSV ストレージエンジン を参照してください。
EXAMPLE	例エンジン詳しくは EXAMPLE ストレージエンジン を参照してください。
FEDERATED	リモート テーブルにアクセスするストレージ エンジン詳しくは FEDERATED ストレージエンジン を参照してください。
HEAP	これは MEMORY の同義語です。
ISAM (OBSOLETE)	MySQL 5.1 では無効です。もし古いバージョンから MySQL 5.1 にアップグレードするなら、その作業の 前に 全ての既存 ISAM テーブルを MyISAM に変換しなければいけません。
InnoDB	行ロックと外部キーを持つトランザクション セーフ テーブル詳しくは InnoDB ストレージ エンジン を参照してください。
MEMORY	このストレージ エンジンのデータはメモリの中だけに格納されます。詳しくは MEMORY (HEAP) ストレージエンジン を参照してください。

MERGE	1つのテーブルとして利用される MyISAM テーブルの集まり。また、 MRG_MyISAM としても知られています。詳しくは MERGE ストレージエンジン を参照してください。
MyISAM	MySQL に利用されるデフォルト ストレージ エンジンであるバイナリ ポータブル ストレージ エンジン。詳しくは MyISAM ストレージエンジン を参照してください。
NDBCLUSTER	クラスタ化された、耐障害性の、メモリ ベースのテーブル。また、 NDB としても知られています。詳しくは 14章MySQL Cluster を参照してください。

もし適応しないストレージ エンジンが指定されると、MySQL は代わりにデフォルト エンジンを利用します。通常は [MyISAM](#) です。例えば、テーブル定義が [ENGINE=INNODB](#) オプションを含み、MySQL サーバが [INNODB](#) テーブルをサポートしなければ、そのテーブルは [MyISAM](#) テーブルとして作成されます。これは、トランザクショナル テーブルをマスタ上に持つが、スレーブ上に作成されたテーブルが非トランザクショナルである場合(スピードを速める為)の複製設定を可能にします。MySQL 5.1 では、ストレージ エンジン仕様が支持されていなければ警告が表示されます。

注意:古い [TYPE](#) オプションは [ENGINE](#) と同義語です。MySQL 4.0 以降、[TYPE](#) は廃止されましたが、MySQL 5.1 (MySQL 5.1.7 となる予定)ではまだ後方互換性の為にサポートされています。MySQL 5.1.8 より、警告が表示されるようになりました。これはMySQL 5.2 では廃止される予定です。新しいアプリケーションの中では [TYPE](#) は利用すべきではありません、また、代わりに [ENGINE](#) を利用する為に、既存アプリケーションの変換を今すぐ始めるべきです。(詳しくは [Changes in release 5.1.8 \(Not released\)](#) を参照してください。)

その他のテーブル オプションはテーブルの動作を最適化する為に利用されます。ほとんどの場合、それらのうちのどれも指定する必要はありません。これらのオプションは、指示されない限り全てのストレージ エンジンに適応します。与えられたストレージ エンジンに適応しないオプションは、受け入れられ、テーブル定義の一部として記憶されるでしょう。そのようなオプションは、後程もし異なるストレージ エンジンの利用の為にテーブルを変換する時 [ALTER TABLE](#) を利用すれば適応されます。

• [AUTO_INCREMENT](#)

テーブルの初期 [AUTO_INCREMENT](#) 値。MySQL 5.1 では、これは [MyISAM](#)、[MEMORY](#)、そして [InnoDB](#) テーブルに対して機能します。これはまた、MySQL 5.1.6 以降も [ARCHIVE](#) テーブルに対しても機能します。[AUTO_INCREMENT](#) テーブル オプションをサポートしないエンジンに最初の自動インクリメント値を設定する為に、テーブルを作成した後、求められる値よりの1少ない値の「dummy」行を挿入し、その後ダミー行を削除してください。

[CREATE TABLE](#) ステートメント内の [AUTO_INCREMENT](#) テーブル オプションをサポートするエンジンには、[AUTO_INCREMENT](#) 値をリセットする為に [ALTER TABLE tbl_name AUTO_INCREMENT = N](#) を利用する事もできます。その値は、現在カラム内にある最大値よりも小さく設定する事はできません。

• [AVG_ROW_LENGTH](#)

テーブルの平均行長近似値です。可変サイズ行を持つ大きいテーブルに対してのみ、これを設定する必要があります。

[MyISAM](#) テーブルを作成する時、MySQL はテーブルが最終的にどの程度の大きさになるのかを決める為に [MAX_ROWS](#) と [AVG_ROW_LENGTH](#) オプションの製品を利用します。もしどちらのオプションも指定しなければ、テーブルの最大サイズはデフォルトで256 TB になります。(もし使用している OS がその大きさのファイルをサポートしていなければ、テーブル サイズはファイル サイズ制限に制約されます。)もし大きいファイルが必要無く、インデックスを小さく早くする為にポインタサイズを小さくしたければ、[myisam_data_pointer_size](#) システム変数を設定する事でデフォルトのポインタ サイズを小さくする事ができます。(詳しくは [システム変数](#) をご確認ください。) もし全てのテーブルをデフォルトの制限よりも大きくする事を希望し、必要以上にテーブルが遅く、大きくなっても良いのであれば、この変数を設定する事でデフォルトのポインタサイズを増やす事ができます。

• [\[DEFAULT\] CHARACTER SET](#)

テーブルにデフォルト文字セットを指定します。[CHARSET](#) は [CHARACTER SET](#) の同義語です。

• [CHECKSUM](#)

MySQL に全ての行のライブ チェックサムを維持させたいければこれを1に設定してください。(これはテーブルが変更される度に MySQL が自動的に更新するチェックサムです。)これはテーブルの更新スピードを少し遅くします

が、壊れたテーブルを見つけるのが早くなります。CHECKSUM TABLE ステートメントはチェックサムをレポートします。(MyISAM のみです。)

- **COLLATE**

テーブルにデフォルト照合を指定します。

- **COMMENT**

最高60文字のテーブルに対するコメントです。

- **CONNECTION**

FEDERATED テーブルの接続文字列です。(注意:MySQL の古いバージョンは COMMENT オプションを接続文字列に利用していました。)

- **DATA DIRECTORY、INDEX DIRECTORY**

DATA DIRECTORY='directory' か INDEX DIRECTORY='directory' を利用する事で、MyISAM ストレージ エンジンがテーブルのデータ ファイルとインデックス ファイルをどこに置く必要があるのかを指定する事ができます。ディレクトリは、相対パス名ではなくフルパス名でなければいけません。

これらのオプションは --skip-symbolic-links オプションを利用していない時だけ機能します。OS にはまた、有効なスレッドセーフな realpath() コールがなければいけません。詳細については、[Unix 上のテーブルに対するシンボリックリンクの使用](#) をご参照ください。

- **DELAY_KEY_WRITE**

キー更新をテーブルが閉じられる時まで遅らせたければこれを1に設定してください。システム変数 内の delay_key_write システム変数についての説明を参照してください。(MyISAM のみです。)

- **INSERT_METHOD**

もしデータを MERGE テーブルに挿入したければ、その行が挿入されるべきテーブルのINSERT_METHOD を利用して指定する必要があります。INSERT_METHOD は MERGE テーブルにのみ有効なオプションです。最初か最後のテーブルに挿入する為には FIRST か LAST 値を、また挿入を防ぐ為には NO 値を利用してください。詳しくは [MERGE ストレージエンジン](#) を参照してください。

- **KEY_BLOCK_SIZE**

このオプションはインデックス キー ブロックに利用するサイズについてストレージ エンジンにヒントを提供します。このエンジンは必要に応じて値を変更する事が可能です。0という値は、デフォルト値を利用しなければいけないという事を表しています。テーブル値を無効にする為に、個々のインデックス定義はそれ自身の KEY_BLOCK_SIZE 値を指定する事ができます。KEY_BLOCK_SIZE は MySQL 5.1.10 で追加されました。

- **MAX_ROWS**

テーブル内に格納する予定の最大行数。これは、厳しい制限というよりは、ストレージ エンジンに対して、テーブルが少なくともこの程度の行数を格納できる必要があるという事を表すヒントのような物です。

- **MIN_ROWS**

テーブル内に格納する予定の最小行数

- **PACK_KEYS**

PACK_KEYS は MyISAM テーブルとだけ効果を発揮します。小さいインデックスを持ちたければ、このオプションを1に設定してください。これは通常更新スピードを遅くし、読み込みを早くします。オプションを0に設定すると、全てのキー パッキングが無効になります。これを DEFAULT に設定すると、ストレージ エンジンには長い CHAR や VARCHAR カラムだけをパックするように指令が出ます。

もし PACK_KEYS を利用しなければ、デフォルトでは文字列をパックしますが、数字はパックしません。もし PACK_KEYS=1 を利用すると、数字もパックされます。

バイナリ数値キーをパックする時、MySQL はプリフィックス圧縮を利用します。

- 前のキーのいくつかのバイト分が次のキーと同じであることを示す為に、全てのキーは1バイト分多く必要とします。
- 行のポインタは、圧縮を強化する為に、キーの直後に高バイト順で直接格納されます。

これは、もし2つの連続した行上に複数の同等なキーを持っていたら、全ての後続する「同じ」キーは通常2バイトしか利用しないという事を意味します。(行のポインタを含む)これを、後続キーが `storage_size_for_key + pointer_size` を取る通常のケースと比べてみてください。(ポインタサイズは通常4)反対に、同じ数値を多く持つ場合のみ、プリフィックス圧縮の恩恵を多いに受ける事ができます。もし全てのキーが全く異なり、`NULL` 値を持つ事ができるキーでなければ、1つのキーに対してもう1バイト多く利用する事になります。(この場合、もしキーが `NULL` であれば、パックされたキー長はマークする為に利用された物と同じバイト数で格納されます。)

- **PASSWORD**

パスワードを持つ `.frm` ファイルを暗号化します。このオプションは、スタンダード MySQL バージョン内では何も行いません。

- **RAID_TYPE**

RAID サポートは MySQL 5.0 以降は削除されました。RAID の情報に関しては、<http://dev.mysql.com/doc/refman/4.1/en/create-table.html> を参照してください。

- **ROW_FORMAT**

行がどのように格納されるべきかを定義します。`MyISAM` テーブルに対しては、静的、または可変長行フォーマットのオプション値は `FIXED` か `DYNAMIC` になり得ます。`myisampack` はタイプを `COMPRESSED` に設定します。詳しくは [MyISAM テーブルストレージフォーマット](#) を参照してください。

`InnoDB` テーブルに対しては、行はデフォルトでコンパクト フォーマットに格納されます。
(`ROW_FORMAT=COMPACT`)`ROW_FORMAT=REDUNDANT` を指定する事により、MySQL の古いバージョンで利用される非コンパクト フォーマットをリクエストする事もできます。

- **UNION**

`UNION` は、同一の `MyISAM` テーブルの集まりを1つの物としてアクセスしたい時に利用する事ができます。これは `MERGE` テーブルとのみ機能します。詳しくは [MERGE ストレージエンジン](#) を参照してください。

`MERGE` テーブルにマップするテーブルに、`SELECT`、`UPDATE`、そして `DELETE` 権限を持たなければいけません。(注意:以前は、全てのテーブルは、`MERGE` テーブルそのものと同じデータベース内にある必要がありました。この制限はもう適応されません。)

`partition_options` は `CREATE TABLE` を利用して作成されたテーブルの領域確保をコントロールする為に利用できます。重要:このセクションの冒頭で紹介された `partition_options` の構文内に表された全てのオプションが、全ての領域確保タイプに有効であるわけではありません。テーブル作成と MySQL 領域確保に関係する他のステートメントの追加例だけでなく、MySQL 内での領域確保の機能と使用に関しての完全な情報については、各タイプの情報仕様の為の個別タイプをリストした物、そして [15章パーティショニング](#) をみてください。

`partition_options` は、もし利用されるなら最小の `PARTITION BY` 条項を含む必要があります。この条項はパーティションを決めるのに利用される関数を含んでいます。その関数は、`num` が分割数の時、1から `num` の整数値を返します。(1つのテーブルが含むユーザー定義パーティションの最高数は1024です。このセクション — の後半で紹介される — サブパーティションはこの最高数に含まれています。)MySQL 5.1 でこの機能に対して有効な物は次のリストに表されています。

- **HASH(expr):**行を置く為のキーを作成する為に1つ、または複数のカラムをハッシュします。`expr` は1つ、または複数のテーブルカラムを利用する式です。これは、単一整数値を生む正当な MySQL 式(MySQL 関数を含む)であればどれでもよいです。例えば、これらは全て `PARTITION BY HASH` を利用した有効な `CREATE TABLE` ステートメントです。

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
PARTITION BY HASH(col1);

CREATE TABLE t1 (col1 INT, col2 CHAR(5))
PARTITION BY HASH( ORD(col2) );

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
```



```
PARTITION BY HASH ( YEAR(col3) );
```

PARTITION BY HASH と、**VALUES LESS THAN** や **VALUES IN** 条項は一緒に利用しません。

PARTITION BY HASH はパーティション数によって分割された (モジュール)**expr** の残りを利用します。追加情報については [HASH パーティショニング](#) を参照してください。

LINEAR キーワードは異なるアルゴリズムを若干必要とします。この場合、行が格納されるパーティション数は、1つ、または複数の論理的な **AND** 操作の結果計算されます。線形ハッシングについての説明と例に関しては、[LINEAR HASH パーティショニング](#) を参照してください。

- **KEY(column_list)**:MySQL がハッシング機能を均等なデータ分布を保障する為に提供するという事以外、これは **HASH** と似ています。**column_list** 引数は単にテーブル カラムのリストです。この例は、キーによって4つのパーティションに分割された単純なテーブルを表しています。

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY KEY(col3)
PARTITIONS 4;
```

キーによって分割されたテーブルには、**LINEAR** キーワードを利用した線形領域確保を採用する事ができます。これは **HASH** によって分割されたテーブルと同じ効果を持ちます。これは、分割数はモジュールではなく **&** 演算子を利用して求められるという事を意味します。(詳細に関しては [LINEAR HASH パーティショニング](#)、と [KEY パーティショニング](#) を参照してください。)この例は、5つのパーティション間でデータを分布する為にキーによる線形領域確保を利用しています。

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR KEY(col3)
PARTITIONS 5;
```

PARTITION BY KEY と、**VALUES LESS THAN** や **VALUES IN** 条項は一緒に利用しません。

- **RANGE**:この場合、**expr** は **VALUES LESS THAN** 演算子のセットを利用して値の範囲を表します。範囲の領域確保を利用する時、**VALUES LESS THAN** を利用して最低1つのパーティションを定義する必要があります。**VALUES IN** を範囲の領域確保に利用する事はできません。

VALUES LESS THAN は直定数値、または単一値を評価する式のどちらかと一緒に利用する事ができます。

次のスキームに従って、年次値を含むカラム上で分割したいテーブルを持っていると仮定します。

パーティション数:	年次範囲:
0	1990以前
1	1991 – 1994
2	1995 – 1998
3	1999 – 2002
4	2003 – 2005
5	2006以降

そのような領域確保スキームを実施するテーブルはここに表されている **CREATE TABLE** ステートメントによって実現されます。

```
CREATE TABLE t1 (
  year_col INT,
  some_data INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999),
  PARTITION p3 VALUES LESS THAN (2002),
  PARTITION p4 VALUES LESS THAN (2006),
  PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

PARTITION ... VALUES LESS THAN ... ステートメントは連続法で機能します。**VALUES LESS THAN MAXVALUE** は、指示されない限り、最大値よりも大きい「leftover」値を指定する為に機能します。

VALUES LESS THAN 条項は **switch ... case** ブロックの **case** 部分と似た方法で連続的に機能するという事を覚えて置いてください。(C、Java、そしてPHP等のような多くのプログラム言語内で見られるのと同じように)これは、それぞれの連続した **VALUES LESS THAN** 内で指定された上限はその前の物よりも大きく、**MAXVALUE** に参照を付ける物がリストの一番最後に来るという方法で条項を配列しなければいけない、という事を意味します。

- **LIST(expr)**:これは州や国のコードなどのような、制限された値のセットを持つテーブル カラムに基づいたパーティションを割り当てる時に便利です。このような場合、特定の州や国に関係している行を1つのパーティションに指定したり、または、特定の州や国のセットに対してパーティションを用意しておく事ができます。これは、**VALUES IN** だけが各パーティションに許容値を指定するのに利用されるという事を除いて、**RANGE** と似ています。

VALUES IN はマッチする値のリストと共に利用されます。例えば、次のように領域確保スキームを作成する事ができます。

```
CREATE TABLE client_firms (
  id INT,
  name VARCHAR(35)
)
PARTITION BY LIST (id) (
  PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
  PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
  PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
  PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

リストの領域確保を利用する時、**VALUES IN** を利用して最低1つのパーティションを定義する必要があります。**VALUES LESS THAN** を **PARTITION BY LIST** と一緒に利用する事はできません。

注意:現在は、**VALUES IN** と共に利用される値のリストは整数値のみで構成されています。

- パーティション数は、**num** がパーティション数である **PARTITIONS num** 条項を利用して任意に指定されます。この条項、そして **PARTITION** 条項が両方利用されたら、**num** は **PARTITION** 条項を利用して宣言されたパーティションの合計数と同一でなければいけません。

注意:**RANGE** か **LIST** によって領域確保されたテーブルを作成するのに **PARTITIONS** 条項を利用したかどうかに関わらず、テーブル定義の中に最低1つ以上の **PARTITION VALUES** 条項を含む必要があります。(下記参照)

- パーティションは任意でサブパーティションに分解する事もあります。これは任意の **SUBPARTITION BY** 条項を利用して指示する事ができます。サブパーティションは **HASH** か **KEY** によって行われるでしょう。これらのどちらかは **LINEAR** でしょう。これらは、既に説明された同等な領域確保のタイプと同じように機能します。(**LIST** や **RANGE** でサブパーティションするのは不可能です。)

整数値が後に続く **SUBPARTITIONS** キーワードを利用してサブパーティション数を指示する事ができます。

- MySQL 5.1.12 で、**PARTITIONS** や **SUBPARTITIONS** 条項で利用された値の厳密なチェックを紹介しています。このバージョンから、この値は次のルールを遵守します。
 - 値は正数、ゼロ以外の整数でなければいけません。
 - ゼロが前に付いてはいけません。
 - 値は整数直定数である必要があり、式にはなり得ません。例えば、**0.2E+01** が **2** であると評価されたとしても、**PARTITIONS 0.2E+01** は許されません。(バグ #15890)

各パーティションは **partition_definition** 条項を利用して個別に定義されるでしょう。この条項を形成するそれぞれの部分は次のような物です。

- **PARTITION partition_name**:これはパーティションの論理名を指定します。
- **VALUES** 条項:範囲の領域確保に関しては、各パーティションが **VALUES LESS THAN** 条項を含む必要があり、リストの領域確保に関しては、各パーティションに対して **VALUES IN** 条項を指定する必要があります。これは、どの行がこのパーティションに格納されるのかという事を決める為に利用されます。構文例に関しては、[15章パーティショニング](#) のパーティション タイプに関する説明を参照してください。

- 任意の **COMMENT** 条項はパーティションを説明する為に利用されるでしょう。このコメントは単一引用句で始まらなければいけません。例:

```
COMMENT = 'Data for the years previous to 1999'
```

- DATA DIRECTORY** と **INDEX DIRECTORY** は、このパーティションのデータとインデックスがそれぞれどこに格納されるのかを指示する為に利用されます。 `data_dir` to `index_dir` の両方は、絶対的なシステムパスネームでなければいけません。例:

```
CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
PARTITION BY LIST(YEAR(adate))
(
  PARTITION p1999 VALUES IN (1995, 1999, 2003)
    DATA DIRECTORY = '/var/appdata/95/data'
    INDEX DIRECTORY = '/var/appdata/95/idx',
  PARTITION p2000 VALUES IN (1996, 2000, 2004)
    DATA DIRECTORY = '/var/appdata/96/data'
    INDEX DIRECTORY = '/var/appdata/96/idx',
  PARTITION p2001 VALUES IN (1997, 2001, 2005)
    DATA DIRECTORY = '/var/appdata/97/data'
    INDEX DIRECTORY = '/var/appdata/97/idx',
  PARTITION p2000 VALUES IN (1998, 2002, 2006)
    DATA DIRECTORY = '/var/appdata/98/data'
    INDEX DIRECTORY = '/var/appdata/98/idx'
);
```

DATA DIRECTORY と **INDEX DIRECTORY** は、**MyISAM** テーブルで利用される **CREATE TABLE** ステートメントの `table_option` 条項と同じ形で機能します。

各パーティションには、1つのデータディレクトリとインデックスディレクトリが指定されます。もし指定されなければ、データとインデックスはデフォルトでMySQLデータディレクトリに格納されます。

- パーティション内に格納する行の最大、最小数をそれぞれ指定する為に **MAX_ROWS** と **MIN_ROWS** が利用されます。 `max_number_of_rows` と `min_number_of_rows` の値は正整数でなければいけません。同名のテーブルレベルオプションと同様に、これらはサーバに対してただの「提案」として機能し、厳しい制限ではありません。
- 任意の **TABLESPACE** 条項は、パーティションのテーブルスペースを指定するのに利用されるでしょう。MySQL クラスタにのみ利用されます。
- 任意の **[STORAGE] ENGINE** 条項は、このMySQLサーバにサポートされたエンジンであるこのパーティション内のテーブルが、指定されたストレージエンジンを利用するように機能します。**STORAGE** キーワードとイコールのサイン(=)の両方は任意です。もしこのオプションを利用して、パーティションを特定するストレージエンジンが設定されなければ、テーブル全体に適用するエンジンがこのパーティションで利用されます。

注意:領域確保ハンドラは **PARTITION** と **SUBPARTITION** の両方に対して **[STORAGE] ENGINE** オプションを受け入れます。現在これを利用する唯一の方法は、全てのパーティション、サブパーティションを同じストレージエンジンに設定する事です。もし同じテーブル内で、パーティション、サブパーティションに異なるストレージエンジンを設定しようとするエラーが発生します。**エラー 1469 (HY000): MySQL のこのバージョンではパーティション内でハンドラを混在させる事は許可されていません。**。将来のMySQL 5.1 リリース時には、領域確保に関するこの制限を取り除く予定です。

- NODEGROUP** オプションは `node_group_id` によって確認されたノードグループの一部としてこのパーティションを機能させる為に利用できます。このオプションはMySQLクラスタに対してだけ利用可能です。
- パーティション定義は、1つかそれ以上の `subpartition_definition` 条項を含みます。これらはそれぞれ、`name` が識別子のサブパーティションである **SUBPARTITION name** の最小値で構成されます。**SUBPARTITION** を利用した **PARTITION** キーワードの入れ替え以外は、サブパーティション定義の構文はパーティション定義の構文と同一です。

サブパーティションは **HASH** か **KEY** によって行われなければならない必要があり、そして **RANGE** か **LIST** パーティション上のみで行われます。詳しくは **サブパーティショニング** を参照してください。

パーティションは修正し、マージし、テーブルに追加し、テーブルからドロップする事ができます。これらのタスクを成し遂げる為の基本的なMySQLステートメントについての情報は、**ALTER TABLE 構文** を参照してください。更なる詳細説明や例に関しては、**パーティショニング管理** を参照してください。

CREATE TABLE ステートメントの最後に **SELECT** を追加する事で、1つのテーブルから別のテーブルを作成する事ができます。

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQLは **SELECT** 内の全ての要素に対して新しいカラムを作成します。例:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,  
-> PRIMARY KEY (a), KEY(b))  
-> ENGINE=MyISAM SELECT b,c FROM test2;
```

これは、これらの3つのカラム **a**、**b**、そして **c** を利用して **MyISAM** テーブルを作成します。**SELECT** ステートメントからのカラムは、テーブル上に重ねられるのではなくテーブルの右側に添付される事を覚えて置いてください。次の例を参考にして下さい。

```
mysql> SELECT * FROM foo;  
+---+  
| n |  
+---+  
| 1 |  
+---+  
  
mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;  
Query OK, 1 row affected (0.02 sec)  
Records: 1 Duplicates: 0 Warnings: 0  
  
mysql> SELECT * FROM bar;  
+-----+  
| m | n |  
+-----+  
| NULL | 1 |  
+-----+  
1 row in set (0.00 sec)
```

foo テーブル内のそれぞれの行には、**foo** からの値と新しいカラムのデフォルト値と共に **bar** に行が挿入されます。

CREATE TABLE ... SELECT の結果出来るテーブル内では、**CREATE TABLE** 部分の中でのみ名づけられたカラムが最初に来ます。両方で名づけられたカラムが **SELECT** 部分の中でのみ名づけられたカラムがその後に来ます。**SELECT** カラムのデータ タイプは **CREATE TABLE** 部分の中でカラムを指定する事によって無効にする事もできます。

もしデータをテーブルにコピーしている間にエラーが発生すると、それは自動的にドロップされるので作成されません。

CREATE TABLE ... SELECT は自動的にインデックスを作成しません。これはステートメントを可能な限りフレキシブルにする為に意図的に行われます。もし作成したテーブルの中でインデックスを持ちたければ、**SELECT** ステートメントの前にこれらを指定しなければいけません。

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

データ タイプの変換が行われるかもしれませんが。例えば、**AUTO_INCREMENT** 属性は保管されず、**VARCHAR** カラムが **CHAR** カラムになる事ができます。

CREATE ... SELECT でテーブルを作成する時、必ずクエリの中では 全ての関数呼び出しや式をエイリアスにしてください。もしそれをしなければ、**CREATE** ステートメントは失敗するか、望まないカラム名になってしまいます。

```
CREATE TABLE artists_and_works  
SELECT artist.name, COUNT(work.artist_id) AS number_of_works  
FROM artist LEFT JOIN work ON artist.id = work.artist_id  
GROUP BY artist.id;
```

発生したカラムに対して、データ タイプを明示的に指定する事もできます。

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

元テーブルの中で指定されたカラム属性やインデックスを含む、他のテーブルの定義に基づき空のテーブルを作成するには、**LIKE** を利用してください。

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

コピーは元テーブルと同じバージョンのテーブル ストレージ フォーマットを利用して作成されます。

CREATE TABLE ... LIKE は、元テーブルに対して、または外部キー定義に対して指示された **DATA DIRECTORY** や **INDEX DIRECTORY** テーブル オプションを保管しません。

ユニーク キー値を複製する行をどのように扱うかを指示する為に、**IGNORE** か **REPLACE** によって **SELECT** を先行させる事ができます。**IGNORE** 利用すると、ユニーク キー値上に既存の行を複製する新しい行は廃棄されます。**REPLACE** を利用すると、新しい行は同じユニーク キー値を持つ行を置き換えます。もし **IGNORE** も **REPLACE** も指示されなければ、複製ユニーク キー値はエラーになります。

バイナリ ログが元テーブルを再作成する為に利用できる事を保障する為に、MySQL は **CREATE TABLE ... SELECT** の最中の並列挿入を許可しません。

サイレント カラム仕様変更

いくつかのケースでは、MySQL は **CREATE TABLE** か **ALTER TABLE** ステートメント内で与えられたカラム仕様を静かに変更します。これらは、データ タイプ、データ タイプに関連する属性、またはインデックス仕様にとっての変更になります。

- **TIMESTAMP** ディスプレイ サイズは廃棄されます。
- **PRIMARY KEY** の一部であるカラムは、そのように宣言されなくても **NOT NULL** にされます。
- テーブルが作成された時、**ENUM** と **SET** メンバー値から後続スペースが自動的に削除されます。
- MySQL は他の SQL データベース ベンダーに利用された特定のデータ タイプを MySQL タイプにマップします。詳しくは [その他のデータベースエンジンのデータタイプの利用](#) を参照してください。
- もし規定のストレージ エンジンに対して正当ではないインデックス タイプを指定する為に **USING** を含み、しかしクエリに影響を与えずにそのエンジンが利用できる別の有効なインデックス タイプが存在すれば、エンジンはその有効なタイプを利用します。

MySQL が、指定したデータ タイプ以外の物を利用したかどうかを確認する為には、テーブルを作成、または変更した後に **DESCRIBE** か **SHOW CREATE TABLE** ステートメントを発行してください。

もし **myisampack** を利用してテーブルを圧縮すると、特定の他のデータタイプの変更を行う事ができます。詳しくは [圧縮テーブルの特徴](#) を参照してください。

CREATE LOGFILE GROUP 構文

```
CREATE LOGFILE GROUP logfile_group
ADD UNDOFILE 'undo_file'
INITIAL_SIZE [=] initial_size
[UNDO_BUFFER_SIZE [=] undo_buffer_size]
ENGINE [=] engine_name
```

このステートメントは、'undo_file' と名づけられた単一 **UNDO** ファイルを持つ **logfile_group** という名前の新しいログ ファイル グループを作成する事ができます。**CREATE LOGFILE GROUP** ステートメントはたった一つの **ADD UNDOFILE** 条項を持ちます。

MySQL 5.1.8 から、いつでも1つのクラスタに付き1つだけのログ ファイル グループを持つ事ができるようになりました。(バグ #16386を参照して下さい。)

INITIAL_SIZE パラメータは **UNDO** ファイルの初期サイズを設定します。任意の **UNDO_BUFFER_SIZE** パラメータは、ログ ファイル グループに対して **UNDO** バッファに利用されるサイズを設定します。**UNDO_BUFFER_SIZE** のデフォルト値は **8M** (8メガバイト)で、この値は有効なシステム メモリの量を超える事はできません。**initial_size** と **undo_buffer_size** の両方はバイトで指示されます。**my.cnf** で利用されている物と同様、大きさによって一文字の省略形を持つこれらのうちのどちらか、または両方に従う事もできます。通常これは **M** (メガ バイト) か **G** (ギガ バイト)のどちらかの文字です。

ENGINE パラメータが、このログ ファイル グループによって利用されるストレージ エンジンを決め、その名前は **engine** となります。MySQL 5.1 では **engine** が **NDB** か **NDBCLUSTER** の値の1つにならなければいけません。

ENGINE = NDB と一緒に利用された時、ログ ファイル グループと、関連する **UNDO** ログ ファイルは、各クラスター データ ノード上に作成されます。**INFORMATION_SCHEMA.FILES** テーブルに問い合わせる事によって **UNDO** ファイルが作成され、それらの情報を得た事を証明する事ができます。例:

```
mysql> SELECT LOGFILE_GROUP_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'undo_10.dat';
```

LOGFILE_GROUP_NAME	LOGFILE_GROUP_NUMBER	EXTRA
lg_3	11	CLUSTER_NODE=3
lg_3	11	CLUSTER_NODE=4

2 rows in set (0.06 sec)

(詳しくは [INFORMATION_SCHEMA FILES テーブル](#) を参照してください。)

MySQL 5.1.6 では **CREATE LOGFILE GROUP** が追加されました。MySQL 5.1 では MySQL クラスターの ディスク データ ストレージと一緒にの時のみ利用する事ができます。詳しくは [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

CREATE TABLESPACE 構文

```
CREATE TABLESPACE tablespace
  ADD DATAFILE 'file'
  USE LOGFILE GROUP logfile_group
  [EXTENT_SIZE [=] extent_size]
  [INITIAL_SIZE [=] initial_size]
  [ENGINE [=] engine]
```

このステートメントは、テーブルに格納スペースを提供しながら1つ、または複数のデータ ファイルを含む事ができるテーブルスペースを作成する為に利用されます。1つのデータ ファイルはこのステートメントを利用してテーブルスペースに作成、または追加されます。**ALTER TABLESPACE** ステートメントを利用してテーブルスペースにデータ ファイルを追加する事ができます。([ALTER TABLESPACE 構文](#) を参照してください。)

1つ、または複数の **UNDO** ログ ファイルのログ ファイル グループは **USE LOGFILE GROUP** 条項を利用して作成する為にテーブルスペースに割り当てする必要があります。**logfile_group** は **CREATE LOGFILE GROUP** で作成した既存のログ ファイル グループでなければいけません。([CREATE LOGFILE GROUP 構文](#) を参照してください。)複数のテーブルスペースは **UNDO** ロギングに同じログ ファイル グループを利用するでしょう。

EXTENT_SIZE は、テーブル スペースに属する全てのファイルに利用される範囲で、サイズをバイトで設定します。デフォルト値は4バイトです。

extent はディスク スペース割り当てのユニットです。1つの範囲は、別の範囲が利用されるまで、可能な限りのできるだけ多くのデータで満たされます。**INFORMATION_SCHEMA.FILES** テーブルをクエリする事によって、与えられたファイルでいくつの範囲がフリーであるかを確認する事ができるので、そのファイルの中でどれくらいのフリー スペースがあるのかの概算が導かれます。更なる情報や例に関しては、[INFORMATION_SCHEMA FILES テーブル](#) を参照してください。

INITIAL_SIZE パラメータはデータ ファイルの合計サイズをバイトで設定します。一度データ ファイルが作成されると、そのサイズは変更できませんが、追加の **ALTER TABLESPACE ... ADD DATAFILE** を利用する事によりテーブルスペースにより多くのデータ ファイルを追加する事ができます。詳しくは [ALTER TABLESPACE 構文](#) を参照してください。

EXTENT_SIZE や **INITIAL_SIZE** を設定する時(片方、または両方)、**my.cnf** で利用されている物と同様、大きさによって一文字の省略形を持つ数字に従う事もできます。通常これは **M** (メガ バイト) か **G** (ギガ バイト)のどちらかの文字です。

ENGINE パラメータが、このテーブルスペースが利用するストレージ エンジンを決め、その名前は **engine** となります。MySQL 5.1 では **engine** が **NDB** か **NDBCLUSTER** の値の1つにならなければいけません。

CREATE TABLESPACE ... ADD DATAFILE が **ENGINE = NDB** と共に利用された時、テーブルスペースと関連するデータ ファイルがそれぞれのクラスター データ ノード上に作成されます。**INFORMATION_SCHEMA.FILES** テーブルに問い合わせる事によってデータ ファイルが作成され、それらの情報を得た事を証明する事ができます。例:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
```



```

-> FROM INFORMATION_SCHEMA.FILES
-> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | FILE_NAME | EXTRA |
+-----+-----+-----+
| lg_3              | newdata.dat | CLUSTER_NODE=3 |
| lg_3              | newdata.dat | CLUSTER_NODE=4 |
+-----+-----+-----+
2 rows in set (0.01 sec)

```

(詳しくは [INFORMATION_SCHEMA FILES テーブル](#) を参照してください。)

MySQL 5.1.6 では [CREATE TABLESPACE](#) が追加されました。MySQL 5.1 では MySQL クラスタのディスク データ ストレージと一緒にの時のみ利用する事ができます。詳しくは [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

CREATE SERVER 構文

```

CREATE SERVER server_name
  FOREIGN DATA WRAPPER wrapper_name
  OPTIONS (option ...)

option:
{ HOST character-literal
| DATABASE character-literal
| USER character-literal
| PASSWORD character-literal
| SOCKET character-literal
| OWNER character-literal
| PORT numeric-literal }

```

このステートメントは [FEDERATED](#) ストレージ エンジンと共に利用する為のサーバの定義を作成します。[CREATE SERVER](#) ステートメントは [mysql](#) データベース内の [servers](#) テーブル内の新しい行を作成します。

server_name はサーバの固有の参照でなければいけません。サーバ定義は、サーバの領域内では広範囲です。特定のデータ ベースにサーバ定義を適応するのは不可能です。*server_name* は最長63文字の長さを持ち(63文字以上の名前は静かに切り捨てられる)、大文字と小文字を区別しません。単一引用句を利用して名前を指定します。

wrapper_name は [mysql](#) となる必要があり、そして1つの引用句を利用して引用されます。*wrapper_name* のその他の値は現在はサポートされていません。

各 *option* に対しては、文字直定数が数値定数のどちらかを指定しなければいけません。文字直定数は UTF 8で、最高64文字の長さでデフォルトを空の文字列にサポートします。文字列定数は静かに64文字まで切り捨てられます。数値定数は0から9999の間の数字である必要があり、デフォルト値は0です。

[CREATE SERVER](#) ステートメントを利用する為に特別な権限は必要ありません。

[CREATE SERVER](#) ステートメントは [mysql.server](#) テーブル内に [FEDERATED](#) テーブルを作成する時に [CREATE TABLE](#) ステートメントと共に利用する事ができるエントリを作成します。指定するオプションは [mysql.server](#) テーブル内にカラムを投入する為に利用されます。テーブルカラムは [Server_name](#)、[Host](#)、[Db](#)、[Username](#)、[Password](#)、[Port](#) そして [Socket](#) です。

例:

```

CREATE SERVER s
  FOREIGN DATA WRAPPER mysql
  OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');

```

テーブルに格納されたデータは [FEDERATED](#) テーブルへの接続を作成する時に利用できます。

```

CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';

```

更なる情報については [FEDERATED ストレージエンジン](#) を参照してください。

[CREATE SERVER](#) は自動コミットを引き起こしません。

DROP DATABASE 構文

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE はデータベース内の全てのテーブルをドロップし、データベースを削除します。このステートメントには 十分 注意してください!**DROP DATABASE** を利用する為には、データベース上の **DROP** 権限が必要です。**DROP SCHEMA** は **DROP DATABASE** の同義語です。

重要:データベースがドロップされる時、データベース上のユーザ権限は自動的にドロップされません。詳しくは **GRANT 構文** を参照してください。

IF EXISTS は、データベースが存在しない時にエラーの発生を防ぎます。

もし象徴的にデータベースとリンクした **DROP DATABASE** を利用すると、リンクと元データベースの両方が削除されます。

DROP DATABASE は削除されたテーブル数を返します。これは、削除された **.frm** ファイルの数と対応しています。

DROP DATABASE ステートメントは、通常作業の最中に MySQL が作成するファイルやディレクトリを与えられたデータベースから削除します。

- これらの拡張子を持つ全てのファイル:

.BAK	.DAT	.HSH	.MRG
.MYD	.MYI	.TRG	.TRN
.db	.frm	.ibd	.ndb
.par			

- もし存在するなら、**db.opt** ファイルです。

MySQL がリストされたばかりのファイルやディレクトリを削除した後、もしデータベース ディレクトリ内に別のファイルやディレクトリが残れば、そのデータベース ディレクトリは削除する事ができません。この場合、残っている全てのファイルやディレクトリを手作業で削除し、**DROP DATABASE** ステートメントをもう一度発行しなければいけません。

mysqladmin でデータベースをドロップする事もできます。詳しくは **mysqladmin — MySQL サーバの管理を行うクライアント** を参照してください。

DROP INDEX 構文

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX はテーブル **tbl_name** から **index_name** と名づけられたインデックスをドロップします。このステートメントは、インデックスをドロップする為に **ALTER TABLE** ステートメントにマップされます。詳しくは **ALTER TABLE 構文** を参照してください。

DROP TABLE 構文

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

DROP TABLE は1つまたは複数のテーブルを削除します。各テーブルごとに **DROP** 権限を持つ必要があります。全てのテーブル データとテーブル定義が 削除されますので、このステートメントには 注意してください!引数リストの中に名前を付けたテーブルのいずれも存在しない場合、MySQL は、存在していなかった為にドロップできなかったテーブルを名称別に表し、エラーを戻しますが、MySQL は既存リスト中のテーブルもすべてドロップします。

重要:テーブルがドロップされる時、テーブル上のユーザー権限は自動的にドロップされません。詳しくは **GRANT 構文** を参照してください。

パーティション テーブルに対しては、**DROP TABLE** がテーブル定義と、その全てのパーティションと、それらのパーティションに格納された全てのデータを恒久的に削除する事を覚えておいて下さい。これはまた、ドロップされたテーブルに関連する領域確保定義(**.par**)ファイルも削除します。

[IF EXISTS](#)を利用して、存在していないテーブルに対してエラーが発生するのを防いでください。[IF EXISTS](#)を使用すると、実在していないテーブルに対して [NOTE](#) が生成されます。詳しくは [SHOW WARNINGS 構文](#) を参照してください。

[RESTRICT](#) と [CASCADE](#) がポーティングを簡単にすることができます。MySQL 5.1 ではそれらは何もしません。

注意:[DROP TABLE](#) は、[TEMPORARY](#) キーワードを利用しない限り自動的に現在のアクティブなトランザクションを行います。

[TEMPORARY](#) キーワードは次の効果を持ちます。

- ステートメントは [TEMPORARY](#) テーブルだけをドロップします。
- ステートメントは進行中のトランザクションを終了しません。
- アクセス権をチェックしません。(TEMPORARY テーブルはそれを作成したクライアントだけが見る事ができる物なので、チェックは必要ありません。)

[TEMPORARY](#) を利用するのは、非 [TEMPORARY](#) テーブルを誤ってドロップしない事を保障するのに有効な方法です。

DROP LOGFILE GROUP 構文

```
DROP LOGFILE GROUP logfile_group
ENGINE [=] engine
```

このステートメントは [logfile_group](#) 名前が付いたログ ファイル グループをドロップします。ログ ファイル グループが既に存在していなければエラーが発生します。(ログ ファイル グループの作成についての詳細は [CREATE LOGFILE GROUP 構文](#) を参照してください。)

重要:ログ ファイル グループをドロップする前に、[UNDO](#) ロギングにそのログ ファイル グループを利用する全てのテーブルスペースをドロップしなければいけません。

必要とされる [ENGINE](#) 条項は、ドロップされるログ ファイル グループが利用するストレージ エンジン名を提供します。MySQL 5.1 では、[engine](#) に許可される値は [NDB](#) と [NDBCLUSTER](#) だけです。

MySQL 5.1.6 では [DROP LOGFILE GROUP](#) が追加されました。MySQL 5.1 では MySQL クラスタのディスク データ ストレージと一緒にの時のみ利用する事ができます。詳しくは [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

DROP TABLESPACE 構文

```
DROP TABLESPACE tablespace
ENGINE [=] engine
```

このステートメントは [CREATE TABLESPACE](#) を利用して予め作成されたテーブルスペースをドロップします。(CREATE TABLESPACE 構文 を参照してください。)

重要:ドロップされるテーブルスペースはいかなるデータ ファイルも含む事はできません。言い換えると、テーブルスペースをドロップする前に、[ALTER TABLESPACE ... DROP DATAFILE](#) を利用してそれぞれのデータ ファイルをドロップしなければいけない、という事です。(ALTER TABLESPACE 構文 を参照してください。)

[ENGINE](#) 条項(要求された)はテーブルスペースに利用されるストレージ エンジン指定します。MySQL 5.1では、[engine](#) に受け入れられる値は [NDB](#) と [NDBCLUSTER](#) だけです。

MySQL 5.1.6 では [DROP TABLESPACE](#) が追加されました。MySQL 5.1 では MySQL クラスタのディスク データ ストレージと一緒にの時のみ利用する事ができます。詳しくは [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

DROP SERVER 構文

```
DROP SERVER [ IF EXISTS ] server_name
```

[server_name](#) と名づけられたサーバの、サーバ定義をドロップします。[mysql.servers](#) テーブル内の対応する行は削除されます。

テーブルの為にサーバをドロップする事は、作成時にこの接続情報を利用した **FEDERATED** テーブルに影響を与えません。詳しくは **CREATE SERVER 構文** を参照してください。

DROP SERVER を利用する為に特別な権限は必要ありません。

DROP SERVER は自動コミットを引き起こしません。

RENAME DATABASE 構文

```
RENAME {DATABASE | SCHEMA} db_name TO new_db_name;
```

このステートメントはデータベースをリネームします。これは、データベースの **ALTER** と **DROP** 権限、そして新しいデータベースの **CREATE** 権限を必要とします。**RENAME SCHEMA** は **RENAME DATABASE** の同義語です。

サーバがこのステートメントを受け取る時、新しいデータベースを作成します。そしてそれは、テーブルと、トリガなどのようなその他のデータベース オブジェクトを新しいデータベースに移動します。それはまた、格納されたルーチンやイベントなどのようなオブジェクトのシステム テーブルに **Db** カラムを更新します。最後に、サーバは古いデータベースをドロップします。

現在はこれらの制限がありますので注意してください。

- **RENAME DATABASE** はシステム テーブル内にリストされたアカウント権限を変更しません。それらは手動で変更しなければいけません。
- **RENAME DATABASE** は、格納されたルーチンやイベントを新しいスキーマ名に移動しません。これは、次のような事を意味します。
 - 格納されたルーチンに対しては、**INFORMATION_SCHEMA.ROUTINES** テーブルの **ROUTINE_SCHEMA** カラム内の、そして **mysql.proc** テーブルの **db** カラムの値の変更を行いません。
 - イベントに対しては、**INFORMATION_SCHEMA.EVENTS** テーブルの **EVENT_SCHEMA** カラム内の、そして **mysql.event** テーブルの **db** カラムの値の変更を行いません。

このステートメントは、MySQL 5.1.7 で追加されました。

RENAME TABLE 構文

```
RENAME TABLE tbl_name TO new_tbl_name
[, tbl_name2 TO new_tbl_name2] ...
```

このステートメントは1つ、または複数のテーブルのリネームをします。

リネームは自動的に行われますので、その他のスレッドはリネーム作業中はどのテーブルにもアクセスできません。例えば、もし既存テーブル **old_table** があるとしたら、同じ構成で中身が空の別のテーブル **new_table** を作成し、その後、次のように既存テーブルと空のテーブルを入れ替える事ができます。(**backup_table** は存在していないと仮定する):

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

もしこのステートメントが複数のテーブルをリネームすると、残りの作業は左から右に行われます。もし2つのテーブル名の入れ替えをしたければ、このように行う事ができます。(**tmp_table** が存在しないと仮定する):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

2つのデータベースが同じファイル システム上にある限り、テーブルを1つのデータベースから別のものに移動するのに **RENAME TABLE** を利用する事ができます。

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

テーブルと関連し、**RENAME TABLE** を利用して別のデータベースに移動されたトリガがあると、ステートメントは **Trigger in wrong schema** というエラーになります。

RENAME TABLE はまた、別のデータベース内にビューをリネームしようとしなければ、ビューに対しても機能します。

リネームされたテーブルやビューに与えられた権限は、新しい名前に移動しません。それらは手動で変更しなければいけません。

RENAME を実行する時、ロックされたテーブルやアクティブなトランザクションを持つ事はできません。また、元テーブル上に **ALTER** と **DROP** 権限を、新しいテーブル上には **CREATE** と **INSERT** 権限を持つ必要があります。

もし MySQL が複合テーブルのリネームをしている最中にエラーが発生すると、全てを元に戻す為に全てのリネームされたテーブルにリバース リネームを行います。

データ取り扱いステートメント

DELETE 構文

単一テーブル構文:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

複合テーブル構文:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
tbl_name[*] [, tbl_name[*]] ...
FROM table_references
[WHERE where_condition]
```

または:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM tbl_name[*] [, tbl_name[*]] ...
USING table_references
[WHERE where_condition]
```

単一テーブル構文に対しては、**DELETE** ステートメントが **tbl_name** から行を削除し、削除された行数を返します。もし **WHERE** 条項が与えられたら、それはどの行を削除するべきかを決定します。**WHERE** 条項が無ければ、全ての行が削除されます。もし **ORDER BY** 条項が指定されると、指定された順に行が削除されます。**LIMIT** 条項は、削除できる行数に制限を設定します。

複合テーブル条項に対しては、**DELETE** はそれぞれの **tbl_name** から条件を満たしている行を削除します。この場合、**ORDER BY** と **LIMIT** を利用する事はできません。

where_condition は削除される各行に対して正しい結果の式です。それは **SELECT 構文** で述べられている通りに指定されます。

述べられているように、**WHERE** を持たない **DELETE** ステートメントは全ての行を削除します。削除された行数を知る必要がない場合にこの作業を速く行うには、**TRUNCATE TABLE** を利用してください。詳しくは **TRUNCATE 構文** を参照してください。

もし **AUTO_INCREMENT** カラムに対して最大値を持つ行を削除すると、その値は **MyISAM** か **InnoDB** テーブルに再度利用される事はありません。**AUTOCOMMIT** モードで **DELETE FROM tbl_name** を利用してテーブルの中の全ての行を削除すると(**WHERE** 条項なしで)、**InnoDB** と **MyISAM** 以外は全てのストレージ エンジンに対してシーケンスが始まります。**AUTO_INCREMENT カラムが InnoDB 内でどのように機能するか** で紹介されているように、**InnoDB** テーブルに対するこの動作にはいくつか例外があります。

MyISAM テーブルには、複合カラム キー内の **AUTO_INCREMENT** セカンダリ カラムを指定する事ができます。この場合、**MyISAM** テーブルに対しても、シーケンスの先頭から削除された値の再利用が行われます。詳しくは **Using AUTO_INCREMENT** を参照してください。

DELETE ステートメントは次の修飾因子をサポートします。

- もし **LOW_PRIORITY** を指定すると、サーバは別のクライアントがテーブルからの読み込みをしなくなるまで、**DELETE** の実行を遅らせます。

- **MyISAM** テーブルに対しては、もし **QUICK** キーワードを利用すると、ストレージ エンジンはインデックスをマージしませんので、いくつかの削除操作のスピードが速くなります。
- **IGNORE** キーワードは、行削除の作業中に MySQL が全てのエラーを無視するように働きかけます。(解析段階で起きたエラーは通常通りの扱いになります。) **IGNORE** を利用した為に無視されたエラーは、警告として返されます。

削除操作のスピードは **DELETE ステートメントの速度** で紹介されている要因によっても影響を受ける可能性があります。

MyISAM テーブルでは、削除された行はリンクされたリスト内で保存され、後続の **INSERT** 操作は古い行の位置を再利用します。利用されていないスペースを再利用しファイル サイズを縮小するには、テーブルを確認する為に **OPTIMIZE TABLE** ステートメントか **myisamchk** ユーティリティを利用してください。**OPTIMIZE TABLE** の方が簡単ですが、**myisamchk** の方が速いです。**OPTIMIZE TABLE 構文**、**myisamchk — MyISAM テーブル メンテナンス ユーティリティ** を参照して下さい。

QUICK 修飾因子は、インデックスが削除操作の為にマージされるかどうかに影響を与えます。**DELETE QUICK** は、削除された行のインデックス値が後で挿入された行に似ているインデックス値と置き換えられるアプリケーションに、最も有効です。この場合、削除された値によって作られた穴は再利用されます。

DELETE QUICK は、削除された値が、新しい挿入が再び行われるインデックス値の範囲にかかるアンダーフィルされたインデックスのブロックを導く場合には、有効ではありません。この場合、**QUICK** の利用は、未使用のままインデックス内に残る無駄なスペースを作り出します。ここに、同じようなシナリオの例があります。

1. インデックスされた **AUTO_INCREMENT** カラムを含むテーブルを作成します。
2. テーブル内に多くの行を挿入します。各挿入は、インデックスのハイ エンドに追加されるインデックス値となります。
3. **DELETE QUICK** を利用してカラム範囲のロー エンドにある行のブロックを削除します。

このシナリオの中では、削除されたインデックス値に関連するインデックス ブロックはアンダーフィルされますが、**QUICK** を利用する事によって別のインデックス ブロックとマージされる事はありません。新しい行は削除された範囲内にインデックス値を持たないので、新しい挿入が行われてもそれらはアンダーフィルされたままです。その上、後で **QUICK** 無しで **DELETE** を利用しても、アンダーフィルされたブロック内またはその隣に、削除されたインデックス ブロックのいくつかは偶然残っていない限り、それらはアンダーフィルされたままです。このような条件下で、利用されていないインデックス スペースを再利用するには、**OPTIMIZE TABLE** を利用してください。

もしテーブルからたくさんの行を削除しようとしているのなら、**OPTIMIZE TABLE** が後に続く **DELETE QUICK** を利用する事で処理が速くできるかもしれません。これは、たくさんのインデックス ブロックのマージ操作を行うというよりは、インデックスを再構築する作業です。

MySQL 特有の、**DELETE** の為の **LIMIT row_count** オプションは、コントロールがクライアントに戻される前に、削除されるべき最大行数をサーバに伝えます。これは、与えられた **DELETE** ステートメントに時間がかかりすぎない事を保障します。影響される行数が **LIMIT** 値よりも少なくなるまで **DELETE** ステートメントを繰り返す事ができます。

もし **DELETE** ステートメントが **ORDER BY** 条項を含むなら、行は条項に指示された順番で削除されます。これは、**LIMIT** を持つ接続詞の中でだけ本当に役に立ちます。例えば、次のステートメントは **WHERE** 条項にマッチする行を見つけ、**timestamp_column** を利用してそれらをソートし、そして最初の (一番古い) 物を削除します。

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

WHERE 条項内の特別な条件に依存する1つ、または複数のテーブルから行を削除する為に、**DELETE** ステートメントの中で複数のテーブルを指定する事ができます。しかし、複合テーブル **DELETE** の中で **ORDER BY** や **LIMIT** を利用する事はできません。**table_references** 条項は接合箇所に含まれるテーブルをリストします。その構文は **JOIN 構文** で説明されています。

最初の複合テーブル構文には、**FROM** 条項の前にリストされた、テーブルのマッチする行だけが削除されます。2番目の複合テーブル構文では、**FROM** 条項(**USING** 条項の前) の前にリストされた、テーブルのマッチする行だけが削除されます。その効果は、複数のテーブルから同時に行を削除でき、検索の為だけに利用される追加テーブルを持つ事ができるという事です。

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```


または:

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

これらのステートメントは、削除する行を検索する時に3つのテーブル全てを利用しますが、テーブル **t1** と **t2** からのマッチする行だけを削除します。

前出の例はカンマ演算子を利用する内部接合を表しますが、複合テーブルの **DELETE** ステートメントは、**LEFT JOIN** のような、**SELECT** ステートメント内で許容される接合タイプを利用する事ができます。

構文は **Access** を持つ互換性のテーブル名の後に ***** を許容します。

外部キー制限があるテーブルに **InnoDB** テーブルを含む複合テーブル **DELETE** ステートメントを利用すると、MySQL のオプティマイザは、それらの親子関係の順番と違う順番でテーブルを処理するかもしれません。この場合、ステートメントは失敗し、ロールバックされます。代わりに、単一テーブルから削除し、他のテーブルが適宜修正されるように **InnoDB** が働きかける **ON DELETE** 性能に頼る必要があります。

注意:もしテーブルにエイリアスを付けるなら、そのテーブルを参照する時はそのエイリアスを利用しなければいけません。

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

データベース間をまたがる削除は、複合テーブル削除にサポートされていますが、この場合は、エイリアスを使わずにテーブルを参照する必要があります。例:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

現在は、サブクエリの中で1つのテーブルから削除し、同じテーブルから選択する事はできません。

DO 構文

```
DO expr [, expr] ...
```

DO は式を実行しますが結果は返しません。あらゆる点で **DO** は **SELECT expr, ...** の省略表現ですが、結果を気にしない場合にはスピードが少し速いという利点があります。

DO は、**RELEASE_LOCK()** のような、副作用を持つ関数と共に利用すると効果的です。

HANDLER 構文

```
HANDLER tbl_name OPEN [ AS alias ]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name CLOSE
```

HANDLER ステートメントは、テーブル ストレージ エンジン インターフェースへの直接アクセスを供給します。これは **MyISAM** と **InnoDB** テーブルに有効です。

HANDLER ... OPEN ステートメントはテーブルをオープンし、それに続く **HANDLER ... READ** ステートメントを通してテーブルをアクセス可能にします。このテーブルは他のスレッドと共有されず、スレッドが **HANDLER ... CLOSE** をコールするか、終了するまでは閉じられません。もしエイリアスを利用してテーブルをオープンすると、別の **HANDLER** ステートメントを利用しているオープンされたテーブルへの更なる参照時には、テーブル名ではなくエイリアスを利用しなければいけません。

最初の **HANDLER ... READ** 構文は、指定されたインデックスが与えられた値を満たし **WHERE** 条件が一致する行をフェッチします。もし複合カラム インデックスがあるなら、インデックス カラム値をカンマで区切られたリストで指示してください。インデックス内の全てのカラムに値を指定するか、インデックス カラムの左端のプリフィックスに値を指定してください。インデックス **my_idx** が **col_a**、**col_b**、そして **col_c** という名前の3つのカラムをこの通りの順番で含むと仮定してください。**HANDLER** ステートメントはインデックス内の3つ全てのカラム、または左端のプリフィックス内のカラムに値を指定する事ができます。例:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
```

```
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

テーブルの **PRIMARY KEY** を参照する為に **HANDLER** インターフェースを利用するには、引用識別子の **'PRIMARY'** を利用してください。

```
HANDLER tbl_name READ 'PRIMARY' ...
```

2つ目の **HANDLER ... READ** 構文は **WHERE** 条件に合うインデックス順でテーブルから行をフェッチします。

2つ目の **HANDLER ... READ** 構文は **WHERE** 条件に合うインデックス順でテーブルから行をフェッチします。これは、フル テーブル スキャンが必要な時には **HANDLER tbl_name READ index_name** よりも速いです。自然な行の順番というのは、**MyISAM** テーブル データ ファイルの中で行が格納されている順番です。このステートメントは **InnoDB** テーブルにも機能しますが、別のデータ ファイルがないので同じようなコンセプトはありません。

HANDLER ... READ の全ての形は単列が空いていれば **LIMIT** 無しでそれをフェッチします。指定した行数を返す為に **LIMIT** 条項を含んでください。これは、**SELECT** ステートメントと同じ構文を持ちます。詳しくは **SELECT 構文** を参照してください。

HANDLER ... CLOSE は **HANDLER ... OPEN** がオープンしたテーブルを閉じます。

HANDLER は若干低レベルなステートメントです。例えば、これは一貫性を持ちません。これは、**HANDLER ... OPEN** はテーブルのスナップショットを撮らない、またテーブルを ロックしない という事です。これは、**HANDLER ... OPEN** ステートメントが発行された後、テーブル データを変更する事ができ (現在のスレッドか別のスレッドを利用して)、そしてそれらの変更点は **HANDLER ... NEXT** か **HANDLER ... PREV** スキャンで部分的にだけ見る事ができるかもしれない、という事を意味します。

通常の **SELECT** ステートメントの代わりに、**HANDLER** インターフェースを利用するのにはいくつかの理由があります。

- **HANDLER** は **SELECT** よりも速いです。
 - 指定されたストレージ エンジン ハンドラ オブジェクトは **HANDLER ... OPEN** に割り当てられます。そのオブジェクトは、そのテーブルの後続 **HANDLER** ステートメントに再利用されます。それぞれに対して再初期化する必要はありません。
 - 解析はあまり行われません。
 - オプティマイザやクエリ チェックなどのオーバーヘッドはありません。
 - 2つのハンドラ要求間でテーブルがロックされる必要はありません。
 - ハンドラ インターフェースはデータの統一をする必要がありませんので (例えばダーティ リードが許されています)、ストレージ エンジンは **SELECT** が通常は許容しない最適化機能を利用する事ができます。
- **ISAM** のようなインターフェースを利用するアプリケーションに対しては、**HANDLER**のおかげでそれらを MySQL にポートするのが簡単になります。
- **HANDLER**のおかげで、**SELECT** では難しい (または不可能な) 方法でデータベースをスキャンする事ができます。データベースに対話式のユーザ インターフェースを提供するアプリケーションを利用する時には、**HANDLER** インターフェースの方がより自然な方法です。

INSERT 構文

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...){,...},...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

または:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

または:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

INSERT は既存テーブルに新しい行を挿入します。**INSERT ... VALUES** と **INSERT ... SET** 型のステートメントは、明示的に指定された値に基づいて行を挿入します。**INSERT ... SELECT** 型は別のテーブルから選択された行を挿入します。**INSERT ... SELECT** については **INSERT ... SELECT 構文** でさらに詳しく説明されています。

古い行に上書きする為に、**INSERT** の代わりに **REPLACE** を利用する事ができます。**REPLACE** は、古い行を複製する固有キー値を含む新しい行の取り扱いの中では **INSERT IGNORE** と同等になります。新しい行は捨てられるのではなく、古い行を置き換えるのに利用されます。詳しくは **REPLACE 構文** を参照してください。

tbl_name は行が挿入されるべきテーブルです。ステートメントが値を提供しなければいけないカラムは次のように指定する事ができます。

- テーブル名の後にカンマで区切られたカラム名のリストを提供する事ができます。この場合、名前が付けられた各カラムの値には **VALUES** リストか **SELECT** ステートメントが与えられなければいけません。
- もし **INSERT ... VALUES** か **INSERT ... SELECT** にカラム名のリストを指定しなければ、**VALUES** リストか **SELECT** ステートメントがテーブル内全てのカラムの値を提供する必要があります。もしテーブル内のカラムの順番がわからなければ、それを見つける為に **DESCRIBE tbl_name** を利用してください。
- **SET** 条項はカラム名を明示的に指示します。

カラム値を与える方法はいくつかあります。

- もしストリクト SQL モードを利用していない場合、値を明示的に与えられていないカラムはそのデフォルト値（明示的、または暗黙の）に設定されます。例えば、もしテーブル内の全てのカラムに名前を付けないカラムのリストを指定すると、名づけられていないカラムはそのデフォルト値に設定されます。デフォルト値の割り当てについては **データタイプデフォルト値** で説明しています。**無効値の制約** もご参照ください。

もし、デフォルト値を持たない全てのカラムの値を明示的に指定しない限りエラーを発生させる **INSERT** ステートメントが必要であれば、ストリクト モードを利用する必要があります。詳しくは **SQL モード** を参照してください。

- 明示的にカラムをそのデフォルト値に設定するには、キーワード **DEFAULT** を利用してください。これはテーブル内の各カラムの値を持たない不完全な **VALUES** リストの書き込みを防ぐ事ができるので、一部を除く全てのカラムに値を割り当てる **INSERT** ステートメントの書き込みを簡単にします。そうでなければ、**VALUES** リスト内のそれぞれの値に対応するカラム名のリストを書かなければいけません。

また、与えられたカラムのデフォルト値を生成する式の中で利用されるより一般的な形として、**DEFAULT(col_name)** を利用する事もできます。

- もしカラム リストと **VALUES** リストの両方が空なら、**INSERT** は各カラム セットを利用してそのデフォルト値に行を作成します。

```
INSERT INTO tbl_name () VALUES();
```

ストリクト モードでは、もしどのカラムもデフォルト値を持っていないければエラーが発生します。そうでなければ、MySQL は明示的に指定されたデフォルト値を持たない全てのカラムに対して暗黙のデフォルト値を利用します。

- カラム値を提供する為に **expr** 式を指定する事ができます。もし式のタイプがカラムのタイプに合わなければ、タイプの変換が行われる可能性があり、そしてデータのタイプによっては、与えられた値の変換が別の挿入値を生み出す事があります。例えば、文字列 '1999.0e-2' を、**INT**、**FLOAT**、**DECIMAL(10,6)**、または **YEAR** カラムに挿入すると、それぞれ 1999、19.9921、19.992100、そして 1999 が挿入される事になります。**INT** と **YEAR** カラムに格納された値が 1999 である理由は、文字列から整数への変換時には、文字列の最初の部分の、有効な整数や年として判断される部分だけを見るからです。浮動小数点と固定小数点カラムに対しては、文字列から浮動小数点への変換時には、文字列全体を有効な浮動小数点として判断します。

expr 式は、値のリストに早いうちに設定された全てのカラムを参照する事ができます。例えば、**col2** の値は、既に割り当てられた **col1** を参照する事ができるので、上記のような事が可能なのです。

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

しかし、col1 の値は、col1 の後で割り当てられた col2 を参照する為、次のような物は正当ではありません。

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

1つの例外が **AUTO_INCREMENT** 値を持つカラムに関係しています。別の値の割り当て後に **AUTO_INCREMENT** 値が発生するので、その割り当ての中での **AUTO_INCREMENT** カラムへの参照は 0 を返すのです。

VALUES 構文を利用する **INSERT** ステートメントは複数行を挿入する事ができます。これをする為には、それぞれが括弧で囲まれカンマで区切られている、カラム値の複数リストを含んでください。例:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

各行の値のリストは括弧で囲まれている必要があります。次のステートメントは、カラム名の数とリストの中の値の数が合わない為、不正となります。

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

INSERT の行に影響された値は **mysql_affected_rows()** C API 関数を利用して取得できます。詳しくは **mysql_affected_rows()** を参照してください。

もし **INSERT ... VALUES** ステートメントを複数値リストか **INSERT ... SELECT** と共に利用すると、そのステートメントはこのフォーマットで文字列情報を戻します。

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records はステートメントによって生成された行数を指示します。(Duplicates がゼロ以外の数値になりえる為、実際に挿入された行数でなければいけないという訳ではありません。)既存の固有インデックス値を複製するので、**Duplicates** は挿入できなかった行数を指示します。**Warnings** は、何らかの形で問題があった、カラム値挿入の試みの回数を指示します。警告は次のような条件時に発生します。

- **NOT NULL** を宣言していたカラムに **NULL** を挿入する。複合行 **INSERT** ステートメントや **INSERT INTO ... SELECT** ステートメントに対しては、カラムはカラム データ タイプの暗黙のデフォルト値に設定されます。数値タイプ、文字列タイプの空の文字列(""),そして日付、時間タイプの「ゼロ」値に対してのこの値は 0 です。サーバは単列を戻すかどうかを確認する為に **SELECT** からの結果セットを検査しない為、**INSERT INTO ... SELECT** ステートメントは複合行挿入と同じ方法で扱われます。(単列 **INSERT** に対しては、**NULL** が **NOT NULL** カラムに挿入された時、警告は発生しません。代わりに、ステートメントはエラーになり失敗します。)
- 数値カラムを、範囲外の値に設定する。値はその範囲の終点に一番近いところでクリップされます。
- 数値カラムに '10.34 a' のような値を割り当てる。後続の非数値テキストは取り除かれ、残りの数値部分が挿入されます。もし文字列値がその最初の部分に数値を持たない場合は、そのカラムは 0 に設定されます。
- 文字列を、カラムの最大長さを上回る文字列カラムに (**CHAR**、**VARCHAR**、**TEXT**、または **BLOB**)挿入する。値はカラムの最大長さまで切り捨てられます。
- 日付や時間カラムに不正である値を挿入する。カラムはそのタイプにとって適正であるゼロの値に設定されます。

もしC API を利用していれば、**mysql_info()** 関数を呼び出す事で、情報文字列を得ることができます。詳しくは **mysql_info()** を参照してください。

もし **AUTO_INCREMENT** カラムを持つテーブルに **INSERT** が行を挿入すれば、SQL **LAST_INSERT_ID()** 関数を利用する事でそのカラムに利用される値を見つける事ができます。C API の内部から、**mysql_insert_id()** 関数を利用してください。しかし、2つの関数がいつでも全く同じ働きをする訳ではない事に注意してください。**AUTO_INCREMENT** カラムに関しての、**INSERT** ステートメントの動作の更なる情報は [情報関数](#) と **mysql_insert_id()** で紹介されています。

INSERT ステートメントは次の修飾因子をサポートします。

- もし **DELAYED** キーワードを利用すると、サーバはバッファに挿入される行を置くので、**INSERT DELAYED** ステートメントを発行するクライアントは即座に再開されます。もしテーブルが使用中であれば、サーバが行を保持します。テーブルがフリーであればサーバは行の挿入を始め、新しいリクエストがないかを調べる為に定

期的にテーブルを確認します。もしあれば、遅れた行の列はテーブルが再度フリーになるまでサスペンドされます。[INSERT DELAYED 構文](#) を参照してください。

サーバは、[INSERT ... SELECT](#) か

[INSERT ... ON DUPLICATE KEY UPDATE](#) に対して

[DELAYED](#) を無視します。

- もし [LOW_PRIORITY](#) キーワードを利用すると、別のクライアントがテーブルからの読み込みをしなくなるまで、[INSERT](#) の実行が遅れます。これには、既存クライアントが読み込みをしている最中、そして [INSERT LOW_PRIORITY](#) ステートメントが待っている最中に読み込みを始めてしまう別のクライアントが含まれます。しかし、[INSERT LOW_PRIORITY](#) ステートメントを発行するクライアントがリードヘビー環境の中で長時間（または永遠に）待つ事は可能です。（これは、クライアントにそのまま作業を続けさせる [INSERT DELAYED](#) とは逆の機能です。並列挿入ができなくなるので、通常 [LOW_PRIORITY](#) は [MyISAM](#) テーブルと一緒に利用されません。詳しくは [同時挿入](#) を参照してください。
- もし [HIGH_PRIORITY](#) を指定すると、サーバが `--low-priority-updates` オプションでスタートされている場合その効果が無効になります。また、並行挿入も利用されなくなります。
- もし [IGNORE](#) キーワードを利用したら、[INSERT](#) ステートメントの実行中に起きたエラーは警告として扱われます。例えば、[IGNORE](#) が無いと、テーブル内に既存の [UNIQUE](#) インデックスや [PRIMARY KEY](#) 値を複製する行に複製キー エラーが起き、そのステートメントは異常終了されます。[IGNORE](#) を利用すると、行の挿入はされませんが、エラーも発行されません。もし [IGNORE](#) が指定されなければ、データ変換はステートメントに関するエラーを引き起こします。[IGNORE](#) を利用すると、不正な値は一番近い値に調節されて挿入され、警告が発生されますがステートメントは異常終了しません。いくつかのテーブルが実際に挿入されたのか、[mysql_info\(\)](#) C API 関数を利用して調べる事ができます。
- もし [ON DUPLICATE KEY UPDATE](#) を指定し、[UNIQUE](#) インデックスか [PRIMARY KEY](#) 内で複製値を引き起こす行が挿入されると、古い行の [UPDATE](#) が実行されます。詳しくは [INSERT ... ON DUPLICATE KEY UPDATE 構文](#) を参照してください。

INSERT ... SELECT 構文

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

[INSERT ... SELECT](#) を利用すると、1つ、または複数のテーブルから多くの行をすばやく挿入する事ができます。例:

```
INSERT INTO tbl_temp2 (fld_id)
SELECT tbl_temp1.fld_order_id
FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

次の条件は [INSERT ... SELECT](#) ステートメントを保持します。

- 複製キー違反を引き起こす行を無視する為に [IGNORE](#) を指定してください。
- [DELAYED](#) は [INSERT ... SELECT](#) と共に無視されます。
- [INSERT](#) ステートメントの対象テーブルはクエリの [SELECT](#) 部の [FROM](#) 条項内に現れます。（これは MySQL の古いバージョンでは不可能でした。）この場合、MySQL は行を保持する為に [SELECT](#) からテンポラリ テーブルを作成し、そして対象テーブルにそれらの行を挿入します。
- [AUTO_INCREMENT](#) カラムは通常通り機能します。
- バイナリ ログが元テーブルを再作成する為に利用できる事を保障する為に、MySQL は [INSERT ... SELECT](#) ステートメントへの並列挿入を許可しません。
- 現在は、サブクエリの中で1つのテーブルに挿入し、同じテーブルから選択する事はできません。
- [SELECT](#) と [INSERT](#) が同じテーブルを参照した時に、曖昧なカラム参照の問題を防ぐ為に、[SELECT](#) 部分で利用される各テーブルに固有のエイリアスを与え、その部分の中でのカラム名を適切なエイリアスに限定してください。

ON DUPLICATE KEY UPDATE の値部分の中で、SELECT 部分の中で GROUP BY を利用しない限り、別のテーブル内でカラムの参照をする事ができます。値の部分で非固有カラム名を指定しなければいけない、という副作用が1つあります。

INSERT DELAYED 構文

INSERT DELAYED ...

INSERT ステートメントの DELAYED オプションは、もし INSERT が完了するのを待つ事ができない、または待つ必要がないクライアントを持っている場合に大変有効となる、スタンダード SQL の MySQL 拡張子です。これは、MySQL をログに利用し、完了までに長時間かかる SELECT と UPDATE ステートメントを定期的に起動させる時によく起こる状態です。

クライアントが INSERT DELAYED を利用する時、サーバからすぐに OK が出て、テーブルが別のスレッドによって使用中でなければ行が挿入される為にキューを作ります。

INSERT DELAYED を利用する事のそれ以外の大きな利益は、たくさんのクライアントからの挿入は一緒にまとめられ、ひとつのブロックに書き込まれると言う事です。これは、別々の挿入を何度も行うよりも早く機能します。

INSERT DELAYED は、もしテーブルが他の形で利用されていないのであれば、通常の INSERT よりも遅いという事に注意してください。また、サーバには、遅れている行を持つ各テーブルに別々のスレッドを扱う為の、追加オーバーヘッドもあります。これは、本当に INSERT DELAYED が必要だと言う事が確実な時だけ利用するべきであるという事を意味します。

キューを作った行は、テーブルに挿入されるまでメモリ内だけで保持されます。これは、もし mysqld を強制的に終了させたり (例えば、kill -9 を利用して)、mysqld が突然停止してしまったりすると、ディスクに書き込まれる前のキューを作った行は全て失われてしまう という事を意味します。

DELAYED の利用に関しては、いくつかの制限があります。

- INSERT DELAYED は MyISAM、MEMORY、そして ARCHIVE テーブルとのみ機能します。MyISAM ストレージエンジン、MEMORY (HEAP) ストレージエンジン、そして ARCHIVE ストレージエンジンを参照してください。

もしデータファイル中にフリー ブロックがなければ、MyISAM テーブルには並列 SELECT と INSERT ステートメントがサポートされます。これらの条件下では、INSERT DELAYED を MyISAM と一緒に利用しなければいけない事はほとんどありません。

- INSERT DELAYED は、値リストを指定する INSERT ステートメントにだけ利用されなければいけません。サーバは、INSERT ... SELECT か INSERT ... ON DUPLICATE KEY UPDATE に対して DELAYED を無視します。
- INSERT DELAYED ステートメントがすぐに返されるので、そのステートメントが生成するであろう AUTO_INCREMENT 値を得る為に、行が挿入される前に、LAST_INSERT_ID() を利用する事はできません。
- DELAYED 行は、実際に挿入されるまでは SELECT ステートメントには見えません。
- DELAYED は、スレーブにマスタとは異なるデータを持たせる事があるので、スレーブ複製サーバ上では無視されます。
- テーブルが書き込みロックされ、ALTER TABLE がテーブル構造を変更するのに利用されると、保留中の INSERT DELAYED ステートメントは失われてしまいます。
- INSERT DELAYED は画面をサポートしません。

次に、INSERT や REPLACE に DELAYED を利用した時に何が起こるかを詳しく説明しています。この説明の中では、「スレッド」は INSERT DELAYED ステートメントを受け取ったスレッドで、「ハンドラ」は特定のテーブルの為に全ての INSERT DELAYED ステートメントを扱うスレッドを表しています。

- スレッドが DELAYED ステートメントをテーブルに実行した時、もし同じようなハンドラが既に存在していなければ、全ての DELAYED ステートメントをテーブルに生成する為にハンドラ スレッドが作成されます。
- スレッドは、ハンドラが以前に DELAYED ロックを習得したかどうかを確認します。もし習得していなければ、ハンドラ スレッドに対して習得するよう命令します。もし他のスレッドが READ か WRITE ロックをテーブル上に持っていたり、DELAYED ロックを得る事ができます。しかし、ハンドラはテーブル構造が最新であるかどうかを確認する為に、全ての ALTER TABLE ロックや FLUSH TABLES ステートメントが終了するのを待ちます。

- スレッドは **INSERT** ステートメントを実行しますが、行をテーブルに書き込む代わりに、ハンドラ スレッドに管理されているキューに最終行のコピーを置きます。構文エラーは全てスレッドに見つけれ、クライアント プログラムにリポートされます。
- クライアントは、挿入操作が完了する前に **INSERT** が返る為、複製行の数や、結果として生じる行の **AUTO_INCREMENT** 値をサーバから得る事ができません。(もし C API を利用すると、同じ理由で **mysql_info()** 関数からは意味のある答えが返りません。)
- 行がテーブルに挿入された時、バイナリ ログはハンドラ スレッドによって更新されます。複合行挿入の場合、最初の行が挿入された時にバイナリ ログが更新されます。
- **delayed_insert_limit** 行が書かれる度に、ハンドラはまだ保留中の **SELECT** ステートメントがないかどうかを確認します。もしあれば、続ける前にそれらを実行させます。
- ハンドラのキューに行が無くなると、テーブルのロックは外されます。もし新しい **INSERT DELAYED** ステートメントが **delayed_insert_timeout** 秒以内に受信されたら、ハンドラは終了します。
- もし **delayed_queue_size** 以上の行が、特定のハンドラ キューの中で保留中だったら、**INSERT DELAYED** をリクエストしているスレッドは、キューの中にスペースができるまで待ちます。これは、遅れたメモリのキューの為に **mysqld** が全てのメモリを使わない事を保障する為に行われます。
- ハンドラスレッドは、**Command** カラム内の **delayed_insert** と共に、MySQL プロセス リスト内に現れます。これは、もし **FLUSH TABLES** ステートメントを実行したり、**KILL thread_id** を利用したりすると中止されます。しかし、終了する前にまずテーブル内でキューを作っている全ての行を格納します。この最中は、別のスレッドから新しい **INSERT** ステートメントを受け入れません。もしこの後に **INSERT DELAYED** ステートメントを実行すると、新しいハンドラ スレッドが作成されます。

もし起動中の **INSERT DELAYED** ハンドラがあったら、**INSERT DELAYED** ステートメントは通常の **INSERT** ステートメントより高い優先権を持つという事を意味します。その他の更新ステートメントは、**INSERT DELAYED** キューが空になるか、誰かがハンドラスレッドを終了させるか (**KILL thread_id** を利用して)、誰かが **FLUSH TABLES** を実行するまで待たなければいけません。

- 次の状態変数は **INSERT DELAYED** ステートメントの情報を提供します。

状態変数	意味
Delayed_insert_threads	ハンドラ スレッド数
Delayed_writes	INSERT DELAYED で書かれた行数
Not_flushed_delayed_rows	書き込みを待つ行数

SHOW STATUS ステートメントか、**mysqladmin extended-status** コマンドを実行する事でこれらの変数を見る事ができます。

INSERT ... ON DUPLICATE KEY UPDATE 構文

もし **ON DUPLICATE KEY UPDATE** を指定し、**UNIQUE** インデックスか **PRIMARY KEY** 内で複製値を引き起こす行が挿入されると、古い行の **UPDATE** が実行されます。例えば、もしカラム **a** が **UNIQUE** として宣言され、それが値 **1** を含んでいたら、次の2つのステートメントは同一効果を持ちます。

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=c+1;

UPDATE table SET c=c+1 WHERE a=1;
```

もしその行が新しいレコードとして挿入されると、行に影響される値は1となり、もし既存レコードが更新されると2になります。

もしカラム **b** も固有であれば、**INSERT** は代わりにこの **UPDATE** ステートメントと同等になります。

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

もし **a=1 OR b=2** がいくつかの行とマッチすれば、1つの 行だけが更新されます。通常、複数の固有インデックスを持つテーブル上で **ON DUPLICATE KEY** 条項を利用するのは避けるべきです。

INSERT ... UPDATE ステートメントの **INSERT** 部分からカラム値を参照する為に、**UPDATE** 条項の中で **VALUES(col_name)** 関数を利用する事ができます。言い換えると、**UPDATE** 条項の中の **VALUES(col_name)** は、挿入される **col_name** の値を参照し、複製キーの矛盾は起きないという事です。この関数は、時に複合行挿入をする時に有効です。**VALUES()** 関数は、**INSERT ... UPDATE** ステートメントの中でだけ意味を持ち、そうでなければ **NULL** を返します。例:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

そのステートメントは次の2つと同一です。

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
ON DUPLICATE KEY UPDATE c=9;
```

もしテーブルが **AUTO_INCREMENT** カラムを含み **INSERT ... UPDATE** が行を挿入すると、**LAST_INSERT_ID()** 関数は **AUTO_INCREMENT** 値を返します。もしステートメントが代わりに行を更新すると、**LAST_INSERT_ID()** は無意味になります。しかし、**LAST_INSERT_ID(expr)** を利用する事でこれに対処する事ができます。**id** が **AUTO_INCREMENT** カラムだと仮定してください。**LAST_INSERT_ID()** が更新に意味を持つようにするには、次のように行を挿入してください。

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE id=LAST_INSERT_ID(id), c=3;
```

ON DUPLICATE KEY UPDATE を利用する時は **DELAYED** オプションは無視されます。

LOAD DATA INFILE 構文

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[FIELDS
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
[IGNORE number LINES]
[[col_name_or_user_var,...]]
[SET col_name = expr,...]
```

LOAD DATA INFILE ステートメントは高スピードでテキスト ファイルからテーブルに行を読み込みます。ファイル名は直定数文字列として与えられなければいけません。

LOAD DATA INFILE は **SELECT ... INTO OUTFILE** の補数です。(詳しくは **SELECT 構文** をご確認ください。) テーブルからファイルにデータを書き込むには、**SELECT ... INTO OUTFILE** を利用してください。テーブルにファイルをリードバックするには、**LOAD DATA INFILE** を利用してください。両方のステートメントに対して **FIELDS** と **LINES** 条項の構文は同じです。条項は両方とも任意ですが、もし両方が指定された場合 **FIELDS** は **LINES** に先行しなければいけません。

INSERT 対 **LOAD DATA INFILE** の性能と、**LOAD DATA INFILE** のスピードアップの更なる情報については、**INSERTステートメントの速度** を参照してください。

character_set_database システム変数によって指示された文字セットはかつてファイルの中の情報を解明していました。**SET NAMES** と **character_set_client** の設定はインプットの解明に影響を与えません。

現在は **ucs2** 文字セットを利用するデータ ファイルをロードするのは不可能だという事に気がつけてください。

MySQL 5.1.6 以降のバージョンでは、**character_set_filesystem** システム変数は、ファイル名の解明をコントロールします。

mysqlimport ユーティリティを利用する事でデータ ファイルをロードする事もできます。これは、サーバに **LOAD DATA INFILE** ステートメントを送信する事で機能します。**--local** オプションは **mysqlimport** がクライアント ホストからデータファイルを読み込むよう働きかけます。もしクライアントとサーバが圧縮されたプロトコルをサポートする

なら、スピードが遅いネットワークにより良い性能を得る為に `--compress` オプションを指定する事ができます。詳しくは [mysqlimport — データインポートプログラム](#) を参照してください。

もし `LOW_PRIORITY` を利用すると、別のクライアントがテーブルからの読み込みをしなくなるまで、`LOAD DATA` の実行が遅れます。

並列挿入の条件を満たす(フリーブロックが途中で無いという事) `MyISAM` テーブルと共に `CONCURRENT` を指定すると、`LOAD DATA` の実行中に他のスレッドがテーブルからデータを検索する事ができます。もし他のスレッドがテーブルを同時に利用していなくても、このオプションを利用する事は `LOAD DATA` の性能に少しだけ影響を与えます。

もし `LOCAL` キーワードが指定されたら、それは接続の最後にクライアントに関して説明されます。

- もし `LOCAL` が指定されると、ファイルはクライアント ホスト上のクライアント プログラムによって読み込まれ、サーバに送られます。ファイルは、その明確な場所を指定する為の完全なパス名として与えられます。もしそのパス名が相対的な物として与えられると、その名前はクライアントプログラムが開始されたディレクトリと比較して説明されます。
- もし `LOCAL` が指定されなければ、ファイルはサーバ ホスト上に置かれ、サーバによって直接読み込まれる必要があります。サーバはファイルを置く為に次のルールを利用します。
 - もしファイル名が完全なパス名であれば、サーバはそれをそのまま利用します。
 - もしファイル名が1つ、または複数の主要コンポーネントを持つ相対的なパス名であれば、サーバはそのデータディレクトリに関連するファイルを検索します。
 - もし主要コンポーネントを持たないファイル名が与えられると、サーバはデフォルト データベースのデータベースディレクトリ内のファイルを探します。

これらのルールは、非 `LOCAL` の場合、`./myfile.txt` としてのファイル名はサーバのデータ ディレクトリから読み込まれ、その一方、`myfile.txt` としてのファイル名はデフォルト データベースのデータベース ディレクトリから読み込まれるという事を意味しますので、注意してください。例えば、もし `db1` がデフォルト データベースなら、ステートメントが `db2` データベース内のテーブルにファイルを明示的にロードしたとしても、次の `LOAD DATA` ステートメントが `db1` のデータベース ディレクトリからファイル `data.txt` を読み込みます。

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

ウィンドウズのパス名は、バックスラッシュではなくフォーワードスラッシュで指定されます。もしバックスラッシュを利用すると、それらを2倍にする必要があります。

安全上の理由で、サーバ上にあるテキスト ファイルを読み込む時、ファイルはデータベース ディレクトリ上にあるか、全てによって読み込み可能である必要があります。また、サーバ ファイル上で `LOAD DATA INFILE` を利用する時は、`FILE` 権限が必要です。詳しくは [MySQL 提供の権限](#) を参照してください。

`LOCAL` を利用すると、ファイルのコンテンツはクライアントからサーバに、その接続全体に送られなければいけないので、サーバがファイル ディレクトリにアクセスするのを許可するのよりも少し速度が遅くなります。反対に、ローカル ファイルをロードするのに `FILE` 特権は必要ありません。

`LOCAL` は、サーバとクライアントの両方が、これを許容できる場合のみ機能します。例えば、もし `mysqld` が `--local-infile=0` と共に開始された場合、`LOCAL` は機能しません。詳しくは [LOAD DATA LOCAL のセキュリティ関連事項](#) を参照してください。

Unix上では、もしパイプから読み込む為に `LOAD DATA` が必要であれば、次のテクニックを利用する事ができます。(ここでは、テーブルの中に / ディレクトリのリストをロードします。)

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
find / -ls > /mysql/db/x/x &
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

ロードされるデータを生成するコマンドと `mysql` を、別々のターミナルのどちらかで起動させる、または背景でデータ生成プロセスを起動させなければいけない(上記の例で表されているように)、という事に注意してください。もしこれを実行しないと、パイプはデータが `mysql` プロセスによって読み込まれるまでブロックします。

`REPLACE` と `IGNORE` キーワードは、固有のキー値上に既存行を複製するインプット行の扱いをコントロールします。

- もし **REPLACE** を指定すると、インプット行は既存行を置き換えます。言い換えると、主キーや固有インデックスに対して同じ値を持つ、既存行であるという事です。詳しくは **REPLACE 構文** を参照してください。
- もし **IGNORE** を指定すると、固有キー値上の、既存行を複製するインプット行はスキップされます。もしどちらのオプションも指定しなければ、その動作は **LOCAL** キーワードが指定されたかどうかによって決まります。**LOCAL** を利用すると、複製キー値が見つかった時点でエラーが発生し、テキスト ファイルの残りは無視されます。**LOCAL** を使用しなければ、デフォルトの動作は **IGNORE** が指定された時と同じです。これは、サーバは操作の最中にファイルの送信を中止する事ができないからです。

もしロード操作中に外部キー制約を無視したければ、**LOAD DATA** を実行する前に **SET FOREIGN_KEY_CHECKS=0** ステートメントを発行する事ができます。

もし空の **MyISAM** テーブル上で **LOAD DATA INFILE** を利用すると、別のバッチに全ての非ユニーク インデックスが作成されます。(**REPAIR TABLE** に関して)通常この操作は、インデックスが多くある時に **LOAD DATA INFILE** のスピードを速くします。いくつかの極端な場合では、テーブルにファイルをロードする前に **ALTER TABLE ... DISABLE KEYS** を利用してインデックスをオフにしたり、ファイルをロードした後にインデックスを再作成する為に **ALTER TABLE ... ENABLE KEYS** を利用する事で、インデックスをさらに速く作成できます。詳しくは **INSERTステートメントの速度** を参照してください。

LOAD DATA INFILE と **SELECT ... INTO OUTFILE** ステートメントの両方に対して、**FIELDS** と **LINES** 条項の構文は同じです。条項は両方とも任意ですが、もし両方が指定された場合 **FIELDS** は **LINES** に先行しなければいけません。

もし **FIELDS** 条項を指定すると、少なくともどれか1つを指定する必要がありますが、その各サブ条項 (**TERMINATED BY**、**[OPTIONALLY] ENCLOSED BY**、そして **ESCAPED BY**) もまた任意になります。

もし **FIELDS** 条項を指定しなければ、デフォルトは、このように書き込んだ場合と同じようになります。

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '"' ESCAPED BY '\\'
```

もし **FIELDS** 条項を指定しなければ、デフォルトは、このように書き込んだ場合と同じようになります。

```
LINES TERMINATED BY '\n' STARTING BY ''
```

言い換えると、インプットを読み込む時、デフォルトは **LOAD DATA INFILE** が次のように機能するよう働きかけるという事です。

- 改行部分のラインの境界を探してください。
- ライン プリフィックスを飛び越えないでください。
- タブのところでラインをフィールドに分割してください。
- フィールドが引用文字によって囲まれていると思わないでください。
- '\ ' が先に来るタブ、改行、または '\ ' を、フィールド値の一部である直定文字として解釈してください。

反対に、デフォルトはアウトプットを書き込む時に **SELECT ... INTO OUTFILE** が次のように機能するよう働きかけます。

- フィールドの間にタブを書いてください。
- 引用文字でフィールドを囲まないでください。
- フィールド値内で起こるタブのインスタンス、改行、または '\ ' を避ける為に '\ ' を利用してください。
- 行の最後に改行を書き込んでください。

バックスラッシュは、MySQL の中では文字列内の拡張文字ですので、**FIELDS ESCAPED BY '\'** を書き込むには、単一バックスラッシュだと認識させる為に2つのバックスラッシュを指定しなければいけません。

注意:もしウィンドウズ システム上でテキスト ファイルを生成したら、ウィンドウズのプログラムは通常ラインのターミネータとして2つの文字を利用するので、ファイルを正確に読み込む為には **LINES TERMINATED BY '\r\n'** を利用しなければいけないでしょう。**WordPad** のようないくつかのプログラムは、ファイルを書き込む時 '\r' をラインターミネータとして利用するでしょう。そのようなファイルを読み込む時は、**LINES TERMINATED BY '\r'** を利用してください。

もし読み込みたいライン全てに、共通の無視したいプリフィックスがあれば、そのプリフィックスと、その前にある全ての物を飛び越える為に `LINES STARTING BY 'prefix_string'` を利用する事ができます。もしラインがプリフィックスを持たなければ、ライン全体がスキップされます。このステートメントを発行すると仮定すると：

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx';
```

もしデータ ファイルがこのようになっていたら：

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

行は結果として `("abc",1)` と `("def",2)` のようになります。ファイル内の3行目は、プリフィックスを含まないのでスキップされます。

`IGNORE number LINES` オプションはファイルの先頭のラインを無視する為に利用する事ができます。例えば、カラム名を含む冒頭のヘッダ ラインを飛び越える為に `IGNORE 1 LINES` を利用する事ができます。

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

データベースからのデータをファイルに書き込み、その後そのファイルをデータベースに読み返す為に `LOAD DATA INFILE` と並行して `SELECT ... INTO OUTFILE` を利用する時、両方のステートメントの field- と line-handling オプションはマッチする必要があります。そうでなければ、`LOAD DATA INFILE` はファイルの内容を正確に解釈しません。カンマで区切られたフィールドを持つファイルを書き込む為に、`SELECT ... INTO OUTFILE` を利用すると仮定してください。

```
SELECT * INTO OUTFILE 'data.txt'
FIELDS TERMINATED BY ','
FROM table2;
```

カンマで区切られたファイルを読み返す為の正しいステートメントは次のようになります。

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
FIELDS TERMINATED BY ',';
```

もし代わりに、次に表されているステートメントを利用してファイルを読み込もうとすると、`LOAD DATA INFILE` にフィールドの間にあるタブを探すよう指示を出すので、そのステートメントは機能しません。

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
FIELDS TERMINATED BY '\t';
```

各インプット ラインが単一フィールドとして解釈されるという結果がよく出されます。

`LOAD DATA INFILE` は、外部ソースから得たファイルを読見込むのに利用する事ができます。例えば、多くのプログラムは、ラインがカンマで区切られたフィールドを持ち、2つの引用句で囲まれている というような、カンマで区切られた値 (CSV) のフォーマットでデータをエクスポートする事ができます。もしそのようなファイルの中のラインが改行で終了していたら、ここに表されているステートメントはファイルをロードする為に利用するであろう field- と line-handling オプションを説明します。

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

field- または line-handling オプションは空の文字列を指定する事ができます。 (")もし空でなければ、`FIELDS [OPTIONALLY] ENCLOSED BY` と `FIELDS ESCAPED BY` 値は単一文字という事になります。`FIELDS TERMINATED BY`、`LINES STARTING BY`、そして `LINES TERMINATED BY` 値は一文字以上になり得ます。例えば、キャリッジ リターンと改行のペアで終わっているラインを書いたり、そのようなラインを含むファイルを読み込む為には、`LINES TERMINATED BY '\r\n'` 条項を指定してください。

`%%` で成り立つラインによって区切られているジョークを含むファイルを読み込むには、これを実行する事ができます。

```
CREATE TABLE jokes
(a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
FIELDS TERMINATED BY "
LINES TERMINATED BY '\n%%\n' (joke);
```


FIELDS [OPTIONALLY] ENCLOSED BY はフィールドの引用句をコントロールします。アウトプットには(**SELECT ... INTO OUTFILE**)、もし **OPTIONALLY** を省略すると全てのフィールドが **ENCLOSED BY** 文字によって囲まれます。そのようなアウトプット(フィールド デリミタとしてカンマを利用している)の例がここに表されています。

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

もし **OPTIONALLY** を指定すると **ENCLOSED BY** 文字は (**CHAR**、**BINARY**、**TEXT**、または **ENUM** のような) 文字列データタイプを持つカラムからの値を囲む為だけに利用されます。

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

フィールド値内の **ENCLOSED BY** 文字の発生は、**ESCAPED BY** 文字と共にそれらをプリフィックスする事で拡張する事ができるという事を覚えておいて下さい。また、もし空の **ESCAPED BY** 値を指定すると、**LOAD DATA INFILE** で正しく読み込む事ができないアウトプットを気づかずに生成してしまう可能性がある事も覚えておいて下さい。例えば、もし拡張文字が空なら、表示されただけの先行するアウトプットは次のように現れます。4つ目のラインの2つ目のフィールドの引用句の後に、フィールドを(誤って)終了させるカンマが含まれている事を確認してください。

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

インプットに関しては、もし **ENCLOSED BY** 文字があれば、フィールド値の最後から剥ぎ取られます。(**OPTIONALLY** が指定されているかどうかは関係なくこれは事実です。 **OPTIONALLY** はインプットの解釈に対して影響しません。) **ESCAPED BY** 文字に先行された **ENCLOSED BY** 文字の発生は、現在のフィールド値の一部として解釈されます。

もしフィールドが **ENCLOSED BY** 文字で始まると、その文字のインスタンスはフィールド、またはライン **TERMINATED BY** シーケンスが後に続いている場合のみ、そのフィールド値を終了させていると判断されます。曖昧さを防ぐ為に、フィールド値内の **ENCLOSED BY** 文字の発生を2倍にすると、その文字の単一インスタントとして解釈されます。例えば、もし **ENCLOSED BY** **"** が指定されると、引用句はここに表されているように扱われます。

```
"The ""BIG"" boss" -> The "BIG" boss
The "BIG" boss    -> The "BIG" boss
The ""BIG"" boss  -> The ""BIG"" boss
```

FIELDS ESCAPED BY は、特別な文字をどのように書き込み、読み込むのかをコントロールします。もし **FIELDS ESCAPED BY** 文字が空でなければ、それはアウトプット上に次の文字をプリフィックスする為に利用されます。

- **FIELDS ESCAPED BY** 文字
- **FIELDS [OPTIONALLY] ENCLOSED BY** 文字
- **FIELDS TERMINATED BY** と **LINES TERMINATED BY** 値の最初の文字
- ASCII 0 (実際に拡張文字の後ろに書かれているのは、ゼロの値のバイトではなく、ASCII '0' です。)

もし **FIELDS ESCAPED BY** 文字が空なら、文字が拡張される事はなく、**NULL** は **\N** ではなく **NULL** としてアウトプットされます。特に、もしデータ中にフィールド値が先ほどのリストの中の文字を含んでいる場合は、空の拡張文字を指定するのは良い考えではないかも知れません。

インプットに関しては、もし **FIELDS ESCAPED BY** 文字が空でなければ、その文字の発生は削除され、後続文字はフィールド値の一部として文字通りに取り込まれます。例外は、拡張された **'0'** または **'N'** です。(例えば、もし拡張文字が **'\'** なら **\0** または **\N** です。)これらのシーケンスは、ASCII NUL (ゼロの値のバイト) と **NULL** として解釈されます。**NULL** の扱いのルールについてはこのセクションの後半で説明します。

'\'-escape 構文の更なる情報については、[リテラル値](#) を参照してください。

特定の場合には、field- と line-handling オプションは互いに影響しあいます。

- もし **LINES TERMINATED BY** が空の文字列で、**FIELDS TERMINATED BY** がそうでない場合、ラインもまた **FIELDS TERMINATED BY** で終わります。

- もし **FIELDS TERMINATED BY** と **FIELDS ENCLOSED BY** 値の両方が空なら、(")、固定行(区切られていない)フォーマットが利用されます。固定行フォーマットでは、フィールド間で区切り文字は利用されません。(ラインターミネータを持つ事はできます。)代わりに、フィールド内に全ての値を保持するのに十分なフィールド幅を利用して、カラム値が読み込み、書き込みされます。**TINYINT**、**SMALLINT**、**MEDIUMINT**、**INT**、そして **BIGINT** のフィールド幅はそれぞれ、宣言されているディスプレイ幅に関わらず、4、6、8、11、そして20です。

LINES TERMINATED BY はラインを分割する為に利用されます。もしラインが全てのフィールドを含んでいなければ、残りのカラムはデフォルト値に設定されます。もしラインターミネータがなければ、これを " に設定しなければいけません。この場合、テキストファイルは各行に対して全てのフィールドを含まなければいけません。

固定行フォーマットはまた、後ほど説明があるように、**NULL** 値の扱いに影響を与えます。固定サイズフォーマットは、もしマルチバイト文字セットを利用する場合は機能しないという事を覚えておいて下さい。

NULL 値の扱いは、利用中の **FIELDS** と **LINES** オプションによって変わります。

- デフォルトの **FIELDS** と **LINES** 値には、**NULL** はアウトプットには **\N** のフィールド値として書かれ、**\N** のフィールド値は、インプットには **NULL** として読まれます。(**ESCAPED BY** 文字は **'\'** と仮定する。)
- もし **FIELDS ENCLOSED BY** が空でなければ、直定数文字 **NULL** をその値として含むフィールドは **NULL** 値として読まれます。これは、**'NULL'** 文字列として読み込まれる **FIELDS ENCLOSED BY** 文字の間に囲まれた単語 **NULL** とは異なります。
- もし **FIELDS ESCAPED BY** が空なら、**NULL** は単語 **NULL** として書き込まれます。
- 固定行フォーマットでは、(**FIELDS TERMINATED BY** と **FIELDS ENCLOSED BY** の両方が空の時に利用される)、**NULL** は空の文字列として書き込まれます。この場合、**NULL** 値と空の文字列は、ファイルに書き込まれた時には両方とも空の文字列として書き込まれるので、テーブル内では見分けがつかないという事を覚えておいて下さい。もしファイルをリードバックする時にその2つを見分ける必要があれば、固定行フォーマットは利用するべきではありません。

NULL を **NOT NULL** カラムにロードしようとする、カラムのデータタイプと警告の為に暗黙のデフォルト値の割り当て、またはストリクト SQL モードでのエラーが発生します。暗黙のデフォルト値に関しては [データタイプデフォルト値](#) で説明されています。

LOAD DATA INFILE によってサポートされない場合もあります。

- 固定サイズ行(**FIELDS TERMINATED BY** と **FIELDS ENCLOSED BY** の両方が空)と **BLOB** か **TEXT** カラム。
- もし、別の物と同じ、またはそのプリフィックスであるセパレータを指定すると、**LOAD DATA INFILE** はインプットを正確に解釈する事ができません。例えば、次の **FIELDS** 条項は問題を引きこします。

```
FIELDS TERMINATED BY "" ENCLOSED BY ""
```

- もし **FIELDS ESCAPED BY** が空なら、**FIELDS TERMINATED BY** 値が後に続く **FIELDS ENCLOSED BY** か **LINES TERMINATED BY** の発生を含むフィールド値は、**LOAD DATA INFILE** のフィールドやラインの読み込みを早く止めてしまいます。これは、**LOAD DATA INFILE** はフィールドやライン値がどこで終わるのかを正しく判断する事ができない為に起こります。

次の例は、**persondata** テーブルの全てのカラムをロードします。

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

デフォルトでは、**LOAD DATA INFILE** ステートメントの最後にカラムリストがない場合、インプットラインが各テーブルカラムに対してフィールドを含みます。もし1つのテーブルのカラムをいくつかロードしたければ、カラムリストを指定してください。

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);
```

インプットファイル内のフィールドの順番が、テーブル内のカラムの順番と異なる場合は、カラムリストも指定しなければいけません。そうでなければ、MySQL はインプットフィールドとテーブルカラムをどのようにマッチさせるのか判断できません。

カラムリストはカラム名かユーザ変数のどちらかを含みます。ユーザ変数を利用すると、カラムに結果を割り当てる前に **SET** 条項で値を変換する事ができます。

SET 条項の中のユーザ変数は、いくつかの方法で利用する事ができます。次の例は、`t1.column1` の値に対して直接最初のインプット カラムを利用し、そして2番目のインプット カラムを `t1.column2` の値に利用される前に、分割操作に影響されるユーザ変数に割り当てます。

```
LOAD DATA INFILE 'file.txt'
INTO TABLE t1
(column1, @var1)
SET column2 = @var1/100;
```

SET 条項は、インプット ファイルから派生した物ではない値を提供する事ができます。次のステートメントは、現在の日付と時間に `column3` を設定する事ができます。

```
LOAD DATA INFILE 'file.txt'
INTO TABLE t1
(column1, column2)
SET column3 = CURRENT_TIMESTAMP;
```

インプット値をユーザ変数に割り当て、変数をテーブル カラムに割り当てない事で、インプット値を廃棄する事ができます。

```
LOAD DATA INFILE 'file.txt'
INTO TABLE t1
(column1, @dummy, column2, @dummy, column3);
```

カラム/変数リストと **SET** 条項の利用は、次の制約を受けます。

- **SET** 条項の中の割り当ては、割り当て演算子の左側にカラム名だけを持たなければいけません。
- **SET** 割り当ての右側でサブクエリを利用する事ができます。カラムに割り当てられる値を返すサブ クエリは、スカラサブ クエリのみでしょう。また、ロードされたテーブルから選択する為にサブクエリを利用する事はできません。
- **IGNORE** 条項に無視されるラインは、カラム/変数リストや **SET** 条項の為に処理されません。
- ユーザ変数は表示幅を持たないので、固定行フォーマットを持つデータのロード中には利用する事ができません。

インプット ラインを処理している時、**LOAD DATA** はそれをフィールドに分割し、カラム/変数リストと **SET** 条項があれば、それらに従って値を利用します。そして、結果としてできた行がテーブルに挿入されます。もしテーブルに **BEFORE INSERT** か **AFTER INSERT** トリガがあれば、それらはそれぞれ、行の挿入前か挿入後に起動されます。

もしインプット ラインのフィールドが多すぎたら、余分なフィールドは無視され、警告数が増加されます。

もしインプット ラインのフィールドが少なすぎたら、インプットフィールドがないテーブル カラムがそれらのデフォルト値として設定されます。デフォルト値の割り当てについては [データタイプデフォルト値](#) で説明しています。

空のフィールド値は、フィールド値がない場合とは異なって解釈されます。

- 文字列タイプには、カラムは空の文字列に設定されます。
- 数値タイプには、カラムは `0` に設定されます。
- 日付と時間タイプには、カラムはそのタイプに適切な「zero」値に設定されます。詳しくは [日付と時刻タイプ](#) を参照してください。

これらは、**INSERT** や **UPDATE** ステートメントの中で、空の文字列を明示的に文字列、数値、または数値と時間タイプに割り当てた時の結果と同じ値です。

TIMESTAMP カラムは、カラムに **NULL** 値がある時(**IN**)、または、**TIMESTAMP** カラムのデフォルト値が現在のタイムスタンプの時だけ現在の日付と時間に設定され、カラムのデフォルト値はフィールド リストが指定された時に削除されます。

LOAD DATA INFILE は全てのインプットを文字列とみなしますので、**INSERT** ステートメントと同じ方法で、**ENUM** か **SET** カラムにも数値の値を利用する事ができます。全ての **ENUM** と **SET** 値は文字列として指定される必要があります。

BIT 値はバイナリ表記を利用してロードする事はできません。(例えば `b'011010'`)これに対処するには、値を標準整数として指定し、それらの変換には、MySQL が数値タイプの変換を行う為に、**SET** 条項を利用し、そしてそれらを **BIT** カラムに正確にロードしてください。

```

shell> cat /tmp/bit_test.txt
2
127
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
-> INTO TABLE bit_test (@var1) SET b= CAST(@var1 AS SIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+-----+
| bin(b+0) |
+-----+
| 10      |
| 1111111 |
+-----+
2 rows in set (0.00 sec)

```

LOAD DATA INFILE ステートメントが終了する時、次のフォーマットで情報文字列を返します。

```
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

もしC API を利用していれば、`mysql_info()` 関数を呼び出す事で、ステートメントの情報を得る事ができます。詳しくは `mysql_info()` を参照してください。

LOAD DATA INFILE もまた、インプット行のフィールドが多すぎる、または少なすぎる時に警告を生成するという事を除き、値が **INSERT** ステートメントを通して挿入された時と同じ状況下で警告が発生します。(**INSERT 構文** を参照してください。) 警告はどこにも格納されません。警告数は、全てが順調であるかどうかを示す為だけに利用されます。

失敗した内容の情報を表す最初の `max_error_count` 警告のリストを得る為に、**SHOW WARNINGS** を利用する事ができます。詳しくは **SHOW WARNINGS 構文** を参照してください。

REPLACE 構文

```

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...

```

または:

```

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...

```

または:

```

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
SELECT ...

```

REPLACE は、もしテーブル内の古い行が **PRIMARY KEY** か **UNIQUE** インデックスの新しい行と同じ値を持っていれば、古い行は新しい行が挿入される前に削除されるという事以外、**INSERT** と全く同じように機能します。詳しくは **INSERT 構文** を参照してください。

REPLACE は SQL スタンドアードの MySQL 拡張子です。それは挿入、または 削除 と挿入を行います。挿入、または 更新 — を行うスタンダード SQL — の別の MySQL 拡張子に関しては、**INSERT ... ON DUPLICATE KEY UPDATE 構文** を参照してください。

テーブルが **PRIMARY KEY** か **UNIQUE** インデックスを持たなければ、**REPLACE** ステートメントの利用は何の意味も持たないという事を覚えておいてください。新しい行が別の行を複製するかどうかを決める為に利用するインデックスが無い為、それは **INSERT** と同等になります。

全てのカラムの値は、**REPLACE** ステートメントの中で指定された値から取られています。紛失したカラムは、**INSERT** と同じように、デフォルト値に設定されます。現在の行から値を参照し、それらを新しい行の中で利用する事はできません。もし、**SET col_name = col_name + 1** のような割り当てを利用すると、右側のカラム名の参照は **DEFAULT(col_name)** として扱われるので、その割り当ては **SET col_name = DEFAULT(col_name) + 1** と同等になります。

REPLACE を利用する為には、テーブルに対して **INSERT** と **DELETE** 権限の両方を持つ必要があります。

REPLACE ステートメントは、影響を受けた行数を表す為に総数を返します。これは、削除、挿入された行の総数です。もし単列 **REPLACE** の総数が1であれば、行が1つ挿入され、削除された行はないという事になります。もし総数が1よりも大きければ、新しい行が挿入される前に、1つまたはそれ以上の行が削除されたという事になります。もしテーブルが複数の固有インデックスを含んでいれば、単列が複数の古い行を置き換える事が可能であり、そして新しい行は異なる固有のインデックス内の異なる古い行に値を複製します。

影響を受けた行の総数によって、**REPLACE** が行を追加しただけなのか、それとも行の置き換えも行ったのか、という事を簡単に知る事ができます。総数が1 (追加された) か、それよりも大きい (置き換えが行われた) かを確認してください。

もしC API を利用していれば、`mysql_affected_rows()` 関数を利用する事で、影響を受けた行の総数を得る事ができます。

現在は、サブクエリの中で1つのテーブルに置き換え、同じテーブルから選択する事はできません。

MySQL は次のアルゴリズムを **REPLACE** (と **LOAD DATA ... REPLACE**)に利用します。

1. テーブルに新しい行の挿入を試みてください。
2. 主キーが固有インデックスに複製キー エラーが起きた為に挿入に失敗したら:
 - a. 複製キー値を持つ矛盾した行をテーブルから削除してください。
 - b. テーブルに新しい行の挿入をもう一度試みてください。

SELECT 構文

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr, ...
  [FROM table_references
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
  [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
  [ASC | DESC], ...]
  [LIMIT [{offset}, row_count | row_count OFFSET offset]]
  [PROCEDURE procedure_name(argument_list)]
  [INTO OUTFILE 'file_name' export_options
  | INTO DUMPFILE 'file_name'
  | INTO @var_name [, @var_name]]
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT は、1つまたは複数のテーブルから選択した行を検索するために利用され、**UNION** ステートメントとサブクエリを含む事ができます。**UNION 構文**、**サブクエリ構文** を参照して下さい。

一番良く利用される **SELECT** ステートメントの条項はこれらです。

- 各 **select_expr** は、検索したいカラムを指示します。少なくとも1つの **select_expr** が必要です。
- **table_references** はどのテーブルから行の検索をするかを指示します。その構文は **JOIN 構文** で説明されています。
- **WHERE** 条項がもしあれば、それは行が選択される為に満たさなければいけない条件を指示します。**where_condition** は選択される行が真であることを確認する式です。ステートメントは、もし **WHERE** 条項がなければ全ての行を選択します。

WHERE 条項の中では、総計 (要約) 関数以外の、MySQL がサポートする関数や演算子の全てを利用する事ができます。詳しくは **11章関数と演算子** を参照してください。

SELECT もまた、別のテーブルへの参照無しで算出された行を検索する為に利用する事ができます。

例:

```
mysql> SELECT 1 + 1;
-> 2
```

テーブルが参照されていない場合に、**DUAL** をダミー テーブルとして指定する事が許されています。

```
mysql> SELECT 1 + 1 FROM DUAL;
-> 2
```

DUAL は純粋に、全ての **SELECT** ステートメントが **FROM** と、別の条項を持津事を要求する人々の為に役立つ物です。MySQL は条項を無視するかもしれませんが。MySQL は、もしテーブルが参照されなければ **FROM DUAL** を要求しません。

通常、利用される条項は構文説明に表されるのと全く同じ順番で与える必要があります。例えば、**HAVING** 条項は **GROUP BY** 条項の前、**ORDER BY** 条項の後に来なければいけません。例外は、**INTO** 条項が構文の説明どおりに表される事も、**FROM** 条項の直前に先行して表される事も両方可能であるという事です。

- **select_expr** に **AS alias_name** を利用したエイリアスを与える事ができます。そのエイリアスは、式のカラム名として利用され、**GROUP BY**、**ORDER BY**、または **HAVING** 条項内で利用する事ができます。例:

```
SELECT CONCAT(last_name,',','first_name') AS full_name
FROM mytable ORDER BY full_name;
```

AS キーワードは、**select_expr** をエイリアスを指定する時には任意です。前出の例はこのように書く事が可能でした。

```
SELECT CONCAT(last_name,',','first_name') full_name
FROM mytable ORDER BY full_name;
```

しかし、**AS** が任意なので、2つの **select_expr** 式の間のカンマを忘れると、わずかなエラーが発生する事があります。MySQL は2番目をエイリアスだと解釈します。例えば次のステートメントの中で、**columnb** はエイリアスとして扱われています。

```
SELECT columna columnb FROM mytable;
```

この理由の為、カラムのエイリアスを指定する時には **AS** を明示的に利用する癖をつけておくとい良いでしょう。

- **WHERE** 条項が実行された時にはまだカラム値が決定されていない可能性があるので、**WHERE** 条項の中でカラムのエイリアスを利用するのは許されていません。詳しくは [Problems with Column Aliases](#) を参照してください。
- **FROM table_references** はどのテーブルから行の検索をするかを指示します。もし複数のテーブルに名前をつけると、それは結合を実行するという事になります。結合構文の情報に関しては、[JOIN 構文](#) を参照してください。指定された各テーブルにエイリアスを任意で指定する事ができます。

```
tbl_name [[AS] alias]
[(USE|IGNORE|FORCE) INDEX (key_list)]
```

どのようにインデックスを選択するかについてのオプティマイザ ヒントを与える **USE INDEX**、**IGNORE INDEX**、**FORCE INDEX** の利用については [JOIN 構文](#) で説明しています。

MySQL がテーブル スキャンの代わりにキー スキャンを好むように仕向ける代替法として **SET max_seeks_for_key=value** を利用する事ができます。詳しくは [システム変数](#) を参照してください。

- データベースを明示的に指定する為に、デフォルト データベース内で **tbl_name**、または **db_name.tbl_name** としてテーブルを参照する事ができます。**col_name**、**tbl_name.col_name**、または **db_name.tbl_name.col_name** としてカラムを参照する事ができます。参照が曖昧にならなければ、カラムの参照に **tbl_name** か **db_name.tbl_name** プリフィックスを指定する必要はありません。さらに明確なカラム参照フォームを必要とする例に関しては、[識別子の修飾語](#) を参照してください。
- テーブル参照では **tbl_name AS alias_name** か **tbl_name alias_name** 利用してエイリアスを指定する事ができます。

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
WHERE t1.name = t2.name;
```



```
SELECT t1.name, t2.salary FROM employee t1, info t2
WHERE t1.name = t2.name;
```

- アウトプットに選択されたカラムは、カラム名、カラムエイリアス、またはカラム位置を利用して **ORDER BY** と **GROUP BY** 条項の中で参照する事ができます。カラム位置は整数で、1から始まります。

```
SELECT college, region, seed FROM tournament
ORDER BY region, seed;
```

```
SELECT college, region AS r, seed AS s FROM tournament
ORDER BY r, s;
```

```
SELECT college, region, seed FROM tournament
ORDER BY 2, 3;
```

逆の順番にソートする為には、ソートに利用している **ORDER BY** 条項の中で、カラム名に **DESC** (降順) キーワードを追加してください。デフォルトは昇順です。これは **ASC** キーワードを利用して明示的に指定する事ができます。

カラム位置の利用は、構文が SQL スタンドアードから削除された為に今後廃止される可能性があります。

- もし **GROUP BY** を利用すると、アウトプット行は **GROUP BY** に従って、まるで同じカラムに **ORDER BY** を持っていたかのようにソートされます。**GROUP BY** が生成するソートのオーバーヘッドを防ぐには、**ORDER BY NULL** を追加してください。

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- MySQL は、条項の中で名づけられたカラムの後ろに **ASC** と **DESC** を指定する事もできるように、**GROUP BY** 条項を拡張します。

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- MySQL は、**GROUP BY** 条項内で言及されていないフィールドの選択を許可する為に、**GROUP BY** の利用を拡張します。もしクエリから期待通りの結果を得る事ができないのであれば、**GROUP BY 句との関数および修飾子の使用** 内の **GROUP BY** の説明を読んでください。
- GROUP BY** は **WITH ROLLUP** 修飾因子を許容します。詳しくは **GROUP BY 修飾子** を参照してください。
- HAVING** 条項は、最後の方で項目がクライアントに送られる直前に、最適化無しで適応されます。(LIMIT は **HAVING** の後で適応されます。)

SQL スタンドアードは **HAVING** が **GROUP BY** 条項の中、または総計関数の中で利用されるカラムだけを参照する事を要求します。しかし、MySQL はこの動作に拡張子をサポートし、**HAVING** が **SELECT** リストの中のカラムと外部のサブクエリの中のカラムを参照する事を許容します。

もし **HAVING** 条項が曖昧なカラムを参照すると、警告が発生します。次のステートメントの中では、エイリアスとカラム名の両方として利用されている為 **col2** は曖昧です。

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

スタンダード SQL の動作には優先権が与えられるので、もし **HAVING** カラム名が **GROUP BY** と、アウトプットカラムリスト内のエイリアスカラムの両方で利用されると、優先権は **GROUP BY** カラム内のカラムに与えられます。

- HAVING** は、**WHERE** 条項内になければいけない項目に対しては利用しないでください。例えば、次のような物を書かないでください。

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

代わりにこのように書いてください。

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- HAVING** 条項は **WHERE** 条項が参照できない総計関数を参照する事ができます。

```
SELECT user, MAX(salary) FROM users
GROUP BY user HAVING MAX(salary) > 10;
```


(これは MySQL の古いバージョンでは機能しませんでした。)

- MySQL は複製カラム名を許容します。これは、同じ名前の `select_expr` が複数存在できるということです。これはスタンダード SQL の拡張子です。MySQL はまた `GROUP BY` と `HAVING` が `select_expr` 値を参照する事を許容するので、この結果は曖昧になり得ます。

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

このステートメントの中では、両方のカラムが `a` という名前を持ちます。グループ分けに正しいカラムを利用する事を保障する為に、各 `select_expr` に異なる名前を利用してください。

- MySQL は `select_expr` 値の中、そして `FROM` 条項内のテーブル カラムの中を検索する事で `ORDER BY` 条項内の無条件のカラムやエイリアス参照を解決します。`GROUP BY` か `HAVING` 条項に対しては、`select_expr` 値内を検索する前に `FROM` 条項を検索します。(`GROUP BY` と `HAVING` に対しては、`ORDER BY` に対するのと同じルールを利用した MySQL 5.0 以前の動作とは異なります。)
- `LIMIT` 条項は `SELECT` ステートメントに返された行数を制限するのに利用する事ができます。`LIMIT` は、負数以外の整数定数でなければいけない、1つか2つの数値引数を取ります。(準備されたステートメントを利用している時以外)

その2つの引数のうち、最初の物は返される最初の行のオフセットを指定し、2つめの物は返される行の最高数を指定します。冒頭の行のオフセットは0です。(1ではありません)

```
SELECT * FROM tbl LIMIT 5,10; # Retrieve rows 6-15
```

全ての行を一定のオフセットから結果セットの最後まで検索するには、2つめのパラメータに大きい数字を利用する事ができます。このステートメントは96番目の行から最後まで全ての行を検索します。

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

1つの引数で、その値は結果セットの最初から返される行数を指定します。

```
SELECT * FROM tbl LIMIT 5; # Retrieve first 5 rows
```

言い換えると、`LIMIT row_count` は `LIMIT 0, row_count` と同等だという事になります。

用意されたステートメントには、プレースホルダを利用する事ができます。次のステートメントは `tbl` テーブルから行を1つ返します。

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
EXECUTE STMT USING @a;
```

次のステートメントは `tbl` テーブルから2行目から6行目を返します。

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

PostgreSQL との互換性に対しては、MySQL はまた `LIMIT row_count OFFSET offset` 構文もサポートします。

- `SELECT` の `SELECT ... INTO OUTFILE 'file_name'` 型は選択された行をファイルに書き込みます。ファイルはサーバソフト上に作成されるので、この構文を利用するには `FILE` 権限を持たなければいけません。`file_name` は、`/etc/passwd` のようなファイルやデータベース テーブルが、その他の物の間で破壊されるのを防ぐ既存ファイルにはなり得ません。MySQL 5.1.6 以降のバージョンでは、`character_set_filesystem` システム変数は、ファイル名の解明をコントロールします。

`SELECT ... INTO OUTFILE` ステートメントはそもそも、サーバマシン上のテキスト ファイルにテーブルをすばやく捨てさせる事を意図しています。もしサーバ ホストではなく、クライアント ホスト上に結果ファイルを作成したければ、`SELECT ... INTO OUTFILE` を利用する事はできません。その場合、クライアント ホスト上にファイルを生成する為には、代わりに `mysql -e "SELECT ..." > file_name` のようなコマンドを利用しなければいけません。

`SELECT ... INTO OUTFILE` は `LOAD DATA INFILE` の補数です。ステートメントの `export_options` 部分の構文は、`LOAD DATA INFILE` ステートメントと共に利用される物と同じ `FIELDS` と `LINES` 条項で成り立っています。詳しくは `LOAD DATA INFILE 構文` を参照してください。

FIELDS ESCAPED BY は、特別な文字をどのように書き込むのかをコントロールします。もし **FIELDS ESCAPED BY** 文字が空でなければ、それはアウトプット上で次の文字に先行するプリフィックスとして利用されます。

- **FIELDS ESCAPED BY** 文字
- **FIELDS [OPTIONALLY] ENCLOSED BY** 文字
- **FIELDS TERMINATED BY** と **LINES TERMINATED BY** 値の最初の文字
- ASCII **NUL** (ゼロの値のバイト;実際に拡張文字の後ろに書かれているのは、ゼロの値のバイトではなく、ASCII '0'です。)

FIELDS TERMINATED BY、**ENCLOSED BY**、**ESCAPED BY**、または **LINES TERMINATED BY** 文字は、ファイルを実際に読み返す事ができるように、拡張されなければいけません。ASCII **NUL** は、ポケベルで見やすくする為に拡張されています。

結果のファイルは SQL 構文と一致する必要がないので、他の物は拡張される必要はありません。

もし **FIELDS ESCAPED BY** 文字が空なら、文字が拡張される事はなく、**NULL** は **\N** ではなく **NULL** としてアウトプットされます。特に、もしデータ中にフィールド値が先ほどのリストの中の文字を含んでいる場合は、空の拡張文字を指定するのは良い考えではないかも知れません。

ここに、多くのプログラムで利用されるカンマで区切られた値 (CSV)のフォーマットのファイルを生成する例があります。

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

- もし **INTO OUTFILE** の代わりに **INTO DUMPFILE** を利用すると、MySQL は、カラムやラインのターミネーションや、拡張操作を行う事無く、ファイルの中に1つだけ書き込みます。これは、もしファイルの中に **BLOB** 値を格納したいのであれば有効です。
- **INTO** 条項は、複数のユーザ定義変数のリストに名前をつける事ができます。選択された値は変数に割り当てられます。変数の数はカラム数と一致しなければいけません。

ストアド ルーチンの中では、その変数はルーチン パラメータがローカル変数になり得ます。詳しくは **SELECT ... INTO ステートメント** を参照してください。

- 注意:**INTO OUTFILE** か **INTO DUMPFILE** によって作成されたファイルは、サーバホスト上で全てのユーザによって書き込まれます。この理由は、MySQL サーバは、それを起動させているアカウントの持ち主であるユーザ以外によって所有されているファイルを作成する事はできない という事です。(これらの理由の為に、**mysqld** を **root** として起動する事は絶対にしてはいけません。)ですので、このファイルは内容を真似する事ができるように、誰でも修正ができる物である必要があります。
- このセクションの最初の **SELECT** 構文の説明で、ステートメントの終わりの方の **INTO** 条項が表されています。**FROM** 条項に先行して、**INTO OUTFILE** か **INTO DUMPFILE** を直接利用する事も可能です。
- **PROCEDURE** 条項は、結果セットの中にデータを処理しなければいけないプロシージャに名前をつけます。(例については、**Procedure Analyse** をご覧ください。)
- もしページカ行ロックを利用するストレージ エンジンと **FOR UPDATE** を一緒に利用するなら、クエリに検査された行は現在のトランザクションが終わるまで書き込みロックされます。 **LOCK IN SHARE MODE** を利用すると、他のトランザクションが検査された行を読む事は許容しますが、それらを更新や削除する事は許容しない共通ロックを設定します。詳しくは **SELECT ... FOR UPDATE** と **SELECT ... LOCK IN SHARE MODE ロック読み取り** を参照してください。

SELECT キーワードの後で、ステートメントの操作に影響を与える幾つかのオプションを利用する事ができます。

ALL、**DISTINCT**、そして **DISTINCTROW** オプションは、複製行が返されるべきかどうかを指定します。もしこれらのオプションが何も与えられなければ、デフォルトは **ALL** です。(全ての一致する行が返されます。)**DISTINCT** と **DISTINCTROW** は同義語で、結果セットから複製行を削除する指示を出します。

[HIGH_PRIORITY](#)、[STRAIGHT_JOIN](#)、そして [SQL_](#) で始まるオプションは、スタンダード SQL の MySQL 拡張子です。

- [HIGH_PRIORITY](#) は [SELECT](#) に、テーブルを更新するステートメントよりも高い優先順位を与えます。これは、スピードがとても速く、一度に実行されなければいけないクエリに対してだけ利用して下さい。読み込みの為にテーブルがロックされている間に発行された [SELECT HIGH_PRIORITY](#) クエリは、テーブルがフリーになるのを待っている更新ステートメントがあったとしても実行します。

[HIGH_PRIORITY](#) は、[UNION](#) の一部である [SELECT](#) ステートメントと一緒に利用できません。

- [STRAIGHT_JOIN](#) は、オプティマイザが [FROM](#) 条項内にリストされている順番でテーブルに結合するよう働きかけます。もし最適化ツールが、最適ではない順番でテーブルに接合した時、クエリのスピードを早くする為にこれを利用する事ができます。詳しくは [EXPLAINを使用して、クエリを最適化する](#) を参照してください。[STRAIGHT_JOIN](#) はまた [table_references](#) リストの中でも利用できます。詳しくは [JOIN 構文](#) を参照してください。
- [SQL_BIG_RESULT](#) は、オプティマイザに結果セットが行を多く持っている事を教える為に [GROUP BY](#) か [DISTINCT](#) と共に利用する事ができます。この場合、MySQL は必要であればディスクベースのテンポラリ テーブルを直接利用し、[GROUP BY](#) 要素上のキーを持つテンポラリ テーブルを利用してソートします。
- [SQL_BUFFER_RESULT](#) は結果がテンポラリ テーブルの中に置かれるよう働きかけます。これは、MySQL がテーブル ロックを早く解除するのを助け、クライアントに結果セットを送るのに時間がかかる場合に補助します。
- [SQL_BIG_RESULT](#) は、オプティマイザに結果セットが小さい事を教える為に [GROUP BY](#) か [DISTINCT](#) と共に利用する事ができます。この場合、MySQL は結果テーブルを格納する為、ソート機能を利用する代わりに高速のテンポラリ テーブルを利用します。通常これは必要ではないでしょう。
- [SQL_CALC_FOUND_ROWS](#) は全ての [LIMIT](#) 条項を無視して、結果セットの中にいくつ行があるかを計算するように MySQL に指示します。行数は [SELECT FOUND_ROWS\(\)](#) を利用して検索する事ができます。詳しくは [情報関数](#) を参照してください。
- 2 or [DEMAND](#) の [query_cache_type](#) 値を利用している場合、[SQL_CACHE](#) は、クエリ キャッシュの中にクエリの結果を格納するよう MySQL に指示します。このオプションは、[UNION](#) かサブクエリを利用するクエリに対して、クエリ中の全ての [SELECT](#) に影響を与えます。詳しくは [MySQL クエリ キャッシュ](#) を参照してください。
- [SQL_NO_CACHE](#) は MySQL に対して、クエリ キャッシュ内のクエリの結果を格納しないように指示します。詳しくは [MySQL クエリ キャッシュ](#) を参照してください。[UNION](#) かサブクエリを利用するクエリに対して、このオプションはクエリ中の全ての [SELECT](#) に影響を与えます。

JOIN 構文

MySQL は、[SELECT](#) ステートメントの [table_references](#) 部分と、複合テーブル [DELETE](#) と [UPDATE](#) ステートメント

に対して、次の [JOIN](#) 構文をサポートします。

```
table_references:
    table_reference [, table_reference] ...

table_reference:
    table_factor
    | join_table

table_factor:
    tbl_name [[AS] alias]
        [{USE|IGNORE|FORCE} INDEX (key_list)]
    | ( table_references )
    | { OJ table_reference LEFT OUTER JOIN table_reference
        ON conditional_expr }

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
    | table_reference STRAIGHT_JOIN table_factor
    | table_reference STRAIGHT_JOIN table_factor ON condition
    | table_reference LEFT [OUTER] JOIN table_reference join_condition
    | table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
    | table_reference RIGHT [OUTER] JOIN table_reference join_condition
```

```
| table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor
join_condition:
  ON conditional_expr
| USING (column_list)
```

テーブル参照は、接合式としても知られています。

`table_factor` の構文は、SQL スタンドアードと比較して拡張されます。後者は `table_reference` だけを許容し、カッコ内のそれらのリストは許容しません。

もし `table_reference` のリスト内の各カラムが内側の接合と同等であると考え、これは保守的な拡張子という事になります。例:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
  ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

は次の物と同等です:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
  ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

MySQL では、`CROSS JOIN` は `INNER JOIN` と構文的に同等です。(お互いに置き換える事ができます。)スタンダード SQL では、それらは同等ではありません。`INNER JOIN` は `ON` 条項と共に利用され、そうでなければ `CROSS JOIN` が利用されます。

通常、内側結合演算だけを含む結合式内のカッコは無視する事ができます。MySQL はネスト化した接合をサポートします。(入れ子結合最適化を参照してください。)

次のリストには、接合を書く時に考慮に入れる通常の要因が説明されています。

- テーブル参照では `tbl_name AS alias_name` か `tbl_name alias_name` を利用してエイリアスを指定する事ができます。

```
SELECT t1.name, t2.salary
  FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

SELECT t1.name, t2.salary
  FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
```

- `INNER JOIN` と、(カンマ)は結合条件がない場合には意味的に同等となります。両方とも、指示されたテーブルの間にデカルト結果を作り出します。(これは、最初のテーブル内の行1つ1つが、2番目のテーブルの行1つ1つに接合されるという事です。)

しかし、カンマ演算子の先行は、`INNER JOIN`、`CROSS JOIN`、`LEFT JOIN` 等のそれよりも少ないです。もし接合条件がある場合にカンマ接合と別のタイプの接合を混合すると、`Unknown column 'col_name' in 'on clause'` という形のエラーが発生するかもしれません。この問題の対処法は、このセクションの後半で紹介します。

- `ON` 条件文は `WHERE` 条項の中で利用する事ができる形の条件文です。通常、テーブルをどのように接合するのかを指定する条件には `ON` 条項を、結果セットの中にどの行が必要であるかを制限するには `WHERE` 条項を利用する必要があります。
- もし `LEFT JOIN` 内の `ON` か `USING` 部分内に右側のテーブルに一致する行がなければ、全てのカラムが `NULL` に設定されている行が右側のテーブルに利用されます。この事実は、別のテーブル内に対応する物を持たないテーブル内の行を見つける為に利用する事ができます。

```
SELECT table1.* FROM table1
  LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

この例は、`table2` の中に存在しない `id` 値を持つ `table1` 内全ての行を見つけます。(table2 内に対応する行を持たない table1 内全ての行)これは、`table2.id` が `NOT NULL` を宣言したと仮定します。詳しくは `LEFT JOIN` と `RIGHT JOIN` 最適化を参照してください。

- `USING(column_list)` 条項は、両方のテーブルに存在しなければいけないカラムのリストに名前をつけます。もしテーブル `a` と `b` の両方がカラム `c1`、`c2`、そして `c3` を含むと、次の接合は二つのテーブルの対応するカラムを比較します。

```
a LEFT JOIN b USING (c1,c2,c3)
```

- 2つのテーブルの **NATURAL [LEFT] JOIN** は **INNER JOIN** か、両方のテーブルに存在する全てのカラムに名前を付ける **USING** 条項を持つ **LEFT JOIN** と意味的に同等になるよう定義されます。
- **RIGHT JOIN** は **LEFT JOIN** と同じように機能します。コードがデータベース全体に移植できる状態を保つ為に、**RIGHT JOIN** の代わりに **LEFT JOIN** を利用する事をお勧めします。
- 接合構文の説明で表されている **{ OJ ... LEFT OUTER JOIN ... }** 構文は ODBC を利用した互換性に対してだけ存在します。構文内のカールした中括弧は文字通り書き込まれる必要があります。それらは構文説明の別の部分で利用されているようなメタシンタックスではありません。
- **STRAIGHT_JOIN** は、左側のテーブルがいつも右側のテーブルの前に読み込まれるという事以外は **JOIN** と全く同じです。これは、接合オプティマイザがテーブルを間違った順番で置いてしまうという（数少ない）場合に利用する事ができます。

接合の例:

```
SELECT * FROM table1, table2;

SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
LEFT JOIN table3 ON table2.id=table3.id;
```

テーブルから情報を検索する時、MySQL がどのインデックスを利用すべきか、ヒントを与える事ができます。**USE INDEX (key_list)** を指定する事で、MySQL がテーブル内の行を見つける為に、有効なインデックスを1つだけ利用するように指示する事ができます。既存構文 **IGNORE INDEX (key_list)** は、MySQL がいくつかの特定のインデックスを利用しないように指示する事ができます。これらのヒントは、MySQL が可能なインデックスのリストの中から、間違ったインデックスを利用している事を、**EXPLAIN** が表示した時に便利な物です。

テーブル スキャンは とても 高いと仮定されますが、**USE INDEX (key_list)** のように機能する **FORCE INDEX** を利用する事もできます。言い換えると、テーブル内の行を見つける為に与えられたインデックスを利用できない場合、テーブル スキャンを利用する事ができるという事です。

USE INDEX、**IGNORE INDEX**、そして **FORCE INDEX** は、MySQL がどのようにテーブルの中の行を見つけ、接合を行うのかを決定する時に、どのインデックスが利用されるのかという事にだけ影響を与えます。それらは、**ORDER BY** か **GROUP BY** を解決する時にインデックスを利用するかどうかという事に影響を与えます。

USE KEY、**IGNORE KEY**、そして **FORCE KEY** は **USE INDEX**、**IGNORE INDEX**、そして **FORCE INDEX** の同義語です。

例：

```
SELECT * FROM table1 USE INDEX (key1,key2)
WHERE key1=1 AND key2=2 AND key3=3;

SELECT * FROM table1 IGNORE INDEX (key3)
WHERE key1=1 AND key2=2 AND key3=3;
```

接合処理は MySQL 5.0.12 で変更されました。

注意:自然接合と、外部接合異形を含む **USING** を利用した接合は、SQL:2003 スタンドードに従って処理されます。その目的は、SQL:2003 に従い **NATURAL JOIN** と **JOIN ... USING** について、MySQL の構文と動作を提携させる事でした。しかし、接合処理に関してのこれらの変更は、いくつかの接合に関して異なるアウトプット カラムをもたらす可能性があります。また、古いバージョン (5.0.12 以前の物) で正しく機能していたいくつかのクエリも、スタンダードに適合する為に書き直される必要があります。

これらの変更には、主に5つの特徴があります。

- MySQL が **NATURAL** か **USING** 接合操作の結果カラムを決定する方法。(従って **FROM** 条項の結果という事)
- 選択されたカラムのリストの中への **SELECT *** と **SELECT tbl_name.*** の拡大。
- **NATURAL** か **USING** 接合内でのカラム名の決定。
- **NATURAL** か **USING** 接合の **JOIN ... ON** への変形。
- **JOIN ... ON** の **ON** 条件内のカラム名の決定。

次のリストに、現在のバージョンと古いバージョンの接合処理の効果について比べた詳細が紹介されています。「以前は」という言葉は「MySQL 5.0.12 以前」という意味です。

- **NATURAL** 接合や **USING** 接合のカラムは以前と異なるかもしれません。特に、余分なアウトプット カラムはもう現れません、そして、**SELECT *** 拡大のカラムの順番は以前とは異なるかもしれません。

このステートメントのセットを検討してください。

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

以前は、このステートメントはこのアウトプットを産出しました。

```
+---+---+---+---+
|i |j |k |j |
+---+---+---+---+
| 1| 1| 1| 1|
+---+---+---+---+
|i |j |k |j |
+---+---+---+---+
| 1| 1| 1| 1|
+---+---+---+---+
```

最初の **SELECT** ステートメントの中で、カラム **j** は両方のテーブル内に現れた為に接合カラムになります。という事は、スタンダード SQL によると、それはアウトプット内に2回ではなく1回のみ現れる必要があるという事になります。同じように、2番目の **SELECT** ステートメントの中で、カラム **j** は **USING** 条項の中で名前が付けられ、2回ではなく1回だけアウトプットの中に現れる必要があります。しかし、この両方で余分なカラムは排除されていません。また、スタンダード SQL によると、カラムの順番は正しくありません。

そして、ステートメントはこのアウトプットを産出します。

```
+---+---+---+
|j |i |k |
+---+---+---+
| 1| 1| 1|
+---+---+---+
|j |i |k |
+---+---+---+
| 1| 1| 1|
+---+---+---+
```

余分なカラムは排除され、スタンダード SQL によると、このカラムの順番は正しいです。

- 最初に、1つ目のテーブルの順番で、2つの接合したテーブルに共通するカラムを合体させました。
- 次に、テーブルの順番で、最初のテーブル固有のカラムを合体させました。
- 最後に、テーブルの順番で、2番目のテーブル固有のカラムを合体させました。

2つの共通カラムを置き換えられる単一結果カラムは、合体操作を通して定義されました。これは、次のステートメントで、**t1.a** と **t2.a** の2つに対して、導き出された1つの接合カラム **a** は **a = COALESCE(t1.a, t2.a)** として定義される、という事です。


```
COALESCE(x, y) = (CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END)
```

もし接合操作がそれ以外の接合であれば、その接合の結果カラムは、接合されたテーブルの全てのカラムの連続で構成されます。これは以前と同じです。

外部接合に関する合体したカラム定義の結論は、合体したカラムはもし2つのうち1つのカラムがいつも `NULL` であれば、非 `NULL` カラムの値を含む、という事です。もしどちらのカラムも `NULL` でない、または両方がそうである場合、両方の共通カラムは同じ値を持つので、どちらが合体したカラムの値として選択されるかというのは特に問題にはなりません。これを理解する簡単な方法は、外部接合の合体したカラムは `JOIN` の内側テーブルの共通カラムによって表される、と考える事です。テーブル `t1(a,b)` と `t2(a,c)` が次のコンテンツを持つと仮定してください。

```
t1  t2
---  ---
1 x  2 z
2 y  3 w
```

すると:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+-----+-----+
| a | b | c |
+-----+-----+
| 1 | x | NULL |
| 2 | y | z |
+-----+-----+
```

ここでは、カラム `a` は `t1.a` の値を含んでいます。

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+-----+-----+
| a | c | b |
+-----+-----+
| 2 | z | y |
| 3 | w | NULL |
+-----+-----+
```

ここでは、カラム `a` は `t2.a` の値を含んでいます。

これらの結果を `JOIN ... ON` を利用した他の同等のクエリと比較してください。

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+-----+-----+-----+
| a | b | a | c |
+-----+-----+-----+
| 1 | x | NULL | NULL |
| 2 | y | 2 | z |
+-----+-----+-----+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+-----+-----+-----+
| a | b | a | c |
+-----+-----+-----+
| 2 | y | 2 | z |
| NULL | NULL | 3 | w |
+-----+-----+-----+
```

- 以前、`USING` 条項は、対応するカラムを比較する `ON` 条項として再度書き込む事ができました。例えば、次の2つの条項は意味的に全く同じでした。

```
a LEFT JOIN b USING (c1,c2,c3)
a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
```

今はもうこの2つの条項は同じではありません。

- どの行が接合条件を満たすかの判断に関しては、両方の接合は意味的に全く同じままです。

- **SELECT *** 拡大に対してどのカラムを表示するかの判断に関しては、両方の接合は意味的に全く同じではありません。**ON** 接合が全てのテーブルから全てのカラムを選択するのに対して、**USING** 接合は対応するカラムの合体した値を選択します。先行する **USING** 接合に対しては、**SELECT *** はこれらの値を選択します。

```
COALESCE(a.c1,b.c1), COALESCE(a.c2,b.c2), COALESCE(a.c3,b.c3)
```

ON 接合に対しては、**SELECT *** が次の値を選択します。

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

内部結合では、両方のカラムが同じ値を持つので **COALESCE(a.c1,b.c1)** は **a.c1** か **b.c1** と同じです。外部接合では(**LEFT JOIN** のような)、2つのうち1つのカラムが **NULL** になり得ます。そのカラムは結果から排除されます。

- 多方向自然接合の評価は、**NATURAL** か **USING** 接合の結果に影響を与え、クエリの再書き込みを必要とするような、大変重要な形で異なっています。それぞれが行を1つ持つ3つのテーブル **t1(a,b)**、**t2(c,b)**、そして **t3(a,c)** があると仮定してください。**t1(1,2)**、**t2(10,2)**、そして **t3(7,10)** です。また、その3つのテーブル上にこの **NATURAL JOIN** も持っているとは仮定してください。

```
SELECT ... FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
```

以前は、2つめの接合の左のオペランドは、ネスト化した接合 (**t1 NATURAL JOIN t2**) とならなければいけない一方、**t2** となると考えられていました。その結果、**t3** のカラムは **t2** の中だけで共通カラムに関して確認され、そしてもし **t3** が **t1** を持つ共通カラムを持っていれば、これらのカラムは等価接合カラムとして利用されません。従って、以前は先行クエリは次の等価接合に変形されていました。

```
SELECT ... FROM t1, t2, t3
WHERE t1.b = t2.b AND t2.c = t3.c;
```

その接合では、もう1つの等価接合述語 (**t1.a = t3.a**) がなくなっています。その結果、それはもう1つ行を作成するので、結果は空にはなりません。正しい同等のクエリはこれです。

```
SELECT ... FROM t1, t2, t3
WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
```

もし現在の MySQL のバージョンの中で、古いバージョンと同じクエリの結果が必要であれば、自然接合を最初の等価接合として書き換えてください。

- 以前は、カンマ演算子(,)と **JOIN** の両方は同じ優先順位だったので、接合式 **t1, t2 JOIN t3** は **((t1, t2) JOIN t3)** として解釈されました。現在は **JOIN** が高い優先順位を持つので、式は **(t1, (t2 JOIN t3))** として解釈されます。この変更は、**ON** 条項が接合の演算子内のカラムだけを参照する事ができ、優先順位の変更はそれらの演算子が何であるかについての解釈を変えてしまうので、この条項を利用するステートメントに影響を与えます。

例:

```
CREATE TABLE t1 (i1 INT, j1 INT);
CREATE TABLE t2 (i2 INT, j2 INT);
CREATE TABLE t3 (i3 INT, j3 INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
INSERT INTO t3 VALUES(1,1);
SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
```

以前は、**(t1,t2)** としての **t1,t2** の暗黙のグループ分けのおかげで、**SELECT** は正当でした。現在は **JOIN** が優先順位を持つので **ON** 条項の演算子は **t2** と **t3** です。**t1.i1** がどちらの演算子でもないの、結果は **Unknown column 't1.i1' in 'on clause'** エラーになります。接合を実行させるには、**ON** 条項の演算子が **(t1,t2)** と **t3** となるように、括弧を利用して最初の2つのテーブルを明示的にグループ分けして下さい。

```
SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
```

または、カンマ演算を利用するのを避け、その代わりに **JOIN** を利用してください。

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
```

この変更は、カンマ演算子とそれよりも高い優先順位を持つ **INNER JOIN**、**CROSS JOIN**、**LEFT JOIN** または **RIGHT JOIN** を混合するステートメントにも適応します。

- 以前は、ON 条項はその右側で名前が付けられたテーブル内のカラムを参照する事ができました。現在は ON 条項はその演算子だけ参照する事ができます。

例:

```
CREATE TABLE t1 (i1 INT);
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

以前は、SELECT ステートメントは正当でした。現在は、i3 は、ON 条項の演算子ではない t3 内のカラムなので、ステートメントは `Unknown column 'i3' in 'on clause'` エラーで失敗します。ステートメントは次のように書き換えられなければいけません。

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- NATURAL か USING 接合内でのカラム名の決定は、以前とは違います。FROM 条項の外にあるカラム名に対しては、MySQL は以前と比べると上位集合であるクエリを扱います。それは、以前は MySQL がいくつかのカラムが曖昧であるというエラーを発行したような場合でも、現在はクエリが正確に扱われるという事です。これは、現在は MySQL が NATURAL や USING 接合の共通カラムを単一カラムとして扱う為、クエリがそのようなカラムを参照した時、クエリ コンパイラがそれらを曖昧だとは認識しないという事実によるものです。

例:

```
SELECT * FROM t1 NATURAL JOIN t2 WHERE b > 1;
```

以前は、このクエリは `ERROR 1052 (23000)` を導いていました。場所条項内の、カラム 'b' が曖昧です。現在は、そのクエリは正しい結果を導きます。

```
+-----+-----+-----+
| b | c | y |
+-----+-----+-----+
| 4 | 2 | 3 |
+-----+-----+-----+
```

SQL:2003 スタンダードと比較した MySQL の拡張機能の1つは、スタンダードは NATURAL や USING 接合(以前のような)の共通(合体した)カラムを修飾する事を許可しなかったのに対して、MySQL はそれを許可するという事です。

UNION 構文

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

UNION は、結果を複数 SELECT ステートメントから単一結果セットに結合させる為に利用されます。

最初の SELECT ステートメントからのカラム名は、返された結果のカラム名として利用されます。各 SELECT ステートメントの対応する位置にリストされている選択されたカラムは、同じデータタイプを持つ必要があります。(例えば、最初のステートメントに選択された最初のカラムは、別のステートメントに選択された最初のカラムと同じデータタイプを持つ必要があります。)

もし、対応する SELECT カラムのデータタイプが一致しなければ、UNION 結果内のタイプとカラムの長さは、全ての SELECT ステートメントによって検索された値を考慮する必要があります。例えば、次の物を検討してみてください。

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a             |
| bbbbbbbbbbb   |
+-----+
```

(MySQL の初期のバージョンでは、最初の SELECT のタイプと長さだけが利用され、2つ目の行は長さ1まで切り捨てられていました。)

SELECT ステートメントは通常の選択ステートメントですが、次の制約があります。

- 最後の **SELECT** ステートメントだけが **INTO OUTFILE** を利用できます。
- HIGH_PRIORITY** は、**UNION** の一部である **SELECT** ステートメントと一緒に利用できません。

もしそれを最初の **SELECT** に指定しても、効果はありません。もしそれを後に続く **SELECT** ステートメントに指定すると、構文エラーが起こります。

UNION のデフォルトの動作は、複製行は結果から削除されるという事です。任意の **DISTINCT** キーワードは、複製行の削除の指定もするので、デフォルト以外に何も効果は持ちません。任意の **ALL** キーワードを利用すると、複製行の削除は行われず、結果には全ての **SELECT** ステートメントからの一致する行が含まれます。

UNION ALL と **UNION DISTINCT** を同じクエリの中で混合する事ができます。混合された **UNION** タイプは **DISTINCT** ユニオンが全ての **ALL** ユニオンをその左側に上乗せするような形で扱われます。**DISTINCT** ユニオンは **UNION DISTINCT** を利用して明示的に、また後に **DISTINCT** や **ALL** キーワードがない **UNION** を利用して暗黙的に作成されます。

ORDER BY や **LIMIT** 条項を、**UNION** 結果全体をソートしたり制限したりする為に利用するには、各 **SELECT** ステートメントを括弧で囲み、最後の物の後に **ORDER BY** か **LIMIT** を置いて下さい。次の例は、両方の条項を利用しています。

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

この種の **ORDER BY** はテーブル名を含むカラム参照を利用する事ができません。(それは、`tbl_name.col_name` フォーマット内の名前です。)その代わりに、最初の **SELECT** ステートメント内でカラムエイリアスを提供し、**ORDER BY** 内でそのエイリアスを参照します。(あるいは、そのカラムの位置を利用して **ORDER BY** 内でカラムを参照します。しかし、カラム位置の使用は今後廃止予定です。)

また、もし格納されるカラムがエイリアスされると、**ORDER BY** 条項は、カラム名ではなく、そのエイリアスを参照しなければいけません。次のステートメントの1つ目の物は機能しますが、2つ目は `Unknown column 'a' in 'order clause'` エラーで失敗します。

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

個々の **SELECT** に **ORDER BY** か **LIMIT** を適用するには、**SELECT** を囲む括弧内に条項を置いて下さい。

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

デフォルトの **UNION** が不規則な順番の行セットを作り出す為、個々の **SELECT** ステートメントへの **ORDER BY** の利用は、最終結果の中で行がどのような順番で現れるのかを暗示しません。もし **ORDER BY** が **LIMIT** と共に現れると、検索の為に選択された行のサブセットを **SELECT** に決定する為に利用されますが、それは最終的な **UNION** の結果内の行の順番に影響を与えるとは限りません。もし **ORDER BY** が **SELECT** 内に **LIMIT** 無しで現れても、何の効果も持たない為最適化されて切り離されます。

UNION の結果内の行が、各 **SELECT** によって1つずつ検索された行で構成されるようにするには、各 **SELECT** からソートカラムとして利用する為の追加カラムを選択し、最後の **SELECT** の後に **ORDER BY** を追加してください。

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

さらに、個々の **SELECT** の結果の中でソートの順番を維持する為には、**ORDER BY** 条項に補助的なカラムを追加してください。

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

サブクエリ構文

サブクエリは、別のステートメント内の **SELECT** ステートメントです。

MySQL 4.1 から、MySQL 特有のいくつかの特徴と同様に、SQL スタンドアードが要求する全てのサブクエリ型と演算子がサポートされています。

ここに、同じようなサブクエリの例があります。

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

この例の中では、**SELECT * FROM t1 ...** は 外部クエリ (または 外部ステートメント) であり、**(SELECT column1 FROM t2)** は サブクエリ です。これは、サブクエリが外部クエリ内で ネスト化された という事であり、実際、サブクエリを別のサブクエリ内で、相当な深さまでネスト化する事が可能です。サブクエリは必ずカッコ内に表示されなければいけません。

サブクエリの主な利点は次のような物になります。

- それらは、ステートメントのそれぞれの部分を分離させる事ができるように、構造化された クエリを許容します。
- それらは、複雑な接合や合併を要求されないように、演算を行う為の代替法を提供します。
- 多くの人の意見によると、それらは複雑な接合や合併と比べると、読み込みやすいという事です。実際これは、初期 SQL の「Structured Query Language.」の創案を人々に与えたサブクエリの発明でした。

ここに、SQL スタンドアードによって指定され、MySQL 内でサポートされているサブクエリ構文に関する主なポイントを説明するステートメントの例があります。

```
DELETE FROM t1
WHERE s11 > ANY
(SELECT COUNT(*) /* no hint */ FROM t2
WHERE NOT EXISTS
(SELECT * FROM t3
WHERE ROW(5*t2.s1,77)=
(SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
(SELECT * FROM t5) AS t5)));
```

サブクエリは、スカラ(単一値)、単一行、単一カラム、またはテーブル(1つ、または複数カラムの、1つ、または複数行)を返す事ができます。これらはスカラ、カラム、行、そしてテーブル サブクエリと呼ばれます。次のセクションで説明されているように、頻繁に特定の種類の結果を返すサブクエリは、特定のコンテキストの中だけで利用する事ができます。

サブクエリを利用する事ができるステートメントのタイプには、いくつかの制限があります。サブクエリは、普通の **SELECT** が含む事のできるキーワードや条項を全て含む事ができます。それは **DISTINCT**、**GROUP BY**、**ORDER BY**、**LIMIT**、接合、インデックスヒント、**UNION** 構成、コメント、関数などです。

制限の1つは、サブクエリの外部ステートメントが **SELECT**、**INSERT**、**UPDATE**、**DELETE**、**SET**、または **DO** のうちのどれか1つでなければいけないという事です。その他の制限は、現在はサブクエリの中でテーブルを変更したり、同じテーブルから選択する事ができないという事です。これは、**DELETE**、**INSERT**、**REPLACE**、**UPDATE**、そして(サブクエリは **SET** 条項内で利用できる為) **LOAD DATA INFILE** のようなステートメントに適応します。

サブクエリ構文の特定型に関する性能問題を含む、サブクエリ利用に関する制限のさらなる総合的な説明に関しては、[サブクエリの規制](#) で紹介されています。

スカラ演算子としてのサブクエリ

簡単に言うと、サブクエリは単一値を返すスカラ サブクエリという事になります。スカラ サブクエリは単純な演算子で、単一カラム値や直定数が正当であればほとんどどこでも利用する事ができ、データタイプ、長さ、**NULL** になり得るかどうかという指示など、全ての演算子が持つ特徴も全て持っています。ここにその例があります。

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

この **SELECT** 内のサブクエリは、**CHAR** のデータタイプを持ち、長さが5で、**CREATE TABLE** 時にデフォルトと同等の文字セットと照合が実施されており、そしてカラム内の値が **NULL** になり得るという指示を持つ単一値('abcde')

を返します。実際は、ほとんどのサブクエリが **NULL** になり得ます。もし例で使用されたテーブルが空であれば、サブクエリの値は **NULL** になるでしょう。

スカラ サブクエリを利用できないコンテキストがいくつかあります。もしステートメントが直定数値だけを許容するなら、サブクエリを利用する事はできません。例えば、**LIMIT** が直定整数引数を要求し、**LOAD DATA INFILE** が直定数文字列ファイル名を要求します。これらの値を供給するのにサブクエリを利用する事はできません。

次のセクションにある、より質素な構造を含む (**SELECT column1 FROM t1**) の例を見る時、自分自身のコードが、それよりもさらに多様で、複雑な構造を含んでいると想像してください。

2つテーブルを作成すると仮定します。

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

次に、**SELECT** を実行します。

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

2 の値を持つカラム **s1** を含む行が **t2** にあるので、その結果は **2** となります。

スカラ サブクエリは式の一部になり得ますが、もしそのサブクエリが関数に引数を与える演算子だとしても、括弧を忘れないでください。例:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

サブクエリを利用した比較

サブクエリの最もよく利用される方法はこの形の中にあります。

```
non_subquery_operand comparison_operator (subquery)
```

comparison_operator がこれらの演算子の1つであるところではこうです。

```
= > < >= <= <>
```

例:

```
... 'a' = (SELECT column1 FROM t1)
```

かつては、サブクエリが正当であるたった1つの場所は比較の右側でしたし、いまだにいくつかの古い DBMS がこれを主張しています。

ここに、接合と共に実行できない共通形サブクエリ比較の例があります。これは、テーブル **t2** の最大値と同等の、テーブル **t1** 内の全ての値を検出します。

```
SELECT column1 FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

ここに、テーブルの1つに対する凝集を含む為、接合と共に利用する事ができないもうひとつの例があります。これは、与えられたカラムの中で2回現れる値を含む、テーブル **t1** の中の全ての行を検出します。

```
SELECT * FROM t1 AS t
WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

これらの演算子の1つを利用して実行された比較には、**=** は行サブクエリと共に利用できるという例外がありますが、サブクエリはスカラを返さなければいけません。詳しくは [行サブクエリ](#) を参照してください。

ANY、IN、そして SOME を持つサブクエリ

構文:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```


比較演算子の後に続かなければいけない **ANY** キーワードは、「もしサブクエリが返すカラム内の値の **ANY** に対する比較が **TRUE** であれば、**TRUE** を返す」という事を意味します。例:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

テーブル (10) を含むテーブル **t1** 内に行があると仮定してください。10 以下である値 7 が **t2** の中にあるので、もしテーブル **t2** が (21,14,7) を含むなら、その式は **TRUE** です。もしテーブル **t2** が (20,10) を含むか、テーブル **t2** が空であれば、その式は **FALSE** です。もしテーブル **t2** が (NULL,NULL,NULL) を含むなら、その式は **UNKNOWN** です。

サブクエリと共に利用される時、**IN** という言葉は **= ANY** のエイリアスとなります。従って、これら2つのステートメントは同じになります。

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

式のリストと共に利用される時、**IN** と **= ANY** は同義語ではありません。**IN** は式のリストを取る事ができますが、**= ANY** はできません。詳しくは [比較関数と演算子](#) を参照してください。

NOT IN は **<> ANY** のエイリアスではありませんが、**<> ALL** のエイリアスです。詳しくは [ALL を持つサブクエリ](#) を参照してください。

SOME という言葉は **ANY** のエイリアスです。従って、これら2つのステートメントは同じになります。

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

SOME が利用される事はほとんどありませんが、なぜこれが役に立つのか、この例が表しています。通常、英語で「a はどの b とも同等ではない」と言うと、「a と等しい b は無い」と解釈されますが、SQL 構文ではそのような意味ではないのです。その構文は、「a と同等ではない b がいくつかある」と意味します。**<> SOME** を代わりに利用すると、そのクエリが本当に意味している事を全員がきちんと理解する助けになります。

ALL を持つサブクエリ

構文:

```
operand comparison_operator ALL (subquery)
```

比較演算子の後に続かなければいけない **ALL** という言葉は、「もしサブクエリが返すカラム内の値の **ALL** に対する比較が **TRUE** であれば、**TRUE** を返す」という事を意味します。例:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

テーブル (10) を含むテーブル **t1** 内に行があると仮定してください。10 は **t2** の中の3つの値全てよりも大きいので、もし **t2** が (-5,0,+5) を含むなら、その式は **TRUE** です。10 よりも大きい単一値 12 がテーブル **t2** にあるので、もしテーブル **t2** が (12,6,NULL,-100) を含むなら、その式は **FALSE** です。もしテーブル **t2** が (0,NULL,1) を含むなら、その式は 不明 (**NULL**) です。

最後に、もしテーブル **t2** が空なら、その結果は **TRUE** です。従って、テーブル **t2** の時、次のステートメントは **TRUE** です。

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

しかし、テーブル **t2** が空の時、このステートメントは **NULL** です。

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

そして、テーブル **t2** が空の時、次のステートメントは **NULL** です。

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

通常、**NULL** 値を含むテーブルと空のテーブルは「エッジ ケースです。」サブクエリ コードを書き込む時、必ずそれら2つの可能性を考慮したかどうか、確認してください。

NOT IN は **<> ALL** のエイリアスです。従って、これら2つのステートメントは同じになります。

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

行サブクエリ

この時点までの話は、単一値や値のカラムを返すサブクエリなどのような、スカラーやカラム サブクエリに関しての物でした。行サブクエリ は単列を返し、ひいては複数のカラム値を返す事ができるサブクエリ異型です。ここに2つ例があります。

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

もしテーブル `t2` が `column1 = 1` と `column2 = 2` の場所に行を持っていれば、ここにあるクエリは両方 `TRUE` です。

式 `(1,2)` と `ROW(1,2)` は時々 row constructors と呼ばれます。それら2つは同等の物です。それらは、別のコンテキストの中でも正当です。例えば、次の2つのステートメントは意味的に同等です。(1つ目は MySQL 5.1.12 まで最適化する事ができません)

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

行コンストラクタは通常、2つ以上のカラムを返すサブクエリを持つ比較に対して利用します。例えば、次のクエリは要求に答え、「テーブル `t2` にも存在する テーブル `t1` 内の全ての行を検出します。」

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
(SELECT column1,column2,column3 FROM t2);
```

EXISTS と NOT EXISTS

もしサブクエリが行を返せば、`EXISTS subquery` は `TRUE` で、`NOT EXISTS subquery` は `FALSE` です。例:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

もともと、`EXISTS` サブクエリは `SELECT *` で開始しますが、`SELECT 5` や `SELECT column1`、またそれ以外のどんな物でも開始する事ができます。MySQL はそのようなサブクエリの中では `SELECT` リストを無視するので、何も変わらないのです。

先ほどの例では、もし `t2` が `NULL` 値しか含まない物でも良いので、何かの行を含むのなら、`EXISTS` の条件は `TRUE` となります。`[NOT] EXISTS` サブクエリは通常相互関係を持つので、実際はこれはよくあるような例ではありません。ここに、もう少し現実的な例があります。

- 複数の町には、どんな種類のお店がありますか？

```
SELECT DISTINCT store_type FROM stores
WHERE EXISTS (SELECT * FROM cities_stores
WHERE cities_stores.store_type = stores.store_type);
```

- 町ではないところには、どんな種類のお店がありますか？

```
SELECT DISTINCT store_type FROM stores
WHERE NOT EXISTS (SELECT * FROM cities_stores
WHERE cities_stores.store_type = stores.store_type);
```

- 全ての町には、どんな種類のお店がありますか？

```
SELECT DISTINCT store_type FROM stores s1
WHERE NOT EXISTS (
SELECT * FROM cities WHERE NOT EXISTS (
SELECT * FROM cities_stores
WHERE cities_stores.city = cities.city
AND cities_stores.store_type = stores.store_type));
```

最後の例は、二重にネスト化された `NOT EXISTS` クエリです。それは、`NOT EXISTS` 条項の中に `NOT EXISTS` 条項を持っている、という事です。これは、正式に次の質問「`Stores` にはないお店がある町は存在しますか？」という質問に答えます。しかし、ネスト化した `NOT EXISTS` が、次の質問「`x` は全ての `y` に `TRUE` ですか？」に答える、と言う方が簡単です。

関連サブクエリ

関連サブクエリ は、外部クエリ内にも現れるテーブルの参照を含むサブクエリです。例:

```
SELECT * FROM t1 WHERE column1 = ANY
(SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

サブクエリの **FROM** 条項がテーブル **t1** に言及しなくても、サブクエリは **t1** のカラムへの参照を含むという事を覚えておいて下さい。ですので、MySQL はサブクエリの外側を見て、外部クエリ内の **t1** を見付けます。

テーブル **t1** が **column1 = 5** と **column2 = 6** の場所で行を含み、一方、テーブル **t2** は **column1 = 5** と **column2 = 7** の場所で行を含むと仮定してください。単純な式 ... **WHERE column1 = ANY (SELECT column1 FROM t2)** は **TRUE** となるでしょうが、例の中では、サブクエリ内の **WHERE** 条項は **FALSE** ですので、**((5,6)** が **(5,7)** と同等ではない為) このサブクエリ全体としては **FALSE** です。

スコープ ルール: MySQL は内側から外側まで評価します。例:

```
SELECT column1 FROM t1 AS x
WHERE x.column1 = (SELECT column1 FROM t2 AS x
WHERE x.column1 = (SELECT column1 FROM t3
WHERE x.column2 = t3.column1));
```

このステートメントの中では、**SELECT column1 FROM t2 AS x ...** が **t2** をリネームするので、**x.column2** はテーブル **t2** 内のカラムでなければいけません。**SELECT column1 FROM t1 ...** は とても遠くにある 外部クエリなので、これはテーブル **t1** 内のカラムではありません。

HAVING か **ORDER BY** 条項内のサブクエリに対しては、MySQL は外部選択リストからもカラム名を探します。

特定の場合には、関連サブクエリは最適化されます。例:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

そうでなければ、それらは役に立たず、スピードも遅くなりがちです。クエリを接合として書き換える事で、性能を向上させる事ができるかもしれません。

関連サブクエリは、外部クエリからの総計関数の結果を参照する事ができません。

FROM 条項内のサブクエリ

サブクエリは **SELECT** ステートメントの **FROM** 条項内で正当です。実際の構文はこれです。

```
SELECT ... FROM (subquery) [AS] name ...
```

[AS] name 条項は強制なので、**FROM** 条項内の全てのテーブルは名前を持つ必要があります。**subquery** 選択リスト内の全てのカラムは固有の名前を持たなければいけません。このマニュアルの中で「派生テーブル」という言葉が利用されている他の場所で、この構文の説明を見付ける事ができます。

説明する為に、このテーブルを持っていると仮定してください。

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

ここに、この例のテーブルを利用して、**FROM** 条項の中でサブクエリを利用する方法の説明があります。

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

結果: 2, '2', 4.0.

ここに別の例があります。グループ分けされたテーブルの、合計セットの平均を知りたいと仮定します。これは機能しません。

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

しかし、このクエリは要求された情報を提供します。

```
SELECT AVG(sum_column1)
```

```
FROM (SELECT SUM(column1) AS sum_column1
FROM t1 GROUP BY column1) AS t1;
```

サブクエリの中で利用されたカラム名(`sum_column1`) が外部クエリの中で認められている事に注意してください。

FROM 条項内のサブクエリは、スカラ、カラム、行、そしてテーブルを返す事ができます。**FROM** 条項内のサブクエリは、**JOIN** 演算の **ON** 条項内で利用されない限り、**FROM** 条項内のサブクエリになる事ができません。

FROM 条項内のサブクエリは、**EXPLAIN** ステートメント(派生テンポラリ テーブルが作られた) に対しても実行する事ができます。これは、上位レベルクエリが最適化の段階で全てのテーブルの情報を必要とする為に起こります。

サブクエリ エラー

これらはサブクエリにだけ適応するエラーです。このセクションでは、それらについて説明していきます。

- サポートされていないサブクエリ構文

```
ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
Message = "This version of MySQL does not yet support
'LIMIT & IN/ALL/ANY/SOME subquery'"
```

これは、次の形のステートメントはまだ機能しないという意味です。

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

- サブクエリからの不正カラム数

```
ERROR 1241 (ER_OPERAND_COL)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"
```

このエラーは、このような場合に起こります。

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

もし比較する事が目的であれば、複合カラムを帰すサブクエリを利用すると良いでしょう。詳しくは [行サブクエリ](#) を参照してください。しかし、別のコンテキスト内では、サブクエリはスカラ演算子でなければいけません。

- サブクエリからの不正行数

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

このエラーは、サブクエリが複数の行を返すステートメントで起こります。次の例を考えてみて下さい。

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

もし `SELECT column1 FROM t2` が行を1つだけ返せば、その前のクエリは機能します。もしサブクエリが複数の行を返せば、エラー 1242が起きます。その場合、クエリは次のように書き直されなければいけません。

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- サブクエリ内の不正使用されたテーブル

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

このエラーは次のような場合に起きます。

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

サブクエリは、**SELECT** ステートメント同様 **UPDATE** と **DELETE** ステートメント内で正当なので、**UPDATE** ステートメント内で、割り当ての為にサブクエリを利用する事ができます。しかし、同じテーブルを(この場合、テーブル `t1`)サブクエリの **FROM** 条項と更新ターゲットの両方に対して利用する事はできません。

トランザクション ストレージ エンジンに対しては、サブクエリの失敗は、ステートメント全体の失敗を引き起こします。非トランザクション ストレージ エンジンに対しては、エラーが起こる前に行われたデータ修正が保持されます。

最適化サブクエリ

開発が進行中なので、最適化についての情報は長期的に信頼性があります。次のリストは、利用してみたいいくつかの興味深いトリックを紹介しています。

- サブクエリ内の行数や行の順番に影響を与えるサブクエリ条項を利用してください。例:

```
SELECT * FROM t1 WHERE t1.column1 IN
(SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
(SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
(SELECT * FROM t2 LIMIT 1);
```

- サブクエリと接合を置き換えてください。例えば、次の例を

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
SELECT column1 FROM t2);
```

この次の物と置き換えます。

```
SELECT DISTINCT t1.column1 FROM t1, t2
WHERE t1.column1 = t2.column1;
```

- サブクエリをサポートしない古いバージョンの MySQL の互換性で、いくつかのサブクエリは接合に変形する事ができます。しかし、いくつかの場合で、サブクエリを接合に変換する事で性能を向上させる事ができます。詳しくは [MySQL 初期バージョンにおいて、サブクエリの接合としての書き換え](#) を参照してください。
- 条項をサブクエリの外から中に移動させて下さい。例えば、次の例を

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

この次のクエリの代わりに利用します。

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

別の例として、このクエリを、

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

この次のクエリの代わりに利用します。

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- 相関サブクエリの代わりに行サブクエリを利用してください。例えば、次の例を

```
SELECT * FROM t1
WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

この次のクエリの代わりに利用します。

```
SELECT * FROM t1
WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
AND t2.column2=t1.column2);
```

- [a <> ALL \(...\)](#) ではなく [NOT \(a = ANY \(...\)\)](#) を利用してください。
- [x=1 OR x=2](#) ではなく [x = ANY \(table containing \(1,2\)\)](#) を利用してください。
- [EXISTS](#) ではなく [= ANY](#) を利用してください。
- 必ず1つの行を返す相関サブクエリに対しては、[IN](#) は必ず [=](#) よりも遅いです。例えば、次の例を

```
SELECT * FROM t1 WHERE t1.col_name
```

```
= (SELECT a FROM t2 WHERE b = some_const);
```

この次のクエリの代わりに利用します。

```
SELECT * FROM t1 WHERE t1.col_name
IN (SELECT a FROM t2 WHERE b = some_const);
```

これらのトリックは、プログラムを早くしたり、遅くしたりする可能性があります。BENCHMARK() 関数のような MySQL 機能を利用すると、今の状況を改善する為のアイデアを見付ける事ができます。詳しくは [情報関数](#) を参照してください。

MySQL 自体が行う最適化のいくつかは次のような物です。

- MySQL は非相関サブクエリを一度だけ実行します。EXPLAIN を利用して、与えられたサブクエリが本当に非相関であるかどうかを確認してください。
- MySQL は、サブクエリ中の選択リスト カラムがインデックスされる可能性の利点を利用する為に IN、ALL、ANY、そして SOME サブクエリを書き換えます。
- MySQL は、EXPLAIN が特別な接合タイプ(unique_subquery または index_subquery)として表現する、インデックス検索機能を利用した次の形のサブクエリを置き換えます。

```
... IN (SELECT indexed_column FROM single_table ...)
```

- MySQL は、NULL 値か空のセットが関連していない限り、MIN() または MAX() を含む式を利用した次の形の式を強化します。

```
value {ALL|ANY|SOME} {> | < | >= | <=} (non-correlated subquery)
```

例えば、次の WHERE 条項ですが、

```
WHERE 5 > ALL (SELECT x FROM t)
```

オプチマイザによってこのように扱われるでしょう。

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

「MySQL がどのようにサブクエリを変形させるか」というタイトルの章が、MySQL 内部マニュアルの <http://dev.mysql.com/doc/> で参照可能です。

MySQL 初期バージョンにおいて、サブクエリの接合としての書き換え

MySQL の初期バージョンでは(MySQL 4.1以前)、INSERT ... SELECT ... と REPLACE ... SELECT ... の形のネスト化されたクエリだけがサポートされていました。MySQL 5.1 ではそうでないとしても、値のセットの中でメンバーシップをテストする別の方法があるという事も事実です。また、場合によっては、クエリをサブクエリ無しで再書き込みする事だけでなく、先ほどの別のテクニックを利用した方がより効果的であるというのも事実です。その1つが IN() コンストラクトです。

例えば、次の例は

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

次のように書き換えられます。

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

次のようなクエリは、

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

IN() を利用して次のように書き換えられます。

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id
WHERE table2.id IS NULL;
```

サーバの方が上手最適化できるかもしれないので、— MySQL サーバだけに特有であるというわけではない、LEFT [OUTER] JOIN はそれに同等のサブクエリよりも早いかもしれません。SQL-92 以前は外部接合が存在しな

かったので、特定の作業をする為には、サブクエリが唯一の方法でした。現在は、MySQL サーバやその他のデータベースシステムが様々なタイプの外部接合を提供しています。

MySQL サーバは、1つのテーブル、または複数のテーブルから一度に出される情報に基づき、行を効果的に削除する為に利用する事ができる複合テーブル **DELETE** ステートメントをサポートします。複合テーブル **UPDATE** ステートメントもまたサポートされています。

TRUNCATE 構文

```
TRUNCATE [TABLE] tbl_name
```

TRUNCATE TABLE はテーブルを完全に空にします。論理的には、これは全ての行を削除する **DELETE** ステートメントと同等ですが、いくつかの条件下では、違いがあります。

もしテーブルを参照する外部キー制約があれば、**InnoDB** テーブルに対しては、**TRUNCATE TABLE** が **DELETE** にマップされ、そうでなければ、高速切断(テーブルのドロップと再作成)が利用されます。外部キー制約の有無に関わらず、**AUTO_INCREMENT** カウンタが **TRUNCATE TABLE** によってリセットされます。

その他のストレージ エンジンに対しては、MySQL 5.1 の中では次のような方法で、**TRUNCATE TABLE** と **DELETE** が異なります。

- 切り捨て操作は、テーブルをドロップ、再作成します。それは、行を1つ1つ削除するよりも処理が速くできます。
- 切り捨て操作はトランザクション セーフではありませんので、実行中のトランザクションやテーブル ロックの途中で行おうとするとエラーが発生します。
- 削除された行数は返されません。
- テーブル フォーマット ファイル `tbl_name.frm` が有効である限り、データやインデックス ファイルが破損しても、テーブルは **TRUNCATE TABLE** を利用して空のテーブルとして再作成する事ができます。
- テーブル ハンドラは最後に利用された **AUTO_INCREMENT** 値を記憶していませんが、また最初から数えます。通常はシーケンス値を再利用しない **MyISAM** と **InnoDB** にも同じ事が言えます。
- パーティション テーブルと共に利用される時には、**TRUNCATE TABLE** はその分割を保管します。それは、パーティション定義(.par)ファイルは影響を受けませんが、データとインデックス ファイルはドロップされ、再作成されるという意味です。
- テーブルの切り捨てが **DELETE** を利用しない為、**TRUNCATE** ステートメントは **ON DELETE** トリガを呼び出しません。

TRUNCATE TABLE は、MySQL 5.1.16 以降から **DROP** 権限を必要とします。(5.1.16 以前では **DELETE** 権限を必要としています。

TRUNCATE TABLE は MySQL に導入されたオラクル SQL 拡張子です。

UPDATE 構文

単一テーブル構文:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2 ...]
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

複合テーブル構文:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
SET col_name1=expr1 [, col_name2=expr2 ...]
[WHERE where_condition]
```

単一テーブル構文には、**UPDATE** ステートメントは新しい値を利用して `tbl_name` 内に既存行のカラムを更新します。**SET** 条項は、どのカラムを変更し、それらにはどの値が与えられるべきかという事を指示します。もし **WHERE**

条項が与えられたら、それはどの行を更新するべきかを決定します。**WHERE** 条項が無ければ、全ての行が更新されます。もし **ORDER BY** 条項が指定されると、指定された順に行が更新されます。**LIMIT** 条項は、更新できる行数に制限を設定します。

複合テーブル構文には、**UPDATE** が、条件を満たす **table_references** で名づけられたそれぞれのテーブルの行を更新します。この場合、**ORDER BY** と **LIMIT** を利用する事はできません。

where_condition は更新される各行に対して正しい結果の式です。それは **SELECT 構文** で述べられている通りに指定されます。

UPDATE ステートメントは次の修飾因子をサポートします。

- もし **LOW_PRIORITY** キーワードを利用すると、別のクライアントがテーブルからの読み込みをしなくなるまで、**UPDATE** の実行が遅れます。
- もし **IGNORE** キーワードを利用すると、更新中にエラーが発生しても更新ステートメントは異常終了しません。複製キーの矛盾が起きた行は更新されません。データ変換エラーを起こす値にカラムが更新された行は、代わりに一番近い有効値に更新されます。

もし式の中で **tbl_name** からカラムにアクセスするなら、**UPDATE** はカラムの現在の値を利用します。例えば、次のステートメントは **age** カラムを現在の値よりも1大きく設定します。

```
UPDATE persondata SET age=age+1;
```

単一テーブル **UPDATE** 割り当ては通常左から右に評価されます。複合テーブルの更新に関しては、割り当てが特定の順番で行われるという保証はありません。

もし現在カラムが持つ値に設定するなら、MySQL はそれに気づくので更新はしません。

NULL に設定する事で **NOT NULL** を宣言されたカラムを更新すると、カラムはそのデータ タイプに適切なデフォルト値に設定され、警告カウントはインクリメントされます。数値タイプ、文字列タイプの空の文字列(""), そして日付と時刻タイプの「ゼロ」値のデフォルト値は **0** です。

UPDATE は実際に変更された行数を返します。**mysql_info()** C API 関数は、一致し更新された行数と、**UPDATE** の最中に起きた警告数を返します。

UPDATE の領域を制限する為に **LIMIT row_count** を利用する事ができます。**LIMIT** 条項は行に一致した制限です。ステートメントは、実際に変更されたかどうかに関わらず、**WHERE** 条項の条件を満たす **row_count** 行を見付けるとすぐに止まります。

もし **UPDATE** ステートメントが **ORDER BY** 条項を含むなら、行は条項に指示された順番で更新されます。これは、エラーが起こるかもしれない特定の場合に有効です。テーブル **t** が固有インデックスを持つカラム **id** を含むと仮定してください。次のステートメントは、行が更新される順番によって、複製キー エラーとなり失敗するかもしれません。

```
UPDATE t SET id = id + 1;
```

例えば、もしテーブルが **id** カラム内に1と2を含み、2が3に更新される前に1が2に更新されると、エラーが起きます。この問題を防ぐには、大きい **id** 値を持つ行が、小さい値を持つ行よりも先に更新されるように **ORDER BY** 条項を追加してください。

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

複合テーブルをカバーする **UPDATE** 演算を行う事もできます。しかし、複合テーブル **UPDATE** と共に **ORDER BY** や **LIMIT** を利用する事はできません。**table_references** 条項は接合箇所に含まれるテーブルをリストします。その構文は **JOIN 構文** で説明されています。ここに1つ例があります。

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

前出の例はカンマ演算子を利用する内部接合を表しますが、複合テーブルの **UPDATE** ステートメントは、**LEFT JOIN** のような、**SELECT** ステートメント内で許容される接合タイプを利用する事ができます。

実際に更新された複合テーブル **UPDATE** の中で参照されたカラムに対してだけ、**UPDATE** 権限が必要です。読み込みはされても、変更はされないカラムには、**SELECT** 権限だけが必要です。

外部キー制限があるテーブルに [InnoDB](#) テーブルを含む複合テーブル [UPDATE](#) ステートメントを利用すると、MySQL の最適マイザは、それらの親子関係の順番と違う順番でテーブルを処理するかもしれません。この場合、ステートメントは失敗し、ロールバックされます。代わりに、単一テーブルを更新し、他のテーブルが適宜修正されるように [InnoDB](#) が働きかける [ON UPDATE](#) 性能に頼ってください。

詳しくは [FOREIGN KEY 制約](#) を参照してください。

現在は、サブクエリの中で1つのテーブルを更新し、同じテーブルから選択する事はできません。

MySQL ユーティリティ ステートメント

DESCRIBE 構文

```
[DESCRIBE | DESC] tbl_name [col_name | wild]
```

[DESCRIBE](#) はテーブル内のカラムについて情報を提供します。これは、[SHOW COLUMNS FROM](#) へのショートカットです。これらのステートメントもまたビューの情報を表示します。(詳しくは [SHOW COLUMNS 構文](#) を参照してください。)

[col_name](#) は、カラム名、または SQL `'%'` を含む文字列、そして文字列と一致する名前を持つカラムにだけアウトプットを取得する `'_'` ワイルドカード文字になる事ができます。文字列がスペースやその他の特別な文字を含んでいない限り、それを引用句で囲む必要はありません。

```
mysql> DESCRIBE city;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Id    | int(11) | NO | PRI | NULL | auto_increment |
| Name  | char(35) | NO | | | |
| Country | char(3) | NO | UNI | | |
| District | char(20) | YES | MUL | | |
| Population | int(11) | NO | | 0 | |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

[Field](#) はカラム名を指示します。

[Null](#) フィールドは [NULL](#) 値がカラムの中に格納する事ができるかどうかを指示します。

[Key](#) フィールドはカラムがインデックスされるかどうかを指示します。[PRI](#) の値は、カラムがそのテーブルの主キーの一部であるかどうかを指示します。[UNI](#) はそのカラムが [UNIQUE](#) インデックスの一部である事を指示します。[MUL](#) 値は与えられた値のカラム内での複合発生が許容されている事を指示します。

[MUL](#) が [UNIQUE](#) インデックス上に表示されるひとつの理由は、いくつかのカラムが複合 [UNIQUE](#) インデックスを形成するという事です。カラムの組み合わせが固有であっても、それぞれのカラムは与えられた値の複合発生を保持する事ができます。複合インデックスの中では、インデックスの左端のカラムだけが [Key](#) フィールド内でエントリーを持つという事に注意してください。

[Default](#) フィールドは、カラムに割り当てられたデフォルト値を指示します。

[Extra](#) フィールドは与えられたカラムについて有効な追加情報を含んでいます。表示された例の中で、[Extra](#) フィールドは [Id](#) カラムが [AUTO_INCREMENT](#) キーワードを利用して作成されたという事を指示しています。

[CREATE TABLE](#) ステートメントに基づいていると思っていた物とデータ タイプがもし異なっていたら、MySQL はデータ タイプを変更する事があるという事に注意してください。詳しくは [サイレント カラム仕様変更](#) を参照してください。

[DESCRIBE](#) ステートメントにはオラクルの互換性が提供されています。

[SHOW CREATE TABLE](#) と [SHOW TABLE STATUS](#) ステートメントもテーブルについての情報を提供します。詳しくは [SHOW 構文](#) を参照してください。

HELP 構文

HELP 'search_string'

HELP ステートメントは MySQL リファレンス マニュアルからオンライン情報を返します。その正しい操作の為に [mysql](#) データベース内のヘルプ テーブルが、ヘルプ トピック情報で初期化する必要があります。(サーバ サイド ヘルプ を参照してください。)

HELP ステートメントは、ヘルプ テーブルの中で与えられた検索文字列を検索し、その結果を表示します。検索文字列は、大文字と小文字を区別しません。

HELP ステートメントはいくつかの検索文字列タイプを理解します。

- 一般的なレベルでは、最高レベルのヘルプ カテゴリのリストを検索する為に [contents](#) を利用してください。

HELP 'contents'

- [Data Types](#) のような与えられたヘルプカテゴリ内のトピック リストには、カテゴリ名を利用してください。

HELP 'data types'

- [ASCII\(\)](#) 関数や [CREATE TABLE](#) ステートメントのような特定のヘルプ トピックには、関連キーワードを利用してください。

HELP 'ascii'

HELP 'create table'

言い換えると、検索文字列はカテゴリ、多くのトピック、または単一トピックに一致するという事です。あらかじめ、与えられた検索文字列が、項目リストや単一ヘルプ トピックのヘルプ情報を返すかどうかを知る事はできません。しかし、結果セットの中のカラムと行数を調べる事で、[HELP](#) がどのような答えを返したのかを知る事ができます。

次の説明は結果セットが取る事ができる形を指示します。ステートメント例のアウトプットは、[mysql](#) クライアントを利用する時に見る事ができる為になじみのある「tabular」や「vertical」フォーマットを利用して表示されますが、[mysql](#) 自体は [HELP](#) 結果セットを違う方法で再フォーマットする事に注意してください。

- 空の結果セット

検索文字列に一致する物が見つかりませんでした。

- 3つのカラムを持つ単列を含む結果セット

これは、検索文字列がヘルプ トピックにヒットをもたらしたという意味です。結果は3つのカラムを持っています。

- [name](#): トピック名。
- [description](#): トピックに対しての記述的なヘルプテキスト。
- [example](#): 使用法例。このカラムは空白である可能性があります。

例: [HELP 'replace'](#)

下記をもたらします。

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

- 2つのカラムを持つ複合行を含む結果セット

これは、検索文字列がたくさんのヘルプ トピックに一致したという意味です。結果セットはヘルプ トピック名を指示します。

- **name**: ヘルプ トピック名
- **is_it_category**: もしその名前がヘルプ カテゴリを表していれば **Y**、もしそうでなければ **N**。もしそうでなければ、**HELP** ステートメントへの引数として指定された時の **name** 値は、名づけられた項目に対する説明を含む単列結果セットをもたらす必要があります。

例: **HELP 'status'**

下記をもたらします。

name	is_it_category
SHOW	N
SHOW ENGINE	N
SHOW INNODB STATUS	N
SHOW MASTER STATUS	N
SHOW PROCEDURE STATUS	N
SHOW SLAVE STATUS	N
SHOW STATUS	N
SHOW TABLE STATUS	N

- 3つのカラムを持つ複合行を含む結果セット

これは、検索文字列がカテゴリに一致するという事を意味します。結果セットはカテゴリ入力を含んでいます。

- **source_category_name**: ヘルプカテゴリ名。
- **name**: カテゴリまたはトピック名
- **is_it_category**: もしその名前がヘルプ カテゴリを表していれば **Y**、もしそうでなければ **N**。もしそうでなければ、**HELP** ステートメントへの引数として指定された時の **name** 値は、名づけられた項目に対する説明を含む単列結果セットをもたらす必要があります。

例: **HELP 'functions'**

下記をもたらします。

source_category_name	name	is_it_category
Functions	CREATE FUNCTION	N
Functions	DROP FUNCTION	N
Functions	Bit Functions	Y
Functions	Comparison operators	Y
Functions	Control flow functions	Y
Functions	Date and Time Functions	Y
Functions	Encryption Functions	Y
Functions	Information Functions	Y
Functions	Logical operators	Y
Functions	Miscellaneous Functions	Y
Functions	Numeric Functions	Y
Functions	String Functions	Y

USE 構文

USE db_name

USE db_name ステートメントは、MySQL に対して、後に続くステートメントのデフォルト データベースとして **db_name** データベースを利用するように指示します。そのデータベースは、そのセッションが終わるまで、または別の **USE** ステートメントが発行されるまでデフォルトのままです。

```
USE db1;
SELECT COUNT(*) FROM mytable; # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable; # selects from db2.mytable
```

[USE](#) ステートメントを利用して特定のデータベースをデフォルトにする事によって、別のデータベースの中でテーブルにアクセスする時に邪魔をする事はありません。次の例は、[db1](#) データベースから [author](#) テーブルへ、そして [db2](#) データベースから [editor](#) テーブルへアクセスします。

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
WHERE author.editor_id = db2.editor.editor_id;
```

[USE](#) ステートメントには Sybase の互換性が提供されています。

MySQL トランザクションとロッキング関連のステートメント

MySQL は [SET AUTOCOMMIT](#)、[START TRANSACTION](#)、[COMMIT](#)、そして [ROLLBACK](#) のようなステートメントを通してローカル トランザクション(与えられたクライアント接続内の)をサポートします。詳しくは [START TRANSACTION](#)、[COMMIT](#)、そして [ROLLBACK 構文](#) を参照してください。XA トランザクション サポートは、MySQL が分散型のトランザクションにも参加できるように働きかけます。詳しくは [XA トランザクション](#) を参照してください。

START TRANSACTION、COMMIT、そして ROLLBACK 構文

```
START TRANSACTION | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET AUTOCOMMIT = {0 | 1}
```

[START TRANSACTION](#) と [BEGIN](#) ステートメントは新しいトランザクションを始めます。[COMMIT](#) は、その変更を恒久的な物にし、現在のトランザクションを行います。[ROLLBACK](#) はその変更をキャンセルし、現在のトランザクションをロールバックします。[SET AUTOCOMMIT](#) ステートメントは現在の接続にデフォルト自動コミット モードを無効に、そして有効にします。

任意の [WORK](#) キーワードは、[CHAIN](#) と [RELEASE](#) 条項のように [COMMIT](#) と [ROLLBACK](#) に対してサポートされています。[CHAIN](#) と [RELEASE](#) はトランザクション完了に対する追加コントロールに利用されます。[completion_type](#) システム変数の値はデフォルトの完了動作を決定します。詳しくは [システム変数](#) を参照してください。

[AND CHAIN](#) 条項は、現在のトランザクションが終わったらすぐに新しいトランザクションが始まるよう働きかけ、その新しいトランザクションは終わったばかりのトランザクションと同じ分離レベルを持ちます。[RELEASE](#) 条項は、現在のトランザクションを終了した後、サーバが現在のクライアント接続を切るよう働きかけます。[NO](#) キーワードを含む事は、もし [completion_type](#) システム変数がデフォルトで変更やリリース完了を引き起こすよう設定されれば有効となる、[CHAIN](#) や [RELEASE](#) 完了を食い止めます。

デフォルトにより、MySQL は自動コミットモードが有効な状態で起動します。これは、テーブルを更新 (変更) するステートメントを実行したとたん、MySQL がディスク上に更新を格納する、という意味です。

もしトランザクション セーフ ストレージ エンジン([InnoDB](#)や [NDB Cluster](#) のような物)を利用していたら、次のステートメントを利用して自動コミットモードを無効にする事ができます。

```
SET AUTOCOMMIT=0;
```

[AUTOCOMMIT](#) 変数をゼロに設定する事で自動コミット モードを無効にした後、ディスクに変更を格納する為には [COMMIT](#) を、またはもしそのトランザクションの最初から行ってきた変更を無視したければ [ROLLBACK](#) を利用しなければいけません。

一連のステートメントに対して自動コミット モードを無効にする為には、[START TRANSACTION](#) ステートメントを利用してください。

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

[START TRANSACTION](#) を利用すると、自動コミットは [COMMIT](#) や [ROLLBACK](#) を利用してトランザクションを終了するまで無効のままです。そして自動コミット モードはその前の状態に戻ります。

BEGIN と BEGIN WORK は、トランザクションを始める START TRANSACTION のエイリアスとしてサポートされています。START TRANSACTION はスタンダード SQL 構文で、アドホック トランザクションを始める方法として推奨されています。

重要:MySQL クライアント アプリケーションを書く為に利用されている多くの API (JDBC のような)は、トランザクションを始める為に、クライアントから START TRANSACTION ステートメントを送る代わりに利用する事ができる (場合によっては利用しなければならない) 独自の方法を提供します。更なる情報については、[23章APIとライブラリー](#) か、ご利用の API の説明書を参照してください。

BEGIN ステートメントは、BEGIN ... END 複合ステートメントを開始する BEGIN キーワードの利用とは異なります。後者はトランザクションを開始しません。詳しくは [BEGIN ... END 複合ステートメント構文](#) を参照してください。

このようにトランザクションを開始する事もできます。

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

WITH CONSISTENT SNAPSHOT 条項は、一貫性のある読み取りが可能であるストレージ エンジンに対して、それを開始します。現在は、これは InnoDB にだけ適応しています。その効果は、InnoDB テーブルから SELECT が後に続く START TRANSACTION を発行することと同じです。詳しくは [一貫非ロック読み取り](#) を参照してください。

WITH CONSISTENT SNAPSHOT 条項は現在のトランザクションの分離レベルを変更しないので、現在の分離レベルが一貫性のある読み取りを許容する物である場合のみ(REPEATABLE READ か SERIALIZABLE)、一貫性のあるスナップショットを提供します。

トランザクションを開始すると、未解決のトランザクションを引き起こします。詳細については、[暗黙のコミットを引き起こすステートメント](#) をご参照ください。

トランザクションを開始すると、LOCK TABLES を利用して行ったテーブル ロックを、まるで UNLOCK TABLES を実行したかのように解除してしまいます。トランザクションを開始しても、FLUSH TABLES WITH READ LOCK を利用して行われたグローバル リード ロックの解除はしません。

一番良い結果を得る為には、単一トランザクショナル ストレージ エンジンによって管理されたテーブルだけを利用してトランザクションを行うべきです。そうでなければ、次のような問題が起きます。

- もし複数のトランザクション セーフ ストレージ エンジンのテーブルを利用して(InnoDB や Falcon のような)、またトランザクションの分離レベルが SERIALIZABLE でないならば、1つのトランザクションが行われた時に同じテーブルを利用する別の実行中のトランザクションが、最初のトランザクションによって行われた変更だけを見るという事が可能です。これは、トランザクションのアトミック性が、混合エンジンに保障されておらず、矛盾した結果になり得るという事です。(もし混合エンジン トランザクションが頻繁でなければ、必要に応じて分離レベルを SERIALIZABLE に設定する為に、1つのトランザクション単位で SET TRANSACTION ISOLATION LEVEL を利用する事ができます。)
- もし非トランザクション セーフ テーブルをトランザクション内で利用すれば、自動コミットの状態に関わらず、それらのテーブルへの変更は一度に格納されます。

もし非トランザクション テーブルをトランザクション内で更新した後に ROLLBACK ステートメントを発行すると、ER_WARNING_NOT_COMPLETE_ROLLBACK 警告が発生します。トランザクション セーフ テーブルへの変更はロールバックされますが、非トランザクション セーフ テーブルへの変更はされません。

各トランザクションは COMMIT によって、1つの塊でバイナリ ログの中に格納されています。ロールバックされたトランザクションはログされません。(例外:非トランザクション テーブルへの変更はロールバックされません。もしロールバックされたトランザクションが、非トランザクション テーブルへの変更を含んでいたら、それらのテーブルへの変更が複製される事を保障する為に、最後にそのトランザクション全体が ROLLBACK ステートメントでログされます。)詳しくは [バイナリ ログ](#) を参照してください。

SET TRANSACTION ISOLATION LEVEL を利用してトランザクションの分離レベルを変更する事ができます。詳しくは [SET TRANSACTION 構文](#) を参照してください。

ロールバックは、ユーザーが明示的に求めなくても行われる、ゆっくりとした操作になり得ます。(例えば、エラーが発生した時。)この為、SHOW PROCESSLIST は、明示的、暗黙的な(ROLLBACK SQL ステートメント)ロールバックの最中の接続に対する State カラム内の Rolling back を表示します。

ロールバックできないステートメント

いくつかのステートメントはロールバックできません。通常、それらはデータベースを作成したりドロップしたりする物や、テーブルやストアドルーチンを作成、ドロップ、変更する物のような、データ定義言語(DDL)ステートメントを含みます。

そのような物をトランザクション内に含まないようにデザインする必要があります。もしロールバックできないトランザクション内で、早いうちにステートメントを発行し、そして別のステートメントがその後失敗すると、そのような場合 **ROLLBACK** ステートメントを発行する事によってそのトランザクション全体の効果をロールバックする事はできません。

暗黙のコミットを引き起こすステートメント

次の各ステートメント(そしてそれらの同義語)は、まるでステートメントを実行する前に **COMMIT** を行ったかのように、暗黙にトランザクションを終了します。

- **ALTER FUNCTION**, **ALTER PROCEDURE**, **ALTER TABLE**, **BEGIN**, **CREATE DATABASE**, **CREATE FUNCTION**, **CREATE INDEX**, **CREATE PROCEDURE**, **CREATE TABLE**, **DROP DATABASE**, **DROP FUNCTION**, **DROP INDEX**, **DROP PROCEDURE**, **DROP TABLE**, **LOAD DATA INFILE** **LOCK TABLES**, **RENAME TABLE**, **SET AUTOCOMMIT=1**, **START TRANSACTION**, **TRUNCATE TABLE**, **UNLOCK TABLES**.
- MySQL 5.1.3 から、**ALTER VIEW**、**CREATE TRIGGER**、**CREATE USER**、**CREATE VIEW**、**DROP TRIGGER**、**DROP USER**、**DROP VIEW**、そして **RENAME USER** ステートメントは暗黙的なコミットを引き起こすようになりました。
- **UNLOCK TABLES** は、もしテーブルが現在 **LOCK TABLES** でロックされていたらトランザクションを行います。これは、**FLUSH TABLES WITH READ LOCK** ステートメントがテーブルレベル ロックを取得しない為、これに続く **UNLOCK TABLES** に対しては行われません。
- InnoDB 内の **CREATE TABLE** ステートメントは単一トランザクションとして生成されます。これは、ユーザからの **ROLLBACK** はトランザクションの最中にユーザが作成した **CREATE TABLE** ステートメントを解除しないという意味です。
- **CREATE TABLE** と **DROP TABLE** は、もし **TEMPORARY** キーワードが利用されたらトランザクションを実行しません。(これは、コミットを引き起こさない **CREATE INDEX** のようなテンポラリ テーブルへのその他の操作には当てはまりません。)
- MySQL 5.1.15 から、**CREATE TABLE ... SELECT** はステートメントが実行される前後に暗黙的なコミットを引き起こすようになりました。しかし、**CREATE TEMPORARY TABLE ... SELECT** には何のコミットも起きません。
- MySQL 5.1.11 以前では、**LOAD DATA INFILE** は全てのストレージ エンジンに対して暗黙的なコミットを引き起こしました。MySQL 5.1.12 からは、**NDB** ストレージエンジンを利用しているテーブルに対してだけ暗黙的なコミットを引き起こすようになりました。更なる情報については、バグ #11151を参照してください。
- MySQL 5.1.15 以降では、非テンポラリ テーブルを作成している時 **CREATE TABLE ... SELECT** ステートメントは暗黙的なコミットを含みます。これは、ロールバック後にマスタ上でテーブルを作成する事ができてバイナリログ内での記録に失敗する為スレーブには複製されない、という複製の最中での発行を防ぐ為の物です。更なる情報については、バグ #22865を参照してください。

トランザクションはネスト化されません。これは、**START TRANSACTION** ステートメントかその同義語の1つを発行する時点でのトランザクションに対して実行される、暗黙的な **COMMIT** の結果です。

トランザクションが **ACTIVE** 状態の間は、暗黙的な作業を引き起こすステートメントは XA トランザクションでは利用する事はできません。

SAVEPOINT と ROLLBACK TO SAVEPOINT 構文

```
SAVEPOINT identifier
ROLLBACK [WORK] TO SAVEPOINT identifier
RELEASE SAVEPOINT identifier
```

InnoDB は SQL ステートメント **SAVEPOINT**、**ROLLBACK TO SAVEPOINT**、**RELEASE SAVEPOINT** そして **ROLLBACK** の任意 **WORK** キーワードをサポートします。

SAVEPOINT ステートメントは **identifier** の名前を利用して、名前が付けられたトランザクション セーブポイントを設定します。もし現在のトランザクションが同名のセーブポイントを持っていれば、古いセーブポイントは削除され新しいものが設定されます。

ROLLBACK TO SAVEPOINT ステートメントは、名づけられたセーブポイントにトランザクションをロールバックします。セーブポイントの設定後に行に対して現在のトランザクションが行った変更はロールバック内で解除されますが、**InnoDB** はセーブポイントの後にメモリに格納された新しい行ロックのリリースは行いません。(新しく挿入された行には、その行内に格納されているトランザクション ID によってロック情報が保持されるので、そのロックはメモリの中で別々に格納される訳ではないという事に注意してください。この場合、行ロックは取り消しの中でリリースされます。)名づけられたセーブポイントよりも後で設定されたセーブポイントは削除されます。

もし **ROLLBACK TO SAVEPOINT** ステートメントが次のエラーを返したら、指定された名前のセーブポイントは存在しないという事を意味します。

```
ERROR 1181: Got error 153 during ROLLBACK
```

RELEASE SAVEPOINT ステートメントは、現在のトランザクションのセーブポイント セットから、名づけられたセーブポイントを除外します。コミットもロールバックも起きません。もしセーブポイントが存在しなければ、これはエラーになります。

もし **COMMIT** や、セーブポイントに名前をつけない **ROLLBACK** を実行すると、現在のトランザクションの全てのセーブポイントは削除されます。

MySQL 5.0.17 から、ストアド ファンクションが呼び出されたりトリガが作動した時に、新しいセーブポイント レベルが作成されます。以前のレベルのセーブポイントは無効になる為、新しいレベルとの間でセーブポイントの衝突が起きません。ファンクションやトリガが終了する時に、それらが作成したセーブポイントはリリースされ、以前のセーブポイント レベルが保持されます。

LOCK TABLES と UNLOCK TABLES 構文

```
LOCK TABLES
tbl_name [AS alias]
{READ [LOCAL] | [LOW_PRIORITY] WRITE}
[, tbl_name [AS alias]
{READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
UNLOCK TABLES
```

LOCK TABLES は、現在のスレッドに対してベース テーブル (ビュー以外) をロックします。もしテーブルが1つでも他のスレッドによってロックされていたら、それは全てのロックを入手するまでブロックします。

UNLOCK TABLES は現在のスレッドによって行われたロックを全て明示的にリリースします。スレッドが別の **LOCK TABLES** を発行した時、またはサーバへの接続が閉じられた時に、現在のスレッドにロックされている全てのテーブルは暗黙的にロック解除されます。**UNLOCK TABLES** はまた、**FLUSH TABLES WITH READ LOCK** を利用してグローバル リード ロックを取得した後に、そのロックをリリースする為に利用されます。(FLUSH TABLES WITH READ LOCK ステートメントによってリード ロックされている全てのデータベース内の全てのテーブルをロックすることができます。詳しくは [FLUSH 構文](#) を参照してください。これは、もし Veritas のような時間内にスナップショットを撮る事ができるファイルシステムを持っていると、バックアップを取るのが大変便利な方法になります。)

LOCK TABLES を利用する為には、関連するテーブルに対して **LOCK TABLES** 権限と **SELECT** 権限を持つ必要があります。

LOCK TABLES を利用する主な理由は、テーブルを更新する際にトランザクションをエミュレートしたり、スピードを早くしたりする事です。後ほどもう少し詳しく説明します。

テーブル ロックは、別のクライアントによる不適切な読み込みや書き込みに対してのみ保護します。ロックを保持しているクライアントは、それがリード ロックだとしても、**DROP TABLE** のようなテーブル レベル操作を行う事ができます。切り捨て操作はトランザクション セーフではないので、もし、アクティブなトランザクションの実行中や、テーブル ロックを保持している最中にクライアントがそれを行おうとすると、エラーが発生します。

LOCK TABLES とトランザクション テーブルを共に利用する際には、次の事に注意してください。

- **LOCK TABLES** はトランザクション セーフではないので、テーブルをロックしようとする前に、全てのアクティブなトランザクションを暗黙的にコミットします。また、トランザクションの開始をする(例えば **START**

TRANSACTION を利用して)、明示的に **UNLOCK TABLES** を実行します。(詳しくは [暗黙のコミットを引き起こすステートメント](#) を参照してください。)

- **LOCK TABLES** を InnoDB のようなトランザクション テーブルと共に利用する正しい方法は、**AUTOCOMMIT = 0** を設定し、トランザクションを明示的にコミットするまでは **UNLOCK TABLES** をコールしないという方法です。**LOCK TABLES** をコールする時、InnoDB は内部的にそれ自身のテーブル ロックを取り、そして MySQL もそれ自身のテーブル ロックを取ります。InnoDB は次のコミットでテーブル ロックを解除しますが、MySQL がそのテーブル ロックを解除するには、**UNLOCK TABLES** をコールする必要があります。InnoDB は **LOCK TABLES** のコールの後そのテーブル ロックを即座にリリースし、その為にデッドロックが簡単に起きてしまうので、**AUTOCOMMIT = 1** を持つべきでは有りません。もし **AUTOCOMMIT=1** であれば、古いアプリケーションの不必要なデッドロックを防ぐ為に、InnoDB テーブル ロックを全く取得しないという事に注意してください。
- **ROLLBACK** は MySQL の非トランザクション テーブル ロックを解除しません。
- **FLUSH TABLES WITH READ LOCK** は、グローバル リード ロックは取得しますがテーブル ロックはしないので、テーブル ロックと暗黙的なコミットに関しては **LOCK TABLES** と **UNLOCK TABLES** と同じような動作の制約は受けません。詳しくは [FLUSH 構文](#) を参照してください。

LOCK TABLES を利用する時、ステートメントの中で利用する予定の全てのテーブルをロックしなければいけません。**LOCK TABLES** はビューをロックしないので、もし今実行している操作がビューを利用するなら、それらのビューが依存する全てのベース テーブルもロックしなければいけません。**LOCK TABLES** ステートメントで取得したロックが有効な間は、そのステートメントによってロックされていないテーブルにアクセスする事はできません。また、単一エリの中でロックされたテーブルを複数回使用する事はできません。各エイリアスに対して別々にロックを取得する必要がある場合には、代わりにエイリアスを利用してください。

```
mysql> LOCK TABLE t WRITE, t AS t1 WRITE;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

もしステートメントがエイリアスを利用してテーブルを参照するなら、同じエイリアスを利用してテーブルをロックする必要があります。エイリアスを指定しないでテーブルをロックする事はできません。

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

反対に、もしエイリアスを利用してテーブルをロックすると、そのエイリアスを利用してステートメント内でそのテーブルを参照する必要があります。

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

もしスレッドがテーブル上で **READ** ロックを取得したら、そのスレッド(そしてそれ以外の全てのスレッド)はテーブルからは読み込む事ができません。もしスレッドがテーブル上で **WRITE** ロックを取得したら、そのロックを保持するスレッドのみがそのテーブルに書き込みができます。ロックが解除されるまで、その他のスレッドはテーブルの読み込み、書き込みからブロックされます。

READ LOCAL と **READ** の違いは、**READ LOCAL** は、ロックされている間に非対立 **INSERT** ステートメント(並列挿入)が実行される事を許容するという事です。しかし、もしロックを保持している間に MySQL の外部でデータベース ファイルを複製しようとする、これを利用する事はできません。InnoDB テーブルに対しては、**READ LOCAL** は **READ** と同じです。

WRITE ロックは通常、更新がなるべく速く行われるよう、**READ** ロックよりも高い優先順位を持ちます。これは、もし1つのスレッドが **READ** ロックを取得し、そして別のスレッドが **WRITE** ロックをリクエストすると、後続の **READ** ロックリクエストは、**WRITE** スレッドがロックを得て、それをリリースするまで待つ、という意味になります。**WRITE** ロックを待っているスレッドの前に、別のスレッドが **READ** ロックを取得するのを許容する為に、**LOW_PRIORITY WRITE** を利用する事ができます。**READ** ロックを持つスレッドが無い時に、時間が残っている事が確実である時だけ、**LOW_PRIORITY WRITE** ロックを利用しなければいけません。(例外:トランザクション モード(自動コミット = 0)の InnoDB テーブルに対しては、**LOW_PRIORITY WRITE** ロックは通常の **WRITE** ロックのように機能し、待機中の **READ** ロックに先行します。)

LOCK TABLES は次のように機能します。

1. ロックされる全てのテーブルを内部定義順に並べ替えます。ユーザの立場からは、この順番は定義されていません。
2. もしそのテーブルが読み込みと書き込みの両方についてロックされるなら、読み込みロックの前に書き込みロックを行ってください。
3. スレッドが全てのロックを得るまで、テーブル1つずつにロックをして下さい。

この方法は、テーブル ロックにデッドロックが起きない事を保証します。:しかし、この方法について知っておかなければいけない事があります。もしテーブルに対して [LOW_PRIORITY WRITE](#) ロックを利用していたら、MySQL は [READ](#) ロックを必要とするスレッドがなくなるまで、この特定のロックを待つ、という意味になります。スレッドが [WRITE](#) ロックを得て、ロックテーブル リストの中の次のテーブルのロックを得るのを待っている時、他の全てのスレッドは [WRITE](#) ロックが解除されるのを待ちます。もしこれが、ご利用のアプリケーションにとって深刻な問題となってしまうたら、いくつかのテーブルをトランザクション セーフ テーブルに変換する事を考慮するべきです。

テーブル ロックを待っているスレッドを終了させる為に、[KILL](#) を安全に使用する事ができます。詳しくは [KILL 構文](#) を参照してください。

[INSERT](#) が別のスレッドによって実行されてしまう為、[INSERT DELAYED](#) と共に利用しているテーブルをロックしてはいけない という事に注意してください。

通常、全ての単一 [UPDATE](#) ステートメントはアトミックなので、テーブルをロックする必要はありません。別のスレッドが現在実行中の SQL ステートメントの邪魔をする事はできません。しかし、テーブルをロックする事が利益をもたらす場合もいくつかあります。

- [MyISAM](#) テーブル セット上でたくさんの操作を行おうとしているのであれば、利用する予定のテーブルをロックしたほうが操作が速くできます。[UNLOCK TABLES](#) がコールされるまで、MySQL はロックされたテーブルのキー キャッシュをフラッシュしないので、[MyISAM](#) テーブルをロックすると、挿入、更新、削除のスピードを速くします。通常、キー キャッシュは各 SQL ステートメントの後でフラッシュされます。

テーブルをロックする事のマイナス面は、[READ](#) ロックされたテーブル(ロックを保持している物も含む)を更新できるスレッドが無く、またロックを保持しているテーブル以外は [WRITE](#) ロックされたテーブルにアクセスできるスレッドが無いという事です。

- 非トランザクション ストレージ エンジンに対してテーブルを利用している場合に、もし [SELECT](#) と [UPDATE](#) の間に別のスレッドがテーブルを変更しない事を保証したければ、[LOCK TABLES](#) を利用する必要があります。ここに表されている例は、安全に実行する為に [LOCK TABLES](#) を必要とします。

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

[LOCK TABLES](#) 無しで、別のスレッドが [SELECT](#) と [UPDATE](#) ステートメントの実行の間に、[trans](#) テーブル内に新しい行を挿入する事が可能です。

多くの場合、相対更新([UPDATE customer SET value=value+new_value](#))または [LAST_INSERT_ID\(\)](#) 関数を利用する事で、[LOCK TABLES](#) の利用を避ける事ができます。詳しくは [トランザクションとアトミックオペレーション](#) を参照してください。

ユーザ レベルの通知ロック機能 [GET_LOCK\(\)](#) と [RELEASE_LOCK\(\)](#) を利用する事で、テーブルのロックを避ける事ができる場合があります。これらのロックはサーバ内のハッシュ テーブルの中に保存され、スピードを速くする目的で [pthread_mutex_lock\(\)](#) と [pthread_mutex_unlock\(\)](#) を利用して実施されます。詳しくは [その他の関数](#) を参照してください。

ロックの規定に関しての更なる情報は [MySQL のテーブルロック方法](#) を参照してください。

注意:もしロックされたテーブル上に [ALTER TABLE](#) を利用すると、ロックが解除される可能性があります。詳しくは [Problems with ALTER TABLE](#) を参照してください。

SET TRANSACTION 構文

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

このステートメントは、次のトランザクションにグローバルに分離レベルを設定したり、現在のセッションに分離レベルを設定したりします。

SET TRANSACTION のデフォルト動作は、次の（まだ始まっていない）トランザクションの分離レベルを設定する事です。もし **GLOBAL** キーワードを利用すると、そのステートメントはその時点以降に作成された全ての新しい接続に対して、デフォルトのトランザクション レベルをグローバルに設定します。既存接続には影響はありません。これには **SUPER** 権限が必要です。**SESSION** キーワードの利用は、現在の接続上で今後行われる全てのトランザクションに対して、デフォルト トランザクション レベルを設定します。

各 InnoDB トランザクション分離レベルの説明に関しては、**InnoDB と TRANSACTION ISOLATION LEVEL** を参照してください。InnoDB は MySQL 5.1 内でこれらそれぞれのレベルをサポートします。デフォルト レベルは **REPEATABLE READ** です。

mysqld に冒頭のデフォルト グローバル分離レベルを設定する為には、**--transaction-isolation** オプションを利用してください。詳しくは **コマンド オプション** を参照してください。

XA トランザクション

InnoDB ストレージ エンジンでは、XA トランザクションがサポートされています。MySQL XA インプリメンテーションは X/Open CAE ドキュメント Distributed Transaction Processing:XA 仕様 に基づいています。この文書はオープン グループによって出版されていて、<http://www.opengroup.org/public/pubs/catalog/c193.htm> で入手可能です。現在の XA インプリメンテーションの制限については、**XA トランザクションの規制** で説明されています。

クライアント側には、特別な事は何も要求されません。MySQL サーバへの XA インターフェースは、**XA** キーワードで始まる SQL ステートメントで構成されています。MySQL クライアント プログラムは、SQL ステートメントを送り、XA ステートメント インターフェースの動作を理解する事が可能である必要があります。それらは最近のクライアント ライブラリに違反してリンクされる必要はありません。古いクライアント ライブラリもまた機能します。

現在、MySQL コネクタの中では、MySQL コネクタ/J 5.0.0 が XA ディレクトリをサポートします。(XA SQL ステートメント インターフェースを扱うクラス インターフェースを利用して)

XA は分散型のトランザクションをサポートします。それは、複数の別々のトランザクション リソースを、グローバル トランザクションに参加させる事ができる能力です。トランザクション リソースは RDBMS である事が多いのですが、別の種類である可能性もあります。

グローバル トランザクションは、それら自身の中でトランザクションのアクションをいくつか含んでいますが、それらは全て、グループとして成功するか、グループとしてロールバックされる必要があります。要するに、これは複数の ACID トランザクションが、ACID 特性を持つグローバル操作のコンポーネントとして一斉に実行されるように ACID 特性を「1レベル上げる」という事です。(しかし、分散型のトランザクションには、ACID 特性を達成する為に、**SERIALIZABLE** 分離レベルを利用しなければいけません。非分散型のトランザクションには **REPEATABLE READ** の利用で十分ですが、分散型のトランザクションに対しては十分では有りません。)

分散型トランザクションのいくつかの例

- アプリケーションは、メッセージ サービスと RDBMS を合体させる統合ツールとして機能します。アプリケーションは、メッセージの送信、検索、さらにトランザクションのデータベースも巻き込んで処理を行うトランザクションが、全て確実にグローバル トランザクションの中で行われるようにします。これを「トランザクションの電子メール」だと考える事ができます。
- アプリケーションは、複数のサーバを含むアクションが、各サーバに対する別々のローカル トランザクションではなく、グローバル トランザクションの一部として発生する、MySQL サーバやオラクル サーバ(または複合 MySQL サーバ)のような、異なるデータベースを含むアクションを実行します。
- 銀行は、RDBMS 内に口座情報を保持し、現金自動支払機(ATM)を通してお金を配布、受け取りします。ATM のアクションは口座に正確に反映されなければいけません、これは RDBMS だけでは行えません。グローバル トランザクション マネージャーが、金融取引全体の整合性を保障する為に、現金 ATM とデータベース リソースを統合します。

グローバル トランザクションを利用するアプリケーションは、1つ、または複数のリソース マネージャと、1つのトランザクション マネージャを含んでいます。

- リソース マネージャ(RM)はトランザクションのリソースにアクセスを供給します。データベース サーバはリソース マネージャの1つです。RM によって管理されたトランザクションをコミットしたり、ロールバックしたりする事が可能なはずですが、
- トランザクション マネージャ(TM)はグローバル トランザクションの一部であるトランザクションを調整します。これは、これらの各トランザクションを扱う RM と通信します。グローバル トランザクション内の各トランザクションは、グローバル トランザクションの「ブランチ」です。グローバル トランザクションと、それらのブランチは、後ほど紹介される名づけスキームによって識別されます。

XA MySQL の MySQL インプリメンテーションは、MySQL サーバがグローバル トランザクション内で XA トランザクションを扱うリソース マネージャとして機能する事を可能にします。MySQL サーバに接続するクライアントプログラムは、トランザクション マネージャとして機能します。

グローバル トランザクションを実行するには、どのコンポーネントが関係しているかを知り、それらをそれぞれ、コミットまたはロールバックできる時にポイントまで持ってくる事が必要です。各コンポーネントがそれぞれの能力についてどのようなレポートをするかによって、それらは全て1つのアトミック グループとしてコミット、またはロールバックしなければいけません。それは、全てのコンポーネントがコミットするか、または全てのコンポーネントがロールバックするという事です。グローバル トランザクションを管理するには、コンポーネントが、接続ネットワークが失敗する可能性があるという事を考慮に入れておく事が必要です。

グローバル トランザクションを実行するプロセスは、二相コミットを利用します。(2PC)これは、グローバル トランザクションのブランチによって行われるアクションが実行された後に起こります。

- 第一段階で、全てのブランチが準備されます。それは、TM からコミットの準備の命令が出るという事です。通常、これはブランチが管理する各 RM が、安定したストレージの中にブランチの為のアクションを記録するという事を意味します。ブランチは、それらが利用可能かどうかを指示し、その結果は第二段階に利用されます。
- 第二段階では、TM が RM にロールバックを行うかどうかの指示を出します。もし全てのブランチの準備が整った時それらがコミットが可能だという連絡を出せば、それら全てにコミットの指示が出されます。もしブランチの準備が整った時にどれか1つでもコミットが不可能であるという連絡があれば、全てのブランチにロールバックの指示が出されます。

場合によっては、グローバル トランザクションは単相コミットを利用します。(1PC)例えば、トランザクション マネージャがたった1つのトランザクション リソースで成り立っているグローバル トランザクションを見つけたら(それを単一ブランチといいます)、そのリソースには準備とコミットを同時にするように指示が出されます。

XA トランザクション SQL 構文

MySQL 内で XA トランザクションを行うには、次のステートメントを利用してください。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER
```

XA START に対しては、JOIN と RESUME 条項はサポートされていません。

XA END に対しては、SUSPEND [FOR MIGRATE] 条項はサポートされていません。

各 XA ステートメントは XA キーワードで始まり、それらのほとんどが xid 値を必要とします。xid は XA トランザクション識別子です。それは、ステートメントがどのトランザクションに適応するのか指示します。xid 値はクライアントから提供されるか、MySQL サーバから発生します。xid 値は1つから3つの部分を持っています。

```
xid: gtrid [, bqual [, formatID ]]
```

gtrid はグローバル トランザクション識別子、bqual はブランチ修飾子、そして formatID は gtrid と bqual 値によって利用されるフォーマットを識別する数値です。構文によって指示されているように、bqual と formatID は任意です。デフォルトの bqual 値は、指示されていない限り " です。デフォルトの formatID 値は、指示されていない限り1です。

`gtrid` と `bqual` は文字列直定数である必要があり、それぞれが最長64バイトです(文字数ではない)。`gtrid` と `bqual` の指定方法は、いくつかあります。引用された文字列('ab')、16進数列 (0x6162、X'ab')、またはビット値(b'nnnn')を利用する事ができます。

`formatID` は符号無し of 整数です。

`gtrid` と `bqual` 値は、MySQL サーバに内在する XA サポート ルーチンによって、バイトで解釈されます。しかし、XA ステートメントを含む SQL ステートメントが解析されると、サーバはいくつかの特定の文字セットと共に機能します。安全の為に、`gtrid` と `bqual` を16進数列として書いてください。

`xid` 値は一般的にトランザクション マネージャによって生成されます。ある TM によって生成された値は、別の TM によって生成された値とは異なります。規定の TM は **XA RECOVER** ステートメントによって返された値のリスト内にある、それ自身の `xid` 値を識別できなければいけません。

XA START `xid` は、規定の `xid` 値を利用して XA トランザクションをスタートします。各 XA トランザクションは固有の `xid` 値を持つ必要があるため、その値は同時に XA トランザクションによって利用されてはいけません。一意性は `gtrid` と `bqual` 値を利用して評価されます。XA トランザクションに続く全ての XA ステートメントは、**XA START** ステートメント内で規定されているように、同じ `xid` 値を利用して指定されなければいけません。もし、それらのステートメントのどれかを利用しながら、既存 XA トランザクションに対応しない `xid` 値を指定すると、エラーが起こります。

ひとつ、または複数の XA トランザクションは同じグローバル トランザクションの一部になる事ができます。規定のグローバル トランザクション内の全ての XA トランザクションは、`xid` 値内で同じ `gtrid` 値を利用しなければいけません。この理由により、規定の XA トランザクションがどのグローバル トランザクションの一部であるかをはっきりさせる為に、`gtrid` 値はグローバルに一意である必要があります。`xid` 値の `bqual` 部は、グローバル トランザクションの中で、各 XA トランザクションに対して異なっている必要があります。(bqual 値が異なっている必要があるという要求は、現在の MySQL XA インプリメンテーションの制限です。これは XA 仕様の一部ではありません。)

XA RECOVER ステートメントは、**PREPARED** 状態にある MySQL サーバ上の XA トランザクションに情報を返します。(詳しくは **XA トランザクションの状態** をご確認ください。) アウトプットは、どのクライアントがスタートしたかに関わらず、サーバ上の XA トランザクションなどの行を含みます。

XA RECOVER アウトプット行はこのような形になります。('abc'、'def'、そして 7 部分で成り立っている `xid` 値の例):

```
mysql> XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
| 7 | 3 | 3 | abcdef |
+-----+-----+-----+-----+
```

アウトプット カラムは次の意味を持っています。

- `formatID` はトランザクション `xid` の `formatID` 部です。
- `gtrid_length` は `xid` の `gtrid` 部の、バイトでの長さです。
- `bqual_length` は `xid` の `bqual` 部の、バイトでの長さです。
- `data` は `xid` の `gtrid` と `bqual` 部の連結です。

XA トランザクションの状態

次の状態での XA トランザクションの進歩

1. XA トランザクションをスタートさせる為に **XA START** を利用し、それを **ACTIVE** な状態にして下さい。
2. **ACTIVE** XA トランザクションには、トランザクションを形成する SQL ステートメントを発行し、それから **XA END** ステートメントを発行してください。**XA END** はトランザクションを **IDLE** の状態にします。
3. **IDLE** XA トランザクションには、**XA PREPARE** ステートメントか **XA COMMIT ... ONE PHASE** ステートメントのどちらかを発行できます。**ONE PHASE** ステートメント:
 - **XA PREPARE** はトランザクションを

PREPARED の状態にします。

- この時点の **XA RECOVER** ステートメントは、**XA RECOVER** が **PREPARED** な状態にある全ての XA トランザクションをリストにする為、そのアウトプットの中にトランザクションの **xid** 値を含みます。
 - **XA COMMIT ... ONE PHASE** はトランザクションを準備し、コミットします。**xid** 値は、トランザクションが終了する為 **XA RECOVER** によってリストされません。
4. **PREPARED** XA トランザクションに対して、トランザクションをコミットし、終了させる為に **XA COMMIT** ステートメントを発行するか、またはトランザクションをロールバックして終了させる為に **XA ROLLBACK** を発行できます。

ここに、行をテーブルにグローバル トランザクションの一部として挿入するシンプルな XA トランザクションの例があります。

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

既存のクライアント接続のコンテキストの中で、XA トランザクションとローカル トランザクション(非 XA)は、お互いに排他的です。例えば、もし XA トランザクションを始める為に **XA START** が発行されたら、XA トランザクションがコミットされるかロールバックされるまで、ローカル トランザクションはスタートできません。逆に、もしローカル トランザクションが **START TRANSACTION** でスタートされたら、トランザクションがコミットされるかロールバックされるまで、XA トランザクションは利用できません。

もし XA トランザクションが **ACTIVE** な状態であれば、暗黙的なコミットを引き起こすステートメントの発行はできないという事に注意してください。XA トランザクションをロールバックする事はできないので、これは XA 規約に違反するという事になります。もしこのようなステートメントを実行しようとする、次のようなエラーが発生します。

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

ステートメントが上記のどの注意事項に当てはまるのかは、**暗黙のコミットを引き起こすステートメント** でリストアップされています。

データベース管理ステートメント

アカウント管理ステートメント

MySQL アカウント情報は **mysql** データベースのテーブルに格納されています。このデータベースとアクセス コントロール システムについては **4章データベース管理** で詳しく説明されていますので、更なる詳細についてはこれを参照して下さい。

重要:MySQL のいくつかのリリース版は、新しい権限や特徴を追加する為に供与テーブルの構造の変更を紹介しています。MySQL の新しいバージョンを更新する時はいつでも、新しい機能を有効に利用できるようにする為に、供与テーブルを最新の構造に更新しなければいけません。詳しくは **mysql_upgrade — MySQL アップグレードのテーブルチェック** を参照してください。

MySQL Enterprise

```
##### MySQL #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

CREATE USER 構文

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
[, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

CREATE USER ステートメントは新しい MySQL アカウントを作成します。それを利用する為には、**mysql** データベースにグローバル **CREATE USER** 権限が **INSERT** 権限を持つ必要があります。それぞれのアカウントに対して、**CREATE USER** は権限を持たない **mysql.user** テーブル内に新しい行を作成します。もしアカウントが既に存在すると、エラーが発生します。各アカウントは、例えば、'**jeffrey**'@'**localhost**' のように **GRANT** ステートメントと同じフォーマットを利用して名づけられます。もしアカウント名のユーザ名部分だけを指定すると、ホスト名の '**%**' 部分が利用されます。アカウント名の指定についての追加情報に関しては、**GRANT 構文** を参照してください。

アカウントには、任意の **IDENTIFIED BY** 条項を利用してパスワードを与える事ができます。**user** 値とパスワードは、**GRANT** ステートメントと同じ方法で与える事ができます。特に、プレーン テキスト内でパスワードを指定するには、**PASSWORD** キーワードを省略してください。パスワードを、**PASSWORD()** 関数によって返されたようにハッシュ化された値として指定するには、**PASSWORD** キーワードを含んでください。詳しくは **GRANT 構文** を参照してください。

DROP USER 構文

```
DROP USER user [, user] ...
```

DROP USER ステートメントは1つ、または複数の MySQL アカウントを削除します。それは、全ての供与テーブルからアカウントの権限行を削除します。このステートメントを利用する為には、**mysql** データベースにグローバル **CREATE USER** 権限が **DELETE** 権限を持つ必要があります。各アカウントは、例えば、'**jeffrey**'@'**localhost**' のように **GRANT** ステートメントと同じフォーマットを利用して名づけられます。もしアカウント名のユーザ名部分だけを指定すると、ホスト名の '**%**' 部分が利用されます。アカウント名の指定についての追加情報に関しては、**GRANT 構文** を参照してください。

DROP USER を利用すると、アカウントとその権限を次のように削除する事ができます。

```
DROP USER user;
```

重要:**DROP USER** は自動的にユーザ セッションを閉じません。それよりも、オープン セッションを持つユーザがドロップした時には、そのユーザのセッションが閉じられるまでそのステートメントは効果を発揮しません。一度セッションが閉じられると、そのユーザはドロップされ、そのユーザが次にログインしようとする時失敗します。これには意図があります。

DROP USER は、ユーザが作成したデータベース オブジェクトを自動的に削除したり、無効にしたりしません。これは、テーブル、ビュー、ストアド ルーチン、トリガ、そしてイベントに適応します。

GRANT 構文

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON [object_type] {tbl_name | * | *.* | db_name.*}
TO user [IDENTIFIED BY [PASSWORD] 'password']
[, user [IDENTIFIED BY [PASSWORD] 'password']] ...
[REQUIRE
  NONE |
  [{SSL} X509]
  [CIPHER 'cipher' [AND]]
  [ISSUER 'issuer' [AND]]
  [SUBJECT 'subject']]
[WITH with_option [with_option] ...]
```

```
object_type =
  TABLE
  | FUNCTION
  | PROCEDURE
```

```
with_option =
  GRANT OPTION
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
```

GRANT ステートメントは、システム管理者が MySQL ユーザ アカウントを作成し、アカウントに権利を与える事を可能にします。**GRANT** を利用する為には、**GRANT OPTION** 権限を持ち、また自分が供与している権限を持つ必要があります。**REVOKE** ステートメントは、アカウント管理者が権限を削除する事に関連し、それを可能にします。詳しくは [REVOKE 構文](#) を参照してください。

MySQL アカウント情報は `mysql` データベースのテーブルに格納されています。このデータベースとアクセス コントロール システムについては [4章データベース管理](#) で詳しく説明されていますので、更なる詳細についてはこれを参照して下さい。

重要:MySQL のいくつかのリリース版は、新しい権限や特徴を追加する為に供与テーブルの構造の変更を紹介しています。MySQL の新しいバージョンを更新する時はいつでも、新しい機能を有効に利用できるようにする為に、供与テーブルを最新の構造に更新しなければいけません。詳しくは [mysql_upgrade — MySQL アップグレードのテーブルチェック](#) を参照してください。

もし供与テーブルが、大文字と小文字が混在するデータベースかテーブル名を含む権限行を保持し、`lower_case_table_names` システム変数がゼロではない値に設定されたら、**REVOKE** をこれらの権限を廃止する為に利用する事はできません。供与テーブルを直接コントロールする必要があるかも知れません。(GRANT は `lower_case_table_names` が設定された時にそのような行を作成しませんが、そのような行は変数を設定する前に作成されていた可能性があります。)

権限はいくつかのレベルで供与する事ができます:

- グローバル レベル

既存サーバ上では、グローバル権限は全てのデータベースに適用します。これらの権限は `mysql.user` テーブル内に格納されています。**GRANT ALL ON *.*** と **REVOKE ALL ON *.*** はグローバル権限だけを供与、廃止します。

- データベース レベル

データベース権限は既存データベース内の全てのオブジェクトに適用します。これらの権限は `mysql.db` と `mysql.host` テーブル内に格納されています。**GRANT ALL ON db_name.*** と **REVOKE ALL ON db_name.*** はグローバル権限だけを供与、廃止します。

- テーブル レベル

テーブル権限は既存テーブル内の全てのカラムに適用します。これらの権限は `mysql.tables_priv` テーブル内に格納されています。**GRANT ALL ON db_name.tbl_name** と **REVOKE ALL ON db_name.tbl_name** はグローバル権限だけを供与、廃止します。

- カラム レベル

カラム権限は既存テーブル内の単一カラムに適用します。これらの権限は `mysql.columns_priv` テーブル内に格納されています。**REVOKE** を利用している時は、供与されたカラムと同じカラムを指定する必要があります。

- ルーチン レベル

CREATE ROUTINE、**ALTER ROUTINE**、**EXECUTE**、そして **GRANT** 権限はストアド ルーチンに適用します。(ファンクションとプロシージャ)それらは、グローバルとデータベース レベルで供与されます。また、**CREATE ROUTINE** 以外は、これらの権限は各ルーチンに対してルーチン レベルで供与する事ができ、`mysql.procs_priv` テーブル内で格納されます。

`object_type` 条項は、次のオブジェクトがテーブル、ストアド ファンクション、またはストアド ルーチンの時には **TABLE**、**FUNCTION**、または **PROCEDURE** 条項として指定されなければいけません。

GRANT と **REVOKE** ステートメントに対しては、`priv_type` は次の表にある物として指定する事ができます。

権限	意味
ALL [PRIVILEGES]	GRANT OPTION 以外の全てのシンプルな権限を設定します。
ALTER	ALTER TABLE の使用を可能にします。
ALTER ROUTINE	ストアド ルーチンの変更、ドロップを可能にします。
CREATE	CREATE TABLE の使用を可能にします。

CREATE ROUTINE	ストアド ルーチンの作成を可能にします。
CREATE TEMPORARY TABLES	CREATE TEMPORARY TABLE の使用を可能にします。
CREATE USER	CREATE USER、DROP USER、RENAME USER、そして REVOKE ALL PRIVILEGES の使用を可能にします。
CREATE VIEW	CREATE VIEW の使用を可能にします。
DELETE	DELETE の使用を可能にします。
DROP	DROP TABLE の使用を可能にします。
EVENT	イベントスケジューラがイベントを作成するのを可能にします。
EXECUTE	ユーザがストアドルーチンを起動させるのを可能にします。
FILE	SELECT ... INTO OUTFILE と LOAD DATA INFILE の使用を可能にします。
INDEX	CREATE INDEX と DROP INDEX の使用を可能にします。
INSERT	INSERT の使用を可能にします。
LOCK TABLES	SELECT 権限を持つテーブル上の LOCK TABLES の使用を可能にします。
PROCESS	SHOW FULL PROCESSLIST の使用を可能にします。
REFERENCES	インプリメントされていません。
RELOAD	FLUSH の使用を可能にします。
REPLICATION CLIENT	ユーザがスレーブとマスタの場所を問い合わせる事を可能にします。
REPLICATION SLAVE	複製スレーブが必要とします。(マスタからバイナリ ログ イベントを読み込む為)
SELECT	SELECT の使用を可能にします。
SHOW DATABASES	SHOW DATABASES は全てのデータベースを表示します。
SHOW VIEW	SHOW CREATE VIEW の使用を可能にします。
SHUTDOWN	mysqladmin shutdown の使用を可能にします。
SUPER	CHANGE MASTER、KILL、PURGE MASTER LOGS、そして SET GLOBAL ステートメントの使用を可能にし、the mysqladmin debug コマンドは max_connections が達成していても接続を (一回) 許可します。
TRIGGER	ユーザがトリガを作成、ドロップするのを可能にします。
UPDATE	UPDATE の使用を可能にします。
USAGE	「権限が無い」 の同義語です。
GRANT OPTION	権限を与えるのを可能にします。

EVENT と TRIGGER 権限は MySQL 5.1.6 で追加されました。トリガはテーブルと関連しているので、トリガを作成したり、ドロップしたりするには、トリガではなくテーブルに TRIGGER 権限を持つ必要があります。(MySQL 5.1.6 以前では SUPER 権限はトリガを作成したりドロップしたりする為に必要でした。)

REFERENCES 権限は現在は利用されていません。

USAGE は、権限を持たないユーザを作成した時に指定する事ができます。

アカウントがどんな権限を持つのかを決定する為には SHOW GRANTS を利用してください。詳しくは SHOW GRANTS 構文 を参照してください。

ON *.* 構文を利用してグローバル権限を、または、ON db_name.* 構文を利用してデータベース権限を割り当てる事ができます。もし ON * を指定し、デフォルト データベースを選択したら、権限はそのデータベース内に供与されます。(警告: もし ON * を指定し、デフォルト データベースを選択 しなければ、供与される権限はグローバルになります。)

FILE、PROCESS、RELOAD、REPLICATION CLIENT、REPLICATION SLAVE、SHOW DATABASES、SHUTDOWN、そして SUPER 権限は、グローバルとしてのみ供与される(ON *.* 構文を利用して)管理権限です。

その他の権限は、グローバルに、またはより特定のレベルで供与することができます。

テーブルに指定できる `priv_type` 値は、`SELECT`、`INSERT`、`UPDATE`、`DELETE`、`CREATE`、`DROP`、`GRANT OPTION`、`INDEX`、`ALTER`、`CREATE VIEW`、`SHOW VIEW` そして `TRIGGER` です。

カラムに指定できる `priv_type` 値は (`column_list` 条項を利用する時)、`SELECT`、`INSERT`、そして `UPDATE` です。

ルーチン レベルで指定できる `priv_type` 値は `ALTER ROUTINE`、`EXECUTE`、そして `GRANT OPTION` です。`CREATE ROUTINE` は、最初にルーチンを作成する時にこの権限を持たなければいけないので、ルーチン レベル権限ではありません。

`GRANT ALL` は、与えようとしているレベルに存在する権限のみをグローバル、データベース、テーブル、そしてルーチン レベルに対して割り当てます。例えば、`GRANT ALL ON db_name.*` はデータベース レベル ステートメントなので、`FILE` のようなグローバルのみの権限は供与しません。

MySQL は、存在しないデータベース オブジェクト上にも権限を与える事を許容します。そのような場合、供与される権限は `CREATE` 権限を含む必要があります。この動作は意図的であり、また後で作成されるデータベース オブジェクトに対するユーザ アカウントと権限を、データベース管理者が準備する事を可能にします。

重要:MySQL は、テーブルやデータベースをドロップする時に、自動的に権限を廃止しません。しかし、もしルーチンをドロップすると、そのルーチンに供与された全てのルーチン レベル権限は廃止されます。

注意: `'_'` と `'%'` ワイルドカードは、権限をグローバル、またはデータベース レベルで供与する `GRANT` ステートメント内でデータベース名を指定する時に許容されます。これは例えば、データベース名の一部として `'_'` 文字を利用したければ、ワイルドカードのパターンに一致する追加データベースにアクセスする事を不可能にする為に、`GRANT` ステートメント内でこれを `'\_'` として指定必要がるという事を意味します。例えば、`GRANT ... ON 'foo\_bar'.* TO ...` のようになります。

任意ホストからユーザに供与権を適応させる為に、MySQL は `user_name@host_name` の形で `user` 値を指定する事をサポートします。もし `user_name` が `host_name` 値が引用されていない識別子として正当であれば、それを引用する必要はありません。しかし、引用句は特別な文字(`'` 等のような)を含む `user_name` 文字列、または特別な文字やワイルドカード文字(`'%'` 等のような)を含む `host_name` 文字列を指定するのに必要です。例えば、`'test-user'@'test-hostname'` のようになります。ユーザ名とホスト名を別々に引用してください。

ホスト名の中でワイルドカードを指定することができます。例えば、`user_name@'%.loc.gov'` は、`loc.gov` ドメイン内のどのホストに対しても `user_name` に適応し、`user_name@'144.155.166.%'` は `144.155.166` クラス C サブネット内のどのホストに対しても、`user_name` に適応します。

シンプルな形 `user_name` は、`user_name@'%'` の同義語です。

MySQL はユーザ名内でワイルドカードをサポートしません。匿名のユーザは、`User=""` を持つエントリを `mysql.user` テーブル内に挿入する事、または `GRANT` を利用してユーザを空の名前で作成する事で定義されます。

```
GRANT ALL ON test.* TO '@'localhost' ...
```

引用された値、引用データベース、テーブル、カラム、そしてルーチン名を識別子として指定する時は、バッククォートを利用しています。(`'`)ホスト名、ユーザ名、そしてパスワードを文字列として、単一引用句(`'`)を利用して引用してください。

警告:匿名ユーザが MySQL サーバに接続する事を許可するなら、全てのローカル ユーザにも `user_name@localhost` として権限を供与する必要があります。そうでなければ、名づけられたユーザがローカル マシンから MySQL サーバにログインしようとした時に、`mysql.user` テーブル内(MySQL インストールの最中に作成された物)の `localhost` の匿名ユーザ アカウントが利用されます。詳細に関しては [アクセス制御の段階 1：接続確認](#) を参照して下さい。

匿名ユーザをリスト アップしてある次のクエリを実行する事によって、これが当てはまるかどうかを決定することができます。

```
SELECT Host, User FROM mysql.user WHERE User="";
```

もし先ほど説明した問題を防ぐ為に、ローカル匿名ユーザ アカウントを削除したければ、次のステートメントを利用してください。

```
DELETE FROM mysql.user WHERE Host='localhost' AND User="";
FLUSH PRIVILEGES;
```

GRANT は最大60文字のホスト名をサポートします。データベース、テーブル、カラム、そしてルーチン名は最大64文字です。ユーザ名は最大16文字です。注意:ユーザ名の許容長さは、**mysql.user** テーブルを変更しても変える事はできません。もしそれをする、ユーザが MySQL サーバにログインできなくなってしまうような、予想不可能な動作に陥る可能性があります。**mysql_upgrade — MySQL アップグレードのテーブル チェック** に説明のある、MySQL AB で規定されている方法を利用する以外は、**mysql** データベース内のテーブルは、どんな方法でも絶対に変更してはいけません。

テーブル、カラム、ルーチンの権限は、各権限レベルの権限の論理的な **OR** として相加的に形成されています。例えば、もし **mysql.user** テーブルが、ユーザはグローバル **SELECT** 権限を持つ、と指定すると、その権限はデータベース、テーブル、またはカラムレベルのエントリによって否定する事はできません。

カラムの権限は次のように計算する事ができます。

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

ほとんどの場合、たった1つの権限レベルでユーザに対して権限を供与する為、実際にはこれほど複雑では有りません。権限チェックの方法の詳細については、**MySQL アクセス権限システム** で紹介されています。

もし **mysql.user** テーブルに存在しないユーザ名/ホスト名の組み合わせに権限を供与するのであれば、エントリが追加されそれは **DELETE** ステートメントで削除されるまでそこに残ります。言い換えると、**GRANT** は **user** テーブル エントリを作成するかもしれませんが、**REVOKE** はそれらを削除しません。従ってそれは **DROP USER** か **DELETE** を明示的に利用して行わなければいけません。

警告:もし新しいユーザを作成して **IDENTIFIED BY** 条項を指定しないと、そのユーザはパスワードを持ちません。これはとても不安定です。しかし、新規ユーザに空ではないパスワードを提供する為に **IDENTIFIED BY** が与えられない限り、**GRANT** が新規ユーザを作成するのを防ぐ為に **NO_AUTO_CREATE_USER** SQL モードを有効にする事ができます。

MySQL Enterprise

MySQL ##### <http://www-jp.mysql.com/products/enterprise/advisors.html> #####

もし新規ユーザが作成されたり、グローバル供与権限を持っていたら、ユーザのパスワードは、**IDENTIFIED BY** 条項があればそれによって指定された物に設定されます。もしユーザが既にパスワードを持っていたら、それは新しい物と置き換えられます。

パスワードは **SET PASSWORD** ステートメントで設定する事もできます。詳しくは **SET PASSWORD 構文** を参照してください。

IDENTIFIED BY 条項内では、パスワードは直定数パスワード値でなければいけません。**PASSWORD()** 関数を **SET PASSWORD** ステートメントに対する時と同じように使用する必要はありません。例:

```
GRANT ... IDENTIFIED BY 'mypass';
```

もしパスワードを明確なテキストで送信したく無い場合、**PASSWORD()** がそのパスワードに返すハッシュ化された値が分かっている、キーワード **PASSWORD** の前にハッシュ化された値を指定する事ができます。

```
GRANT ...
IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

C プログラムの中では、**make_scrambled_password()** C API 関数を利用する事で、ハッシュ化された値を得る事ができます。

もしデータベースに権限を供与すると、**mysql.db** テーブル内のエントリが必要に応じて作成されます。もしデータベースの全ての権限が **REVOKE** で削除されると、このエントリも削除されます。

SHOW DATABASES 権限は **SHOW DATABASE** ステートメントを発行する事によって、アカウントがデータベース名を見る事を可能にします。この権限を持たないアカウントは、それらが権限を持つデータベースだけを見て、また、もしサーバが **--skip-show-database** オプションでスタートされていれば、ステートメントを利用する事は全できません。

MySQL Enterprise

```
SHOW DATABASES #####MySQL #####MySQL #####
##### --skip-show-database ##### http://www-
jp.mysql.com/products/enterprise/advisors.html #####
```

もしユーザがテーブルに何の権限も持たなければ、ユーザがテーブルのリストを要求した時(例えば、`SHOW TABLES` ステートメントを利用して)テーブル名は表示されません。

`WITH GRANT OPTION` 条項は、ユーザが指定された権限レベルで持つ権限を、別のユーザに与える事ができる機能を与えます。異なる権限を持つ2つのユーザは権限を接合できる事があるので、どのユーザに `GRANT OPTION` 権限を与えるのが注意する必要があります。

自分が持たない権限を別のユーザに与える事はできません。`GRANT OPTION` 権限によって、自分が持つ権限を割り当てる機能のみ与えられます。

特定の権限レベルでユーザに `GRANT OPTION` 権限を与える時は、そのユーザがそのレベルで持つどんな権限も、(または将来与えられる物も)そのユーザから別のユーザに与える事ができるので、注意が必要です。ユーザにデータベース上で `INSERT` 権限を与えると仮定してください。その時も `SELECT` 権限をデータベース上で与え、`WITH GRANT OPTION` を指定すると、そのユーザは別のユーザに `SELECT` 権限だけではなく `INSERT` も与える事ができます。その時もデータベース上でユーザに `UPDATE` 権限を与えると、そのユーザは `INSERT`、`SELECT`、そして `UPDATE` を与える事ができるようになります。

非管理者ユーザに対しては、`ALTER` 権限をグローバルに与えたり、またはそれを `mysql` データベースに与えたりするべきでは有りません。もしそれをしてしまうと、残りのテーブルによってそのユーザが権限システムを破壊する事が可能になってしまいます!

`MAX_QUERIES_PER_HOUR count`、`MAX_UPDATES_PER_HOUR count`、そして

`MAX_CONNECTIONS_PER_HOUR count` オプションは、1時間の間にユーザが行う事ができるクエリ、更新、そしてログインの数を制限します。(クエリ キャッシュから結果が与えられるクエリは `MAX_QUERIES_PER_HOUR` 制限に対してカウントしません。)もし `count` が 0 なら(デフォルト)、そのユーザには制限がないという意味になります。

`MAX_USER_CONNECTIONS count` オプションは、そのアカウントが作成する事ができる同時接続の最大数を制限します。もし `count` が 0 なら(デフォルト)、`max_user_connections` システム変数はそのアカウントの同時接続数を決定します。

注意:既存権限に影響を与えずに、既に存在するユーザにこれらのリソース制限オプションを指定するには、`GRANT USAGE ON *.* ... WITH MAX_...` を利用して下さい。

詳しくは [ユーザ リソースの制限](#) を参照してください。

MySQL は、ユーザ名とパスワードに基づいた通常認証に加え、X509 証明拡張子を確認する事ができます。SSL 関連オプションを MySQL アカウントに指定するには `GRANT` ステートメントの `REQUIRE` 条項を利用してください。(MySQL を利用した SSL 利用の背景情報については、[接続安全](#) を参照してください。)

与えられたアカウントの接続を制限するには、いくつかの異なる方法があります。

- `REQUIRE NONE` は、アカウントが SSL や X509 要求を持たないという事を指示します。もし SSL 関連 `REQUIRE` オプションが指定されれば、これがデフォルトになります。もしユーザ名とパスワードが有効なら、暗号化されていない接続が許可されます。しかし、クライアントのオプションにより、もしそのクライアントが正しい証明とキー ファイルを持っていれば、暗号化された接続もまた利用できます。それは、クライアントは SSL コマンドオプションを指定する必要がなく、その場合接続は暗号化されていない物になる、という事です。暗号化された接続を利用するには、クライアントは `--ssl-ca` オプションか、または `--ssl-ca`、`--ssl-key`、または `--ssl-cert` オプションの3つ全てを指定する必要があります。
- `REQUIRE SSL` オプションは、サーバに対して、SSL 暗号化接続だけをアカウントに許容するよう指示を出します。

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

接続するには、クライアントは `--ssl-ca` オプションを指定し、また追加で `--ssl-key` と `--ssl-cert` オプションを指定する必要があります。

- **REQUIRE X509** は、クライアントが有効な証明書を持つ必要があるが、その証明書、発行者、そして題目は何でもかまわないという事を意味します。要求されるのは、CA 証明の1つによってそのサインを検証できなければいけない、という事だけです。

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

接続するには、クライアントは **--ssl-ca**、**--ssl-key**、**--ssl-cert** オプションを指定する必要があります。

これは **REQUIRE** オプションが **X509** の意味を含む為、**ISSUER** と **SUBJECT** にとっても同じ事が言えます。

- **REQUIRE ISSUER 'issuer'** は、クライアントが CA **'issuer'** によって発行された有効な X509 証明を提示するよう、接続に制限を与えます。もしクライアントが、有効ではあるが異なる発行者による証明を提示したら、サーバは接続を拒否します。X509 証明の利用は、必ず暗号化を暗示しますので、**SSL** オプションはこの場合必要ありません。

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
IDENTIFIED BY 'goodsecret'
REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
```

'issuer' 値は単一文字列として入力されなければいけない事を覚えておいて下さい。

- **REQUIRE SUBJECT 'subject'** は、クライアントがサブジェクト **subject** を含む有効な X509 証明を提示する接続に制限を与えます。もしクライアントが、有効ではあるが異なるサブジェクトによる証明を提示したら、サーバは接続を拒否します。

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
IDENTIFIED BY 'goodsecret'
REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
O=MySQL demo client certificate/
CN=Tonu Samuel/Email=tonu@example.com';
```

'subject' 値は単一文字列として入力されなければいけない事を覚えておいて下さい。

- **REQUIRE CIPHER 'cipher'** は十分な強さの暗号とキー長さが利用される事が保証されていなければいけません。SSL 自体は、短い暗号キーを利用している古いアルゴリズムが利用されると、弱くなる事があります。このオプションを利用すると、接続を許可する為に特定の暗号方法が利用されるように依頼する事ができます。

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
IDENTIFIED BY 'goodsecret'
REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

SUBJECT、**ISSUER**、そして **CIPHER** オプションを、次のように **REQUIRE** 条項の中で組み合わせる事ができます。

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
IDENTIFIED BY 'goodsecret'
REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
O=MySQL demo client certificate/
CN=Tonu Samuel/Email=tonu@example.com'
AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

AND キーワードは **REQUIRE** オプションとの間では任意です。

オプションの順番は重要では有りませんが、2回指定できるオプションはありません。

mysqld がスタートする時、全ての権限はメモリの中に読み込まれます。詳細に関しては [権限の変更が反映するタイミング](#) を参照して下さい。

1つのユーザに対してテーブル、カラム、またはルーチン権限を利用していると、サーバはテーブル、カラム、そしてルーチン権限を全てのユーザに対して分析し、その為に MySQL のスピードが少し遅くなります。同じように、もしユーザに対してクエリ、更新または接続の数を制限すると、サーバはそれらの値を監視しなければいけません。

SQL と MySQL **GRANT** バージョンとの最大の違いは次のような物です。

- MySQL では、権限はユーザ名だけではなく、ホスト名とユーザ名の組み合わせと関連しています。
- スタンダード SQL はグローバル、またはデータベース レベル権限を持たず、MySQL がサポートする全ての権限タイプのサポートもしません。
- MySQL はスタンダード SQL [UNDER](#) 権限をサポートしません。
- スタンダード SQL 権限は、階層的な様式で構造されています。もしユーザを削除すると、そのユーザが持つ全ての権限が与えられ、削除されます。もし [DROP USER](#) を利用するなら、MySQL でも同じです。詳しくは [DROP USER 構文](#) を参照してください。
- スタンダード SQL では、テーブルをドロップすると、そのテーブルの全ての権限は削除されます。スタンダード SQL では、権限を削除すると、その権限に基づいて与えられた全ての権限もまた削除されます。MySQL では、明示的な [REVOKE](#) ステートメントだけの利用、または MySQL 供与テーブル内に格納された値の複製でドロップできます。
- MySQL では、テーブル内のいくつかのカラムに対してだけ [INSERT](#) 権限を持つ事が可能です。この場合、[INSERT](#) 権限を持っていないそれらのカラムを削除するのであれば、テーブル上で [INSERT](#) ステートメントを実行する事ができます。削除されたカラムは、もしストリクト SQL モードが有効でなければ、暗黙的なデフォルト値に設定されます。ストリクト モードでは、もし削除されたカラムがデフォルト値を持たなければ、ステートメントは却下されます。(スタンダード SQL は、全てのカラムに [INSERT](#) 権限を持つように要求します。) [SQL モード](#) で、ストリクト モードについて説明されています。 [データタイプデフォルト値](#) で暗黙のデフォルト値について説明されています。

RENAME USER 構文

```
RENAME USER old_user TO new_user
[, old_user TO new_user] ...
```

[RENAME USER](#) ステートメントは既存 MySQL アカウントをリネームします。それを利用する為には、[mysql](#) データベースにグローバル [CREATE USER](#) 権限か [UPDATE](#) 権限を持つ必要があります。もし古いアカウントが存在しない時、または新しいアカウントが存在する時はエラーが発生します。各アカウントは、例えば、'[jeffrey](#)'@'[localhost](#)' のように [GRANT](#) ステートメントと同じフォーマットを利用して名づけられます。もしアカウント名のユーザ名部分だけを指定すると、ホスト名の '[%](#)' 部分が利用されます。アカウント名の指定についての追加情報に関しては、[GRANT 構文](#) を参照してください。

[RENAME USER](#) は、ユーザが作成したデータベース オブジェクトを自動的に移動させたり、リネーム前にユーザが持っていた権限を移動させたりしません。これは、テーブル、ビュー、ストアド ルーチン、トリガ、そしてイベントに適用します。

REVOKE 構文

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON [object_type] {tbl_name | * | *.* | db_name.*}
FROM user [, user] ...
```

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

[REVOKE](#) ステートメントは、システム管理者が MySQL アカウントから権限を削除する事を可能にします。[REVOKE](#) を利用する為には、[GRANT OPTION](#) 権限を持ち、また自分が削除しようとしている権限を持っている必要があります。各アカウントは、例えば、'[jeffrey](#)'@'[localhost](#)' のように [GRANT](#) ステートメントと同じフォーマットを利用して名づけられます。もしアカウント名のユーザ名部分だけを指定すると、ホスト名の '[%](#)' 部分が利用されます。アカウント名の指定についての追加情報に関しては、[GRANT 構文](#) を参照してください。

どのレベルにどの権限が存在するのか、許容 [priv_type](#) 値、そしてユーザとパスワードを指定する為の構文については、[GRANT 構文](#) を参照してください。

もし供与テーブルが、大文字と小文字が混在するデータベースカテーブル名を含む権限行を保持し、[lower_case_table_names](#) システム変数がゼロではない値に設定されたら、[REVOKE](#) をこれらの権限を廃止する為に利用する事はできません。供与テーブルを直接コントロールする必要があるかも知れません。([GRANT](#) は [lower_case_table_names](#) が設定された時にそのような行を作成しませんが、そのような行は変数を設定する前に作成されていた可能性があります。)

全ての権限を削除するには、名づけられた全てのユーザに対して、全てのグローバル、データベース、テーブル、そしてカラム レベルの権限をドロップする次の構文を利用してください。

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

この **REVOKE** 構文を利用する為には、**mysql** データベースにグローバル **CREATE USER** 権限か **UPDATE** 権限を持つ必要があります。

REVOKE は権限を削除しますが、**user** テーブル エントリのドロップはしません。それは **DELETE** か **DROP USER** を利用して明示的に行う必要があります。(**DROP USER** 構文 を参照してください。)

SET PASSWORD 構文

```
SET PASSWORD [FOR user] = PASSWORD('some password')
```

SET PASSWORD ステートメントは既存 MySQL ユーザ アカウントにパスワードを割り当てます。

FOR 条項無しで、このステートメントは現在のユーザにパスワードを設定します。非匿名アカウントを利用しているサーバに接続したクライアントは、そのアカウントのパスワードを変更する事ができます。

このステートメントは、**FOR** 条項を利用して、現在のサーバ ホスト上の特定のアカウントにパスワードを設定します。**mysql** データベースの **UPDATE** 権限を持つクライアントだけがこれを実行できます。**user** 値は、**user_name** と **host_name** が、**mysql.user** テーブル エントリの **User** と **Host** カラム内でリストされているのと全く同じである **user_name@host_name** フォーマット内で与えられる必要があります。例えば、もし **'bob'** と **'%.loc.gov'** の **User** と **Host** カラム値と共にエントリを持っていたら、ステートメントを次のように書くでしょう。

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

これは、次のステートメントと同等です。

```
UPDATE mysql.user SET Password=PASSWORD('newpass')
WHERE User='bob' AND Host='%.loc.gov';
FLUSH PRIVILEGES;
```

注意:もし MySQL 4.1 または pre-4.1 クライアント プログラムを利用するそれ以降のサーバに接続していたら、**MySQL 4.1 のパスワードハッシュ** を読むまでは、前出の **SET PASSWORD** や **UPDATE** ステートメントを利用しないでください。MySQL 4.1 で変更されたパスワード フォーマット、そして特定の状況下では、もしパスワードを変更するとその後サーバに接続する事ができなくなる、という事が起こり得ます。

SELECT CURRENT_USER() を実行する事によって、どのアカウントとしてサーバが認証したのかを見る事ができます。

テーブル メンテナンス ステートメント

ANALYZE TABLE 構文

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

ANALYZE TABLE はテーブルのキーの分布を分析、格納します。分析の最中に、テーブルは **MyISAM** のリード ロックを利用してロックされます。**InnoDB** には、テーブルは書き込みロックでロックされます。このステートメントは **MyISAM** と **InnoDB** テーブルと共に機能します。**MyISAM** テーブルにとっては、このステートメントは **myisamchk --analyze** を利用する事と同じです。

解析が **InnoDB** 内でどのように機能するのかという事に関する情報については、**InnoDB テーブル上の制約** を参照してください。

MySQL は、定数以外の何かに対して接合を実行した時、どの順番でテーブルが接合されるべきかを決める為に格納されたキー分布を利用します。

このステートメントはテーブルに **SELECT** と **INSERT** 権限を要求します。

ANALYZE TABLE は次のカラムを利用して結果セットを返します。

カラム	値
Table	テーブル名
Op	いつも analyze

Msg_type	status、error、info、または warning の1つ
Msg_text	メッセージ

SHOW INDEX ステートメントを利用して格納されたキー分布を確認する事ができます。詳しくは [SHOW INDEX 構文](#) を参照してください。

もしテーブルが前回の **ANALYZE TABLE** ステートメント以降変更されていなければ、そのテーブルは再解析されません。

ANALYZE TABLE ステートメントは、任意の **NO_WRITE_TO_BINLOG** キーワード(またはそのエイリアス **LOCAL**)が利用されない限り、バイナリ ログに書きこまれます。これは、複製マスタとして機能している MySQL サーバ上で利用される **ANALYZE TABLE** ステートメントが、複製スレーブにデフォルトで複製される為に行われます。

BACKUP TABLE 構文

```
BACKUP TABLE tbl_name [, tbl_name] ... TO 'path/to/backup/directory'
```

注意:このステートメントは廃止予定です。オンライン バックアップ機能を提供する、より良い代替を準備中です。その間、[mysqlhotcopy](#) スクリプトを代わりに利用する事ができます。

BACKUP TABLE は、バッファされた変更をディスクにフラッシュした後、テーブルを格納するのに必要な最低数のテーブル ファイルをバックアップ ディレクトリにコピーします。このステートメントは **MyISAM** テーブルにしか機能しません。それは **.frm** 定義と **.MYD** データ ファイルをコピーします。**.MYI** インデックス ファイルは、それら2つのファイルから回復する事ができます。ディレクトリは、完全なパス名として指定されなければいけません。テーブルを復旧させるには、**RESTORE TABLE** を利用してください。

バックアップの最中に、バックアップされるのに従って、各テーブルに対して1つずつリード ロックが行われます。もしいくつかのテーブルをスナップショットとしてバックアップしたければ(バックアップ操作の最中にそれらが変更されるのを防ぎながら)まずグループ内の全てのテーブルに対してリード ロックを取得する為に、**LOCK TABLES** ステートメントを発行してください。

BACKUP TABLE は次のカラムを利用して結果セットを返します。

カラム	値
Table	テーブル名
Op	いつも backup
Msg_type	status、error、info、または warning の1つ
Msg_text	メッセージ

CHECK TABLE 構文

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...
```

option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

CHECK TABLE はテーブルのエラーを確認します。**CHECK TABLE** は **MyISAM**、**InnoDB**、そして **ARCHIVE** テーブルにのみ機能します。MySQL 5.1.9 から、**CHECK** は **CSV** テーブルにも有効になりました。[CSV ストレージエンジン](#) を参照してください。**MyISAM** テーブルに対しては、キー統計もまた更新されます。

CHECK TABLE は、既に存在していないビュー定義内で参照されるテーブルなどのような、ビューの問題も確認できます。

CHECK TABLE は次のカラムを利用して結果セットを返します。

カラム	値
Table	テーブル名
Op	いつも check
Msg_type	status、error、info、または warning の1つ

Msg_text	メッセージ

ステートメントは各確認済テーブルにたくさんの情報行を作成する可能性がある事に注意してください。最後の行は `status` の `Msg_type` 値を持ち、`Msg_text` は通常 `OK` になります。もし `OK` か `Table is already up to date` が得られなければ、通常はテーブルの修復を起動する必要があります。詳しくは [テーブル保守とクラッシュ リカバリ](#) を参照してください。`Table is already up to date` は、そのテーブルのストレージ エンジンが、そのテーブルを確認する必要は無いと指示しているという意味です。

FOR UPGRADE オプションは、名づけられたテーブルが現在の MySQL バージョンと互換性があるか確認します。このオプションは MySQL 5.1.7 で追加されました。サーバは **FOR UPGRADE** を利用し、作成された時以降、各テーブルに対してデータ タイプやインデックスに互換性の無い変更があったかどうかを確認します。もし無ければ、その確認は成功です。反対に、もし非適合性があれば、サーバはテーブルに対して総確認を行います。(少々時間がかかります。)もし総確認が成功すれば、サーバはテーブルの `.frm` ファイルに現在の MySQL バージョン番号をマークします。`.frm` ファイルをマークする事で、同じバージョンのサーバによるそのテーブルの今後の確認が早くなる事が保証されます。

データタイプの格納フォーマットが変更されたか、そのソート順が変更された為に、非適合性が発生する可能性があります。私たちの目的はそれらの変更を避ける事です。時として各リリースの間に、非適合性よりもさらに深刻である問題を修正する事の方が大切な事もあります。

現在は、**FOR UPGRADE** でこれらの非適合性が見つっています。

- MySQL 4.1 と 5.0 の間で、**InnoDB** と **MyISAM** テーブルの為の **TEXT** カラム内の最後の空白のインデックス順が変更されました。
- 新しい **DECIMAL** データタイプの格納方法は MySQL 5.0.3 と 5.0.5 の間に変更されました。

それ以外のチェックポイントは次のテーブルに表されています。これらのオプションは **MyISAM** テーブルの確認だけに適応し、**InnoDB** テーブルとビューでは無視されます。

タイプ	意味
QUICK	不正リンクの確認の為に行をスキャンしないでください。
FAST	適切に閉じられていないテーブルだけを確認してください。
CHANGED	最後の確認以降変更されたか、または正常に閉じられていないテーブルだけを確認してください。
MEDIUM	削除されたリンクが有効である事を検証する為に行をスキャンしてください。これは行のキーチェックサムも計算し、それをキーの為に計算されたチェックサムを利用して検証します。
EXTENDED	各行の全てのキーに対して総キー参照を行ってください。これはテーブルが100%整合性がある事を保証しますが、時間がかかる作業です。

もし **QUICK**、**MEDIUM**、または **EXTENDED** のどのオプションも指定されなければ、動的フォーマットである **MyISAM** テーブルのデフォルトの確認タイプは **MEDIUM** になります。これは、テーブル上で `myisamchk --medium-check tbl_name` を起動させるのと同じ結果になります。**CHANGED** か **FAST** が指定されない限り、静的フォーマットの **MyISAM** テーブルのデフォルトの確認タイプは、**MEDIUM** です。その場合、デフォルトは **QUICK** です。行はめったに破壊されないで、**CHANGED** と **FAST** の行スキャンは省かれます。

テーブルが正しく閉じられているかどうかの確認の為に簡単なチェックを行っている次の例のように、確認オプションを組み合わせる事も可能です。

```
CHECK TABLE test_table FAST QUICK;
```

注意:場合によっては **CHECK TABLE** でテーブルが変更されます。これは、テーブルが「corrupted」や「not closed properly」と印されているにも関わらず、**CHECK TABLE** がそのテーブル内に何の問題も発見できない時に起こります。この場合、**CHECK TABLE** はテーブルに `OK` の印をつけます。

もしテーブルが破損すると、その問題はデータ部分ではなく、ほとんどインデックス内にあるでしょう。これまでに紹介された全ての確認タイプはインデックスをくまなく確認するので、ほとんどのエラーを見つける事ができるはずです。

もし `OK` であると思われるテーブルを確認したければ、確認オプションは利用しない、または利用するのであれば **QUICK** オプションを利用しなければいけません。急いでいる為、**QUICK** がデータ ファイル中に何のエラーも見し

ないかもしれないという小さいリスクを負う事ができる場合には、後者を利用してください。(ほとんどの場合、通常の利用条件下であれば、MySQL はデータ ファイル中に何らかのエラーを発見するはずでず。その場合、そのテーブルは「corrupted」と印が付けられ、修復されるまでは利用できなくなります。)

FAST と **CHANGED** は、テーブルを頻繁に確認したい場合に、スクリプト(例えば、**cron** から実行されるように)から利用されるようになっています。ほとんどの場合、**FAST** は **CHANGED** より好まれます。(MyISAM コード内にバグを見つけた可能性がある場合のみ、この方法は好ましくありません。)

通常確認を起動した後で、MySQL が行を更新したりキーで行を見つけたりする時に、テーブルから変なエラーが発生する場合のみ **EXTENDED** が利用されます。これは通常確認が成功した場合は、めったに起こらないでしょう。

CHECK TABLE によって報告されたいくつかの問題は自動的に修復されません。

- Found row where the auto_increment column has the value 0.

これは、**AUTO_INCREMENT** インデックス カラムが0の値を含むテーブル内に行を持っている事を意味します。(UPDATE ステートメントを利用して、カラムを明示的に 0 に設定する事で、**AUTO_INCREMENT** カラムが 0 である行を作成する事が可能です。)

これ自体はエラーではないのですが、このテーブルを一度ダンプしそれを復旧させる、またはテーブル上に **ALTER TABLE** を実行する事を決めた時に問題を引き起こします。この場合、**AUTO_INCREMENT** カラムは、複製キーエラーのような問題を引き起こす可能性がある **AUTO_INCREMENT** カラムのルールに従って値を変更します。

警告を除去するには、カラム値を0以外の値に設定する為に **UPDATE** ステートメントを実行してください。

CHECKSUM TABLE 構文

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

CHECKSUM TABLE はテーブル チェックサムを報告します。

QUICK を利用すると、ライブ テーブル チェックサムが有効であればそれが報告され、有効でなければ **NULL** を利用できます。これはとても速いです。ライブ チェックサムは、テーブルを作成した時に **CHECKSUM=1** テーブル オプションを指定する事によって有効になります。これは現在は **MyISAM** テーブルにだけサポートされています。詳しくは **CREATE TABLE 構文** を参照してください。

EXTENDED を利用すると、テーブル全体が一行ごとに読まれ、チェックサムが計算されます。大きいテーブルでは時間がかかります。

もし **QUICK** と **EXTENDED** のどちらも指定されない場合、テーブル ストレージ エンジンがライブ チェックサムをサポートし、またそうでない場合にはテーブルをスキャンするのであれば、MySQL がライブ チェックサムを返します。

存在しないテーブルに対しては、**CHECKSUM TABLE** は **NULL** を返し、警告を生成します。

チェックサムの値はテーブル行フォーマットによって決まります。もし行フォーマットが変更されれば、チェックサムもまた変更されます。例えば、**VARCHAR** の格納フォーマットは MySQL 4.1 と 5.0 の間に変更されたので、もし 4.1のテーブルが MySQL 5.0 にアップグレードされると、チェックサム値も変更されるでしょう。

OPTIMIZE TABLE 構文

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

もしテーブルの大部分を削除したり、変数長行で何箇所もテーブルを変更した場合は(**VARCHAR**、**VARBINARY**、**BLOB**、または **TEXT** カラムを持つテーブル)、**OPTIMIZE TABLE** を利用しなければいけません。

削除された行はリンクされたリスト内で保存され、後続の **INSERT** 操作は古い行の位置を再利用します。使用されていないスペースの再要求をし、データ ファイルをデフラグするには **OPTIMIZE TABLE** を利用する事ができます。

このステートメントはテーブルに **SELECT** と **INSERT** 権限を要求します。

ほとんどの設定で、**OPTIMIZE TABLE** を利用する必要は全くありません。可変長行の更新を頻繁にするとしても、特定のテーブルに対してだけ、この作業を週に一度、または月に一度以上する必要はありません。

OPTIMIZE TABLE は **MyISAM** と **InnoDB** テーブルに対してのみ機能します。これは、**NDB** ディスク データ テーブルを含むその他のストレージ エンジンを利用して作成されたテーブルには機能しません。

MyISAM テーブルに対しては、**OPTIMIZE TABLE** は次のように機能します。

1. もしテーブルが行を削除したり分割したりすると、テーブルを修復します。
2. もしインデックス ページがソートされていなければ、それらをソートします。
3. もしテーブルの統計が最新でなければ、(そしてインデックスをソートする事で修復が完了されなければ)それらを更新します。

InnoDB テーブルに対しては、インデックス統計とクラスタされたインデックス内の使用されていないフリー スペースを更新する為にテーブルを復元する **ALTER TABLE** に **OPTIMIZE TABLE** がマップされます。

--skip-new か **--safe-mode** オプションを利用して **mysqld** をスタートさせる事で、**OPTIMIZE TABLE** が別のストレージ エンジン上で機能させる事ができます。この場合、**OPTIMIZE TABLE** は単に **ALTER TABLE** にマップされます。

OPTIMIZE TABLE は次のカラムを利用して結果セットを返します。

カラム	値
Table	テーブル名
Op	いつも optimize
Msg_type	status 、 error 、 info 、または warning の1つ
Msg_text	メッセージ

MySQL は **OPTIMIZE TABLE** が起動している最中にテーブルをロックするという事に注意してください。

OPTIMIZE TABLE ステートメントは、任意の **NO_WRITE_TO_BINLOG** キーワード(またはそのエイリアス **LOCAL**)が利用されない限り、バイナリ ログに書きこまれます。これは、複製マスタとして機能している MySQL サーバ上で利用される **OPTIMIZE TABLE** ステートメントが、複製スレーブにデフォルトで複製される為に行われます。

OPTIMIZE TABLE は、**POINT** カラム上の空間インデックスのような、R-tree インデックスをソートしません。(バグ #23578)

REPAIR TABLE 構文

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE は破損された可能性があるテーブルを修復します。これはデフォルトで、**myisamchk --recover tbl_name** と同じ効果を持っています。**REPAIR TABLE** は **MyISAM** と **ARCHIVE** テーブルに対して機能します。MySQL 5.1.9 から、**REPAIR** は **CSV** テーブルにも有効になりました。**MyISAM ストレージエンジン**、**ARCHIVE ストレージエンジン**、そして **CSV ストレージエンジン** を参照してください。

このステートメントはテーブルに **SELECT** と **INSERT** 権限を要求します。

通常は、このステートメントは一度も実行する必要はありません。しかし、もし大変な失敗をしてしまった時は **REPAIR TABLE** を利用すると全てのデータを **MyISAM** テーブルから取り戻す事ができます。もしテーブルが頻繁に破損する場合は、**REPAIR TABLE** を利用する必要性を減らす為にその原因を見つける努力が必要です。**What to Do If MySQL Keeps Crashing** と **MyISAM テーブルの問題点** を参照して下さい。

警告:もし **REPAIR TABLE** 操作の最中にサーバが停止してしまったら、再起動した直後に、他の操作をする前にすぐにもう一度 **REPAIR TABLE** ステートメントを実行しなければいけません。(バックアップを作成してスタートさせるのが良いでしょう。)最悪の場合、データ ファイルの情報を何も持たない新しい綺麗なインデックス ファイルを得て、その次に行う操作によってデータ ファイルが上書きされてしまう事もあるでしょう。実際に起こる可能性は低い事ですが、起こり得るシナリオです。

REPAIR TABLE は次のカラムを利用して結果セットを返します。

カラム	値

Table	テーブル名
Op	いつも <code>repair</code>
Msg_type	<code>status</code> 、 <code>error</code> 、 <code>info</code> 、または <code>warning</code> の1つ
Msg_text	メッセージ

`REPAIR TABLE` ステートメントは各修復済テーブルにたくさんの情報行を作成する可能性があります。最後の行は `status` と `Msg_text` の `Msg_type` 値を持ち、`Msg_text` は通常 `OK` になります。もし `OK` でなかったら、`myisamchk --safe-recover` を利用してテーブルを修復してみる必要があります。(`REPAIR TABLE` は `myisamchk` の全てのオプションをまだインプリメントしていません。) `myisamchk --safe-recover` を利用すると、`--max-record-length` のような `REPAIR TABLE` がサポートしていないオプションを利用する事ができます。

もし `QUICK` が与えられたら、`REPAIR TABLE` はインデックス ツリーのみを修復しようと試みます。このタイプの修正は `myisamchk --recover --quick` によって行われた物と似ています。

もし `EXTENDED` を利用すると、MySQL は1つのインデックスをソートしながら一度に作成する代わりに、1つの行ごとに作成します。このタイプの修正は `myisamchk --recover --quick` によって行われた物と似ています。

`REPAIR TABLE` に有効な `USE_FRM` モードもあります。もし `.MYI` インデックス ファイルがなくなっていたり、そのヘッダが破損していたらこれを利用してください。このモードでは、MySQL は `.frm` ファイルからの情報を利用して `.MYI` ファイルを再作成します。この種類の修復は `myisamchk` ではできません。注意:このモードは、通常の `REPAIR` モードを利用できない時のみ 利用してください。`.MYI` ヘッダは `REPAIR ... USE_FRM` 内で失われた重要なテーブルメタデータ(具体的には、現在の `AUTO_INCREMENT` 値と `Delete link`)を含んでいます。この情報は `.MYI` ファイル内にも格納されているので、テーブルが圧縮された時には `USE_FRM` を利用しないでください。

`REPAIR TABLE` ステートメントは、任意の `NO_WRITE_TO_BINLOG` キーワード(またはそのエイリアス `LOCAL`)が利用されない限り、バイナリ ログに書きこまれます。これは、複製マスタとして機能している MySQL サーバ上で利用される `REPAIR TABLE` ステートメントが、複製スレーブにデフォルトで複製される為に行われます。

RESTORE TABLE 構文

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

`RESTORE TABLE` は `BACKUP TABLE` で作成されたバックアップからテーブルを復旧します。ディレクトリは、完全なパス名として指定されなければいけません。

既存テーブルは上書きされません。もしそのようなテーブルを修復させようとするエラーが発生します。`BACKUP TABLE` と同じで、`RESTORE TABLE` は現在 `MyISAM` テーブルにしか機能しません。修復されたテーブルはマスタからスレーブに複製されません。

各テーブルのバックアップは、その `.frm` フォーマット ファイルと `.MYD` データ ファイルで構成されています。修復操作はそれらのファイルを修復し、そして `.MYI` インデックス ファイルを再構築する為にそれらを利用します。修復操作は、インデックスを再構築する必要がある為、バックアップ作業よりも時間がかかります。テーブルが長いインデックスを持っていれば、その分時間も長くなります。

`RESTORE TABLE` は次のカラムを利用して結果セットを返します。

カラム	値
Table	テーブル名
Op	いつも <code>restore</code>
Msg_type	<code>status</code> 、 <code>error</code> 、 <code>info</code> 、または <code>warning</code> の1つ
Msg_text	メッセージ

SET 構文

```
SET variable_assignment [, variable_assignment] ...
```

```
variable_assignment:
  user_var_name = expr
  | [GLOBAL | SESSION] system_var_name = expr
  | [@@global. | @@session. | @@]system_var_name = expr
```


SET ステートメントは、サーバやクライアントの操作に影響を与える、様々なタイプの変数に値を割り当てます。MySQL の古いバージョンは SET OPTION を採用していましたが、OPTION を持たない SET がより好まれるようになった為、この構文は廃止される事になりました。

このセクションでは、システム変数やユーザ変数に値を割り当てる SET の利用について説明します。変数に関するこれらのタイプの一般情報については システム変数 と ユーザによって定義された変数 を参照してください。システム変数も、システム変数の使用 で説明されているように、サーバ起動時に設定する事ができます。

SET 構文異型のいくつか、別のコンテキストで利用されています。

- SET PASSWORD はアカウント パスワードを割り当てます。詳しくは SET PASSWORD 構文 を参照してください。
- SET TRANSACTION ISOLATION LEVEL はトランザクション プロセスに分離レベルを設定します。詳しくは SET TRANSACTION 構文 を参照してください。
- SET は、ローカル ルーチン変数に値を割り当てる為のストアード ルーチン内で利用されます。詳しくは 変数 SET ステートメント を参照してください。

次の説明は、変数を設定する為に利用する事ができる、異なる SET 構文を表しています。例では = 代入演算子を利用していますが、:= 演算子も許容されています。

ユーザ変数は @var_name として書かれ、次のように設定されます。

```
SET @var_name = expr;
```

多くのシステム変数は動的であり、SET ステートメントを利用してサーバが起動している間に変更する事ができます。(リストについては、動的システム変数 をご覧ください。)

SET を利用してシステム変数を変更するには、任意で修飾子が先行する var_name として参照してください。

- 変数がグローバルである事を明示的に指示する為には、その名前に GLOBAL か @@global. を先行させてください。SUPER 権限はグローバル変数を設定する為に要求されます。
- 変数がセッションである事を明示的に指示する為には、その名前に SESSION か @@session. を先行させてください。セッション変数を設定するのに特別な権限は必要とされませんが、クライアントは自分自身のセッション変数以外は変更できませんので、別のクライアントの物は変更できません。
- LOCAL と @@local. は SESSION と @@session. の同義語です。
- もし修飾子が何もなければ、SET はセッション変数を変更します。

A SET ステートメントは、カンマで区切られた複数変数割り当てを含む事ができます。もし複数のシステム変数を設定したら、ステートメント内の一番最近の GLOBAL か SESSION 修飾子が、指定された修飾子を持たない次の変数に利用されます。

例：

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

SET を利用してシステム変数に値を割り当てた時、(起動オプションでは可能なように)値の中でサフィックス文字を利用する事はできません。しかし、その値は式の形を取る事ができます。

```
SET sort_buffer_size = 10 * 1024 * 1024;
```

システム変数の @@var_name 構文は、別のデータベースとの互換性の為にサポートされています。

もしセッション システム変数を変更すると、セッションが終了するまで、または変数を異なる値に変更するまではその値は効果を持ち続けます。別のクライアントは変更を見る事ができません。

もしグローバル システム変数を変更すると、その値はサーバが再起動するまでの間記憶され、次の接続に利用されます。(グローバル システム変数を永久的な物にするには、それをオプション ファイルに設定する必要があります。)そのグローバル変数にアクセスする全てのクライアントが変更を見る事ができます。しかしその変更は、変更が行われ

た後に接続するクライアントの、対応するセッション変数に対してだけ影響を与えます。グローバル変数の変更は、現在接続中のクライアントのセッション変数には影響を与えません。(SET GLOBAL ステートメントを発行するクライアントにも)

不正使用を防ぐ為に、もし SET GLOBAL を SET SESSION とだけ利用できる変数と共に利用したり、グローバル変数を設定する時に GLOBAL (または @@global.)を指定しなかったりすると、MySQL はエラーを作成します。

SESSION 変数を GLOBAL 値に、または GLOBAL 値をコンパイル インされた MySQL デフォルト値に設定するには、DEFAULT キーワードを利用してください。例えば、次の2つのステートメントは max_join_size のセッション値をグローバル値に設定する上で同一です。

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

全てのシステム変数を DEFAULT に設定できる訳では有りません。そのような場合、DEFAULT の利用はエラーを引き起こします。

@@ 修飾子の1つを利用する事で、式の中の特定のグローバル、またはシステム変数の値を参照する事ができます。例えば、このようにして SELECT ステートメント内の値を検索する事ができます。

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

@@var_name (@@global. か @@session. を指定しない場合)のような式でシステム変数を参照する時、MySQL はセッション値が存在すればそれを返し、そうでなければグローバル値を返します。(これは、毎回セッション値を参照する SET @@var_name = value とは異なります。)

システム変数名と値を表示するには、SHOW VARIABLES ステートメントを利用してください。(詳しくは SHOW VARIABLES 構文 を参照してください。)

次のリストは、非スタンダード構文を持つオプションについて説明しています。システム変数のリスト中に無い物は、システム変数 で説明されています。ここで説明されているオプションが SHOW VARIABLES で表示されていないとしても、SELECT を利用してそれらの値を得る事ができます。(CHARACTER SET と SET NAMES は例外です。)例:

```
mysql> SELECT @@AUTOCOMMIT;
+-----+
| @@AUTOCOMMIT |
+-----+
| 1 |
+-----+
```

これらのオプションは、大文字でも小文字でもかまいません。

- **AUTOCOMMIT = {0 | 1}**

オートコミット モードを設定してください。もし1に設定すると、テーブルの全ての変更がすぐに効果を発揮します。もし0に設定すると、トランザクションを受け入れる為には COMMIT、キャンセルするには ROLLBACK を利用しなければいけません。デフォルトで、AUTOCOMMIT で始まるクライアント接続は1を設定します。もし AUTOCOMMIT モードを0から1に変更すると、MySQL は開いているトランザクションの自動 COMMIT を実行します。トランザクションを始める別の方法は、START TRANSACTION か BEGIN ステートメントを利用する方法です。詳しくは START TRANSACTION、COMMIT、そして ROLLBACK 構文 を参照してください。

- **BIG_TABLES = {0 | 1}**

もし1に設定すると、全てのテンポラリ テーブルは、メモリの中ではなくディスク上に格納されます。これはスピードが少し遅いですが、エラー The table tbl_name is full は、大きいテンポラリ テーブルを必要とする SELECT オペレーションにこれは起こりません。新しい接続のデフォルト値は0です。(メモリ内のテンポラリ テーブルを利用してください。)通常、メモリ内のテーブルは要求に従って自動的にディスク ベース テーブルに変換されるので、この変数を設定する必要は全くありません。(注意:この変数は以前は SQL_BIG_TABLES という名前でした。)

- **CHARACTER SET {charset_name | DEFAULT}**

これは与えられたマッピングを利用して全ての文字列をクライアントとの間でマップします。MySQL ソース分布の中で sql/convert.cc を編集する事で新しいマッピングを追加する事ができます。SET CHARACTER SET は3つのセッション システム変数を設定します。character_set_client と character_set_results は与えられた文字セットに

設定され、`character_set_connection` は `character_set_database` の値に設定されます。詳しくは [接続のキャラクタセットおよび照合順序](#) を参照してください。

デフォルトのマッピングは、値 `DEFAULT` を利用する事で復旧できます。デフォルトはサーバの設定によって決まります。

`SET CHARACTER SET` の構文は、別のほとんどのオプションを設定している物とは異なる事を覚えておいて下さい。

- `FOREIGN_KEY_CHECKS = {0 | 1}`

もし1に設定すると(デフォルト)、InnoDB テーブルの外部キー制約が確認されます。もし0に設定すると、それらは無視されます。外部キー確認を無効にする事は、それらの親/子供関係に要求される順番とは異なる順番で InnoDB を再ロードするのに役立ちます。詳しくは [FOREIGN KEY 制約](#) を参照してください。

`FOREIGN_KEY_CHECKS` を0に設定する事もまた、データ定義ステートメントに影響を与えます。`DROP SCHEMA` は、スキーマ外のテーブルに参照される外部キーを持つテーブルを含んでいてもスキーマをドロップし、そして、`DROP TABLE` は別のテーブルに参照される外部キーを持つテーブルをドロップします。

注記

Setting `FOREIGN_KEY_CHECKS` を1に設定する事は、既存テーブル データのスキャンをトリガしません。従って、`FOREIGN_KEY_CHECKS=0` の最中にテーブルに追加された行の一貫性の照合はされません。

- `IDENTITY = value`

この変数は `LAST_INSERT_ID` 変数の同義語です。これは別のデータベース システムとの互換性の為に存在します。この値は `SELECT @@IDENTITY` を利用して読む事ができ、`SET IDENTITY` を利用して設定する事ができます。

- `INSERT_ID = value`

`AUTO_INCREMENT` 値を挿入する時、次の `INSERT` か `ALTER TABLE` ステートメントで使用される値を設定してください。これは主にバイナリ ログと共に利用されます。

- `LAST_INSERT_ID = value`

`LAST_INSERT_ID()` から返される値を設定してください。これは、テーブルを更新するステートメント内で `LAST_INSERT_ID()` を利用する時にバイナリ ログ内に格納されます。この変数を設定する事で、`mysql_insert_id()` C API 関数に返される値は更新されません。

- `NAMES {'charset_name' [COLLATE 'collation_name']} | DEFAULT}`

`SET NAMES` は3つのセッション システム変数 `character_set_client`、`character_set_connection`、そして `character_set_results` を与えられた文字セットに設定します。`character_set_connection` を `charset_name` に設定すると、`collation_connection` も `charset_name` のデフォルト照合に設定します。任意の `COLLATE` 条項は照合を明示的に指定するのに利用されるでしょう。詳しくは [接続のキャラクタセットおよび照合順序](#) を参照してください。

デフォルトのマッピングは、値 `DEFAULT` を利用する事で復旧できます。デフォルトはサーバの設定によって決まります。

`SET NAMES` の構文は、別のほとんどのオプションを設定している物とは異なる事を覚えておいて下さい。

- `ONE_SHOT`

このオプションは変数ではなく修飾子です。これは、文字セット、照合、そしてタイム ゾーンを設定する変数の効果に影響を与える為に利用する事ができます。`ONE_SHOT` は主に複製を目的として利用されます。文字セット、照合、そしてタイム ゾーン変数の値をローラーフォーワードで元の形を反映させる為に、`mysqlbinlog` はそれらをテンポラリに修正できるよう `SET ONE_SHOT` を利用します。`ONE_SHOT` は内部的な利用の為だけの物で、MySQL 5.0 以降のバージョンでは廃止予定です。

`ONE_SHOT` を許容された変数のセット以外の物と利用する事はできません。もし利用を試みると、次のようなエラーが発生します。

```
mysql> SET ONE_SHOT max_allowed_packet = 1;
ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for purposes
internal to the MySQL server
```

もし **ONE_SHOT** が許容された変数と共に利用されたら、それは要求されたとおりに変数を変更しますが、それは次の非 **SET** ステートメントに対してのみ行われます。その後、サーバは全ての文字セット、照合、そしてタイムゾーンに関連したシステム変数を、以前の値に再設定します。例:

```
mysql> SET ONE_SHOT character_set_connection = latin5;

mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;

mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_connection | latin5 |
| collation_connection | latin5_turkish_ci |
+-----+-----+

mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_connection | latin1 |
| collation_connection | latin1_swedish_ci |
+-----+-----+
```

- **SQL_AUTO_IS_NULL = {0 | 1}**

もし1に設定すると(デフォルト)、次の構造を利用する事で **AUTO_INCREMENT** カラムを含む最後にテーブルに挿入された行を見つける事ができます。

```
WHERE auto_increment_column IS NULL
```

この動作は、Access のようないくつかの ODBC プログラムによって利用されます。

- **SQL_BIG_SELECTS = {0 | 1}**

もし0に設定すると、MySQL は、実行するのにとても時間がかかる **SELECT** ステートメントを異常終了します。(これはオプチマイザが、分析された行の数が **max_join_size** の値を超えると判断したステートメントの事です。)これは、得策では無い **WHERE** ステートメントが発行された時に便利な物です。新しい接続のデフォルト値は、全ての **SELECT** ステートメントを許容する1です。

もし **max_join_size** システム変数を **DEFAULT** 以外の値に設定すると、**SQL_BIG_SELECTS** は0に設定されます。

- **SQL_BUFFER_RESULT = {0 | 1}**

もし1に設定すると、**SQL_BUFFER_RESULT** は結果が **SELECT** ステートメントからテンポラリ テーブルに置かれるように強制します。これは、MySQL がテーブル ロックを早く解除するのを助け、クライアントに結果を送るのに時間がかかる場合に有益な物になります。デフォルト値は0です。

- **SQL_LOG_BIN = {0 | 1}**

もし0に設定すると、クライアントの為にバイナリ ログはログされません。クライアントはこのオプションを設定する為に **SUPER** 権限を持つ必要があります。デフォルト値は1です。

- **SQL_LOG_OFF = {0 | 1}**

もし1に設定すると、このクライアントの為に通常のクエリ ログはログされません。クライアントはこのオプションを設定する為に **SUPER** 権限を持つ必要があります。デフォルト値は0です。

- **SQL_LOG_UPDATE = {0 | 1}**

この変数は廃止予定であり、**SQL_LOG_BIN** にマップされています。

- **SQL_NOTES = {0 | 1}**

もし1に設定すると(デフォルト)、**Note** レベルの警告が記録されます。もし0に設定すると **Note** 警告は抑制されます。**mysqldump** はダンプ ファイルの再ロードが、その操作のインテグリティに影響しないイベントに対して警告を発行しないよう、この変数を0に設定する為にアウトプットを含みます。

- **SQL_QUOTE_SHOW_CREATE** = {0 | 1}

もし1に設定すると(デフォルト)、サーバは **SHOW CREATE TABLE** と **SHOW CREATE DATABASE** ステートメントに識別子を引用します。もし0に設定すると、引用は無効になります。このオプションは、引用を必要とする識別子に対して複製が機能するように、デフォルトで使用可能となっています。**SHOW CREATE TABLE 構文** と **SHOW CREATE DATABASE 構文** を参照して下さい。

- **SQL_SAFE_UPDATES** = {0 | 1}

もし1に設定すると、MySQL は **WHERE** 条項が **LIMIT** 条項内でキーを利用しない **UPDATE** か **DELETE** ステートメントを異常終了します。これは、キーが正しく利用されずその為に多数の行を変更、または削除する **UPDATE** か **DELETE** ステートメントをキャッチする事を可能にします。デフォルト値は0です。

- **SQL_SELECT_LIMIT** = {value | DEFAULT}

SELECT ステートメントから返される最大行数。新しい接続のデフォルト値は「無制限」です。もしその制限を変更すると、デフォルト値は **DEFAULT** の **SQL_SELECT_LIMIT** 値を利用して復旧できます。

もし **SELECT** が **LIMIT** 条項を持っていたら、**LIMIT** は **SQL_SELECT_LIMIT** の値を上回ります。

SQL_SELECT_LIMIT は、ストアド ルーチン内で実行された **SELECT** ステートメントに適応しません。それは、クライアントに戻される結果セットを作成しない **SELECT** ステートメントには適応しません。これらはサブクエリの中に **SELECT** ステートメント、**CREATE TABLE ... SELECT**、そして **INSERT INTO ... SELECT** を含んでいます。

- **SQL_WARNINGS** = {0 | 1}

この変数は、もし警告が発生したら、単列 **INSERT** ステートメントが情報を作成するかどうかをコントロールします。デフォルトは0です。情報文字列を作成するには、値を0に設定して下さい。

- **TIMESTAMP** = {timestamp_value | DEFAULT}

このクライアントに時刻を設定してください。これは、もし行の格納にバイナリ ログを利用するなら、元のタイムスタンプを得る為に利用されます。**timestamp_value** は MySQL タイムスタンプではなく、ユニックス エポック タイムスタンプでなければいけません。

SET TIMESTAMP は **NOW()** に返された値に影響しますが、**SYSDATE()** に返された物には影響しません。これは、バイナリ ログ内のタイムスタンプ設定は **SYSDATE()** の起動に影響を持たないという意味です。サーバは、**SYSDATE()** が **NOW()** のエイリアスになるよう働きかける為に **--sysdate-is-now** オプションでスタートでき、その場合、**SET TIMESTAMP** は両方の関数に影響を与えます。

- **UNIQUE_CHECKS** = {0 | 1}

もし1に設定すると(デフォルト)、**InnoDB** テーブル内の第二インデックスの一意性チェックが行われます。もし0に設定すると、ストレージ エンジンにインプット データ内に複製キーが存在しないと仮定する事を許可されます。もし、ご自分のデータに一意性違反が無い事が確かであれば、これを0に設定して **InnoDB** に大きいテーブルをインポートする際のスピードを早くする事ができます。

この変数を0に設定する事は、ストレージ エンジンが複製キーを無視する事を 要求 する訳ではないと覚えておいて下さい。エンジンは複製キーの確認をし、もしそれらを発見したらエラーを発行する事が許可されています。

SHOW 構文

SHOW は、データベース、テーブル、カラム、またサーバのステータス情報などのような様々な情報を提供する多くの形を持っています。このセクションでは次のような物を紹介します。

```
SHOW AUTHORS
SHOW CHARACTER SET [LIKE 'pattern']
SHOW COLLATION [LIKE 'pattern']
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

```

SHOW CONTRIBUTORS
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION funcname
SHOW CREATE PROCEDURE procname
SHOW CREATE TABLE tbl_name
SHOW CREATE VIEW view_name
SHOW DATABASES [LIKE 'pattern']
SHOW ENGINE engine_name {LOGS | STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW [FULL] EVENTS
SHOW FUNCTION CODE sp_name
SHOW FUNCTION STATUS [LIKE 'pattern']
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW PROCEDURE CODE sp_name
SHOW PROCEDURE STATUS [LIKE 'pattern']
SHOW PLUGINS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW SCHEDULER STATUS
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW TRIGGERS
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]

```

SHOW ステートメントも、複製マスタとスレーブ マスタに関する情報を提供する形を持っており、それらは [複製ステートメント](#) で紹介されています。

```

SHOW BINARY LOGS
SHOW BINLOG EVENTS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS

```

もし与えられた **SHOW** ステートメントの構文が **LIKE 'pattern'** 部を含んでいたら、**'pattern'** は SQL **'%'** と **'_'** ワイルドカード文字を含む事ができる文字列になります。そのパターンは、一致する値へのステートメント アウトプットを制限するのに有効です。

いくつかの **SHOW** ステートメントは、どの行を表示するかを指定する事に対して柔軟性を提供する **WHERE** 条項も許容します。詳しくは [SHOW ステートメントへの拡張](#) を参照してください。

SHOW AUTHORS 構文

```
SHOW AUTHORS
```

SHOW AUTHORS ステートメントは、MySQL 上で働く人々の情報を表示します。それぞれの作者に対して、**Name**、**Location**、そして **Comment** 値を表示します。

このステートメントは、MySQL 5.1.3 で追加されました。

SHOW CHARACTER SET 構文

```
SHOW CHARACTER SET [LIKE 'pattern']
```

SHOW CHARACTER SET ステートメントは全ての有効な文字セットを表示します。これは、どの文字セット名が一致するかを指示する任意の **LIKE** 条項を取ります。例:

```

mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description           | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European  | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |

```

```
| latin5 | ISO 8859-9 Turkish      | latin5_turkish_ci | 1 |
| latin7 | ISO 8859-13 Baltic         | latin7_general_ci | 1 |
+-----+-----+-----+-----+
```

Maxlen カラムは、一文字を格納するのに必要な最大バイト数を表示します。

SHOW COLLATION 構文

```
SHOW COLLATION [LIKE 'pattern']
```

SHOW COLLATION からのアウトプットは全ての有効な文字セットを含んでいます。これは、どの **pattern** が、どの照合名と一致するかを指示する任意の **LIKE** 条項を取ります。例:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | | | 0 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 0 |
| latin1_danish_ci | latin1 | 15 | | | 0 |
| latin1_german2_ci | latin1 | 31 | | Yes | 2 |
| latin1_bin | latin1 | 47 | | Yes | 0 |
| latin1_general_ci | latin1 | 48 | | | 0 |
| latin1_general_cs | latin1 | 49 | | | 0 |
| latin1_spanish_ci | latin1 | 94 | | | 0 |
+-----+-----+-----+-----+-----+-----+
```

Default カラムは、照合がその文字セットにとってデフォルトであるかどうかを指示します。**Compiled** はその文字セットがサーバ内にコンパイルされるかどうかを指示します。**Sortlen** は、文字セットの中の文字列式をソートする為に必要とされるメモリの量と関係しています。

SHOW COLUMNS 構文

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

SHOW COLUMNS は与えられたテーブル内のカラムに関する情報を表示します。これはビューに対しても機能します。

FULL キーワードは、各カラムに対するプレ カラム コメントと同じように、アウトプットが今持っている権限を含むように働きかけます。

db_name.tbl_name を **tbl_name FROM db_name** 構文の代替として利用する事ができます。言い換えると、これらの2つのステートメントは同等という事です。

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW FIELDS は

SHOW COLUMNS の同義語です。テーブルのカラムを **mysqlshow db_name tbl_name** コマンドを利用してリストにする事もできます。

DESCRIBE ステートメントは **SHOW COLUMNS** に似た情報を提供します。詳しくは **DESCRIBE 構文** を参照してください。

SHOW CONTRIBUTORS 構文

```
SHOW CONTRIBUTORS
```

SHOW CONTRIBUTORS ステートメントは、MySQL のソ - スに貢献した人や、MySQL AB サポートを引き起こす情報を表示します。それぞれの貢献者に対して、**Name**、**Location**、そして **Comment** 値を表示します。

このステートメントは、MySQL 5.1.12 で追加されました。

SHOW CREATE DATABASE 構文


```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

与えられたデータベースを作成する [CREATE DATABASE](#) ステートメントを表示します。[SHOW CREATE SCHEMA](#) は [SHOW CREATE DATABASE](#) の同義語です。

```
mysql> SHOW CREATE DATABASE test\G
***** 1. row *****
Database: test
Create Database: CREATE DATABASE `test`
/*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
***** 1. row *****
Database: test
Create Database: CREATE DATABASE `test`
/*!40100 DEFAULT CHARACTER SET latin1 */
```

[SHOW CREATE DATABASE](#) は [SQL_QUOTE_SHOW_CREATE](#) オプションの値に従ってテーブルとカラム名を引用します。詳しくは [SET 構文](#) を参照してください。

SHOW CREATE EVENT

```
SHOW CREATE EVENT event_name
```

このステートメントは、与えられたイベントを再作成する為に必要な [CREATE EVENT](#) ステートメントを表示します。例えば、(同じイベント [e_daily](#) が定義され、その後 [SHOW EVENTS](#) に変更された物を利用):

```
mysql> SHOW CREATE EVENT test.e_daily\G
***** 1. row *****
Event: e_daily
Create Event: CREATE EVENT e_daily
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
ENABLE
COMMENT 'Saves total number of sessions and
clears the table once per day.'
DO
BEGIN
INSERT INTO site_activity.totals (when, total)
SELECT CURRENT_TIMESTAMP, COUNT(*)
FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END
```

アウトプットは、それが作成されたステータスよりも、イベントの現在のステータス([ENABLE](#))を反映するという事を覚えておいて下さい。

このステートメントは、MySQL 5.1.6 でインプリメントされました。

SHOW CREATE PROCEDURE と SHOW CREATE FUNCTION 構文

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

これらのステートメントは MySQL 拡張子です。それらは、[SHOW CREATE TABLE](#) と似て、名づけられたルーチンを再作成する為に利用できる精密な文字列を返します。そのステートメントは、あなたがそのルーチンの持ち主になるか、[mysql.proc](#) テーブルに [SELECT](#) アクセスを持つ事を要求します。

```
mysql> SHOW CREATE FUNCTION test.hello\G
***** 1. row *****
Function: hello
sql_mode:
Create Function: CREATE FUNCTION `test`.`hello` (s CHAR(20)) »
RETURNS CHAR(50)
RETURN CONCAT('Hello, 's, '!')
```

SHOW CREATE TABLE 構文

```
SHOW CREATE TABLE tbl_name
```

与えられたテーブルを作成する **CREATE TABLE** ステートメントを表示します。このステートメントも、ビューと共に機能します。

```
mysql> SHOW CREATE TABLE t1G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM
```

SHOW CREATE TABLE は **SQL_QUOTE_SHOW_CREATE** オプションの値に従ってテーブルとカラム名を引用します。詳しくは **SET 構文** を参照してください。

SHOW CREATE VIEW 構文

```
SHOW CREATE VIEW view_name
```

このステートメントは、与えられたビューを作成する **CREATE VIEW** ステートメントを表示します。

```
mysql> SHOW CREATE VIEW v;
+-----+-----+
| View | Create View |
+-----+-----+
| v   | CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b` |
+-----+-----+
```

SHOW CREATE VIEW の利用は、問題になっているビューに **SHOW VIEW** 権限と **SELECT** 権限を必要とします。

VIEWS テーブルを含む **INFORMATION_SCHEMA** から、ビュー オブジェクトに関する情報を得る事ができます。詳しくは **INFORMATION_SCHEMA VIEWS テーブル** を参照してください。

SHOW DATABASES 構文

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

SHOW DATABASES は MySQL サーバ ホスト上のデータベースをリストにします。**SHOW SCHEMAS** は **SHOW DATABASES** の同義語です。

グローバル **SHOW DATABASES** 権限を持っていない限り、自分が何かしらの権限を持つデータベースしか見る事ができません。**mysqlshow** コマンドを利用してこのリストを手に入れる事もできます。

もしサーバが **--skip-show-database** オプションを利用してスタートしたら、**SHOW DATABASES** 権限を持っていない限り、このステートメントを利用する事は絶対にできません。

SHOW SCHEMAS を利用する事もできます。

SHOW ENGINE 構文

```
SHOW ENGINE engine_name {LOGS | STATUS | MUTEX}
```

SHOW ENGINE はストレージ エンジンに関するログやステータス情報を表示します。現在次のステートメントがサポートされています。

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE NDB STATUS
```

SHOW ENGINE INNODB STATUS と **SHOW ENGINE INNODB MUTEX** の古い(または廃止された)同義語は **SHOW INNODB STATUS** と **SHOW MUTEX STATUS** です。

SHOW ENGINE INNODB STATUS は InnoDB ストレージエンジンの状態に関する広範囲な情報を表示します。

InnoDB モニタは InnoDB 処理に関する追加情報を提供します。詳しくは [SHOW ENGINE INNODB STATUS](#) と [InnoDB モニタ](#) を参照してください。

[SHOW ENGINE INNODB MUTEX](#) は InnoDB ミューテックス統計を表示します。アウトプット フィールドは次に紹介されています。

- **Type**

常に InnoDB です。

- **Name**

それがインプリメントされたミューテックス名とソース ファイル。例:&pool->mutex:mem0pool.c

ミューテックス名はその目的を指示します。例えば、[log_sys](#) ミューテックスは InnoDB ログ サブシステムに利用され、ログ活動がどれほど集中しているのかを指示します。[buf_pool](#) ミューテックスは InnoDB バッファ プールを保護します。

- **Status**

ミューテックス ステータスフィールドはいくつかの値を含んでいます。

- [count](#) は、ミューテックスが何回要求されたかを指示します。
- [spin_waits](#) はスピンロックが何回起動しなければいけなかったかを指示します。
- [spin_rounds](#) はスピンロック ラウンドの数を指示します。([spin_rounds](#) を [spin_waits](#) で割ると、平均ラウンド カウントがわかります。)
- [os_waits](#) は OS の待機数を指示します。これは、スピンロックが機能しなかった時に起こります。(ミューテックスはスピンロックの最中にロックされておらず、OSに従い、待つ必要がありました。)
- [os_yields](#) はスレッドがミューテックスをロックしようと試みて、そのタイムスライスを放棄し、OSに従う回数を指示します。(別のスレッドが起動する事を許可すると、ミューテックスをロックする為にそれを自由にするという仮定の下。)
- [os_wait_times](#) は、もし [timed_mutexes](#) システム変数が1であれば(ON)、OS 待機にかかった時間を(分で)指示します。もし [timed_mutexes](#) が0であれば(OFF)タイミングが無効になるので、[os_wait_times](#) は0です。[timed_mutexes](#) はデフォルトでオフになっています。

このステートメントからの情報は、システムの問題を診断するのに利用する事ができます。例えば、[spin_waits](#) と [spin_rounds](#) の大きい値は拡張性の問題を指示するでしょう。

もしサーバが、有効な [NDBCLUSTER](#) ストレージ エンジンを持っていたら、[SHOW ENGINE NDB STATUS](#) は接続されたデータノード、クラスタ接続、そしてクラスタ ビンログ エポックなどのクラスタ ステータス情報を表示します。

[SHOW ENGINE NDB STATUS](#) からのアウトプット例はここに表されています。— MySQL 5.0 内のステートメントによって表示されていた物からかなり変更されている事に注意してください。

```
mysql> SHOW ENGINE NDB STATUS\G
***** 1. row *****
Type: ndbcluster
Name: connection
Status: cluster_node_id=6, connected_host=192.168.0.179,
connected_port=1186, number_of_storage_nodes=4,
number_of_ready_storage_nodes=4, connect_count=0
***** 2. row *****
Type: ndbcluster
Name: binlog
Status: latest_epoch=0, latest_trans_epoch=2226134,
latest_received_binlog_epoch=0, latest_handled_binlog_epoch=0,
latest_applied_binlog_epoch=0
2 rows in set (0.00 sec)
```

In MySQL 5.0 では、[SHOW ENGINE INNODB MUTEX](#) は [SHOW MUTEX STATUS](#) として呼び出されます。後者のステートメントは似たような情報を表示しますが、それは少し異なるアウトプット フォーマットになります。

SHOW ENGINE BDB LOGS は以前は **BDB** ログ ファイルのステータス情報を表示しました。MySQL 5.1.12 にもあるように、**BDB** ストレージ エンジン もうサポートされていませんし、このステートメントは警告を作成します。

SHOW ENGINES 構文

SHOW [STORAGE] ENGINES

SHOW ENGINES はサーバのストレージ エンジンについてのステータス情報を表示します。これは特に、ストレージ エンジンがサポートされているのか、またはデフォルト エンジンが何なのかを確認するのに便利です。**SHOW TABLE TYPES** は廃止予定の同義語です。

```
mysql> SHOW ENGINES\G
***** 1. row *****
  Engine: MEMORY
  Support: YES
  Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
  XA: NO
  Savepoints: NO
***** 2. row *****
  Engine: MyISAM
  Support: DEFAULT
  Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
  XA: NO
  Savepoints: NO
***** 3. row *****
  Engine: InnoDB
  Support: YES
  Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
  XA: YES
  Savepoints: YES
***** 4. row *****
  Engine: EXAMPLE
  Support: YES
  Comment: Example storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 5. row *****
  Engine: ARCHIVE
  Support: YES
  Comment: Archive storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 6. row *****
  Engine: CSV
  Support: YES
  Comment: CSV storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 7. row *****
  Engine: BLACKHOLE
  Support: YES
  Comment: /dev/null storage engine (anything you write »
         to it disappears)
Transactions: NO
  XA: NO
  Savepoints: NO
***** 8. row *****
  Engine: FEDERATED
  Support: YES
  Comment: Federated MySQL storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 9. row *****
  Engine: MRG_MYISAM
  Support: YES
  Comment: Collection of identical MyISAM tables
```

```
Transactions: NO
XA: NO
Savepoints: NO
```

SHOW ENGINES からのアウトプットは、使用される MySQL バージョンや別の要因によって変わります。**Support** カラム内に表されている値は、ここに表されているように、別の機能に対するサーバのサポート レベルを指示します。

値	意味
YES	機能はサポートされており、アクティブです。
NO	機能はサポートされていません。
DISABLED	機能はサポートされていますが、無効になっています。

NO の値は、サーバはその機能に対するサポート無しでコンパイルされた為、ランタイムに起動する事はできないという事を意味します。

DISABLED の値は、サーバがその機能を無効にするオプションを利用してスタートされたか、それを有効にする為に必要な全てのオプションが与えられなかった為に起こります。後者の場合、エラー ログ ファイルは、なぜオプションが無効になったのかを指示する理由を含んでいるはずでです。詳しくは [エラー ログ](#) を参照してください。

もし、サーバがストレージ エンジンの **DISABLED** をサポートする為にコンパイルされたのに、**--skip-engine** オプションを利用してスタートされたら、それも発見するかもしれません。例えば、**--skip-innodb** は **InnoDB** エンジンが無効にします。**NDB Cluster** ストレージ エンジンにとっては、**DISABLED** は、サーバは MySQL クラスタへのサポートを利用してコンパイルされたが、スタートするのに **--ndb-cluster** オプションは利用されなかった、という事を意味します。

全ての MySQL サーバは、**MyISAM** がデフォルトのストレージ エンジンなので、**MyISAM** テーブルをサポートします。

Transactions、**XA**、そして **Savepoints** カラムが MySQL 5.1.2 で追加されました。それらはそれぞれストレージ エンジンが、トランザクション、XA トランザクション、そしてセーブポイントをサポートするかどうかを指示します。

SHOW ERRORS 構文

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

このステートメントは、エラー、警告、そして注意を表示する代わりに、エラーのみを表示するという事以外、**SHOW WARNINGS** と似ています。

LIMIT 条項は **SELECT** ステートメントに対するのと同じ構文を持っています。詳しくは [SELECT 構文](#) を参照してください。

SHOW COUNT(*) ERRORS ステートメントはエラーの数を表示します。**error_count** 変数からもこの数字を検索する事ができます。

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

更なる情報については、[SHOW WARNINGS 構文](#) を参照してください。

SHOW EVENTS

```
SHOW EVENTS [FROM schema_name] [LIKE pattern]
```

SHOW EVENTS は、その一番シンプルな形で、現在のスキーマ内の全てのイベントをリストにします。

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+-----+-----+
| CURRENT_USER() | SCHEMA() |
+-----+-----+
| jon@ghidora   | myschema |
+-----+-----+
```

```

1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
***** 1. row *****
      Db: myschema
     Name: e_daily
    Definer: jon@ghidora
     Type: RECURRING
  Execute at: NULL
Interval value: 10
Interval field: INTERVAL_SECOND
      Starts: 2006-02-09 10:41:23
       Ends: 0000-00-00 00:00:00
      Status: ENABLED
1 row in set (0.01 sec)

```

SHOW EVENTS のアウトプット内のカラム — `INFORMATION_SCHEMA.EVENTS` テーブル内のカラムに似ているけれど同一ではない — がここに表されています。

- **Db:** イベントが定義されるスキーマ(データベース)
- **Name:** イベント名。
- **Definer:** イベントを作成したユーザアカウント。(username@hostname)
- **Type:** `ONE TIME` (一時的な)か `RECURRING` の2つのうちの1つの値。
- **Execute At:** 一時的なイベントが実行される時の日付と時刻。 `DATETIME` 値として表示されます。
自動更新イベントにとっては、このカラムの値はいつでも `NULL` です。
- **Interval Value:** 自動更新イベントの為に、イベント実行の間のインターバルの回数。
一時的イベントにとっては、このカラムの値はいつでも `NULL` です。
- **Interval Field:** 自動更新イベントが、次の実行までの間のインターバルに利用される時間単位。
一時的イベントにとっては、このカラムの値はいつでも `NULL` です。
- **Starts:** 自動更新イベント開始の日付と時間。これは `DATETIME` 値として表示され、これはそのイベントに開始の日付と時間が定義されない時は空白です。(MySQL 5.1.8 以前は、このような場合 `'0000-00-00 00:00:00'` がデフォルトでした。)
一時的イベントにとっては、このカラムの値はいつでも `NULL` です。
- **Ends:** 自動更新イベント終了の日付と時間。これは、そのイベント終了の日付と時刻が定義されていなければ、 `DATETIME` 値として表示され、デフォルトは `'0000-00-00 00:00:00'` になります。
一時的イベントにとっては、このカラムの値はいつでも `NULL` です。
- **Status:** イベント ステータス `ENABLED` か `DISABLED` のうちの1つです。

アクション ステートメントは `SHOW EVENTS` のアウトプット内に表示されていない事に注意してください。

注意: `Starts` と `Ends` (`'0000-00-00 00:00:00'` 以外)を表示する値は、現在はユニバーサル タイムを利用しています。しかし、ユニバーサル タイムのこのコンテキスト内での利用は変更される予定ですので、アプリケーション内でこれに依存するべきでは有りません。(バグ #16420) `INFORMATION_SCHEMA.EVENTS` テーブル もご参照ください。

異なるスキーマのイベントを見るには、`FROM` 条項を利用できます。例えば、もし `test` スキーマが前出の例の中で選択されていたら、次のステートメントを利用して `myschema` 上で定義されたイベントを見る事ができました。

```
SHOW EVENTS FROM myschema;
```

`LIKE` にパターンを1つプラスした物を利用して、イベント名上でこのステートメントに返されたリストをフィルタする事ができます。

このステートメントは、MySQL 5.1.6 で追加されました。

[INFORMATION_SCHEMA EVENTS テーブル](#) もご参照下さい。

注意:MySQL 5.1.11 とそれ以前のバージョンでは、[SHOW EVENTS](#) は現在のユーザが定義者であるイベントだけを表示し、[SHOW FULL EVENTS](#) ステートメントは、与えられたスキーマ上で全てのユーザによって定義されたイベントを見る為に利用されていました。[SHOW FULL EVENTS](#) は MySQL 5.1.12 で削除されました。

SHOW GRANTS 構文

```
SHOW GRANTS [FOR user]
```

このステートメントは、MySQL ユーザ アカウントに供与された権限を複製する為に発行されなければいけない [GRANT](#) ステートメントをリストにします。アカウントは、例えば 'jeffrey'@'localhost' のように [GRANT](#) ステートメントと同じフォーマットを利用して名づけられます。もしアカウント名のユーザ名部分だけを指定すると、ホスト名の '%' 部分が利用されます。アカウント名の指定についての追加情報に関しては、[GRANT 構文](#) を参照してください。

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+-----+
| Grants for root@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----+
```

サーバに接続する為に利用している、アカウントに供与された権限をリストにする為に、次のステートメントを利用する事ができます。

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

MySQL 5.1.12 にも有るように、もし [SHOW GRANTS FOR CURRENT_USER](#) (またはそれと同等な構文)が [DEFINER](#) コンテキスト内で利用されたら、[SQL SECURITY DEFINER](#) を利用して定義されたストアドプロシージャ内などで、表示された供与物は、呼び出し元の物ではなく、定義者の物です。

[SHOW GRANTS](#) は名づけられたアカウントに明示的に供与された権限のみを表示します。そのアカウントに有効なその他の権限もあるかもしれませんが、それらは表示されません。例えば、もし匿名アカウントが存在したら、名づけられたアカウントはその権限を利用する事ができるかもしれませんが、[SHOW GRANTS](#) はそれらを表示しません。

SHOW INDEX 構文

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

[SHOW INDEX](#) はテーブル インデックス情報を返します。そのフォーマットは、ODBC 内の [SQLStatistics](#) コールのそれと似ています。

[SHOW INDEX](#) は次のフィールドを返します。

- [Table](#)
テーブル名。
- [Non_unique](#)
もしインデックスが複製を含む事ができなければ0、もしできるなら1。
- [Key_name](#)
インデックス名
- [Seq_in_index](#)
1から始まる、インデックス内のカラム シーケンス番号
- [Column_name](#)

カラム名

- **Collation**

カラムがインデックス内でどのようにソートされるか。MySQL では、これは値 'A' (昇順)か **NULL** (格納されない)を持つ事ができます。

- **Cardinality**

インデックス内の固有値数の見積もりこれは、**ANALYZE TABLE** か **myisamchk -a** を起動させる事で更新されます。**Cardinality** は、整数として格納された統計に基づいてカウントされるので、小さいテーブルに対してもその値は必ずしも精密ではありません。濃度が高ければ、その分 MySQL が接合を行う時にインデックスを利用する可能性は高くなります。

- **Sub_part**

もしカラムが部分的にだけインデックスされていた時のインデックスされる文字数、もしカラム全体がインデックスされていた時は **NULL** です。

- **Packed**

キーがどのようにパックされるのかを指示します。もしそうでなければ **NULL** です。

- **Null**

もしカラムが **NULL** を含んでいたら、**YES** を含みます。もしそうでなければ、カラムは **NO** を含みます。

- **Index_type**

使用されるインデックス方法(**BTREE**、**FULLTEXT**、**HASH**、**RTREE**)

- **Comment**

様々な意見

db_name.tbl_name を **tbl_name FROM db_name** 構文の代替として利用する事ができます。これらの2つのステートメントは同等です。

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

SHOW KEYS は

SHOW INDEX の同義語です。テーブルのインデックスを **mysqlshow db_name tbl_name** コマンドを利用してリストにする事もできます。

SHOW INNODB STATUS 構文

```
SHOW INNODB STATUS
```

MySQL 5.1 では、これは廃止予定の **SHOW ENGINE INNODB STATUS** の同義語です。詳しくは **SHOW ENGINE 構文** を参照してください。

SHOW OPEN TABLES 構文

```
SHOW OPEN TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW OPEN TABLES は、現在テーブル キャッシュ内で開かれている非 **TEMPORARY** テーブルをリストします。詳しくは **MySQL でのテーブルのオープンとクローズの方法** を参照してください。

SHOW OPEN TABLES は次のフィールドを返します。

- **Database**

テーブルを含むデータベース

- [Table](#)

テーブル名。

- [In_use](#)

クエリによってテーブルが現在使用されている回数。もしカウントがゼロなら、そのテーブルは開いていますが現在には利用されていません。

- [Name_locked](#)

テーブル名がロックされているかどうか名前ロックは、ドロップやテーブルのリネームのような操作に利用されません。

SHOW PLUGINS 構文

SHOW PLUGINS

[SHOW PLUGINS](#) は既知のプラグインの情報を表示します。

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+
| Name      | Status | Type      | Library |
+-----+-----+-----+-----+
| MEMORY    | ACTIVE | STORAGE ENGINE | NULL |
| MyISAM    | ACTIVE | STORAGE ENGINE | NULL |
| InnoDB    | ACTIVE | STORAGE ENGINE | NULL |
| ARCHIVE   | ACTIVE | STORAGE ENGINE | NULL |
| CSV       | ACTIVE | STORAGE ENGINE | NULL |
| BLACKHOLE | ACTIVE | STORAGE ENGINE | NULL |
| FEDERATED | ACTIVE | STORAGE ENGINE | NULL |
| MRG_MYISAM | ACTIVE | STORAGE ENGINE | NULL |
+-----+-----+-----+-----+
```

[SHOW PLUGIN](#) は MySQL 5.1.5 で追加され、5.1.9 で [SHOW PLUGINS](#) という名前に変わりました。(5.1.9 では [SHOW PLUGIN](#) は廃止予定で、警告を發します。)

SHOW PRIVILEGES 構文

SHOW PRIVILEGES

[SHOW PRIVILEGES](#) は MySQL サーバがサポートするシステム権限のリストを表示します。権限リストの詳細内容は、使用サーバのバージョンによって決まります。

```
mysql> SHOW PRIVILEGES\G
***** 1. row *****
Privilege: Alter
Context: Tables
Comment: To alter the table
***** 2. row *****
Privilege: Alter routine
Context: Functions,Procedures
Comment: To alter or drop stored functions/procedures
***** 3. row *****
Privilege: Create
Context: Databases,Tables,Indexes
Comment: To create new databases and tables
***** 4. row *****
Privilege: Create routine
Context: Functions,Procedures
Comment: To use CREATE FUNCTION/PROCEDURE
***** 5. row *****
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
...
```

SHOW PROCEDURE CODE と SHOW FUNCTION CODE 構文

```
SHOW {PROCEDURE | FUNCTION} CODE sp_name
```

これらのステートメントは、デバッグ サポートを利用して構築されたサーバに対してだけ有効な MySQL 拡張子です。それらは名づけられたルーチンの内部インプリメンテーション表現を表示します。そのステートメントは、あなたがそのルーチンの持ち主になるか、`mysql.proc` テーブルに `SELECT` アクセスを持つ事を要求します。

もし名づけられたルーチンが有効なら、各ステートメントは結果セットを作成します。結果セット内の各行は、ルーチン内の1つの「instruction」に対応します。最初のカラムは、0で始まる序数 `Pos` です。2つ目のカラムは、SQL ステートメントや(通常元のソースから変更された物)ストアド ルーチン ヘッダに対してだけ意味を持つコマンドを含む `Instruction` です。

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
-> BEGIN
-> DECLARE fanta INT DEFAULT 55;
-> DROP TABLE t2;
-> LOOP
-> INSERT INTO t3 VALUES (fanta);
-> END LOOP;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW PROCEDURE CODE p1//
+-----+-----+
| Pos | Instruction |
+-----+-----+
| 0 | set fanta@0 55 |
| 1 | stmt 9 "DROP TABLE t2" |
| 2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
| 3 | jump 2 |
+-----+-----+
4 rows in set (0.00 sec)
```

この例の中では、非実行可能 `BEGIN` と `END` ステートメントは無くなっており、`DECLARE variable_name` ステートメントに対しては、実行可能な部分だけが現れています。(デフォルトが割り当てられている部分)ソースから取り出された各ステートメントに対しては、後にタイプが続くコード文字 `stmt` があります。(9は `DROP`、5は `INSERT` を意味する、という感じの物です。)最終行は、`GOTO instruction #2` という意味を持つ `jump 2` 指示を含んでいます。

これらのステートメントは、MySQL 5.1.3 で追加されました。

SHOW PROCEDURE STATUS と SHOW FUNCTION STATUS 構文

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

これらのステートメントは MySQL 拡張子です。これらは、データベース、名前タイプ、作成者、そして作成日と変更日などのような、ルーチンの性質を返します。もしパターンが指定されなければ、どのステートメントを利用しているかによって、全てのストアド プロシージャや全てのストアド ファンクションの情報がリストされます。

```
mysql> SHOW FUNCTION STATUS LIKE 'hello\G
***** 1. row *****
      Db: test
     Name: hello
    Type: FUNCTION
  Definer: testuser@localhost
 Modified: 2004-08-03 15:29:37
   Created: 2004-08-03 15:29:37
Security_type: DEFINER
   Comment:
```

`INFORMATION_SCHEMA` 内の `ROUTINES` テーブルからストアド ルーチンに関する情報を得る事もできます。詳しくは [INFORMATION_SCHEMA ROUTINES テーブル](#) を参照してください。

SHOW PROCESSLIST 構文

```
SHOW [FULL] PROCESSLIST
```

SHOW PROCESSLIST はどのスレッドが起動しているかを表示します。`mysqladmin processlist` コマンドを利用してこの情報を手に入れる事もできます。

もし **PROCESS** 権限を持っていれば、全てのスレッドを見る事ができます。そうでなければ、自分自身のスレッドのみ見る事ができます。(使用中のMySQL アカウントと関連しているスレッド) 詳しくは **KILL** 構文を参照してください。もし **FULL** キーワードを利用しなければ、各ステートメントの最初の100文字だけが **Info** フィールドに表示されます。

MySQL Enterprise

MySQL #####
www-jp.mysql.com/products/enterprise/advisors.html #####

このステートメントは、「too many connections」エラー メッセージを受け取り、何が起きているのを確認したい時に大変役に立ちます。MySQL は、管理者がいつでもシステムに接続して確認できる事を保証する為、**SUPER** 権限を持つアカウントに利用される事ができる接続を1つ余分に確保します。(この権限を全てのユーザには与えていないと仮定しています。)

SHOW PROCESSLIST のアウトプットは、次のようになるでしょう。

```
mysql> SHOW FULL PROCESSLIST
***** 1. row *****
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave »
      I/O thread to update it
Info: NULL
***** 3. row *****
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 4. row *****
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 5. row *****
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

アウトプット カラムは次の意味を持っています。

- **Id**

接続識別子。

- User

ステートメントを発行した MySQL ユーザ。もしこれが `system user` であれば、タスクを内部的に取り扱う為に、サーバによって生み出された非クライアント スレッドを参照します。これは、複製スレーブが遅れた行のハンドラに利用される I/O または SQL スレッドになり得ます。 `event_scheduler` はスケジュールされたイベントをモニタするスレッドを参照する事ができます。 `system user` が `event_scheduler` には、Host カラム内で指定されたホストはありません。

- Host

ステートメントを発行するクライアントのホスト名(ホストが無い `system user` 以外)。 `SHOW PROCESSLIST` は、どのクライアントが何をしているかの究明を簡単にする為に、TCP/IP 接続のホスト名を `host_name:client_port` フォーマットで報告します。

- db

もし選択されれば、これがデフォルト データベースです。そうでなければ `NULL` です。

- Command

クライアント/サーバ プロトコルの `COM_xxx` コマンドに対応するカラムの値。詳しくは [ステータス変数](#) を参照してください。

Command 値は、次のうちのどれかでしょう。 Binlog Dump、Change user、Close stmt、Connect、Connect Out、Create DB、Daemon、Debug、Delayed insert、Drop DB、Error、Execute、Fetch、Field List、Init DB、Kill、Long Data、Ping、Prepare、Processlist、Query、Quit、Refresh、Register Slave、Reset stmt、Set option、Shutdown、Sleep、Statistics、Table Dump、Time

- Time

ステートメントやコマンドの開始から現在までの、秒表示での時間。

- State

次のうちのどれかになり得る、アクション、イベント、またはステート: After create、Analyzing、Changing master、Checking master version、Checking table、Connecting to master、Copying to group table、Copying to tmp table、Creating delayed handler、Creating index、Creating sort index、Creating table from master dump、Creating tmp table、Execution of init_command、FULLTEXT initialization、Finished reading one binlog; switching to next binlog、Flushing tables、Killed、Killing slave、Locked、Making temp file、Opening master dump table、Opening table、Opening tables、Processing request、Purging old relay logs、Queueing master event to the relay log、Reading event from the relay log、Reading from net、Reading master dump table data、Rebuilding the index on master dump table、Reconnecting after a failed binlog dump request、Reconnecting after a failed master event read、Registering slave on master、Removing duplicates、Reopen tables、Repair by sorting、Repair done、Repair with keycache、Requesting binlog dump、Rolling back、Saving state、Searching rows for update、Sending binlog event to slave、Sending data、Sorting for group、Sorting for order、Sorting index、Sorting result、System lock、Table lock、Thread initialized、Updating、User lock、Waiting for INSERT、Waiting for master to send event、Waiting for master update、Waiting for slave mutex on exit、Waiting for table、Waiting for tables、Waiting for the next event in relay log、Waiting on cond、Waiting to finalize termination、Waiting to reconnect after a failed binlog dump request、Waiting to reconnect after a failed master event read、Writing to net、allocating local table、cleaning up、closing tables、converting HEAP to MyISAM、copy to tmp table、creating table、deleting from main table、deleting from reference tables、discard_or_import_tablespace、end、freeing items、got handler lock、got old table、info、init、insert、logging slow query、login、preparing、purging old relay logs、query end、removing tmp table、rename、rename result table、reschedule、setup、starting slave、statistics、storing row into queue、unauthenticated user、update、updating、updating main table、updating reference tables、upgrading lock、waiting for delay_list、waiting for handler insert、waiting for handler lock、waiting for handler open、Waiting for event from ndbcluster

最も一般的な State 値はこのセクションの残りの部分で説明されています。それ以外のほとんどの State 値は、サーバ内のバグを見つける為にだけ役に立ちます。複製サーバのプロセス ステートについての追加情報については、[レプリケーション実装の詳細](#) も参照してください。

SHOW PROCESSLIST ステートメントに対しては `State` の値は `NULL` です。

- Info

スレッドが実行中のステートメント、または、もしそれがステートメントを何も実行していなければ `NULL` になります。

SHOW PROCESSLIST からのアウトプット内で主に見られるいくつかの `State` 値

- Checking table

スレッドがテーブル チェック操作を行っています。

- Closing tables

スレッドが変更されたテーブル データをディスクにフラッシュしている、そして使用されたテーブルを閉じているという意味です。この操作スピードは早いでしょう。もし速くなければ、ディスクがフルではないという事と、ディスクがそれほど頻繁に使用されていないという事を証明しなければいけません。

- Connect Out

複製スレーブはそのマスタに接続しています。

- Copying to group table

もしステートメントが異なる `ORDER BY` と `GROUP BY` 基準を持っていたら、行はグループによってソートされ、テンポラリ テーブルにコピーされます。

- Copying to tmp table

サーバはメモリ内のテンポラリ テーブルにコピーしています。

- Copying to tmp table on disk

サーバはディスク上のテンポラリ テーブルにコピーしています。テンポラリ結果セットは `tmp_table_size` よりも大きく、スレッドはメモリを保存する為にテンポラリ テーブルをイン メモリからディスク ベース フォーマットに変更しています。

- Creating tmp table

スレッドはクエリに結果の一部を保持する為にテンポラリ テーブルを作成しています。

- deleting from main table

サーバは複合テーブル削除の最初の部分を実行しています。最初のテーブルからの削除だけを行い、別の (参照) テーブルからの削除に利用されるフィールドとオフセットを保存しています。

- deleting from reference tables

サーバは複合テーブル削除の2番目の部分を行っており、別のテーブルから一致したテーブルを削除しています。

- Flushing tables

スレッドは `FLUSH TABLES` を実行しており、全てのスレッドがそのテーブルを閉じるのを待っています。

- FULLTEXT initialization

サーバは自然言語フル テキスト サーチを行う準備をしています。

- 停止する

誰かがスレッドに `KILL` ステートメントを送り、そして次にキル フラグを見つけた時異常終了するはずですが、そのフラグは MySQL 内の主要な各グループ内で確認されますが、場合によってはスレッドが停止するまでに少し時間がかかる場合があります。もしスレッドが別のスレッドにロックされていると、停止作業は別のスレッドがそのロックを解除するとすぐに効果を発揮します。

- [Locked](#)

クエリは別のクエリによってロックされています。

- [Sending data](#)

スレッドは [SELECT](#) ステートメントの為に行を作成し、また、クライアントにデータを送っています。

- [Sorting for group](#)

スレッドは [GROUP BY](#) を満足させる為にソートを行っています。

- [Sorting for order](#)

スレッドは [ORDER BY](#) を満足させる為にソートを行っています。

- [Opening tables](#)

スレッドはテーブルをオープンしようと試みています。この操作は、何かにオープンを邪魔されない限り、スピードが速いはずで。例えば、[ALTER TABLE](#) か [LOCK TABLE](#) ステートメントは、そのステートメントが終了するまでテーブルのオープンを邪魔する事ができます。

- [Reading from net](#)

サーバはネットワークからパケットを読み込んでいます。

- [Removing duplicates](#)

クエリは、MySQL が早い段階で独特な操作を最適化する事ができなくなるような方法で [SELECT DISTINCT](#) を利用していました。この為、MySQL は結果をクライアントに送る前に全ての複製行を削除する為の特別な段階を必要とします。

- [Reopen table](#)

スレッドはテーブルの為にロックを得ましたが、その後基礎となるテーブル構造が変更された事に気が付きました。それはロックを解除し、テーブルを閉じ、そして再度オープンしようとしています。

- [Repair by sorting](#)

修復コードはインデックスを作成する為にソートを利用しています。

- [Repair with keycache](#)

修復コードはキー キャッシュを通してキー作成を1つ1つ利用しています。これは [Repair by sorting](#) と比べるとかなり遅い作業です。

- [Searching rows for update](#)

スレッドは、全ての一致する行を更新する前にそれらを見つける為の第一段階を行っています。もし [UPDATE](#) が、関連する行を見つける為に利用されたインデックスを変更していれば、これが行われなければいけません。

- [Sleeping](#)

スレッドは新しいステートメントをスレッドに送る為のクライアントを待っています。

- [statistics](#)

サーバはクエリ実行計画を開発する統計を計算しています。

- [System lock](#)

スレッドは、テーブルの為に外部システム ロックを得るのを待っています。もし同じテーブルにアクセスする複数の [mysqld](#) サーバを利用していなければ、[--skip-external-locking](#) オプションを利用してシステム ロックを無効にする事ができます。

- [unauthenticated user](#)

クライアント接続への関連はしたが、そのクライアント ユーザの認証はまだ行われていないスレッドの状態。

- [Upgrading lock](#)

[INSERT DELAYED](#) ハンドラは行を挿入する為に、テーブルにロックを得ようとしています。

- [Updating](#)

スレッドは更新する行を探していて、それらを更新しています。

- [updating main table](#)

サーバは複合テーブル更新の最初の部分を実行しています。最初のテーブルの更新だけを行い、別の（参照）テーブルの更新に利用されるフィールドとオフセットを保存しています。

- [updating reference tables](#)

サーバは複合テーブル更新の2番目の部分を行っており、別のテーブルから一致したテーブルを更新しています。

- [User Lock](#)

スレッドは [GET_LOCK\(\)](#) 上で待っています。

- [Waiting for event from ndbcluster](#)

サーバは MySQL クラスタ内の SQL ノードとして機能しており、クラスタ管理ノードに接続されています。

- [Waiting for tables](#)

スレッドは、テーブルの基礎構造が変更され、その新しい構造を得る為にテーブルを再度オープンしなければいけないという通知を受け取りました。しかし、テーブルを再度オープンするには、他の全てのスレッドが問題になっているテーブルを閉じるまで待たなければいけません。

もし別のスレッドが [FLUSH TABLES](#) か、次にある問題のテーブル上のステートメントの中のひとつを利用すると、この通知が出されます。[FLUSH TABLES tbl_name](#)、[ALTER TABLE](#)、[RENAME TABLE](#)、[REPAIR TABLE](#)、[ANALYZE TABLE](#)、または [OPTIMIZE TABLE](#)

- [waiting for handler insert](#)

[INSERT DELAYED](#) ハンドラは全ての未解決の挿入を処理し、新しい物を待っています。

- [Writing to net](#)

サーバはネットワークにパケットを書き込んでいます。

ほとんどのステートが大変速い操作に対応します。もしスレッドがこれらのステートのどれかに何秒間も留まれば、調査が必要な問題があるかもしれません。

SHOW SCHEDULER STATUS 構文

SHOW SCHEDULER STATUS

このステートメントは、イベント スケジューラの状態に関連したデバッグ情報を提供します。これは、MySQL 5.1.11 の [-debug](#) 構造内でだけサポートされ、5.1.12 とそれ以降のリリースでは削除されました。

アウトプット例がここに表示されています。

```
+-----+-----+
| Name          | Value          |
+-----+-----+
| scheduler state      | INITIALIZED    |
| thread_id         | NULL           |
| scheduler last locked at | init_scheduler::313 |
| scheduler last unlocked at | init_scheduler::318 |
| scheduler waiting on condition | 0              |
| scheduler workers count | 0              |
```

```

| scheduler executed events | 0 |
| scheduler data locked    | 0 |
| queue element count      | 1 |
| queue data locked        | 0 |
| queue data attempting lock | 0 |
| queue last locked at      | create_event::218 |
| queue last unlocked at    | create_event::222 |
| queue last attempted lock at | ::0 |
| queue waiting on condition | 0 |
| next activation at        | 0-00-00 00:00:00 |
+-----+-----+

```

MySQL 5.1.12 以降の物では、この情報は `mysqladmin debug` を利用して得る事ができます。(詳しくは [mysqladmin — MySQL サーバの管理を行うクライアント](#) をご確認ください。) イベント スケジューラ ステータス情報を得る事に関する情報については、[Event Scheduler Status](#) を参照してください。

SHOW STATUS 構文

```
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
```

SHOW STATUS はサーバ ステータス情報を提供します。この情報は、`mysqladmin extended-status` コマンドを利用して得る事もできます。

部分的なアウトプットがここに表示されています。名前と値のリストは、お使いのサーバとは異なる場合があります。各変数の意味は [ステータス変数](#) で説明しています。

```

mysql> SHOW STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
| Bytes_sent | 1176560426 |
| Connections | 30023 |
| Created_tmp_disk_tables | 0 |
| Created_tmp_tables | 8340 |
| Created_tmp_files | 60 |
...
| Open_tables | 1 |
| Open_files | 2 |
| Open_streams | 0 |
| Opened_tables | 44600 |
| Questions | 2026873 |
...
| Table_locks_immediate | 1920382 |
| Table_locks_waited | 0 |
| Threads_cached | 0 |
| Threads_created | 30022 |
| Threads_connected | 1 |
| Threads_running | 1 |
| Uptime | 80380 |
+-----+-----+

```

LIKE 条項を利用すると、パターンと一致する名前を持つ変数の行のみを表示します。

```

mysql> SHOW STATUS LIKE 'Key%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_used | 14955 |
| Key_read_requests | 96854827 |
| Key_reads | 162040 |
| Key_write_requests | 7589728 |
| Key_writes | 3813196 |
+-----+-----+

```

GLOBAL 修飾子を利用すると、**SHOW STATUS** は MySQL への全ての接続のステータス値を表示します。**SESSION** を利用すると、現在の接続のステータス値を表示します。もし修飾子が無ければ、デフォルトは **SESSION** です。**LOCAL** は **SESSION** の同義語です。

いくつかのステータス変数は、グローバル値しか持っていません。それらに対しては、[GLOBAL](#) と [SESSION](#) の両方に同じ値を得ます。

SHOW TABLE STATUS 構文

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

[SHOW TABLE STATUS](#) は [SHOW TABLES](#) のように機能しますが、各テーブルに関する多くの情報を提供します。[mysqlshow --status db_name](#) コマンドを利用してこのリストを得る事もできます。

このステートメントもまたビューの情報を表示します。

[SHOW TABLE STATUS](#) は次のフィールドを返します。

- [Name](#)

テーブル名。

- [Engine](#)

テーブルのストレージ エンジン。詳しくは [13章ストレージエンジンとテーブルタイプ](#) を参照してください。

- [Version](#)

テーブルの .frm file. のバージョン番号。

- [Row_format](#)

行のストレージ フォーマット([Fixed](#)、[Dynamic](#)、[Compressed](#)、[Redundant](#)、[Compact](#))[InnoDB](#) テーブルのフォーマットは [Redundant](#) が [Compact](#) としてレポートされます。

- [Rows](#)

行数[MyISAM](#) のようないくつかのストレージ エンジンは、正確なカウントを格納します。[InnoDB](#) のような別のストレージ エンジンにとっては、この値はおおよその物であり、実際の値とは40から50%くらい異なります。そのような場合は、正確なカウントを得る為に [SELECT COUNT\(*\)](#) を利用してください。

[Rows](#) 値は [INFORMATION_SCHEMA](#) データベース内のテーブルには [NULL](#) です。

- [Avg_row_length](#)

平均行長

- [Data_length](#)

データ ファイルの長さ

- [Max_data_length](#)

データ ファイルの最大長データ ポインタ サイズが利用されたと仮定して、これはテーブル内に格納できるデータの総バイト数です。

- [Index_length](#)

インデックス ファイルの長さ

- [Data_free](#)

割り当てられたけれど使用されていないバイト数

- [Auto_increment](#)

次の [AUTO_INCREMENT](#) 値。

- [Create_time](#)

テーブルが作成された時。

- **Update_time**

データファイルが最後に更新された時。いくつかのストレージ エンジンに対しては、この値は **NULL** です。例えば、**InnoDB** はそのテーブルスペース内に複数のテーブルを格納し、データ ファイルタイムスタンプは適応しません。

- **Check_time**

テーブルが最後に確認された時。値が毎回 **NULL** の場合、全てのストレージ エンジンはこの時更新しません。

- **Collation**

テーブルの文字セットと照合。

- **Checksum**

ライブチェックサム値 (もしあれば)。

- **Create_options**

CREATE TABLE と共に利用される特別オプション。

- **Comment**

テーブルを作成する時に利用されるコメント(またはなぜ MySQL がテーブル情報にアクセスできなかったのかに関する情報)。

InnoDB テーブルは、テーブル コメントの中で、それが属する所にテーブルスペースのフリー スペースを報告します。共有テーブルスペースの中にあるテーブルには、これが共有テーブルスペースの空きスペースです。もし複数のテーブルスペースを利用して、そのテーブルが専用のテーブルスペースを所有していたら、そのフリー スペースはそのテーブルだけの物です。

MEMORY テーブルに対して、**Data_length**、**Max_data_length**、そして **Index_length** 値は割り当てられたメモリの実際の量を概算します。割り当てアルゴリズムは、割り当て操作の数を減らす為に、大量のメモリを確保します。

NDB Cluster テーブルに対して、**BLOB** カラムは考慮されないという例外はありますが、このステートメントのアウトプットは **Avg_row_length** と **Data_length** カラムのおおよその値を表します。さらに、レプリカの数 **Comment** カラム内に表示されます。(**number_of_replicas** として)

ビューに関しては、**Name** がビュー名を指示し、**Comment** が **view** という事以外、**SHOW TABLE STATUS** に表示される全てのフィールドは **NULL** です。

SHOW TABLES 構文

```
SHOW [FULL] TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW TABLES は、与えられたテーブル内で非 **TEMPORARY** テーブルをリストします。 **mysqlshow db_name** コマンドを利用してこのリストを得る事もできます。

このステートメントはデータベース内のビューもリストします。 **FULL** 修飾子は **SHOW FULL TABLES** が第2のアウトプット カラムを表示するようにサポートされています。第2カラムの値は、テーブルに対しては **BASE TABLE** で、ビューに対しては **VIEW** です。

注意:もしテーブルに権限を持っていなかったら、テーブルは **SHOW TABLES** が **mysqlshow db_name** からのアウトプット内に現れません。

SHOW TRIGGERS 構文

```
SHOW TRIGGERS [FROM db_name] [LIKE expr]
```

SHOW TRIGGERS は最近 MySQL サーバ上で定義されたトリガをリストします。このステートメントは **SUPER** 権限を必要とします。

トリガの使用 内で定義されている、トリガ `ins_sum` に対しては、このステートメントのアウトプットはここに表されているようになります。

```
mysql> SHOW TRIGGERS LIKE 'acc%\G
***** 1. row *****
Trigger: ins_sum
Event: INSERT
Table: account
Statement: SET @sum = @sum + NEW.amount
Timing: BEFORE
Created: NULL
sql_mode:
Definer: myname@localhost
```

注意: `LIKE` 条項を `SHOW TRIGGERS` と利用する時、一致する式(`expr`) は、トリガ名ではなく、トリガが宣言されたテーブル名と比較されます。

```
mysql> SHOW TRIGGERS LIKE 'ins%';
Empty set (0.01 sec)
```

このステートメントのアウトプット内のコラムに関する簡単な説明はここに表されています。

- **Trigger**
トリガ名。
- **Event**
トリガを有効化するイベント: `'INSERT'`、`'UPDATE'`、または `'DELETE'` の1つ。
- **Table**
トリガが定義されるテーブル。
- **Statement**
トリガが有効化された時に実行されるステートメント。これは `INFORMATION_SCHEMA.TRIGGERS` の `ACTION_STATEMENT` 内に表されているテキストと同じです。
- **Timing**
`'BEFORE'` か `'AFTER'` の2つの値の1つ。
- **Created**
現在、このコラムの値はいつでも `NULL` です。
- **sql_mode**
トリガが実行する時に有効な SQL モード。
- **Definer**
トリガを作成したアカウント。

`INFORMATION_SCHEMA TRIGGERS` [テーブル](#) もご参照ください。

SHOW VARIABLES 構文

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

`SHOW VARIABLES` は MySQL システム変数の値を表示します。 `mysqladmin variables` コマンドを利用してこの情報を得る事もできます。

`GLOBAL` 修飾子を利用すると、`SHOW VARIABLES` は MySQL への新しい接続に利用される値を表示します。 `SESSION` を利用すると、現在の接続に有効な値を表示します。 もし修飾子が無ければ、デフォルトは `SESSION` です。 `LOCAL` は `SESSION` の同義語です。

もしデフォルトのシステム変数が不適切であれば、`mysqld` がスタートした時にコマンド オプションを利用してそれらを設定する事ができ、また `SET` ステートメントを利用してほとんどの物をランタイムに変更できます。[システム変数の使用](#) と [SET 構文](#) を参照して下さい。

部分的なアウトプットがここに表示されています。名前と値のリストは、お使いのサーバとは異なる場合があります。[システム変数](#) で各変数の意味が説明されており、[サーバパラメータのチューニング](#) にはそれらを調整する為の情報が紹介されています。

```
mysql> SHOW VARIABLES;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| automatic_sp_privileges | ON |
| back_log | 50 |
| basedir | /home/jon/bin/mysql-5.1/ |
| binlog_cache_size | 32768 |
| bulk_insert_buffer_size | 8388608 |
| character_set_client | latin1 |
| character_set_connection | latin1 |
| ... | ... |
| max_user_connections | 0 |
| max_write_lock_count | 4294967295 |
| multi_range_count | 256 |
| myisam_data_pointer_size | 6 |
| myisam_max_sort_file_size | 2147483647 |
| myisam_recover_options | OFF |
| myisam_repair_threads | 1 |
| myisam_sort_buffer_size | 8388608 |
| ndb_autoincrement_prefetch_sz | 32 |
| ndb_cache_check_time | 0 |
| ndb_force_send | ON |
| ... | ... |
| time_zone | SYSTEM |
| timed_mutexes | OFF |
| tmp_table_size | 33554432 |
| tmpdir | |
| transaction_alloc_block_size | 8192 |
| transaction_prealloc_size | 4096 |
| tx_isolation | REPEATABLE-READ |
| updatable_views_with_limit | YES |
| version | 5.1.6-alpha-log |
| version_comment | Source distribution |
| version_compile_machine | i686 |
| version_compile_os | suse-linux |
| wait_timeout | 28800 |
+-----+-----+
```

`LIKE` 条項を利用すると、パターンと一致する名前を持つ変数の行のみを表示します。

特定の変数に行を得る為には、次に表示されているように `LIKE` 条項を利用してください。

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

名前がパターンと一致する変数のリストを得るには、`LIKE` 条項内の `'%'` ワイルドカード文字を利用してください。

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

ワイルドカード文字は、一致するパターン内のどの場所でも利用する事ができます。厳密に言うと、`'_'` は全ての単一文字と一致するワイルドカードなので、完全に一致させる為に `'\_'` の時は拡張する必要があります。実際には、これはほとんど必要ありません。

SHOW WARNINGS 構文

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS` は、メッセージを作成した最後のステートメントから生じたエラー、警告、そしてノート メッセージを表示、または、もしテーブルを利用した最後のステートメントが何のメッセージも作成しなければ、何も表

示しません。関連ステートメントである [SHOW ERRORS](#) はエラーだけを表示します。詳しくは [SHOW ERRORS 構文](#) を参照してください。

テーブルを利用するそれぞれの新ステートメントに対して、メッセージのリストはリセットされます。

[SHOW COUNT\(*\) WARNINGS](#) ステートメントはエラー、警告、そしてノートの総数を表示します。[warning_count](#) 変数からもこの数字を検索する事ができます。

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

もし [max_error_count](#) システム変数が、全てのメッセージの格納ができないほど低く設定されると、[warning_count](#) の値は [SHOW WARNINGS](#) によって表示されるメッセージ数よりも大きくなります。このセクションの後の方で表示される例で、これがどのように起きるのが紹介しています。

[LIMIT](#) 条項は [SELECT](#) ステートメントに対するのと同じ構文を持っています。詳しくは [SELECT 構文](#) を参照してください。

MySQL サーバは、最後のステートメントから生じたエラー、警告、そしてノートの総数を送り返します。もし C API を利用していれば、この値は [mysql_warning_count\(\)](#) をコールする事で得る事ができます。詳しくは [mysql_warning_count\(\)](#) を参照してください。

警告は、[LOAD DATA INFILE](#) や、また DML [INSERT](#)、[UPDATE](#)、[CREATE TABLE](#)、そして [ALTER TABLE](#) のような DML ステートメントなどのようなステートメントに対して作成されます。

次の [DROP TABLE](#) ステートメントはノートをもたらします:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Note  | 1051 | Unknown table 'no_such_table'             |
+-----+-----+-----+
```

ここに、[CREATE TABLE](#) に対する構文警告と、[INSERT](#) に対する変換警告を表すシンプルな例があります。

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),(NULL,'test'),
-> (300,'Open Source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 4

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 1
***** 2. row *****
Level: Warning
Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
***** 3. row *****
Level: Warning
Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
***** 4. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

エラー警告、そしてノート メッセージの最高格納数は `max_error_count` システム変数によってコントロールされています。デフォルトにより、その値は64です。格納するメッセージ数を変更したければ、`max_error_count` の値を変更してください。次の例では `ALTER TABLE` ステートメントは3つの警告メッセージを発生しますが、`max_error_count` が1に設定されている為、そのうちの1つしか格納されません。

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_error_count | 64 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3

mysql> SELECT @@warning_count;
+-----+
| @@warning_count |
+-----+
| 3 |
+-----+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

警告を無効にするには、`max_error_count` を0に設定してください。この場合、`warning_count` はいくつの警告が起きたか指示しますが、どのメッセージも格納はされません。

`SQL_NOTES` セッション変数を0に設定して、`Note` レベルの警告が記録されないようにできます。

その他の管理ステートメント

CACHE INDEX 構文

```
CACHE INDEX
tbl_index_list [, tbl_index_list] ...
IN key_cache_name

tbl_index_list:
tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

`CACHE INDEX` ステートメントはテーブル インデックスを特定のキー キャッシュに割り当てます。これは `MyISAM` テーブルにしか利用されません。

次のステートメントは、インデックスをテーブル `t1`、`t2`、そして `t3` から `hot_cache` という名前のキー キャッシュに割り当てます。

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+-----+
| Table | Op          | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | assign_to_keycache | status | OK |
| test.t2 | assign_to_keycache | status | OK |
| test.t3 | assign_to_keycache | status | OK |
+-----+-----+-----+-----+
```

`CACHE INDEX` の構文によって、テーブルからの特定のインデックスだけがキャッシュに割り当てられなければならない、と指定する事ができます。現在のインプリメンテーションは、全てのテーブルのインデックスをキャッシュに割り当てるので、テーブル名以外を指定する利用は無いのです。

CACHE INDEX ステートメント内で参照されるキー キャッシュは、パラメータ設定ステートメントを設定するか、サーバパラメータ設定の中で作成できます。例:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

キー キャッシュ パラメータには、構造化システム変数のメンバとしてアクセスできます。詳しくは [構造化システム変数](#) を参照してください。

キー キャッシュは、インデックスをそれに割り当てる前に存在していなければいけません。

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

デフォルトで、テーブル インデックスは、サーバ起動時に作成された主要 (デフォルト) キー キャッシュに割り当てられます。キー キャッシュが破壊される時、そこに割り当てられた全てのインデックスはデフォルト キーキャッシュに再び割り当てられます。

インデックスの割り当ては、サーバに対してグローバルに影響します。もし1つのクライアントが既存のキャッシュにインデックスを割り当てると、どのクライアントがクエリを発行したかに関わらず、このキャッシュはそのインデックスを含む全てのクエリに対して利用されます。

FLUSH 構文

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG]
flush_option [, flush_option] ...
```

FLUSH ステートメントは、MySQL に利用された様々な内部キャッシュをクリア、または再ロードします。**FLUSH** を実行する為には、**RELOAD** 権限を持つ必要があります。

RESET ステートメントは **FLUSH** と似ています。詳しくは [RESET 構文](#) を参照してください。

flush_option は、次のうちのどれかになり得ます。

- **HOSTS**

ホスト キャッシュ テーブルを空にします。もし、いくつかのホストが IP 番号を変えたり、エラー メッセージ **Host 'host_name' is blocked** を受け取ったりしたら、ホスト テーブルをフラッシュする必要があります。MySQL サーバに接続中に、連続して **max_connect_errors** 以上のエラーがホストに発生すると、MySQL は何か異常があると仮定して、それ以上の接続の要求をブロックします。ホスト テーブルをフラッシュすると、ホストは再度接続を試みる事ができます。詳しくは **Host 'host_name' is blocked** を参照してください。このエラー メッセージを避ける為に、**--max_connect_errors=999999999** を利用して **mysqld** をスタートする事ができます。

- **DES_KEY_FILE**

サーバ起動時に **--des-key-file** オプションを利用して指定されたファイルからの DES キーを再ロードします。

- **LOGS**

全てのログ ファイルを閉じ、再オープンします。もしバイナリ ログが有効であれば、バイナリ ログ ファイルのシーケンス番号は前のファイルと比較して1つ増加されます。これは Unix では、**SIGHUP** シグナルを **mysqld** サーバに送信する事と同じです。(mysqld が **SIGHUP** と **SIGQUIT** を無視する Mac OS X 10.3 バージョン以外)

もしサーバが **--log-error** オプションでスタートされたら、それは **FLUSH LOGS** によって **-old** のサフィックスを利用して現在のエラー ログ ファイルをリネームし、新しく空のログ ファイルを作成します。もし **--log-error** オプションがなければリネームは行われません。

- **MASTER** (DEPRECATED)全てのバイナリ ログを削除し、バイナリ ログ インデックス ファイルをリセットし、新しいバイナリ ログを作成します。**FLUSH MASTER** は **RESET MASTER** と置き換えられ廃止予定であり、現在は後方互換だけがサポートされています。詳しくは [RESET MASTER 構文](#) を参照してください。

- **PRIVILEGES**

mysql データベース内で供与テーブルから権限を再ロードします。

- **QUERY CACHE**

メモリ使用を向上させる為にクエリ キャッシュをデフラグします。FLUSH QUERY CACHE は RESET QUERY CACHE とは違い、キャッシュからクエリを削除しません。

- **SLAVE** (DEPRECATED)リレー ログ ファイルやマスタのバイナリ ログ内の複製位置を含む、全ての複製スレーブ パラメータをリセットします。FLUSH SLAVE は RESET SLAVE と置き換えられ廃止予定であり、現在は後方互換 だけがサポートされています。詳しくは [RESET SLAVE 構文](#) を参照してください。

- **STATUS**

このオプションは、グローバル値に現在のスレッドのセッション ステータス変数値を追加し、セッション値をゼロにリセットします。それはキー キャッシュ(デフォルトと名づけられた物)のカウンタをゼロにリセットし、現在 オープンしている接続の数に `Max_used_connections` を設定します。これは、クエリをデバッグしている時のみ利用するべき物です。詳しくは [質問またはバグの報告](#) を参照してください。

- **{TABLE | TABLES} [tbl_name [, tbl_name] ...]**

どのテーブル名づけられていない時に、全てのオープンなテーブルを閉じ、利用中の全てのテーブルを強制的に閉じます。これはクエリ キャッシュもフラッシュします。複数のテーブル名があると、与えられたテーブルだけをフラッシュします。FLUSH TABLES は RESET QUERY CACHE のように、クエリ キャッシュから全てのクエリ結果の削除もします。

- **TABLES WITH READ LOCK**

UNLOCK TABLES を実行して明示的にロックを解除するまで、リード ロックを利用して全てのデータベースの全てのオープン テーブルを閉じ、全てのテーブルをロックします。これは、もし Veritas のような、時間内にスナップショットを撮る事ができるファイルシステムを持っているなら、バックアップを取るのに大変便利な方法になります。

FLUSH TABLES WITH READ LOCK は、グローバル リード ロックは取得しますがテーブル ロックはしないので、テーブル ロックと暗黙的なコミットに関しては LOCK TABLES と UNLOCK TABLES と同じような動作の制約は受けません。

- UNLOCK TABLES は、もしテーブルが現在 LOCK TABLES でロックされていたらトランザクションを行います。これは、FLUSH TABLES WITH READ LOCK ステートメントがテーブル レベル ロックを取得しない為、これに続く UNLOCK TABLES に対しては行われません。
- トランザクションを開始すると、LOCK TABLES を利用して行ったテーブル ロックを、まるで UNLOCK TABLES を実行したかのように解除してしまいます。トランザクションを開始しても、FLUSH TABLES WITH READ LOCK を利用して行われたグローバル リード ロックの解除はしません。

- **USER_RESOURCES**

全ての時間あたりのユーザ リソースをゼロにリセットします。これは、時間ごとの接続、クエリ、更新リミットに達したクライアントがすぐに活動を再開できるようにします。FLUSH USER_RESOURCES は最大同時接続のリミットに適応しません。詳しくは [GRANT 構文](#) を参照してください。

FLUSH ステートメントは、任意の NO_WRITE_TO_BINLOG キーワード(またはそのエイリアス LOCAL) が利用されない限り、バイナリ ログに書きこまれます。これは、複製マスタとして機能している MySQL サーバ上で利用される FLUSH ステートメントが、複製スレーブにデフォルトで複製される為に行われます。

注意:FLUSH LOGS、FLUSH MASTER、FLUSH SLAVE、そして FLUSH TABLES WITH READ LOCK は、スレーブに複製されると問題を引き起こす為、ログインされません。

flush-hosts、flush-logs、flush-privileges、flush-status、または flush-tables コマンドを利用する mysqladmin ユーティリティで、いくつかのステートメントにアクセスする事ができます。

注意:MySQL 5.1 内では、ストアド ファンクションやトリガ内で FLUSH ステートメントを発行する事は不可能です。しかし、ストアド プロシージャ内の FLUSH がストアド ファンクションやトリガにコールされない限り、それらを利用してよいでしょう。詳しくは [ストアド ルーチンとトリガの規制](#) を参照してください。

RESET ステートメントが複製の中でどのように利用されるかという情報については [RESET 構文](#) も参照してください。

KILL 構文

```
KILL [CONNECTION | QUERY] thread_id
```

mysqld への各接続は別々のスレッド内で起動します。どのスレッドが `SHOW PROCESSLIST` ステートメントを利用し、スレッドを停止させるのかを、`KILL thread_id` ステートメントを利用して確認する事ができます。

`KILL` は任意の `CONNECTION` か `QUERY` 修飾子を許容します。

- `KILL CONNECTION` は修飾子を持たない `KILL` と同じです。それは `thread_id` と関連している接続を終了します。
- `KILL QUERY` は現在接続が実行中であるステートメントを終了させますが、その接続自体には手をつけずそのまま残しておきます。

もし `PROCESS` 権限を持っていれば、全てのスレッドを見る事ができます。もし `SUPER` 権限を持っていれば、全てのスレッドとステートメントを停止する事ができます。そうでなければ、自分自身のスレッドとステートメントのみ確認、停止させる事ができます。

`mysqladmin processlist` と `mysqladmin kill` コマンドを利用して、スレッドを検査、停止する事もできます。

注意:埋め込みサーバは、ホスト アプリケーションのスレッド内ではほとんど起動しないので、`KILL` と埋め込み MySQL サーバ ライブラリと一緒に利用する事はできません。それは、それ自身の接続スレッドは作成しません。

`KILL` を利用する時、スレッド固有のキル フラグがスレッドに設定されます。ほとんどの場合、キル フラグは特定のインターバルでしか確認されないの、スレッドが停止するまでに少し時間がかかります。

- `SELECT`, `ORDER BY` そして `GROUP BY` ループ内では、フラグは行のブロックを読み込んだ後に確認されます。もしキル フラグが設定されると、ステートメントは異常終了します。
- `ALTER TABLE` の最中に、行の各ブロックが元テーブルから読み込まれる前にキル フラグが確認されます。もしキル フラグが設定されると、ステートメントは異常終了し、テンポラリ テーブルは削除されます。
- `UPDATE` や `DELETE` 操作の最中に、各ブロックの読み込みや、行の各更新や削除の後でキル フラグが確認されます。もしキル フラグが設定されると、ステートメントは異常終了します。もしトランザクションを利用していなければ、変更はロールバックされない事に注意してください。
- `GET_LOCK()` は `NULL` を異常終了し、返します。
- `INSERT DELAYED` スレッドはメモリ内に持っている全ての行をすばやくフラッシュ(挿入)します。
- もしスレッドがテーブル ロック ハンドラ内にあれば、(状態: `Locked`) テーブルロックは迅速に異常終了します。
- もしスレッドが書き込みコール内でフリー ディスク スペースを待っていたら、その書き込みは「disk full」 エラーメッセージを利用して異常終了されます。
- 警告: `MyISAM` テーブル上で `REPAIR TABLE` や `OPTIMIZE TABLE` 操作を終了させると、テーブルが破損され、利用不可能になります。それを最適化、または修復するまで(割り込み無しで)、そのようなテーブルへの書き込みや読み込みは失敗します。

LOAD INDEX INTO CACHE 構文

```
LOAD INDEX INTO CACHE
tbl_index_list [, tbl_index_list] ...

tbl_index_list:
tbl_name
[[INDEX|KEY] (index_name[, index_name] ...)]
[IGNORE LEAVES]
```

`LOAD INDEX INTO CACHE` ステートメントは、明示的な `CACHE INDEX` ステートメントによって割り当てられたキー キャッシュに、またはそうでなければデフォルトのキー キャッシュに、テーブル インデックスをあらかじめロードしておきます。`LOAD INDEX INTO CACHE` は `MyISAM` テーブルにだけ利用されます。

`IGNORE LEAVES` 修飾子は、インデックスの非リーフ ノードの為のブロックだけがあらかじめロードされるよう働きかけます。

次のステートメントは、(index blocks) テーブル **t1** と **t2** のインデックスのノード(インデックスブロック)をあらかじめロードしておきます。

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
```

Table	Op	Msg_type	Msg_text
test.t1	preload_keys	status	OK
test.t2	preload_keys	status	OK

このステートメントは **t1** から全てのインデックス ブロックをあらかじめロードしておきます。それは、**t2** から非リーフ ノードのブロックだけをあらかじめロードします。

LOAD INDEX INTO CACHE の構文によって、テーブルからの特定のインデックスだけがあらかじめロードされなければいけない、と指定する事ができます。現在のインプリメンテーションは、全てのテーブルのインデックスをキャッシュ内に割り当てるので、テーブル名以外を指定する利用は無いのです。

LOAD INDEX INTO CACHE は、テーブル内の全てのインデックスが同じブロック サイズでなければ失敗します。**myisamchk -dv** を利用し、**Blocksize** カラムを確認する事で、テーブルのインデックス ブロック サイズを決定する事ができます。

RESET 構文

```
RESET reset_option [, reset_option] ...
```

RESET ステートメントは様々なサーバ操作の状態をクリアする為に利用されます。**RESET** を実行する為には **RELOAD** 権限を持つ必要があります。

RESET は **FLUSH** ステートメントの、より強力バージョンとして機能します。詳しくは **FLUSH 構文** を参照してください。

reset_option は、次のうちのどれかになり得ます。

- **MASTER**

インデックス ファイル内にリストされている全てのバイナリ ログを削除し、バイナリ ログ インデックス ファイルをゼロにリセットし、新しいバイナリ ログを作成します。(MySQL 3.23.26 以前のバージョンでは **FLUSH MASTER** として知られています。)詳しくは **マスタ サーバをコントロールする SQL ステートメント** を参照してください。

- **QUERY CACHE**

クエリ キャッシュから全てのクエリ結果を削除します。

- **SLAVE**

スレーブがマスタ バイナリ ログ内の複製位置を忘れるよう働きかけます。また、既存リレー ログ ファイルを削除する事で、リレー ログをリセットし、新しい物を開始します。(MySQL 3.23.26 以前のバージョンでは **FLUSH SLAVE** として知られています。)詳しくは **スレーブ サーバをコントロールする SQL ステートメント** を参照してください。

複製ステートメント

このセクションでは、複製に関連した SQL ステートメントについて説明します。1つのグループは、マスタ サーバをコントロールする為に利用されます。別の物はスレーブ サーバをコントロールする為に利用されます。

マスタ サーバをコントロールする SQL ステートメント

複製は SQL インターフェースを通してコントロールできます。このセクションでは、マスタ複製サーバの管理についてのステートメントの説明をします。**スレーブ サーバをコントロールする SQL ステートメント**では、スレーブ サーバの管理について説明しています。

PURGE MASTER LOGS 構文

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'
PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

ログ インデックス内で指定されたログや日付の前にリストされている全てのバイナリ ログを削除します。与えられたログが最初になるように、ログ インデックス ファイル内に記録されたリストからもログが削除されます。

例:

```
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

BEFORE 異型の **date** 引数は 'YYYY-MM-DD hh:mm:ss' フォーマットになり得ます。**MASTER** と **BINARY** は同義語です。

このステートメントは、スレーブが複製中に起動しても安全です。それらを停止させる必要はありません。現在削除しようとしているログの1つを読み込んでいる、アクティブ スレーブを持っていれば、このステートメントは何もせず、エラーで失敗します。しかし、もしスレーブが休止状態で、まだ読み込まれていないログの1つを消去してしまったら、そのスレーブはその後複製が不可能になります。

ログを安全に消去するには、次の手順に従ってください。

1. 各スレーブ サーバ上で、どのログがそれを読み込んでいるのか確認する為に **SHOW SLAVE STATUS** を利用してください。
2. **SHOW BINARY LOGS** を利用してマスタ サーバ上でバイナリ ログのリストを手に入れてください。
3. 全てのスレーブの中で一番最初のログを確認してください。これがターゲット ログです。もし全てのスレーブが最新であれば、これがリスト上の最後のログになります。
4. 削除しようとしている全てのログのバックアップを作成してください。(このステップは任意ですが、常に推奨されている物です。)
5. ターゲット ログを含まず、そこまでの全てのログを消去してください。

指定した日数後に(**システム変数** を参照)バイナリ ログを自動的に無効にする **expire_logs_days** システム変数も設定できます。もし複製を利用しているなら、ご利用のスレーブがマスタよりも遅れるであろう最大日数よりも低く変数を設定しなければいけません。

RESET MASTER 構文

```
RESET MASTER
```

インデックス ファイル内にリストされている全てのバイナリ ログを削除し、バイナリ ログ インデックス ファイルをゼロにリセットし、新しいバイナリ ログを作成します。

SET SQL_LOG_BIN 構文

```
SET SQL_LOG_BIN = {0|1}
```

もしクライアントが **SUPER** 権限を持っていたら、現在の接続(**SQL_LOG_BIN** がセッション変数です)のバイナリ ログを無効、または有効にします。もしクライアントがその権限を持っていなければ、ステートメントはエラーによって拒否されます。

SHOW BINLOG EVENTS 構文

```
SHOW BINLOG EVENTS
[IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

バイナリ ログ内のイベントを表します。もし 'log_name' を指定しなければ、最初のバイナリ ログが表示されます。

LIMIT 条項は **SELECT** ステートメントに対するのと同じ構文を持っています。詳しくは **SELECT 構文** を参照してください。

注意: **LIMIT** を利用せずに **SHOW BINLOG EVENTS** を発行すると、サーバがクライアントに完全なバイナリ ログのコンテンツ(データを変更するサーバによって実行された全てのステートメントを含む)を返すので、時間、リソースの両方を大量に消費するプロセスを開始する事になってしまいます。**SHOW BINLOG EVENTS** の代替として、後ほど行う調査と分析の為にバイナリ ログをテキスト ファイルに保存する為に **mysqlbinlog** ユーティリティを利用してください。詳しくは **mysqlbinlog — バイナリログファイルを処理するためのユーティリティ** を参照してください。

SHOW BINARY LOGS 構文

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

サーバ上にバイナリ ログをリストします。このステートメントは、どのログを消去できるかを定める方法を表す **PURGE MASTER LOGS** 構文 内で説明されている手順の一部として利用されています。

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name | File_size |
+-----+-----+
| binlog.000015 | 724935 |
| binlog.000016 | 733481 |
+-----+-----+
```

SHOW MASTER LOGS は **SHOW BINARY LOGS** と同等です。

SHOW MASTER STATUS 構文

```
SHOW MASTER STATUS
```

マスタのバイナリ ログ ファイルについてのステータス情報を提供します。例:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003 | 73 | test | manual,mysql |
+-----+-----+-----+-----+
```

SHOW SLAVE HOSTS 構文

```
SHOW SLAVE HOSTS
```

現在マスタと共に登録されている複製スレーブのリストを表示します。このリスト内では、**--report-host=slave_name** オプションを利用してスタートされたスレーブのみ見る事ができます。

このリストはどのサーバ上にも表示されます。(マスタサーバではありません。)アウトプットはこのようになります。

```
mysql> SHOW SLAVE HOSTS;
+-----+-----+-----+-----+
| Server_id | Host | Port | Master_id |
+-----+-----+-----+-----+
| 192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena | 3306 | 192168011 |
+-----+-----+-----+-----+
```

- **Server_id**:サーバのオプション ファイル内、または **--server-id=value** を利用したコマンドライン上で設定された、ユニーク サーバ ID です。
- **Host**:サーバのオプション ファイル内、または **--report-host=value** を利用したコマンドライン上で設定された、スレーブ サーバのホスト名です。

OS 内で設定されたマシン名とは異なる可能性があるという事に注意してください。

- **Port**:スレーブ サーバが聴いているポート
- **Master_id**:スレーブ サーバがそこから複製している、マスタ サーバのユニーク サーバ ID。

いくつかの MySQL バージョンは別の変数 `Rpl_recovery_rank` を報告します。この変数が利用された事はなく、最終的に削除されました。

スレーブ サーバをコントロールする SQL ステートメント

複製は SQL インターフェースを通してコントロールできます。このセクションでは、スレーブ複製サーバの管理についてのステートメントの説明をします。[マスタ サーバをコントロールする SQL ステートメント](#)では、マスタサーバの管理について説明しています。

CHANGE MASTER TO 構文

```
CHANGE MASTER TO master_def [, master_def] ...
```

```
master_def:
  MASTER_HOST = 'host_name'
| MASTER_USER = 'user_name'
| MASTER_PASSWORD = 'password'
| MASTER_PORT = port_num
| MASTER_CONNECT_RETRY = count
| MASTER_LOG_FILE = 'master_log_name'
| MASTER_LOG_POS = master_log_pos
| RELAY_LOG_FILE = 'relay_log_name'
| RELAY_LOG_POS = relay_log_pos
| MASTER_SSL = {0|1}
| MASTER_SSL_CA = 'ca_file_name'
| MASTER_SSL_CAPATH = 'ca_directory_name'
| MASTER_SSL_CERT = 'cert_file_name'
| MASTER_SSL_KEY = 'key_file_name'
| MASTER_SSL_CIPHER = 'cipher_list'
```

CHANGE MASTER TO は、スレーブ サーバがマスタ サーバに接続、また更新する時に利用するパラメータを変更します。それは `master.info` と `relay-log.info` ファイルのコンテンツの更新もします。

`MASTER_USER`、`MASTER_PASSWORD`、`MASTER_SSL`、`MASTER_SSL_CA`、`MASTER_SSL_CAPATH`、`MASTER_SSL_CERT`、そして `MASTER_SSL_CIPHER` はどのようにマスタに接続するかという事について情報を提供します。

SSL オプション

(`MASTER_SSL`、`MASTER_SSL_CA`、`MASTER_SSL_CAPATH`、`MASTER_SSL_CERT`、`MASTER_SSL_KEY`、そして `MASTER_SSL_CIPHER`) は、SSL サポート無しでコンパイルされたスレーブ上でも変更できます。それらは `master.info` ファイルに保存されますが、有効な SSL サポートを持つサーバを利用しなければ無視されます。

もし与えられたパラメータを指定しなければ、次の説明の中で指示されている場合以外は古い値を持ち続けます。例えば、MySQL マスタに接続する為のパスワードが変更されたら、スレーブに新しいパスワードについて指示する為にこれらのステートメントを発行するだけでよいのです。

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

変更しないパラメータを指定する必要はありません。(ホスト、ポート、ユーザ、など)

`MASTER_HOST` と `MASTER_PORT` はマスタホストと、その TCP/IP ポートのホスト名(または IP アドレス)です。もし `MASTER_HOST` が `localhost` と同等であれば、その時は MySQL の別の部分と同じで、ポート番号は無視されるという事に注意してください。(例えば、もし Unix ソケットファイルが利用できる場合)

もし `MASTER_HOST` が `MASTER_PORT` を指定すると、スレーブはマスタサーバが以前とは違うと仮定します。(たとえ現在の値と等しいホストやポート値を指定したとしても)この場合、マスタ バイナリ ログ名と位置の古い値は適応しないとみなされる為、もしステートメント内の `MASTER_LOG_FILE` と `MASTER_LOG_POS` を指定しなければ、`MASTER_LOG_FILE=""` と `MASTER_LOG_POS=4` が静かにそれに加えられます。

`MASTER_LOG_FILE` と `MASTER_LOG_POS` は、次にスレッドがスタートするマスタから、スレーブ I/O スレッドが読み込みを始めなければいけない座標です。もしそれらのどちらもを指定しなければ、`RELAY_LOG_FILE` が `RELAY_LOG_POS` を指定する事ができません。もし `MASTER_LOG_FILE` も `MASTER_LOG_POS` も指定されなければ、**CHANGE MASTER** が発行される前に、スレーブは slave SQL thread の最後の座標を利用します。これは、ただ単に使用するパスワードを変更したいだけの時に、スレーブ I/O スレッドと比べてスレーブ SQL スレッドが遅かったとしても、複製内に切れ目が無い事を保証します。

CHANGE MASTER は、**RELAY_LOG_FILE** か **RELAY_LOG_POS** を指定しない限り、全てのリレー ログ ファイルを削除し、新しい物をスタートします。その場合、リレー ログは保管されます。**relay_log_purge** グローバル変数は静かに0に設定されます。

CHANGE MASTER は、マスタのスナップショットを持っていて、それに対応するログとオフセットを記録した時に、スレーブを設定するのに役立ちます。スナップショットをスレーブにロードした後、スレーブ上で **CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master** を起動する事ができます。

次の例は、マスタとマスタのバイナリ ログ座標を変更します。これは、マスタを複製する為にスレーブを設定したい時に利用します。

```
CHANGE MASTER TO
MASTER_HOST='master2.mycompany.com',
MASTER_USER='replication',
MASTER_PASSWORD='bigs3cret',
MASTER_PORT=3306,
MASTER_LOG_FILE='master2-bin.001',
MASTER_LOG_POS=4,
MASTER_CONNECT_RETRY=10;
```

次の例は、あまり利用されない操作を表しています。これは、何かの理由でもう一度実行したいリレー ログをスレーブが持っている時に利用します。これを行う為には、マスタにはアクセス不可能でなければいけません。**CHANGE MASTER TO** を利用し、SQL スレッドをスタートさせるだけでよいです。(START SLAVE SQL_THREAD)

```
CHANGE MASTER TO
RELAY_LOG_FILE='slave-relay-bin.006',
RELAY_LOG_POS=4025;
```

スタンドアロンを利用した非複製セットアップ内、クラッシュ後の修復の為に非スレーブサーバ内で、2番目の操作を利用する事もできます。サーバがクラッシュして、バックアップを格納したと仮定してください。サーバのバイナリログ（リレーログではなく通常のバイナリログ）、名づけられた（例えば）**myhost-bin.*** を再生したいでしょう。まず、次の手順と全く同じように作業しなかった為にサーバが誤ってバイナリ ログを消去してしまう、という場合に備えて、これらのバイナリ ログのバックアップ コピーを安全なところに作成してください。更なる安全の為に **SET GLOBAL relay_log_purge=0** を利用してください。**--log-bin** オプションは利用せず、その代わりに、**--replicate-same-server-id**、**--relay-log=myhost-bin** (サーバにこれらの通常のバイナリ ログがリレー ログだと信じさせる為)そして **--skip-slave-start** オプションを利用してサーバをスタートさせてください。サーバがスタートしたら、これらのステートメントを発行してください。

```
CHANGE MASTER TO
RELAY_LOG_FILE='myhost-bin.153',
RELAY_LOG_POS=410,
MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

サーバがそれ自身のバイナリ ログを読み込み、実行するので、クラッシュの修復を達成します。修復が終了したら、**STOP SLAVE** を起動し、サーバをシャットダウンし、**master.info** と **relay-log.info** ファイルを削除し、そして元のオプションを利用してサーバを再スタートさせてください。

MASTER_HOST オプション(たとえダミー値を利用していても) を指定するには、それがスレーブだとマスタに信じさせる事が必要です。

LOAD DATA FROM MASTER 構文

LOAD DATA FROM MASTER

この機能は終了しました。今後は使わないことを勧めます。MySQLの将来のバージョンでは取り除かれることもあります。

LOAD DATA FROM MASTER と **LOAD TABLE FROM MASTER** の現在のインプリメンテーションがとても制限されているので、これらのステートメントはMySQLのバージョン4.1以降では廃止予定です。今後のバージョンでは、さらに進歩した技術(「online backup」と呼ばれる物)を紹介する予定です。その技術は、さらに多くのストレージエンジンと機能する更なる利点を持ちます。

MySQL 5.1 以前のバージョンで **LOAD DATA FROM MASTER** か **LOAD TABLE FROM MASTER** を利用する為に推奨する代替方法は、**mysqldump** か **mysqlhotcopy** を利用する事です。後者は Perl と2つの Perl モジュール(DBI と

DBD:mysql)を必要とし、MyISAM と ARCHIVE テーブルにだけ機能します。mysqldump を利用すると、マスタ上に SQL ダンプを作成でき、それらをスレーブ上の mysql クライアントにパイプ(またはコピー)できます。これは全てのストレージ エンジンに機能するという利点を持ちますが、SELECT を利用して機能する為スピードが大変遅いです。

このステートメントは、マスタのスナップショットを撮り、それをスレーブにコピーします。それは、スレーブが正しい位置から複製を始めるように MASTER_LOG_FILE と MASTER_LOG_POS の値を更新します。--replicate-*-do-* と --replicate-*-ignore-* オプションを利用して指定されたテーブルとデータベースの除外ルールは支持されています。マスタからテーブルをロードする時にスレーブを混乱させる --replicate-rewrite-db="db1->db3" と --replicate-rewrite-db="db2->db3" のような非固有マッピングを設定する為に、ユーザはこのオプションを利用できるので、--replicate-rewrite-db は考慮に入れていません。

このステートメントは次の条件に従い利用できます：

- これは MyISAM テーブルにしか機能しません。非 MyISAM テーブルをロードしようとすると、次のエラーが起こります。

```
ERROR 1189 (08S01): Net error reading from master
```

- それはスナップショットを撮っている間にグローバル リード ロックを取得し、それはロード作業中のマスタ上での更新を妨げます。

もし大きいテーブルをロードしていたら、マスタとスレーブの両方で net_read_timeout と net_write_timeout の値を増やす必要があるでしょう。詳しくは [システム変数](#) を参照してください。

LOAD DATA FROM MASTER は mysql データベースから何もコピー しない 事に注意してください。これは、マスタとスレーブ上で異なるユーザと権限を持つ事を簡単にします。

LOAD DATA FROM MASTER を利用する為には、マスタに接続する為に利用される複製アカウントはマスタ上に RELOAD と SUPER 権限を持ち、ロードしたい全てのマスタ テーブルに SELECT 権限を持つ必要があります。ユーザが SELECT 権限を持たない全てのマスタ テーブルは LOAD DATA FROM MASTER に無視されます。これは、マスタがユーザからそれらを隠す為に起こります。LOAD DATA FROM MASTER は、ロードするデータベースを知る為に SHOW DATABASES をコールしますが、SHOW DATABASES はユーザが何かしらの権限を持つデータベースだけを返します。詳しくは [SHOW DATABASES 構文](#) を参照してください。スレーブ サイドでは、LOAD DATA FROM MASTER を発行するユーザはデータベースをドロップし作成する権限と、コピーされるテーブルを持つ必要があります。

LOAD TABLE tbl_name FROM MASTER 構文

```
LOAD TABLE tbl_name FROM MASTER
```

この機能は終了しました。今後は使わないことを勧めます。MySQLの将来のバージョンでは取り除かれることもあります。

LOAD DATA FROM MASTER と LOAD TABLE FROM MASTER の現在のインプリメンテーションがとても制限されているので、これらのステートメントは MySQL のバージョン 4.1 以降では廃止予定です。今後のバージョンでは、さらに進歩した技術(「online backup」と呼ばれる物)を紹介する予定です。その技術は、さらに多くのストレージ エンジンと機能する更なる利点を持ちます。

MySQL 5.1 以前のバージョンで LOAD DATA FROM MASTER か LOAD TABLE FROM MASTER を利用する為に推奨する代替方法は、mysqldump か mysqlhotcopy を利用する事です。後者は Perl と2つの Perl モジュール(DBI と DBD:mysql)を必要とし、MyISAM と ARCHIVE テーブルにだけ機能します。mysqldump を利用すると、マスタ上に SQL ダンプを作成でき、それらをスレーブ上の mysql クライアントにパイプ(またはコピー)できます。これは全てのストレージ エンジンに機能するという利点を持ちますが、SELECT を利用して機能する為スピードが大変遅いです。

マスタからスレーブにテーブルのコピーをトランスファします。このステートメントは主に LOAD DATA FROM MASTER 操作をデバッグしながらインプリメントされます。LOAD TABLE を利用するには、マスタ サーバに接続するのに利用されるアカウントはマスタ上に RELOAD と SUPER 権限を持ち、マスタ テーブルをロードする為に SELECT 権限を持つ必要があります。スレーブ サイドでは、LOAD TABLE FROM MASTER を発行するユーザはテーブルをドロップし作成する権限を持つ必要があります。

LOAD DATA FROM MASTER の条件もこれに適応します。例えば、LOAD TABLE FROM MASTER は MyISAM テーブルに対してのみ機能します。

`LOAD DATA FROM MASTER` のタイム アウト ノートもこれに適応します。

MASTER_POS_WAIT() 構文

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

このオプションはステートメントではなく関数です。これはスレーブがマスタのバイナリ ログ内の指定された位置までのイベントを読み込み、実行した事を保証する為に利用されます。完全な説明は [その他の関数](#) を参照してください。

RESET SLAVE 構文

```
RESET SLAVE
```

`RESET SLAVE` がスレーブがマスタのバイナリ ログ内の複製位置を忘れるよう働きかけます。このステートメントは再スタートの時に利用する為の物です。これは `master.info` と `relay-log.info` ファイル、全てのリレー ログを削除し、そして新しいリレー ログをスタートします。

注意:スレーブ SQL スレッドによって完全に実行されていなくても全てのリレー ログが削除されます。(もし `STOP SLAVE` ステートメントを発行していたり、スレーブが高負荷であると、複製スレーブ上にこの状態が存在しやすくなります。)

`master.info` ファイル内に格納された接続情報は、対応するスタートアップ オプション内で指定された値を利用して速やかにリセットされます。この情報は、マスタ ホスト、マスタ ポート、マスタ ユーザ、そしてマスタ パスワードのような値を含みます。もしスレーブ SQL スレッドが停止した時テンポラリ テーブルの複製の最中で、そして `RESET SLAVE` が発行されると、これらの複製テンポラリ テーブルはスレーブ上で削除されます。

SET GLOBAL SQL_SLAVE_SKIP_COUNTER 構文

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N
```

このステートメントは、マスタから次の `N` イベントをスキップします。これはステートメントによって引き起こされた複製から回復するのに役に立ちます。

このステートメントは、スレーブ スレッドが起動していない時のみ有効です。そうでなければ、エラーを発生させます。

SHOW SLAVE STATUS 構文

```
SHOW SLAVE STATUS
```

このステートメントは、スレーブ スレッドに不可欠なパラメータ上にステータス情報を提供します。`mysql` クライアントを利用してこのステートメントを発行すると、より読みやすい水平方向のレイアウトを得る為に、セミコロンではなく `\G` ステートメント ターミネータを利用する事ができます。

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: localhost
Master_User: root
Master_Port: 3306
Connect_Retry: 3
Master_Log_File: gbichot-bin.005
Read_Master_Log_Pos: 79
Relay_Log_File: gbichot-relay-bin.005
Relay_Log_Pos: 548
Relay_Master_Log_File: gbichot-bin.005
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 79
Relay_Log_Space: 552
```

```
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 8
```

SHOW SLAVE STATUS は次のフィールドを返します。

- **Slave_IO_State**

スレーブ I/O スレッドの為の **SHOW PROCESSLIST** のアウトプットの **State** フィールドのコピー。これで、スレッドが何をしているかを知る事ができます。マスタに接続しようとしている、マスタからイベントを待っている、マスタに再接続している、など。[レプリケーション実装の詳細](#) にステートの例がリストされています。古いバージョンの MySQL では、マスタへの接続が成功していなくてもスレッドが起動し続ける事ができるので、このフィールドを確認する必要があります。もし起動していれば問題は有りませんが、もしそうでなければ、**Last_Error** フィールド内にエラーを見つけることができます。(下記で説明)

- **Master_Host**

現在のマスタ ホスト。

- **Master_User**

マスタに接続する為に利用される現在のユーザ。

- **Master_Port**

現在のマスタ ポート。

- **Connect_Retry**

`--master-connect-retry` オプションの現在の値。

- **Master_Log_File**

I/O スレッドが現在読み込んでいるマスタ バイナリ ログ ファイルの名前。

- **Read_Master_Log_Pos**

現在のマスタ バイナリ ログ内で、I/O スレッドが読み込んだところまでの位置。

- **Relay_Log_File**

現在読み込み、実行をしている SQL スレッドからのリレー ログ ファイルの名前。

- **Relay_Log_Pos**

SQL スレッドが現在のリレー ログ内で読み込み、実行したところまでの位置。

- **Relay_Master_Log_File**

SQL スレッドによって実行された一番最近のイベントを含むマスタ バイナリ ログ ファイルの名前。

- **Slave_IO_Running**

I/O スレッドがスタートされて、マスタに正常に接続したかどうか。MySQL の古いバージョンでは(4.1.14 と 5.0.12 以前)、スレーブがまだマスタに接続されていなくても **Slave_IO_Running** は **YES** です。

- **Slave_SQL_Running**

SQL スレッドがスタートしたかどうか

- [Replicate_Do_DB](#)、[Replicate_Ignore_DB](#)

もしあるなら、`--replicate-do-db` と `--replicate-ignore-db` オプションを利用して指定されたデータベースのリスト。

- [Replicate_Do_Table](#)、[Replicate_Ignore_Table](#)、[Replicate_Wild_Do_Table](#)、[Replicate_Wild_Ignore_Table](#)

もしあるなら、`--replicate-do-table`、`--replicate-ignore-table`、`--replicate-wild-do-table`、そして `--replicate-wild-ignore-table` オプションを利用して指定されたテーブルのリスト。

- [Last_Errno](#)、[Last_Error](#)

一番最近実行されたクエリに返されるエラー数とエラー メッセージ。エラー数が0で、空の文字列のメッセージであれば、「no error.」です。もし [Last_Error](#) 値が空でなければ、それもスレーブのエラー ログ内に現れます。例:

```
Last_Errno: 1051
Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

このメッセージは、テーブル `z` がマスタ上に存在しそこでドロップされたが、それはスレーブ上には存在しなかった為、スレーブ上で `DROP TABLE` が失敗した、という事を含んでいます。(これは例えば、複製をセットアップする時にテーブルをスレーブにコピーするのを忘れた時に起きます。)

- [Skip_Counter](#)

[SQL_SLAVE_SKIP_COUNTER](#) に一番最近利用された値。

- [Exec_Master_Log_Pos](#)

マスタのバイナリ ログから SQL スレッドによって実行された最後のイベントの位置([Relay_Master_Log_File](#))。(マスタのバイナリ ログ内の [Relay_Master_Log_File](#) と [Exec_Master_Log_Pos](#))はリレーログ内の([Relay_Log_File](#) と [Relay_Log_Pos](#))に対応しています。

- [Relay_Log_Space](#)

全ての既存リレー ログを合計したサイズ。

- [Until_Condition](#)、[Until_Log_File](#)、[Until_Log_Pos](#)

`START SLAVE` ステートメントの `UNTIL` 条項内で指定された値。

[Until_Condition](#) は3つの値を持ちます。

- `UNTIL` 条項が指定されなければ `None`。
- もしスレーブがマスタのバイナリ ログ内の指定位置まで読み込んでいたら `Master`。
- もしスレーブがそのリレー ログ内の指定位置まで読み込んでいたら `Relay`。

[Until_Log_File](#) と [Until_Log_Pos](#) は、SQL スレッドが実行を停止するポイントを定義するログ ファイル名と位置の値を指示します。

- [Master_SSL_Allowed](#)、[Master_SSL_CA_File](#)、[Master_SSL_CA_Path](#)、[Master_SSL_Cert](#)、[Master_SSL_Cipher](#)、[Master_SSL_Key](#)

これらのフィールドは、もし SSL パラメータがあれば、マスタに接続するスレーブによって利用されるそれらを表します。

[Master_SSL_Allowed](#) はこれらの値を持ちます。

- もしマスタへの SSL 接続が許可されると `Yes`
- もしマスタへの SSL 接続が許可されないと `No`
- もし SSL 接続が許可されても、スレーブ サーバが有効な SSL サポートを持っていないければ `Ignored`

別の SSL 関連フィールドの値は、`--master-ca`、`--master-capath`、`--master-cert`、`--master-cipher`、そして `--master-key` オプションの値に対応します。

- [Seconds_Behind_Master](#)

このフィールドは、スレーブがどの程度「late」であるかの目安です。

- スレーブ SQL スレッドがアクティブに起動している時(更新を実行している時)、このフィールドはスレッドによって実行されたマスタ上の一番最近のイベントのタイム スタンプから経過した秒数を表します。
- SQL スレッドがスレーブ I/O スレッドに追いつき、そこからの更なるイベントを待ってアイドル状態になると、このフィールドはゼロになります。

基本的に、このフィールドはスレーブ SQL スレッドとスレーブ I/O スレッド間の時差を秒数で計算します。

もしマスタとスレーブ間のネットワーク接続が速ければ、スレーブ I/O スレッドはマスタにとっても近いので、このフィールドはマスタと比べてスレーブ SQL スレッドがどれくらい遅いかを表す近似値と言えます。これは、もしネットワークが遅いと参考になる近似値とは言えません。スレーブ SQL スレッドは頻繁に読み込みが遅いスレーブ I/O スレッドに追いつかれる為、I/O スレッドがマスタよりも遅くても [Seconds_Behind_Master](#) は 0 の値を頻繁に示します。言い換えると、このカラムは速いネットワークに対してだけ有効であるという事です。

この時間差の算出は、マスタとスレーブが同一の時計を持たなくても機能します。(時計の違いはスレーブ I/O スレッドがスタートする時に計算され、その時点から一定であると仮定されます。) [Seconds_Behind_Master](#) は、もしスレーブ SQL スレッドが起動していなかったり、スレーブ I/O スレッドが起動していない、またはマスタに接続されていない時は、[NULL](#) です(「unknown」を意味する)。例えばもしスレーブ I/O スレッドが、再接続前に [--master-connect-retry](#) オプションによって与えられた秒数眠っていたとすると、スレーブはマスタが何をしているか知る事ができないので [NULL](#) が表示され、その為どの程度遅れているか正確に知る事ができません。

このフィールドには1つ制限があります。タイムスタンプは複製中に維持されますので、これは、もしマスタ M1 自体が M0 のスレーブだったら、M0 の-binログからのイベントを複製する上で発生したM1の-binログからのイベントは、M0 のイベントのタイムスタンプを持つ、という事を意味します。これは MySQL が [TIMESTAMP](#) を正常に複製する事を可能にします。しかし、[Seconds_Behind_Master](#) の欠点は、もし M1 もクライアントから直接更新されると、最後の M1 のイベントは M0 からであったり、直接の更新からの一番最近のタイムスタンプであったりする為、その値はランダムに外れる、という事です。

START SLAVE 構文

```
START SLAVE [thread\_type [, thread\_type] ... ]
START SLAVE [SQL_THREAD] UNTIL
  MASTER_LOG_FILE = 'log\_name', MASTER_LOG_POS = log\_pos
START SLAVE [SQL_THREAD] UNTIL
  RELAY_LOG_FILE = 'log\_name', RELAY_LOG_POS = log\_pos

thread\_type: IO_THREAD | SQL_THREAD
```

[thread_type](#) オプションを持たない [START SLAVE](#) は両方のスレーブ スレッドをスタートします。I/O スレッドはマスタ サーバからクエリを読み、それらをリレー ログ内に格納します。SQL スレッドはリレー ログを読みこみ、クエリを実行します。[START SLAVE](#) は [SUPER](#) 権限を必要とします。

もし [START SLAVE](#) がスレーブ スレッドのスタートに成功すると、エラー無しで返ります。しかし、そのような場合でも、スレーブ スレッドがスタートし、その後停止する、という事になり得ます。(例えば、マスタに接続できなかったり、マスタのバイナリログを読めなかったり、それ以外の問題の為)[START SLAVE](#) からはこれについての警告はありません。スレーブ スレッドによって発生したエラー メッセージに対するスレーブのエラー ログの確認、また [SHOW SLAVE STATUS](#) を利用してそれらが満足に機能している事を確認する必要があります。

どのスレッドをスタートするかを指定する為に、ステートメントに [IO_THREAD](#) と [SQL_THREAD](#) オプションを追加する事ができます。

[UNTIL](#) 条項は、SQL スレッドがマスタ バイナリ ログ、またはスレーブ リレー ログ内の指定ポイントに達するまで、スレーブがスタート、起動する事を指定する為に追加されるかもしれません。SQL スレッドがそのポイントに達すると、それは停止します。もし [SQL_THREAD](#) オプションがステートメント内で指定されると、それは SQL スレッドのみをスタートします。そうでなければ、それはスレーブ スレッドを両方スタートします。もし SQL スレッドが起動していると、[UNTIL](#) 条項は無視され、警告が発行されます。

[UNTIL](#) 条項には、ログ ファイル名と位置の両方を指定する必要があります。マスタとリレー ログ オプションを混同しないでください。

全ての **UNTIL** 条件は、後に続く **STOP SLAVE** ステートメント、**UNTIL** 条項を含まない **START SLAVE** ステートメント、またはサーバの再スタートによってリセットされます。

UNTIL 条項は、デバッグの複製や、スレーブがステートメントを複製するのを防ぎたいポイントの直前まで複製が続くように働きかけるのに役に立ちます。例えば、マスタ上でもし分別のない **DROP TABLE** ステートメントが実行されたら、スレーブがそのポイントまでは実行し、それ以上は進まないように指示する為 **UNTIL** を利用する事ができます。イベントが何であるかを調べる為には、マスタ ログかスレーブ リレー ログと共に、**mysqlbinlog** 利用するか、または **SHOW BINLOG EVENTS** ステートメントを利用してしてください。

もしスレーブ プロセス複製クエリをセクションの中で持つ為に **UNTIL** を利用していたら、スレーブ サーバがスタートする時に SQL スレッドが起動するのを防ぐ為に、スレーブを **--skip-slave-start** オプションでスタートする事が推奨されています。予期せぬサーバの再スタートによって忘れられる事を防ぐ為、コマンドライン上ではなく、オプション ファイル内でこのオプションを利用するのが一番良いでしょう。

SHOW SLAVE STATUS ステートメントは **UNTIL** 条件の現在の値を表示するアウトプット フィールドを含んでいます。

MySQL の古いバージョン内では(4.0.5以前)、このステートメントは **SLAVE START** と呼ばれていました。この利用方法は MySQL 5.1 内でいまだに後方互換の為に許容されていますが、廃止予定になっています。

STOP SLAVE 構文

```
STOP SLAVE [thread_type [ , thread_type ] ... ]
```

```
thread_type: IO_THREAD | SQL_THREAD
```

スレーブ スレッドを停止します。**STOP SLAVE** は **SUPER** 権限を必要とします。

START SLAVE のように、このステートメントは、スレッドに名前を付けたり、スレッドを停止したりする為に **IO_THREAD** や **SQL_THREAD** オプションと共に利用されます。

MySQL の古いバージョン内では(4.0.5以前)、このステートメントは **SLAVE STOP** と呼ばれていました。この利用方法は MySQL 5.1 内でいまだに後方互換の為に許容されていますが、廃止予定になっています。

プリペアド ステートメントの為の SQL 構文

MySQL 5.1 はサーバ サイドのプリペアド ステートメントへのサポートを提供します。妥当なクライアント プログラム インターフェースを利用するという条件で、このサポートは MySQL 4.1 内でインプリメントされた有効なクライアント/サーバ バイナリ プロトコルを駆使します。インターフェース候補は、MySQL C API クライアント ライブラリ (C プログラムの為の物)、MySQL コネクタ/J(プログラムの為の物)、そして MySQL コネクタ/NET です。例えば、C API はそのプリペアド ステートメント API を構成する関数呼び出しのセットを提供します。詳しくは [準備されたC APIステートメント](#)。を参照してください。別の言語インターフェースは、バイナリ プロトコルを C クライアント ライブラリ内でリンクさせて利用するプリペアド ステートメントのサポートを提供する事ができます。その1つの例が、PHP 5.0 以降で有効な、[mysqli extension](#) です。

プリペアド ステートメントの代替 SQL インターフェースが有効です。このインターフェースはプリペアド ステートメント API にバイナリ プロトコルを利用する事ほど有効では有りませんが、これは SQL レベルで直接有効なのでプログラミングを必要としません。

- 有効なプログラミング インターフェースが無い時にこれを利用する事ができます。
- **mysql** クライアント プログラムのように、実行されるサーバに SQL ステートメントを送る事を許容する全てのプログラムからこれを利用する事ができます。
- もしクライアントが古いバージョンのクライアント ライブラリを利用していてもこれを利用する事ができます。たった1つ要求される事は、プリペアド ステートメントの SQL 構文をサポートするのに充分新しいサーバに接続できなければいけないという事です。

プリペアド ステートメントの SQL 構文は、次のような場合に利用する為の物です。

- プリペアド ステートメントが、それをコード化する前に自分のアプリケーション内でどのように機能するかをテストしたい。

- アプリケーションが、プリペアド ステートメントを実行するに当たり問題が発生し、何が原因なのかを調査したい。
- バグ リポートをファイルする為に、プリペアド ステートメントについて起きている問題を説明するテスト ケースを作成したい。
- プリペアド ステートメントを利用したいが、それらをサポートするプログラム API へのアクセスが無い。

プリペアド ステートメントの SQL 構文が3つの SQL ステートメントに基づいている。

- **PREPARE stmt_name FROM preparable_stmt**

PREPARE ステートメントはステートメントを準備し、後でそのステートメントを参照する物によって、名前 **stmt_name** を割り当てます。ステートメント名は大文字と小文字を区別しません。 **preparable_stmt** は文字列直定数、またはステートメントのテキストを含むユーザ変数です。テキストは複数ステートメントではなく、単一 SQL ステートメントを表さなければいけません。ステートメント内で '?' 文字は、後でクエリを実行する時にデータ値がクエリのどこに結合されるかを指示するパラメータ メーカとして利用されます。 '?' 文字は、それらを文字列値に結合したいとしても、引用符で囲んではいけません。パラメータ メーカは、SQL キーワードや識別子等ではなく、データ値が現れるところでのみ利用する事ができます。

もし与えられた名前を持つプリペアド ステートメントが既に存在したら、新しいステートメントが準備される前にそれは暗黙的に割り当て解除されます。これは、もし新しいステートメントがエラーを含み準備できないとしたらエラーは返され、与えられた名前のステートメントは存在しない、という意味です。

プリペアド ステートメントの範囲は、それが作成される範囲内のクライアント セッションです。他のクライアントはそれを見る事ができません。

- **EXECUTE stmt_name [USING @var_name [, @var_name] ...]**

ステートメントを準備した後、プリペアド ステートメント名を参照する **EXECUTE** ステートメントを利用してそれを実行します。もしプリペアド ステートメントがパラメータ メーカを含んでいたら、パラメータに結合される値を含むユーザ変数をリストする **USING** 条項を提供する必要があります。パラメータ値はユーザ変数のみで提供でき、**USING** 条項はステートメント内のパラメータ メーカ数と全く同数の変数に名前をつける必要があります。

各実行の前に、与えられたプリペアド ステートメントを複数回実行し、それに異なる変数を与え、また変数を異なる値に設定する事ができます。

- **{DEALLOCATE | DROP} PREPARE stmt_name**

プリペアド ステートメントの割り当てを解除するには、**DEALLOCATE PREPARE** ステートメントを利用してください。割り当て解除した後プリペアド ステートメントを実行しようとするエラーが発生します。

もし以前のプリペアド ステートメントの割り当て解除をしないでクライアント セッションを終了しようとする、サーバがそれを自動的に割り当て解除します。

次の SQL ステートメントはプリペアド ステートメント内で利用できます。 **CREATE TABLE**、**DELETE**、**DO**、**INSERT**、**REPLACE**、**SELECT**、**SET**、**UPDATE**、そしてほとんどの **SHOW** ステートメント。

MySQL 5.1.10 以降、次の追加ステートメントがサポートされています。

```
ANALYZE TABLE
OPTIMIZE TABLE
REPAIR TABLE
```

MySQL 5.1.12 以降、次の追加ステートメントがサポートされています。

```
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
| LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
```

```

REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN

```

その他のステートメントはまだサポートされていません。

次の例は、2辺の長さが分かっている3角形の斜辺を算出するステートメントを準備する為の、2つの同等な方法を表しています。

最初の例は、ステートメントのテキストを提供する為に、文字列直定数を利用してプリペアド ステートメントを作成する方法を表しています。

```

mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|      5 |
+-----+
mysql> DEALLOCATE PREPARE stmt1;

```

2つ目の例も似ていますが、ステートメントのテキストをユーザ変数として提供しています。

```

mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|      10 |
+-----+
mysql> DEALLOCATE PREPARE stmt2;

```

プレースホルダは、プリペアド ステートメントを利用する時に [LIMIT](#) 条項の引数に対して利用できます。詳しくは [SELECT 構文](#) を参照してください。

プリペアド ステートメントの SQL 構文は、ネスト化された種類の中では利用できません。これは、[PREPARE](#) にパスしたステートメント自体は [PREPARE](#)、[EXECUTE](#)、または [DEALLOCATE PREPARE](#) ステートメントになり得ないということです。

プリペアド ステートメントの SQL 構文は、プリペアド ステートメント API コールを利用する事とは違います。例えば、[PREPARE](#)、[EXECUTE](#)、または [DEALLOCATE PREPARE](#) ステートメントを準備する為に [mysql_stmt_prepare\(\)](#) C API 関数を利用する事はできません。

プリペアド ステートメントの SQL 構文はストアード プロシージャ内で利用できますが、ストアード ファンクションやトリガ内では利用できません。

プリペアド ステートメントの SQL 構文は、マルチ ステートメント(';' 文字で分割される、単一文字列内の複数ステートメントの事)をサポートしません。

第13章 ストレージエンジンとテーブルタイプ

目次

MySQLストレージエンジンアーキテクチャの概要	794
共通データベースサーバ層	795
プラグブルなストレージエンジンアーキテクチャ	795
サポートされたストレージエンジン	796
ストレージエンジンの選択	797
トランザクションエンジンと、非トランザクションエンジンの比較	797
その他のストレージエンジン	798
ストレージエンジンの設定	799
MyISAM ストレージエンジン	799
MyISAM スタートアップオプション	801
キーに必要な領域	802
MyISAM テーブルストレージフォーマット	802
MyISAM テーブルの問題点	804
InnoDB ストレージ エンジン	806
InnoDB 概要	806
InnoDB 連絡先情報	806
InnoDB 設定	807
InnoDB 起動オプションとシステム変数	812
InnoDB テーブルスペースを作成する	818
InnoDB テーブルの作成と利用	819
InnoDB データとログ ファイルの追加と削除	825
InnoDB データベースのバックアップと復旧	826
InnoDB データベースを別のマシンに移動する	829
InnoDB トランザクション モデルとロック	829
InnoDB パフォーマンス チューニング ヒント	838
マルチバージョンの実装	843
InnoDB テーブルとインデックス構造	844
InnoDB ファイル領域の管理とディスク I/O	846
InnoDB エラー処理	848
InnoDB テーブル上の制約	853
InnoDB トラブルシューティング	855
MERGE ストレージエンジン	856
MERGE テーブルの問題点	859
MEMORY (HEAP) ストレージエンジン	860
EXAMPLE ストレージエンジン	862
FEDERATED ストレージエンジン	862
FEDERATED ストレージエンジン概要	862
FEDERATED テーブルの作成方法	863
FEDERATED ストレージエンジン 注意とヒント	866
FEDERATED ストレージエンジンリソース	866
ARCHIVE ストレージエンジン	866
CSV ストレージエンジン	867
CSVテーブル修正と確認	868
CSV制限	869
BLACKHOLE ストレージエンジン	869

MySQLは、異なるテーブルタイプのハンドラとして機能するいくつかのストレージエンジンをサポートします。MySQLストレージエンジンは、トランザクションセーフなテーブルを扱うものと、トランザクションセーフではないテーブルを扱うものの両方を含んでいます。

MySQL 5.1で MySQL ABは、動作中のMySQLサーバにストレージエンジンをロードしたりアンロードしたりできる、新しいプラグブルなストレージエンジンアーキテクチャを導入しました。

この章では、14章MySQL Clusterで紹介されているNDB Cluster以外のMySQLストレージエンジンについて説明します。プラグブルなストレージエンジンアーキテクチャについても説明しています。(MySQLストレージエンジンアーキテクチャの概要を参照してください。)

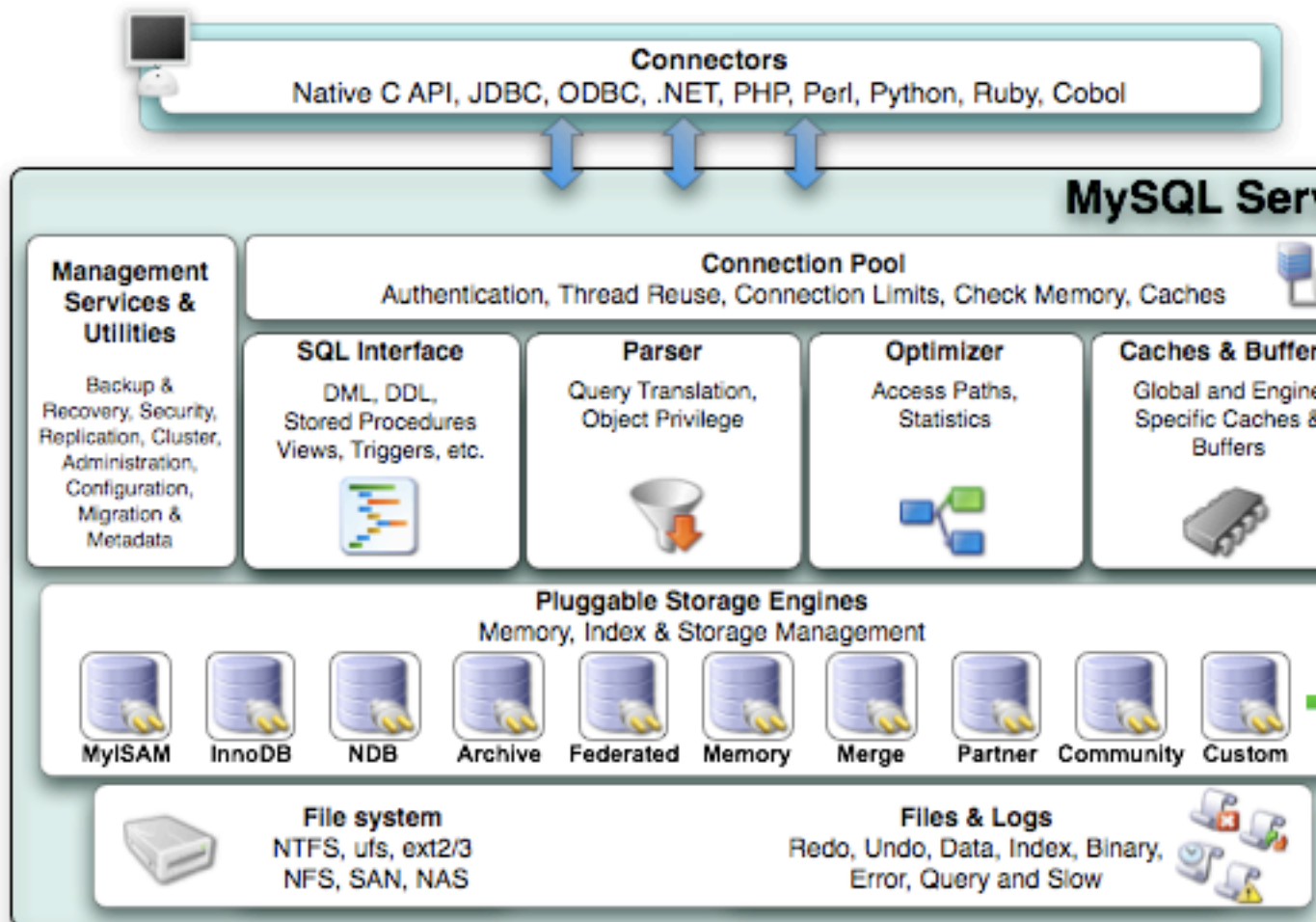
MySQL ストレージエンジンについてのよくある質問に対する答えに関しては、MySQL 5.1 FAQ — Storage Enginesを参照してください。

MySQLストレージエンジンアーキテクチャの概要

MySQLのプラグ可能ストレージエンジンアーキテクチャは、特殊なアプリケーションコーディングを必要とせずに、データベースのプロが専門のストレージエンジンを選択する事を可能にします。MySQLサーバアーキテクチャは、一貫した簡単なアプリケーションモデルとAPIを供給する事によって、アプリケーションプログラマとDBAを、ストレージレベルでの下位レベルの詳細な実装から切り離します。それにより、別々のストレージエンジン間で別々の機能があったとしても、アプリケーションはそれらの違いから守られるのです。

MySQLプラグブルストレージエンジンアーキテクチャは 図13.1「プラグブルストレージエンジンを利用したMySQLアーキテクチャ」で紹介しています。

図13.1 プラグブルストレージエンジンを利用したMySQLアーキテクチャ



プラグブルストレージエンジンアーキテクチャは、全てのストレージエンジンに横断的に共通する標準的な管理とサポートサービスのセットを提供します。ストレージエンジン自体は、物理的サーバレベルで管理されている基礎データに直接働きかけるデータベースサーバのコンポーネントです。

この効果的なモジュール式のアーキテクチャは、特殊なアプリケーションニーズ — データウェアハウス、トランザクションプロセス、またはハイアベイラビリティなど — をターゲットとする人達に甚大な利益をもたらします。

アプリケーションプログラマとDBAは、ストレージエンジンの上位にある接続APIとサービスレイヤを通してMySQLデータベースと対話します。もしアプリケーションの変更が、下位のストレージエンジンの変更を必要とするような要求を引き起こしたり、新しい要求に対応するために複数のストレージエンジンが追加されたりしても、コーディングやプロセスの変更は特に必要ありません。MySQLサーバアーキテクチャは、ストレージエンジン全体に適応する一貫した使いやすいAPIによって、ストレージエンジンの複雑さからアプリケーションを守ります。

共通データベースサーバ層

MySQLプラグブルストレージエンジンは、特別なアプリケーションニーズをターゲットとする特徴集合を有効にし、実行するだけでなく、実際のデータ入出力操作に責任を持つMySQLデータベースサーバの中のコンポーネントです。特殊なストレージエンジンを利用する事の最大の利点は、特定のアプリケーションに必要な特徴だけが供給されるという事です。その結果、最終的にデータベースはより効果的で高性能になり、システムオーバーヘッドが少なくなります。業界標準ベンチマークの独自仕様モノリシック構造に適応し、さらに先を進んでいる事が、MySQLが常に高性能として知られている理由の1つです。

技術的観点から見て、ストレージエンジンの中にある固有サポート構造基盤コンポーネントにはどのような物があるでしょう？ 重要な特徴の区別には次のような物があります。

- 並行処理 — いくつかのアプリケーションは他の物と比べて、より粒度の細かいロックを必要とします。(行レベルロックのような物) 適切なロック方法を選択する事によりオーバーヘッドを減らす事ができ、その為全体的な性能も向上します。この分野はまた、マルチバージョン並行処理制御や「スナップショット」読み込みのような機能もサポートします。
- トランザクションサポート — 全てのアプリケーションがトランザクションを必要とするわけではありませんが、必要とするアプリケーションに対しては、ACIDコンプライアンスのような明確な条件があります。
- 参照整合性 — サーバを持つ事によって、DDLに定義された外部キーを通してリレーショナルデータベース参照整合性が強調されます。
- 物理記憶 — これは、ディスク装置にデータを記憶する為に利用されるフォーマットだけでなく、テーブルとインデックス全体のページサイズを含んでいます。
- インデックスサポート — 異なるアプリケーションシナリオは異なるインデックスストラテジーの恩恵を受ける傾向があります。通常各ストレージエンジンは専用のインデックス方法を持ちますが、いくつかの方法は(Bツリーインデックス等)ほとんど全てのエンジンに共通しています。
- メモリキャッシュ — 異なるアプリケーションは、いくつかのメモリキャッシュストラテジに対して他の物より反応がよいです。その為、いくつかのメモリキャッシュが全てのストレージエンジン(ユーザ接続やMySQLの高速クエリキャッシュに利用される物など)に共通であるとしても、他の物は特定のストレージエンジンが稼働する時だけ一意に定義されます。
- パフォーマンスエイド — これはパネル操作の為に複数の入出力スレッド、スレッド並行処理、データベースチェックポイント、大量挿入操作等を含みます。
- その他のターゲット特性 — これは、地球空間的操作へのサポート、特定データのマニピュレーション操作への安全保障制限、そしてその他の類似特性などを含みます。

プラグブルなストレージエンジン構造基盤コンポーネントのセットはそれぞれ、特定のアプリケーションに選りすぐった利益をもたらすようにデザインされています。反対に、コンポーネント特性を無効にすると、不必要なオーバーヘッドを削減する事ができます。特定のアプリケーションの要求事項を理解し、適切なMySQLストレージエンジンを選択する事が、システム全体の効率と性能に劇的な衝撃を与えるのは明らかです。

プラグブルなストレージエンジンアーキテクチャ

MySQL 5.1では、MySQL ABが稼働中のMySQLサーバにストレージエンジンをロードしたり、そこからアンロードしたりできる、新しいプラグブルなストレージエンジンアーキテクチャを紹介しました。

ストレージエンジンのプラグイン

ストレージエンジンを使用する前に、ストレージエンジンプラグイン共用ライブラリを `INSTALL PLUGIN` ステートメントを利用してMySQLにロードしなければいけません。例えば、`EXAMPLE` エンジンプラグインが `ha_example` と

名づけられ、共有ライブラリが `ha_example.so` と名づけられると、次のステートメントを利用してロードする事になります。

```
INSTALL PLUGIN ha_example SONAME 'ha_example.so';
```

共有ライブラリはMySQLサーバプラグインディレクトリの中に無ければいけません。その場所は `plugin_dir` システム変数によって指示されます。

ストレージエンジンのアンプラグ

ストレージエンジンをアンプラグするには、`UNINSTALL PLUGIN` ステートメントを利用します。

```
UNINSTALL PLUGIN ha_example;
```

もし、テーブルに必要なストレージエンジンをアンプラグすると、それらのテーブルはアクセス不可になりますが、ディスク上には存在し続けます。(アクセス可能な場所)ストレージエンジンをアンプラグする前に、それらを利用しているテーブルが無い事を確認してください。

プラグ可能ストレージのセキュリティインプリケーション

プラグ可能なストレージエンジンをインストールするには、MySQLプラグインディレクトリの中にプラグインファイルがなければいけません。そして、`INSTALL PLUGIN` ステートメントを発行するユーザは `mysql.plugin` テーブルの為の `INSERT` 権限を持つ必要があります。

サポートされたストレージエンジン

MySQL 5.1は次のストレージエンジンをサポートします。

- **MyISAM** — デフォルトのMySQLストレージエンジンと、ウェブ、データウェアハウス、そしてその他のアプリケーション環境で一番利用されるストレージエンジンです。MyISAMは全てのMySQLコンフィギュレーションの中でサポートされています。そして、MySQLに他のストレージエンジンを設定しない限り、これがデフォルトとして利用されます。
- **InnoDB** — トランザクションプロセスアプリケーションに利用され、ACIDトランザクションサポートや外部キーなどを含む、複数の特徴をサポートします。InnoDBは全てのMySQL 5.1 バイナリディストリビューションの中にデフォルトとして含まれています。ソースディストリビューションの中では、好きなようにMySQLを設定する事によって、エンジンを有効にも無効にもできます。
- **Memory** — 参照事項や迅速なデータ検索を必要とする環境で、きわめて高速なアクセスで全てのデータをRAMの中に格納します。このエンジンは以前は **HEAP** エンジンとして知られていました。
- **Merge** — MySQL DBAや開発者が、一連の同一 **MyISAM** テーブルを論理的にグループ化し、それらを1つのオブジェクトとして参照付ける事を可能にします。VLDB データウェアハウスと同じで、VLDBに効果的です。
- **Archive** — サイズが大きいほとんど参照されない履歴、アーカイブ、セキュリティ監査情報を格納したり、検索する為の完璧な解決法を提供します。
- **Federated** — いくつもの物理的サーバーから、別々のMySQLサーバーをリンクさせて1つの論理データベースを作成する能力を提供します。分散、またはデータマート環境に大変効果的です。
- **NDB** — 高い検索機能と、できるだけ長い稼働時間を必要とするアプリケーションにぴったりの、クラスタ化されたデータベースエンジンです。
- **CSV** — ストレージエンジンはコンマ区切りの値を使ったフォーマットでデータをテキストファイルに保存します。CSVエンジンは、CSVフォーマットにインポート・エクスポートする事ができる他のソフトやアプリケーション間でデータを簡単に交換する為に利用する事ができます。
- **Blackhole** — ブラックホールストレージエンジンはデータの受け入れはしますが、格納はせず、検索しても結果は得られません。この機能性は、データが自動的に複製される分散型のデータベースデザインの中で利用できますが、局所的に格納はされません。
- **Example** — このストレージエンジンは「スタブ」エンジンで実装されており、何の機能も持ちません。このエンジンを利用してテーブルを作成できますが、データの格納も検索もできません。このエンジンの目的は、MySQLソースコードの中で新しいストレージエンジンを作成する方法を説明する為の、見本の役割を果たす事です。それ自体は、ソフトウェア開発者向のものです。

この章では、14章MySQL Clusterで紹介されているNDB Cluster以外のMySQLストレージエンジンについて説明します。

サーバやスキーマ全体に同じストレージエンジンを利用しなければならないという制限はないという事を覚えておいて下さい。スキーマの中のそれぞれのテーブルに違うストレージエンジンを利用する事ができます。

ストレージエンジンの選択

MySQLから提供される様々なストレージエンジンは異なるユースケースを想定してデザインされています。プラグ可能ストレージアーキテクチャを効果的に利用する為には、様々なストレージエンジンの利点と欠点を知っておく事が役立ちます。次のテーブルは、MySQLが提供するいくつかのストレージエンジンの概要を表しています。

特徴	MyISAM	メモリ	InnoDB	アーカイブ	NDB
ストレージリミット	256TB	Yes	64TB	No	384EB ^[4]
トランザクション	No	No	Yes	No	Yes
ロック精度	テーブル	テーブル	行	行	行
MVCC (スナップショット読み込み)	No	No	Yes	Yes	No
地球空間サポート	Yes	No	Yes ^[1]	Yes ^[1]	Yes ^[1]
Bツリーインデックス	Yes	Yes	Yes	No	Yes
ハッシュインデックス	No	Yes	No	No	Yes
前文検索インデックス	Yes	No	No	No	No
クラスタ化されたインデックス	No	No	Yes	No	No
データキャッシュ	No	N/A	Yes	No	Yes
インデックスキャッシュ	Yes	N/A	Yes	No	Yes
圧縮データ	Yes	No	No	Yes	No
暗号化されたデータ ^[2]	Yes	Yes	Yes	Yes	Yes
クラスタデータベースサポート	No	No	No	No	Yes
レプリケーションサポート ^[3]	Yes	Yes	Yes	Yes	Yes
外字サポート	No	No	Yes	No	No
バックアップ / ポイントインタイムリカバリ ^[3]	Yes	Yes	Yes	Yes	Yes
クエリキャッシュサポート	Yes	Yes	Yes	Yes	Yes
データディレクトリの更新統計	Yes	Yes	Yes	Yes	Yes

- [1] ストレージエンジンは空間データタイプはサポートしますが、それらをインデックスはしません。
- [2] ストレージエンジンの中よりも、サーバの中で実行されます。(暗号化機能を利用)
- [3] ストレージエンジンの中よりも、サーバの中で実行されます。
- [4] EB = exabyte (1024 * 1024 terabyte)

トランザクションエンジンと、非トランザクションエンジンの比較

トランザクションセーフテーブル(TSTs)は、非トランザクションセーフテーブル(NTSTs)よりも利点がいくつかあります。

- より安全です。もしMySQLがクラッシュしたり、ハードウェアに問題がおきても、自動修復機能がバックアップとトランザクションログでデータを取り戻す事ができます。
- **COMMIT** ステートメントを利用して、いくつものステートメントを組み合わせたり、同時に受け入れたりする事ができます。(自動コミットが無効の時)
- 変更を無視する為に **ROLLBACK** を実行する事ができます。(自動コミットが無効の時)
- もし更新に失敗したら、全ての変更は元に戻ります。(非トランザクションセーフテーブルを利用すると、全ての変更は永久的です。)
- トランザクションセーフストレージエンジンは、読み込みと更新の並行作業が多いテーブルに、より良い並行処理を提供する事ができます。

トランザクションセーフと、非トランザクションセーフテーブルの両方を同じステートメントの中で組み合わせ、両方の利点を利用する事ができます。しかし、MySQLがいくつかのトランザクションセーフストレージエンジンをサポートしていても、良い結果を出す為に、自動コミットが無効の時は異なるストレージエンジンを1つのトランザクションの中に混在させない方がよいです。例えば、もしこれをしてしまうと、非トランザクションセーフテーブルへの変更が行われてしまい、ロールバックできなくなってしまいます。複数のストレージエンジンをミックスして利用する時に起こるこのような問題や、これ以外の問題に関しての情報は、**START TRANSACTION**、**COMMIT**、そして **ROLLBACK** 構文を参照してください。

非トランザクションセーフテーブルには、トランザクションオーバーヘッドが無い為に起こるいくつかの利点があります。

- より速い
- より少ないディスク領域
- 更新時に必要とするメモリがより少ない

その他のストレージエンジン

その他のストレージエンジンは、カスタムストレージエンジンインターフェースを利用した第三者やコミュニティメンバーから入手する事ができるでしょう。

[MySQLフォージストレージエンジン](#) ページにある、第三者ストレージエンジンのリストに、更なる情報が紹介されています。

注記

第三者エンジンはMySQLにサポートされていません。これらのエンジンに関しての更なる情報、文書、インストールガイド、バグレポート、または支援等が必要であれば、エンジンディレクトリの開発者に連絡してください。

利用可能な第三者エンジンは次の物を含んでいます。更なる情報については、フォージリンクを参照してください。

- **PrimeBase XT (PBXT)** — PBXTはモデム、ウェブベース、高い同時並行性を持つ環境の為にデザインされています。
- **RitmarkFS** — RitmarkFSを利用すると、SQLクエリを利用してファイルシステムにアクセスしたり、複製したりできます。RitmarkFSはファイルシステムの複製と、ディレクトリ変更のトラッキングもサポートします。
- **分散データエンジン** — 分散データエンジンは、作業負荷統計による分散データにストレージエンジンを提供するという作業専用の、オープンソースプロジェクトです。
- **mdbtools** — Microsoft Access **.mdb** データベースファイルに読み取り専用アクセスを許可するプラグ可能なストレージエンジンです。
- **solidDB for MySQL** — MySQLのsolidDBストレージエンジンは、オープンソースであり、MySQLサーバーのトランザクションストレージエンジンです。これは、強固なトランザクションデータベースを必要とする基幹インプリメンテーションの為にデザインされています。MySQLのsolidDBストレージエンジンは、予測される全てのトランザクション分離レベル、低レベルロック、そしてノンブロッキングの読み込みと書き込みができるマルチバージョン

並行処理コントロール(MVCC)を持つフルACIDコンプライアンスをサポートする、マルチスレッドストレージエンジンです。

プラグابلなストレージエンジンアーキテクチャと一緒に利用できるカスタムストレージエンジンの開発については、MySQL内部マニュアルの中にある [Writing a Custom Storage Engine on MySQL Forge](#) を参照してください。

ストレージエンジンの設定

新しいテーブルを作成する時、**ENGINE** テーブルオプションを **CREATE TABLE** ステートメントに加える事によって、どのストレージエンジンを利用するかを指定できます。

```
CREATE TABLE t (i INT) ENGINE = INNODB;
```

ENGINE が **TYPE** オプションを省略すると、デフォルトのストレージエンジンが利用されます。通常、これは **MyISAM** ですが、`--default-storage-engine` が `--default-table-type` サーバ始動オプションを利用、または `my.cnf` 構造ファイルの中の `default-storage-engine` が `default-table-type` オプション を設定する事で変更できます。

現在のセッションの途中で `storage_engine` 変数を設定する事によって、利用するデフォルトストレージエンジンを設定する事ができます。

```
SET storage_engine=MYISAM;
```

MySQLコンフィギュレーションウィザードを利用してMySQLがWindowsにインストールされた時は、**MyISAM**の代わりに **InnoDB** ストレージエンジンをデフォルトとして選択する事ができます。詳しくは [データベースの使用ダイアログ](#) を参照してください。

テーブルを別のストレージエンジンに変換するには、新しいエンジンを指示する **ALTER TABLE** ステートメントを利用してください。

```
ALTER TABLE t ENGINE = MYISAM;
```

CREATE TABLE 構文と **ALTER TABLE** 構文を参照して下さい。

もし、編集されていないストレージエンジンや、編集はされていても無効になっているストレージエンジンを利用しようとすると、MySQLは代わりにデフォルトのストレージエンジンを利用してテーブルを作成します。通常その時利用されるのは **MyISAM** です。この機能は、別々のストレージエンジンをサポートするMySQLサーバ間でテーブルをコピーしたい時に便利です。(例えば、複製を設定する時、マスターサーバは安全の為トランザクションストレージエンジンをサポートするでしょう。しかし、スレーブサーバはスピードの為に、非トランザクションストレージエンジンだけを利用します。)

新しいMySQLユーザにとっては、利用不可デフォルトストレージエンジンの自動置換は複雑かもしれません。ストレージエンジンが自動的に変更された時は必ず、警告メッセージが表示されます。

テーブルとカラムの定義を維持する為に、MySQLは毎回新しいテーブルの為に `.frm` を作成します。ストレージエンジンによっては、テーブルのインデックスとデータは複数の別のファイルに格納されるでしょう。サーバがストレージエンジンレベルの上位に `.frm` ファイルを作成します。個々のストレージエンジンは、それらが管理するテーブルが必要とする追加ファイルも作成します。もしテーブル名が特別な文字を含んでいる場合、そのテーブルファイルの名前は [ファイル名への識別子のマッピング](#) に表されているようにそれらの文字が暗号化された形を含んだ物になります。

データベースは異なるタイプのテーブルを含む事があります。これは、テーブルが同じストレージエンジンで作成されなくても良い事を表しています。

MyISAM ストレージエンジン

MyISAM はデフォルトストレージエンジンです。古い **ISAM** コードに基づいていますが、便利な拡張子を多く持っています。(MySQL 5.1 は **ISAM** をサポート しない 事を覚えておってください。)

各 **MyISAM** テーブルはディスク上に3つのファイルとして保管されます。そのファイル名はテーブル名で始まり、ファイルタイプを指示する拡張子が付きます。`.frm` ファイルはテーブルフォーマットを格納します。データファイルには `.MYD` (**MYData**) 拡張子が付きます。インデックスファイルには `.MYI` (**MYIndex**) 拡張子が付きます。

MyISAM テーブルが必要だということを明確に指示したい場合は、**ENGINE** テーブルオプションを指定します。

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

通常、**MyISAM** ストレージエンジンを指示するのに**ENGINE**を使用する必要はありません。変更されない限り、**MyISAM** がデフォルトエンジンです。デフォルトが変更されている可能性がある場合に **MyISAM** を確実に利用する為には、**ENGINE** オプションを確実に包括してください。

mysqlcheck クライアントが **myisamchk** ユーティリティで **MyISAM** テーブルをチェックしたり、修正したりする事ができます。容量を節約する為に **myisampack** を使って **MyISAM** テーブルを圧縮できます。[mysqlcheck — テーブル メンテナンスと修復プログラム](#)、[myisamchk でクラッシュ リカバリ](#)、[myisampack — 圧縮された、読み取り専用 MyISAM テーブルを作成する](#)。を参照して下さい。

MyISAM テーブルには次のような特徴があります。

- 全てのデータ値は最初は下位バイトで格納されます。その為、データマシーンとOSは独立します。バイナリポータビリティに対する唯一の条件は、マシンが2個の補数符号付の整数とIEEE浮動小数点フォーマットを使用するという事だけです。これらの条件は主流マシンの間で広く利用されています。バイナリ互換性は、固有のプロセッサを持つ事がある内蔵システムには適合しない可能性があります。

データを最初に下位バイトで格納する事に関して、スピードに関する重大なペナルティはありません。テーブル行内のバイトは通常非同盟であり、非同盟バイトを順番に読み込むのは、逆の順番で読み込むよりも少し手間がかかります。また、カラム値をフェッチするサーバー内のコードは、他のコードに比べるとタイムクリティカルではありません。

- 全ての数値キー値は、よりよいインデックス圧縮の為に、上位バイトの物から先に格納されます。
- 大きいファイル (最高63-bitファイル) は、専用のファイルシステムとOSによってサポートされています。
- **MyISAM** テーブルには 2^{32} (~4.295E+09) 行の制限があります。MySQL を `--with-big-tables` オプションで作成すると、行の制限を $(2^{32})^2$ (1.844E+19) 行に増加させる事ができます。詳しくは [典型的な configure オプション](#) を参照してください。MySQL 5.0.4バージョンより、全ての標準バイナリはこのオプションで作成されます。
- 1つの**MyISAM** テーブルのインデックス数は最高64です。これは再コンパイルする事によって変更できます。MySQL 5.1.4から、1つの**MyISAM** テーブルに許可されているインデックス数の最高値が **N** の場合、`--with-max-indexes=N` オプションを使って [configure](#) を呼び出す事によって体型を設定できるようになりました。**N** は128以下でなければいけません。MySQL 5.1.4以前のバージョンではソースを変える必要があります。

各インデックスのカラム最高数は16です。

- 最高キー長さは1000バイトです。これも、ソースを変えたり、再コンパイルする事によって変える事ができます。キー長さが250バイト以上の場合、デフォルトの1024バイトよりも大きいキーブロックサイズが使用されます。
- ソートされた順番で行が挿入された時(**AUTO_INCREMENT** カラムを使用している時と同様に)、高ノードが1つのキーだけを含むように、インデックスツリーが分割されます。このおかげでインデックスツリーのスペース利用は向上します。
- 1つのテーブルに対する1つの **AUTO_INCREMENT** カラムの内部操作がサポートされます。**MyISAM** は**INSERT** と**UPDATE** 操作のカラムを自動的に更新します。そのおかげで **AUTO_INCREMENT** カラムは速くなります。(最低10%)シーケンスの最上部の値が削除された後に再利用される事はありません。(**AUTO_INCREMENT** が複数カラムインデックスの最後のカラムとして定義された場合は、シーケンスの最上部から削除された値が再利用される事があります。) **AUTO_INCREMENT** 値は **ALTER TABLE** や **myisamchk** でリセットできます。
- 動的サイズの行は、削除作業をアップデートと挿入でミックスした時には断片化される事が少なくなります。隣のブロックが削除された時に、隣接している削除されたブロックを自動的に一体化したり、ブロックを拡張したりする事でこれを行います。
- もしテーブルのデータファイルの間に開いているブロックがなければ、他のスレッドが読み込みをしているのと同時に新しい行を**INSERT**する事ができます。(これらは並列挿入として知られています。)行を削除する事や、それ自体の現在の内容よりも多くのデータを持つ動的長さの行をアップデートする事によって、フリーブロックが生じます。全てのフリーブロックが使用された(記入された)時、その後の挿入は再度並列になります。詳しくは [同時挿入](#) を参照してください。
- スピードを上げるために**DATA DIRECTORY** と **INDEX DIRECTORY** テーブルオプションへの**CREATE TABLE** を使って、データファイルとインデックスファイルを別々のディレクトリに入れる事ができます。詳しくは [CREATE TABLE 構文](#) を参照してください。

- BLOB と TEXT カラムはインデックスする事ができます。
- NULL 値がインデックスカラムの中で許可されています。1つのキーに対して0-1 バイト使われます。
- それぞれのキャラクタカラムは異なるキャラクタセットを持つ事ができます。詳しくは9章キャラクタセットサポートを参照してください。
- MyISAM インデックスファイルの中に、テーブルが正しく閉じられたかどうかを表すフラグがあります。もし `mysqld` が `--myisam-recover` オプションで開かれると MyISAM テーブルは自動的にチェックされて、もし正しく閉じられてなかった時には修正されます。
- `myisamchk` を `--update-state` オプションで実行すると、チェックしたテーブルにマークをつけます。 `myisamchk --fast` はこのマークがないテーブルだけをチェックします。
- `myisamchk --analyze` はキー全体に対してするのと同様に、キーの一部に統計データを格納します。
- `myisampack` は BLOB と VARCHAR カラムを圧縮する事ができます。

MyISAM は次のような特徴をサポートします。

- 本物の VARCHAR タイプへのサポート; VARCHAR カラムは1バイトか2バイトで格納された長さから始まります。
- VARCHAR カラムを持つテーブルの行の長さは固定、または動的になり得ます。
- 1つのテーブル内の VARCHAR と CHAR カラム長さの合計は、最高で 64KBになるでしょう。
- 任意長さ UNIQUE 制約。

追加情報

- MyISAM ストレージエンジンを専門に扱うフォーラムがあります。 <http://forums.mysql.com/list.php?21>。

MyISAM スタートアップオプション

MyISAM テーブルの性能を変える為に次の `mysqld` オプションを使う事ができます。追加情報については コマンド オプションを参照してください。

- `--myisam-recover=モード`

クラッシュした MyISAM テーブルの自動修復モードをセットします。

- `--delay-key-write=ALL`

MyISAM テーブルへの書き込みの間にキーバッファをフラッシュしないでください。

注:もしこれをするのであれば、テーブルの使用中に、他のプログラムから MyISAM テーブルにアクセスしてはいけません。(他のMySQL サーバーからのアクセスや、 `myisamchk` を利用してのアクセスなど。) もししてしまうと、インデックスが破損します。 `--external-locking` を利用してもこのリスクを排除する事はできません。

次のシステム変数は MyISAM テーブルの性能に影響を与えます。追加情報については システム変数を参照してください。

- `bulk_insert_buffer_size`

大量挿入最適化で使用するツリーキャッシュのサイズ。注:これはそれぞれのスレッドのリミットです!

- `myisam_max_sort_file_size`

MyISAM インデックスを再作成している最中にMySQLが使用を許されているテンポラリファイルの最大サイズ (REPAIR TABLE、ALTER TABLE、または LOAD DATA INFILEの最中)。もしファイルサイズがこの値よりも大きければ、代わりにキーキャッシュを使用してインデックスが作成されますが、この方法のほうが遅くなります。値はバイトで表示されます。

- `myisam_sort_buffer_size`

テーブルをリカバする時に使用されるバッファのサイズを設定します。

もし `--myisam-recover` オプションを利用して `mysqld` をスタートさせれば、自動修復が作動します。この場合、サーバーが `MyISAM` テーブルを開いた時に、テーブルにクラッシュのマークが付いているかどうかや、テーブルのオープンカウント変数が0でないかどうか、そして外部ロックが使用不可能な状態でサーバーを作動させているかどうかを確認します。もしこのような条件が整っていると、次のような事が起こります。

- サーバーがテーブルのエラーをチェックします。
- もしサーバーがエラーを発見すると、高速テーブル修復を試みます。(データファイルの保存はしますが、再作成はしません。)
- データファイル中のエラーのせいで修復が失敗したら、(例えば重複キーエラー)、サーバーは今度はデータファイルを再作成してもう一度修復を行います。
- もし修復がまた失敗したら、サーバーは古い修復オプションの方法でもう一度修復を試みます (ソートせず一行ずつ書く)。この方法であればどんなタイプのエラーも修復し、ディスクのスペースも少しで済みます。

MySQL Enterprise

```
## --myisam-recover #####MySQL Network Monitoring and Advisory Service#####
##### http://www-jp.mysql.com/products/enterprise/advisors.html#####
```

もし以前に完成されたステートメントから全ての行を回復できず、`--myisam-recover` オプション値の中の `FORCE` を指定しなければ、自動修復はエラーログにエラーメッセージを残して異常終了します。

```
Error: Couldn't repair table: test.g00pages
```

もし `FORCE` を指定すると、代わりにこのような警告が書かれます。

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

もし自動修復値が `BACKUP` を含んでいる時、修復プロセスは `tbl_name-datetime.BAK` の名前でファイルを作成します。これらのファイルを自動的にデータベースディレクトリからバックアップメディアに移動する `cron` スクリプトが必要です。

キーに必要な領域

`MyISAM` テーブルはBツリーインデックスを使用します。インデックスファイルのサイズは、全てのキーを大体 $(key\_length+4)/0.67$ のように計算し、それらを合計して大まかに算出する事ができます。これは、全てのキーがソートされた順番に挿入され、かつキーが全く圧縮されない時を想定した最悪のケースです。

文字列インデックスでは空白が圧縮されます。もしインデックスの最初の部分が文字列であれば、プリフィックスもまた圧縮されます。文字列カラムに含まれる後続の空白が長い場合、またはそのカラムが `VARCHAR` カラムであるためにその長さがフルに使用されることがない場合は、空白の圧縮によってインデックスファイルが上記の数値よりも小さくなります。プレフィックス圧縮は文字列で始まるキーに使用されます。同一のプリフィックスを持つ文字列が多数存在する場合は、プリフィックスの圧縮が役立ちます。

`MyISAM` テーブルでは、テーブル作成時に `PACK_KEYS=1` を指定する事で、数値のプリフィックスを圧縮する事もできます。この機能は、数値が上位バイトから順に格納される場合に、同一のプリフィックスを持つ整数キーが多数ある時に役立ちます。

MyISAM テーブルストレージフォーマット

`MyISAM` は3つの異なるストレージフォーマットをサポートします。そのうちの2つである静的、または動的フォーマットは、使用しているカラムタイプによって自動的に選択されます。3つめである圧縮フォーマットは、`mysampack` ユーティリティでしか作成する事ができません。

`BLOB` や `TEXT` カラムを持たないテーブルに対して、`CREATE TABLE` か `ALTER TABLE` カラムを使用する時、`ROW_FORMAT` テーブルオプションで、テーブルフォーマットを `FIXED` か `DYNAMIC` に強制する事ができます。

`ALTER TABLE` を使った `ROW_FORMAT=DEFAULT` を指定する事で、テーブルを復元する事ができます。

ROW_FORMAT についての情報は CREATE TABLE 構文を参照してください。

静的な(固定長) テーブルの特徴

静的フォーマットは MyISAM テーブルのデフォルトです。これは、テーブルが可変長カラムを持たない時に使用されます。(VARCHAR、VARBINARY、BLOB、また TEXT)それぞれの行は固定バイト数を利用して格納されます。

3つの MyISAM ストレージフォーマットの中で、静的フォーマットが一番シンプルで安全です。(一番破損しにくい)これはまた、データファイル中の行がディスク上で見つけられるという簡単さゆえ、オンディスクフォーマットの中でも一番早いです。インデックスの中で行数に基づいて行の位置を探すには、行数に行長をかけてください。また、テーブルをスキャンする時には、それぞれのディスクの読み込み操作を使って簡単に行の定数を読む事ができます。

もしMySQLサーバーが固定フォーマットのMyISAMファイルに書き込んでいる最中にコンピュータがクラッシュしたら、そのセキュリティが証明されます。この場合、myisamchk はそれぞれの行がどこで始まりどこで終わるかを簡単に測定する事ができますので、部分的に書かれた物以外の全ての行を大概再生する事ができます。MyISAM テーブルインデックスは、データ行に基づいていつでも再配列できるという事を覚えておいてください。

注記

固定長の行フォーマットはBLOB か TEXT カラムの無いテーブルでだけ使用する事ができます。明確な ROW_FORMAT 条項を持つカラムを使ってテーブルを作成する事によって、エラーや警告は発生しません。フォーマットの仕様は無視されるのです。

静的フォーマットにはこれらの特徴があります。

- CHAR と VARCHAR カラムは、タイプは変えられていませんが、特定のカラム幅にスペースが埋め込まれた物です。BINARY と VARBINARY カラムは幅が0x00 バイトで埋め込まれています。
- とても速いです。
- キャッシュが簡単です。
- 行が固定位置にあるので、クラッシュした後も修復が簡単です。
- 莫大な数の行を削除してしまい、フリーディスクスペースをOSに戻したい時以外は、再編成の必要はありません。これをするには、OPTIMIZE TABLE か myisamchk -r を使用してください。
- ほとんどの場合、動的フォーマットテーブルよりもディスクの領域を必要とします。

動的テーブルの特徴

もし MyISAM テーブルが可変長カラムを含んでいる場合(VARCHAR、VARBINARY、BLOB、または TEXT)、または ROW_FORMAT=DYNAMIC テーブルオプションで作成された場合は、動的ストレージフォーマットが使用されます。

動的フォーマットは、それぞれの行にその長さを示すヘッダーがあるので、静的フォーマットよりも少しだけ複雑です。更新の結果行が長くなった時には、フラグメント化する事ができます。(非連続単位での格納)

テーブルをデフラグメント化する為に OPTIMIZE TABLE か myisamchk -r を使用する事ができます。頻繁にアクセスや変更がある固定長カラムが、可変長カラムも含むテーブルの中にあるなら、フラグメント化を防ぐ為に、可変長カラムを他のテーブルに移動するとよいでしょう。

動的フォーマットにはこれらの特徴があります。

- 長さが4以下の物以外の全ての文字列カラムは動的カラムです。
- それぞれの行の先頭には、どのカラムが空文字列(文字列カラム)なのか、またはゼロ(数字カラム)なのかを示すビットマップが付いています。NULL 値を含むカラムは、これに含まれていない事を覚えておいてください。後続スペースの除去をした後に文字列カラムの長さがゼロになったり、数字カラムの値がゼロだったりすると、それらはビットマップの中でマークが付けられ、ディスク上には保存されません。空ではない文字列は、長さバイトに加えて文字列コンテンツとして保存されます。

- 通常、固定長テーブルに比べると少量のディスク容量を必要とします。
- それぞれの行は、必要とする容量のみを使用します。しかし、行が大きくなれば、要求されただけの単位に分割され、行のフラグメント化が行われます。例えば、行の長さを延長する情報を使って行を更新すると、その行はフラグメント化されます。このような場合は、性能を上げるために `OPTIMIZE TABLE` か `myisamchk -r` を時々行う必要があるでしょう。テーブル統計を得る為には `myisamchk -ei` を利用してください。
- 行がいくつもの単位にフラグメント化されていて、リンク(フラグメント)がなくなっているかもしれないので、静的フォーマットテーブルよりも、クラッシュ後の再配列は難しいです。
- 動的サイズの行の予想長さは次の表現を使って計算されます。

```

3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8

```

それぞれのリンクには6バイトのペナルティがあります。更新する事で行が拡大されると必ず動的行はリンクされます。新しいリンクはそれぞれ20バイトなので、次に行が拡大される時も同じリンクになるでしょう。そうでなければ別のリンクが作成されます。 `myisamchk -ed` を利用してリンク数を確認する事ができます。 `OPTIMIZE TABLE` か `myisamchk -r` を使って全てのリンクを除去する事ができます。

圧縮テーブルの特徴

圧縮ストレージフォーマットは `mysampack` ツールによって生成される読み取り専用のフォーマットです。圧縮テーブルは `myisamchk` を使って解凍する事ができます。

圧縮テーブルには次のような特徴があります。

- 圧縮テーブルはごくわずかなディスク容量しか必要としません。そのおかげでディスクの使用を最小化する事ができるので、低速ディスク (CD-ROMなど) を利用する時に便利です。
- それぞれの行が別々に圧縮されるので、アクセスオーバーヘッドがほとんどありません。行のヘッダーは、そのテーブルの中の一番大きな行に応じて、1から3バイトに固定されます。それぞれのカラムは違う方法で圧縮されます。それぞれのカラムは通常異なったハフマンツリーを持ちます。以下はいくつかの圧縮タイプの例です。
 - サフィックス空白の圧縮
 - プレフィックス空白の圧縮
 - 値0の数値は1ビットで格納されます。
 - 値の範囲が小さい整数カラムは、可能な限り小さな型を使って格納されます。例えば、`BIGINT` カラム (8バイト) の全ての値が `-128` から `127` の範囲内にある場合は、このカラムを `TINYINT` カラム (1バイト) として格納する事ができます。
 - カラムの可能値が少ない場合は、データの型を `ENUM` に変換します。
 - 上記の圧縮を組み合わせる事もできます。
- 固定長または可変長の行を処理する事ができます。

注: 圧縮テーブルは読み取り専用なので、テーブルを更新したり、行を追加したりはできません。DDL (データ定義言語) 操作は有効です。例えば、テーブルをドロップする為に `DROP` を利用したり、空にする為に `TRUNCATE` を利用する事もできます。

MyISAM テーブルの問題点

MySQL がデータの格納に使用するファイル形式は広範な検査を受けていますが、データベーステーブルの破損を招きかねない状況は常に存在します。次にどのようにしてそれらが起きるのか、そしてどのように対処すればよいのかを紹介합니다。

破損した MyISAM テーブル

MyISAM は信頼性の高いテーブル形式ですが、(テーブルに対するすべての変更は SQL ステートメントから制御が戻る前に書き込まれます)それでも以下の状況が発生した場合はテーブルが破損するおそれがあります。

- `mysqld` プロセスが書き込みの最中に強制終了された場合。
- コンピューターが予期せずシャットダウンされた場合。(例えばコンピューターの電源が切られた時。)
- ハードウェアエラー
- サーバー上でテーブルを修正中に外部プログラム (`myisamchk` など) を利用した時。
- MySQL または MyISAM コードのソフトウェアバグ。

テーブルが破損すると通常次のような現象が見られます。

- テーブルからデータを選択する時に次のようなエラーが表示されます。

```
Incorrect key file for table: '...'. Try to repair it
```

- クエリがテーブルでレコードを検出できない、または不完全なデータを返します。

MyISAM テーブルが破損していないかどうか `CHECK TABLE` ステートメントを利用して確認することができます。また、`REPAIR TABLE` を利用して破損した MyISAM テーブルを修復することができます。また、`mysqld` が稼働していない時は `myisamchk` コマンドを利用して確認や修復ができます。[CHECK TABLE 構文](#)、[REPAIR TABLE 構文](#)、[myisamchk — MyISAM テーブル メンテナンス ユーティリティ](#) を参照して下さい。

もしテーブルが頻繁に破損する場合は、その原因を突き止める必要があります。最も重要なのは、サーバーのクラッシュによってテーブルが破損されたのかどうかを確認する事です。最近のエラーログの中の `restarted mysqld` メッセージを探せば簡単に検証することができます。もしそのようなメッセージがあれば、破損の原因はサーバーの破損によるものである可能性が高いでしょう。そうでなければ、破損は通常作業の最中に起きたという事になるでしょう。その場合はバグです。ですので、その破損のテストケースを作成してみる必要があります。[What to Do If MySQL Keeps Crashing](#)、[Making a Test Case If You Experience Table Corruption](#) を参照して下さい。

MySQL Enterprise

```
#####MySQL#####  
##### http://www-jp.mysql.com/products/enterprise/advisors.html#####
```

適切に閉じられなかったテーブルによる問題

各 MyISAM インデックスファイル (`.MYI` ファイル)には テーブルが適切に閉じられているかをチェックする為のカウンタがあります。もし `CHECK TABLE` が `myisamchk` から次のような警告が表示されたら、このカウンタは同期されていない事を表します。

```
clients are using or haven't closed the table properly
```

この警告は、テーブルが破損されたという意味ではありませんが、少なくともテーブルを確認したほうがよいでしょう。

カウンターは次のように機能します。

- MySQL でテーブルが最初に更新される時に、インデックスファイルのヘッダ内にあるカウンタが増加します。
- その後の更新ではカウンタは変更されません。
- (`FLUSH TABLES` 操作が行われた、またはテーブルキャッシュの中に場所が無い為)テーブルの最後のインスタンスが閉じられると、それまでにテーブルが更新されていればカウンタが減少します。
- テーブルを修復するか、チェックして問題がなかった場合は、カウンタがゼロにリセットされます。
- テーブルを検査する他のプロセスとの相互作用に伴う問題を回避する為、カウンタがゼロである場合は、テーブルを閉じる際にカウンタは減少しません。

つまり、カウンタがずれる可能性があるのは次のような場合です。

- MyISAM テーブルが **LOCK TABLES** と **FLUSH TABLES** を発行せずにコピーされた。
- MySQL が更新されてから閉じられるまでの間にクラッシュした。(ただし、MySQL は各ステートメントで生じたすべての書き込みを次のステートメントまでに発行する為、テーブルが無事である可能性もあります)。
- `mysqld`によって使用されていたのと同時に`myisamchk --recover`が `myisamchk --update-state` によってテーブルが更新された。
- 別のサーバーによって使用されている最中に、複数の `mysqld` サーバーがテーブルを使用し、1つのサーバーが **REPAIR TABLE** か **CHECK TABLE** を実行した。この場合、他のサーバーから警告を受けるかもしれないが、**CHECK TABLE** を使用してもよい。しかし、サーバーがデータファイルを新しい物と交換するとそれが別のサーバーに通知されないの、**REPAIR TABLE** は避ける必要がある。

通常、複数サーバー間でデータディレクトリを共有するのはよくない事です。追加情報に関しては [同じマシン上で複数の MySQL サーバの実行](#) を参照してください。

InnoDB ストレージ エンジン

InnoDB 概要

InnoDB は MySQL に、コミット、ロールバック、クラッシュ復旧機能を持つトランザクション セーフ(ACID 適合) ストレージ エンジンを提供します。InnoDB は行レベルでのロックを行い、**SELECT** ステートメント内で Oracle スタイルの一貫した非ロック リードを提供します。これらの特徴により、複数ユーザによる並行処理とその性能が向上します。InnoDB 内では、行レベル ロックは領域をほとんど利用しないので、ロックを向上させる必要はありません。InnoDB は **FOREIGN KEY** 制約もまたサポートします。同じステートメント内で InnoDB テーブルと別の MySQL ストレージ エンジンからのテーブルを混合する事ができます。

InnoDB は大容量データ処理の最大性能の為に設計されました。その CPU 性能に匹敵するディスク ベースのリレーショナル データベース エンジンは他にはないでしょう。

InnoDB ストレージ エンジンは、MySQL サーバと完全に融和し、メイン メモリ内にデータとインデックスをキャッシュする為の、それ自身のバッファ プールを維持します。InnoDB は、いくつかのファイル(または未加工ディスク パーティション)で構成されるであろうテーブル領域内にそのテーブルとインデックスを格納します。これは例えば、各テーブルが別々のファイルを利用して格納される MyISAM テーブルとは異なります。InnoDB テーブルは、ファイル サイズが 2GB に制限されている OS 上で、どんなサイズにもなり得ます。

InnoDB はバイナリ ディストリビューションの中にデフォルトとして含まれています。Windows Essentials インストーラによって、InnoDB は Windows 上で MySQL のデフォルト ストレージ エンジンになります。

InnoDB は高性能が求められる数々の大型データベース サイトにて、製造に利用されます。有名なインターネット ニュース サイト Slashdot.org は InnoDB で起動しています。Mytrix, Inc. は InnoDB 内に 1TB 以上のデータを格納し、別のサイトは InnoDB 内で一秒に 800 の挿入/更新の平均負荷を扱っています。

InnoDB は、MySQL と同じ GNU GPL ライセンス バージョン 2(1991年6月)によって発行されています。MySQL のライセンスについての更なる情報に関しては、<http://www.mysql.com/company/legal/licensing/> を参照してください。

追加情報

- InnoDB ストレージ エンジンに専門に扱うフォーラムがあります。 <http://forums.mysql.com/list.php?22>

InnoDB 連絡先情報

Innobase Oy の連絡先情報、InnoDB エンジンの作成元:

Web site: <http://www.innodb.com/>
 Email: [<sales@innodb.com>](mailto:sales@innodb.com)
 Phone: +358-9-6969 3250 (office)
 +358-40-5617367 (mobile)

Innobase Oy Inc.
 World Trade Center Helsinki
 Aleksanterinkatu 17
 P.O.Box 800

00101 Helsinki
Finland

InnoDB 設定

InnoDB ストレージ エンジン はデフォルトにて有効になっています。もし InnoDB テーブルを利用したくなければ、お使いの MySQL オプション ファイルに `skip-innodb` オプションを追加する事ができます。

注意: InnoDB は MySQL に、コミット、ロールバック、クラッシュ復旧機能を持つトランザクション セーフ(ACID 適合)ストレージ エンジンを提供します。しかし、もし基礎となる OS やハードウェアが広告どおりに機能しなければ、それを行う事はできません。多くの OS やディスク サブ システムが、性能を向上させる為に書き込み操作を遅らせたり再オーダしたりするでしょう。いくつかの OS 上で、ファイルの全ての未書き込みデータがフラッシュされるまで待つ必要があるそのシステムコール、`fsync()` は、実際にデータが安定したストレージにフラッシュされる前に返されます。この為、OS のクラッシュや停電によって最近コミットされたデータが破損したり、さらに最悪の場合、書き込み操作が再オーダされた為にデータベースが破損する事もありますもしデータの整合性が重要であるなら、製造で何かを利用する前に「pull-the-plug」テストを行うべきです。Mac OS X 10.3 以降のバージョンでは、InnoDB は特別な `fcntl()` ファイル フラッシュ法を利用します。Linux 下では、ライト バック キャッシュを無効にする 事をお勧めします。

ATAPI ハードディスク上では、`hdparm -W0 /dev/hda` のようなコマンドがライト バック キャッシュを無効にする働きをします。いくつかのドライブやディスク コントローラでは、ライト バック キャッシュを無効にできない可能性があるので注意してください。

InnoDB ストレージ エンジンによって管理されている2つの重要なディスク ベース リソースは、そのテーブルスペース データ ファイルとログ ファイルです。

注意: もし InnoDB 設定オプションを全く指定しなければ、MySQL は MySQL データ ディレクトリ内に、`ibdata1` という名前の 10MB の自動延長データ ファイルと、`ib_logfile0` と `ib_logfile1` という名前の 5MB のログ ファイルを作成します。高性能を得るには、次の例にあるように InnoDB パラメータを明示的に提供する必要があります。当然ながら、お使いのハードウェアとその要求に合うように、設定を編集する必要があります。

MySQL Enterprise

#####MySQL ##### http://
www.jp.mysql.com/products/enterprise/advisors.html #####

ここに表されている例は代表的な物です。InnoDB に関連した設定パラメータに関する追加情報は、[InnoDB 起動オプションとシステム変数](#)を参照してください。

InnoDB テーブルスペース ファイルを設定する為には、`my.cnf` オプション ファイルの `[mysqld]` セクション内の `innodb_data_file_path` オプションを利用してください。Windows 上では、代わりに `my.ini` を利用する事ができます。`innodb_data_file_path` の値は、1つまたは複数のデータ ファイル仕様のリストでなければいけません。複数のデータ ファイルに名前を付けたら、セミコロン文字(';')でそれらを区切ってください:

```
innodb_data_file_path=datafile_spec1[:datafile_spec2]...
```

例えば、デフォルトと同じ特徴を持つテーブル スペースを明示的に作成する設定は、次のようになります:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

この設定は、`ibdata1` と名づけられた自動延長の単一10MB データ ファイルを構成します。そのファイルの場所は指定されない為、InnoDB がデフォルトでそれを MySQL データ ディレクトリ内に作成します。

MB か GB の単位を指定する為に、**M** か **G** のサフィックス文字を利用してサイズが指定されます。

データ ディレクトリ内で、`ibdata1` と名づけられた固定サイズ50MB のデータ ファイルと、`ibdata2` と名づけられた50MBの自動拡大ファイルを含むテーブル スペースは、次のように設定されます:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

データ ファイル仕様の完全構文は、ファイル名、そのサイズ、そしていくつかの任意の属性を含んでいます:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```


`autoextend` 属性やそれらに続く物は、`innodb_data_file_path` ライン内の最後のデータ ファイルにのみ利用できます。

最後のデータ ファイルに `autoextend` オプションを指定すると、InnoDB はもしテーブルスペースの中に空き領域がなければデータ ファイルを拡大します。デフォルトで、インクリメントは一回に付き8MB となっています。それは `innodb_autoextend_increment` システム変数を変更する事で修正できます。

もしディスクがいっぱいになると、別のディスク上に別のデータ ファイルを追加したくなるでしょう。InnoDB データとログ ファイルの追加と削除に既存のテーブル スペースの再設定に関する説明があります。

InnoDB にはファイル システムの最大サイズが分からないので、それが 2GB のような小さい値の場合は注意してください。自動拡大データ ファイルの最大サイズを指定するには、`max` 属性を利用してください。次の設定は、`ibdata1` が最大500MB まで大きくなる事を許容します:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB はデフォルトで MySQL データ ディレクトリ内にテーブルスペース ファイルを作成します。場所を明示的に指定するには、`innodb_data_home_dir` オプションを利用してください。例えば、`ibdata1` と `ibdata2` と名づけられた2つのファイルを、`/ibdata` ディレクトリ内で作成して利用するには、InnoDB をこのように設定してください:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

注意:InnoDB はディレクトリを作成しないので、サーバを起動する前に必ず `/ibdata` ディレクトリが存在する事を確認してください。これは、ご自分が設定する別のログ ファイルにも全て当てはまります。必要なディレクトリを作成する為に、Unix が DOS `mkdir` コマンドを利用してください。

InnoDB は `innodb_data_home_dir` の値を原文どおりにデータ ファイル名に連結させ、必要に応じてパス名セパレータ (スラッシュまたはバックスラッシュ) を値の間に追加して、各データ ファイルのディレクトリ パスを形作ります。もし `innodb_data_home_dir` オプションについて `my.cnf` 内で全く触れられなければ、MySQL データ ディレクトリを意味する「dot」 directory `.` がデフォルト値になります。(MySQL サーバは、実行を始める時にその時起動しているディレクトリを、それ自体のデータ ディレクトリに変更します。)

もし `innodb_data_home_dir` を空の文字列として指定すれば、`innodb_data_file_path` 値内にリストされたデータ ファイルに完全なパスを指定する事ができます。次の例は、前出の物と同等です:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

シンプルな`my.cnf` 例。128MB RAM とハードディスクが1つあるコンピュータを持っていると仮定してください。次の例は、`autoextend` 属性を含む、InnoDB の為の `my.cnf` か、`my.ini` 内で可能な設定パラメータを表しています。この例は、InnoDB データ ファイルとログ ファイルをいくつかのディスクに分散する事を希望しない、Unix と Windows 両方のほとんどのユーザに適しています。これは、MySQL データ ディレクトリ内に、自動拡大データ ファイル `ibdata1` と、2つの InnoDB ログ ファイル `ib_logfile0` と `ib_logfile1` を作成します。

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=70M
innodb_additional_mem_pool_size=10M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=20M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

MySQL サーバがデータ ディレクトリ内にファイルを作成する為の正当なアクセス権を持っている事を確認してください。さらに一般的には、サーバはデータ ファイルとログ ファイルを作成しなければいけないディレクトリ内にアクセス権を持っている必要があります。

データ ファイルはいくつかのファイル システム内で 2GB 以下でなければいけない事に注意してください。結合したログ ファイルのサイズは 4GB 以下でなければいけません。結合したデータ ファイルのサイズは最低 10MB でなければいけません。

InnoDB テーブルスペースを初めて作成する時は、MySQL サーバをコマンド プロンプトから起動するのが一番良い方法です。その時 InnoDB はデータベース作成に関する情報をスクリーンにプリントする事ができる為、何が起きているかを知る事ができるのです。例えば、もし Windows 上で `mysqld` が `C:\Program Files\MySQL\MySQL Server 5.1\bin` 内にあったら、次ように起動する事ができます:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld" --console
```

もしスクリーンにサーバ アウトプットを送らないなら、InnoDB がスタートアップ作業の最中に何をプリントするのを確認する為にサーバのエラー ログを確認してください。

InnoDB によって表示される情報の例については、[InnoDB テーブルスペースを作成する](#) を参照してください。

サーバが起動した時に読み込むオプション ファイルの `[mysqld]` グループ内にある InnoDB オプションをおく事ができます。オプション ファイルの場所に関しては、[オプションファイルの使用](#) で紹介されています。

もし MySQL をインストールと設定ウィザードを利用して Windows 上にインストールしたら、そのオプション ファイルはお使いの MySQL インストール ディレクトリ内の `my.ini` ファイルになります。詳しくは [my.ini ファイルのクエション](#) を参照してください。

もしお使いの PC が、`C:` ドライブがブート ドライブではないブート ローダを利用していたら、残されたオプションは Windows ディレクトリ(通常 `C:\WINDOWS` か `C:\WINNT`)内の `my.ini` ファイルを利用する事だけです。WINDIR の値をプリントするには、コンソール ウィンドウ内のコマンド プロンプトで `SET` コマンドを利用する事ができます:

```
C:\> SET WINDIR
windir=C:\WINDOWS
```

`mysqld` が特定のファイルからだけオプションを読み込むようにしたいのであれば、サーバを起動する時 `--defaults-file` オプションをコマンドラインの最初のオプションとして利用する事ができます:

```
mysqld --defaults-file=your_path_to_my.cnf
```

進歩した `my.cnf` 例。ディレクトリパス `/`、`/dr2` そして `/dr3` に 2GB RAM と3つの 60GB ハード ディスクを持つ Linux コンピュータを持っていると仮定してください。次の例は、InnoDB の `my.cnf` 内で可能な設定パラメータを表しています。

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /db/ibdata1:2000M;/dr2/db/ibdata2:2000M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
innodb_log_files_in_group = 2
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next lines if you want to use them
innodb_thread_concurrency=5
```

場合によっては、もし全てのデータが同じディスク上に置かれていなければ、データベース性能が向上します。ログ ファイルをデータとは別のディスク上に置く事で、性能が向上する事が多いです。どのようにするかを例で説明しています。これは、2つのデータ ファイルを別々のディスクに、そしてログ ファイルを3つ目のディスクにおきます。InnoDB は最初のデータ ファイルを先に利用してテーブル スペースを埋めていきます。InnoDB データ ファイル

として、未加工ディスクパーティション（未加工デバイス）を利用する事ができ、そのおかげで I/O のスピードが向上します。[共有テーブルスペースに未加工デバイスを利用する](#) を参照してください。

警告: 32-bit GNU/Linux x86 上では、メモリ使用を高く設定しすぎないように注意してください。[glibc](#) はプロセスヒープがスレッドスタックよりも大きくなる事を許容する可能性があり、その為サーバがクラッシュしてしまうかもしれません。もし次の式の値が2GBに近い、またはそれを上回っていたら危険です:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

各スレッドはスタック(通常2MBですが、MySQL AB バイナリ内ではたったの256KB)を利用し、そして最悪の場合、[sort_buffer_size](#) + [read_buffer_size](#) 追加メモリも利用します。

MySQL をご自分でコンパイルする事により、32ビット Windows 内で最高 64GB の物理メモリを利用する事ができます。[InnoDB 起動オプションとシステム変数](#) 内の [innodb_buffer_pool_awe_mem_mb](#) に関する説明を参照してください。

別の [mysqld](#) サーバ パラメータはどのように調整するのでしょうか？次の値は典型的な物であり、ほとんどのユーザに適応しています:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=value
```

Per-Table テーブルスペースを利用する

各 [InnoDB](#) テーブルとそのインデックスをそれ自体のファイル内に格納する事ができます。この特徴は、実際に各テーブルがそのテーブルスペースを持つ為「multiple tablespaces」と呼ばれています。

複数のテーブルスペースを利用する事は、特定のテーブルを別々の物理ディスクに移動したり、単一テーブルのバックアップを残りの [InnoDB](#) テーブルの利用を邪魔する事なく、素早く復元したいユーザにとって、有益な物です。

このラインを [my.cnf](#) の [\[mysqld\]](#) セクションに追加する事で、複数のテーブルスペースを有効にする事ができます:

```
[mysqld]
innodb_file_per_table
```

サーバを再起動した後、[InnoDB](#) はテーブルが属するデータベースディレクトリ内にある、それ自体のファイル [tbl_name.ibd](#) 内に、それぞれの新しく作成されたテーブルを格納します。これは [MyISAM](#) ストレージエンジンが行う事に似ていますが、[MyISAM](#) はテーブルをデータファイル [tbl_name.MYD](#) とインデックスファイル [tbl_name.MYI](#) に分割します。[InnoDB](#) には、データとインデックスは [.ibd](#) ファイル内で一緒に格納されます。[tbl_name.frm](#) ファイルはまだ通常通り作成されます。

もし [innodb_file_per_table](#) ラインを [my.cnf](#) から削除してサーバを再起動すると、[InnoDB](#) は共有テーブルスペースファイル内にテーブルを再度作成します。

[innodb_file_per_table](#) はテーブル作成だけに影響を与え、既存テーブルにアクセスはしません。もしこのオプションを利用してサーバを起動すると、新しいテーブルは [.ibd](#) ファイルを利用して作成されますが、共有テーブルスペース内に存在するテーブルにアクセスする事もまだ可能です。もしオプションを削除してサーバを再起動すると、新しいテーブルは共有テーブルスペース内に作成されますが、複数のテーブルスペースを利用して作成されたテーブルにもまだアクセスする事ができます。

注意:[InnoDB](#) は、共有テーブルスペースに内部データディレクトリと取り消しログを置くので、いつもそれを必要とします。[.ibd](#) ファイルは [InnoDB](#) の作動に充分ではありません。

注意:[MyISAM](#) テーブルファイルで行えるのと同じように、データベースディレクトリ間で [.ibd](#) ファイルを自由に移動させる事はできません。これは、[InnoDB](#) 共有テーブルスペース内に格納されているテーブル定義がデータベース名を含み、そして [InnoDB](#) がトランザクション ID とログシーケンス番号の一貫性を保持しなければいけない為です。

1つのデータベースから別のデータベースに `.ibd` ファイルとその関連テーブルを移動するには、`RENAME TABLE` ステートメントを利用してください:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

もし `.ibd` の「空の」バックアップを持っていれば、それを次のように、それが発生した場所から MySQL インストールに格納する事ができます:

1. この `ALTER TABLE` ステートメントを発行してください:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

注意:このステートメントは現在の `.ibd` ファイルを削除します。

2. バックアップ `.ibd` ファイルを正しいデータベース ディレクトリ内に戻してください。

3. この `ALTER TABLE` ステートメントを発行してください:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

このコンテキスト内で「空の」`.ibd` ファイル バックアップが意味する物は:

- `.ibd` ファイル内には、トランザクションによってコミットされていない変更はありません。
- `.ibd` ファイル内にマージされていない挿入バッファ エントリはありません。
- ページは `.ibd` ファイルから全ての削除マークされたインデックス レコードを削除しました。
- `mysqld` は、`.ibd` ファイルの全ての変更されたページをバッファ プールからファイルにフラッシュしました。

次の方法を利用して、空のバックアップ `.ibd` ファイルを作る事ができます:

1. `mysqld` サーバからの全てのアクティビティを停止して、全てのトランザクションをコミットしてください。
2. `SHOW ENGINE INNODB STATUS` がデータベース内にアクティブなトランザクションが無いと表示し、`InnoDB` のメイン スレッド ステータスが `Waiting for server activity` となるまで待つてください。すると、`.ibd` ファイルのコピーを作成する事ができます。

`.ibd` ファイルの空のコピーを作成する別の方法は、商業 `InnoDB Hot Backup` ツールを利用する事です:

1. `InnoDB` インストールをバック アップする為に `InnoDB Hot Backup` を利用してください。
2. 2番目の `mysqld` サーバをバック アップ上で起動し、その中で `.ibd` ファイルを掃除させてください。

共有テーブルスペースに未加工デバイスを利用する

共有テーブルスペース内で未加工ディスク パーティションをデータ ファイルとして利用できます。未加工ディスクを利用する事で、Windows といくつかの Unix システム上でファイル システムの負荷が無い非バッファ I/O を実行する事ができ、それで操作性能が向上します。

新しいデータ ファイルを作成する時、`innodb_data_file_path` 内のデータ ファイル サイズの直後にキーワード `newraw` を置く必要があります。パーティションは、少なくとも指定したサイズと同じである必要があります。ディスク仕様の 1MB は通常 1,000,000 バイトを意味するのに対して、`InnoDB` 内の 1MB は 1024 × 1024 バイトである事に注意してください。

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw:/dev/hdd2:2Gnewraw
```

次にサーバを起動する時、`InnoDB` は `newraw` キーワードに気付き、新しいパーティションを開始します。しかし、まだ `InnoDB` テーブルを作成したり変更したりしないでください。そうでなければ、サーバを次に再起動した時 `InnoDB` がパーティションを再開始し、変更が全て失われます。(安全策として、`InnoDB` は `newraw` を持つパーティションが指定された時ユーザがデータを更新する事を防ぎます。)

`InnoDB` が新しいパーティションを開始したら、サーバを停止し、データ ファイル仕様の中の `newraw` を `raw` に変更してください:

```
[mysql]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:5Graw:/dev/hdd2:2Graw
```

そしてサーバを再起動させると、InnoDB は変更を許可します。

Windows 上では、次のようにディスク パーティションを割り当てる事ができます:

```
[mysql]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

// は物理的ドライブにアクセスする為の Windows の構文 \\.\ に対応しています。

未加工ディスク パーティションを利用する時、MySQL サーバを起動するのに利用されるアカウントによって読み込み書き込みアクセスが許可されている事を確認してください。

InnoDB 起動オプションとシステム変数

この章では、InnoDB に関連するコマンド オプションとシステム変数を紹介します。虚実であるシステム変数は、それらを名づける事によってサーバ起動時に有効にされるか、または `skip-` プリフィックスを利用する事で無効にされます。例えば、InnoDB チェックサムを有効、または無効にするには、コマンドライン上で `--innodb_checksums` か `--skip-innodb_checksums` を、またはオプション ファイル内で `innodb_checksums` か `skip-innodb_checksums` を利用する事ができます。数字値を取るシステム変数は、コマンドライン上で `--var_name=value` として、またはオプション ファイル内で `var_name=value` として指定する事ができます。オプションとシステム変数を指定する事に関する更なる情報は、[プログラム・オプションの指定](#) を参照してください。多くのシステム変数は起動時に変更できます。(動的システム変数を参照してください。)

MySQL Enterprise

```
MySQL #####InnoDB #####
# http://www-jp.mysql.com/products/enterprise/advisors.html #####
```

InnoDB コマンド オプション:

- `--innodb`

もしサーバが InnoDB サポートでコンパイルされると InnoDB ストレージ エンジンが有効になります。InnoDB を無効にするには `--skip-innodb` を利用してください。

- `--innodb_status_file`

InnoDB が MySQL データ ディレクトリ内にファイル名 `<datadir>/innodb_status.<pid>` を作成するように働きます。InnoDB は定期的に `SHOW ENGINE INNODB STATUS` のアウトプットをこのファイルに書き込みます。

InnoDB システム変数:

- `innodb_additional_mem_pool_size`

InnoDB がデータ辞書情報と別の内部データ構造を格納する為に利用する、メモリ プールのバイトでのサイズです。より多くのテーブルをアプリケーション内に持っている、ここに割り当てる為により多くのメモリが必要になります。もし InnoDB がこのプール内のメモリを使い果たしてしまったら、これは OS からメモリを割り当て始め、MySQL エラー ログに警告メッセージを書きます。デフォルト値は1MB です。

- `innodb_autoextend_increment`

自動拡大テーブルスペースがいっぱいになった時にサイズを拡大する為のインクリメント サイズ(MB)。デフォルト値は8です。

- `innodb_buffer_pool_ave_mem_mb`

AWE メモリ内に置かれた時の、バッファ プールのサイズ(MB)。これは32ビット Windows 内でだけ関連性があります。もしお使いの32ビット Windows OS が4GB 以上のメモリをサポートするなら、いわゆる「Address Windowing Extensions,」を利用する事で、この変数を利用して InnoDB バッファプールを

AWE 物理的メモリに割り当てる事ができます。この変数の最大可能値は63000です。もしこれが0以上なら、`innodb_buffer_pool_size` は InnoDB がその AWE メモリを マップする `mysqld` の32ビット アドレス領域内のウィンドウです。`innodb_buffer_pool_size` の適正な値は 500MB です。

AWE メモリを活用するには、自分で MySQL をリコンパイルする必要があります。これを行うのに必要な現在のプロジェクト設定は、`storage/innobase/os/os0proj.c` ソース ファイル内で見つける事ができます。

- `innodb_buffer_pool_size`

InnoDB がそのテーブルのデータとインデックスをキャッシュする為に利用する、メモリバッファ のバイトでのサイズです。この値を大きく設定するほど、テーブル内のデータにアクセスするのに必要なディスク I/O は少なくなります。専用のデータベース サーバ上で、これをマシンの物理的メモリ サイズの最大80% に設定すると良いでしょう。しかし、物理的メモリの競合が OS 内でページングを引き起こす可能性があるため、あまり大きく設定しないでください。

- `innodb_checksums`

InnoDB は、壊れたハードウェアやデータ ファイルに対する追加フォールト トレランスを保証するディスクからの全てのページの読み込み上で、チェックサムの妥当性確認を利用する事ができます。この妥当性確認はデフォルトで有効化されています。しかし、まれに(ベンチマークが起動している時等)、この追加安全機能は必要なく、`--skip-innodb-checksums` を利用して無効にする事ができます。

- `innodb_commit_concurrency`

同時にコミットする事ができるスレッドの数。0の値は並行処理制御を無効にします。

- `innodb_concurrency_tickets`

InnoDB に同時に入る事ができるスレッドの数は、`innodb_thread_concurrency` 変数によって決められます。スレッドが InnoDB に入ろうとする時にもし並行処理の限度までスレッド数が達していたら、それらは列になります。スレッドが InnoDB に入るのを許可されると、`innodb_concurrency_tickets` の値と同等の「フリー チケット」をたくさん与えられ、スレッドはそのチケットを使ってしまいうまでは自由に InnoDB に入出入りできます。それ以降は、スレッドが次に InnoDB に入ろうとした時に、再度並行処理チェックの対象となります。(または列に並ぶ可能性もある)

- `innodb_data_file_path`

独立したデータ ファイルとそれらのサイズへのパス。各データ ファイルへの完全ディレクトリパスは、ここに指定された各パスへの `innodb_data_home_dir` を結合する事によって形作られます。ファイル サイズは、サイズ値に `M` か `G` を付加して、MB か GB (1024MB)で指定されます。ファイル サイズの合計は最低10MB 必要です。もし `innodb_data_file_path` を指定しなければ、デフォルト動作で `ibdata1` と名づけられた10MB の単一自動拡大データファイルが作成されます。各ファイルのサイズ制限は OS によって決定されます。大きいファイルをサポートする OS のサイズを4GB 以上に設定する事ができます。未加工ディスク パーティションをデータ ファイルとして利用する事もできます。詳しくは [共有テーブルスペースに未加工デバイスを利用する](#) を参照してください。

- `innodb_data_home_dir`

全ての InnoDB データ ファイルのディレクトリ パスの主な部分。もしこの値を設定しなければ、デフォルトは MySQL データ ディレクトリになります。値を空の文字列として指定する事もでき、その場合は `innodb_data_file_path` 内で完全なファイル パスを利用する事ができます。

- `innodb_doublewrite`

デフォルトで、InnoDB は全てのデータを2回格納します。一回目は二重書き込み バッファに、そして次に実際のデータ ファイルに格納します。この変数はデフォルトで有効化されています。それは、データの整合性や起こり得る失敗に対する心配よりも、ベンチマークや最高性能が要求される時に、`--skip-innodb_doublewrite` を利用して止める事ができます。

- `innodb_fast_shutdown`

もしこの変数を0に設定すると、InnoDB はシャットダウンの前に完全消去と挿入バッファ マージを行います。これらの操作には数分間、または極端な場合には数時間かかる事があります。もしこの変数を1に設定すると、InnoDB はこれらの操作をシャットダウンの時にスキップします。デフォルト値は1です。もしこれを2に設定

すると、InnoDB はそのログをフラッシュし、まるで MySQL がクラッシュしたかのように急にシャットダウンします。コミットされたトランザクションはなくなりませんが、次の起動の際にクラッシュ復旧が行われます。2の値は NetWare 上では利用できません。

- [innodb_file_io_threads](#)

InnoDB 内のファイル I/O スレッド数。通常、これはデフォルト値である4のままにしておくべきですが、Windows 上のディスク I/O にとってはそれよりも大きい値の方がよいかもしれません。Unix 上では、数値を増やしても効果はありません。InnoDB は必ずデフォルト値を利用します。

- [innodb_file_per_table](#)

この変数が有効になると、InnoDB はデータとインデックスを共有テーブルスペースに格納するのではなく、それ自体の .ibd ファイルを利用してそれぞれの新しいテーブルを作成し、そこに格納します。デフォルトでは、共有テーブルスペースにテーブルを作成するという事になっています。詳しくは [Per-Table テーブルスペースを利用する](#) を参照してください。

- [innodb_flush_log_at_trx_commit](#)

[innodb_flush_log_at_trx_commit](#) が0に設定された時は、ログ バッファは1秒に一回ログ ファイルに書き込まれ、ディスク操作へのフラッシュはログ ファイル上で行われますが、トランザクション コミットの際には何も行われません。この値が1(デフォルト)の時は、ログ ファイルは各トランザクション コミットの時にログ ファイルに書き込まれ、ディスク操作へのフラッシュはログ ファイル上で行われます。2に設定された時は、ログ バッファはコミット毎にファイルに書き込まれますが、ディスク操作へのフラッシュはそこでは行われません。しかし、値が2の時もログ ファイル上でのフラッシュは1秒に1回行われます。1秒に1回のフラッシュは、処理スケジュールの発行の為100% 保証された物ではないという事に注意してください。

この変数のデフォルト値は1です。これは ACID 整合性に要求されている値です。より良い性能の為に1以外の値を設定する事もできますが、その場合1つのクラッシュの中で最大1秒分のトランザクションを失う可能性があります。もし値を0に設定すると、全ての [mysqld](#) プロセス クラッシュは最後の秒のトランザクションを消す場合があります。もし値を2に設定すると、OS のクラッシュが停電によって、最後の秒のトランザクションが消されてしまいます。しかし、InnoDB のクラッシュ復旧は影響を受けないので、値に関係なくクラッシュ復旧は行われます。多くの OS といくつかのディスク ハードウェアはディスクへのフラッシュ操作を欺く事があると覚えておいてください。それらはフラッシュが行われていなくても、行われたと [mysqld](#) に伝える可能性があります。1の設定がしてあってもトランザクションの耐久力が保証されないという事になり、さらに悪い事に、停電によって InnoDB データベースが破損する可能性もあります。SCSI ディスク コントローラ内やディスク自体の中での、バッテリーに頼っているディスク キャッシュの利用はファイル フラッシュのスピードを上げ、操作を安全に行う事ができます。ハードウェア キャッシュ内でディスク書き込みのキャッシュを無効にする為に、Unix コマンド [hdparm](#) を利用してみたり、ハードウェア ベンダに対しての特定の別のコマンドを利用したりもできます。

注意:InnoDB とトランザクションを共に利用して複製設定内で最大の耐久力と一貫性を得る為に、お使いのマスタサーバ [my.cnf](#) 内で [innodb_flush_log_at_trx_commit=1](#) と [sync_binlog=1](#) を利用しなければいけません。

- [innodb_flush_method](#)

もし [fdatasync](#) (デフォルト)に設定すると、InnoDB はデータとログ ファイルの両方をフラッシュする為に [fsync\(\)](#) を利用します。もし [O_DSYNC](#) に設定すると、InnoDB はログ ファイルをオープン、フラッシュする為に [O_SYNC](#) を利用しますが、データ ファイルをフラッシュする為に [fsync\(\)](#) を利用します。もし [O_DIRECT](#) が指定されると(GNU/Linux バージョン上で有効)、InnoDB はデータ ファイルをオープンする為に [O_DIRECT](#) を利用し、データとログ ファイルの両方をフラッシュする為に [fsync\(\)](#) を利用します。InnoDB は [fdatasync\(\)](#) の代わりに [fsync\(\)](#) を利用する事、また様々な種類の Unix 上で問題があった為、デフォルトで [O_DSYNC](#) は利用しないという事に注意してください。この変数は Unix に対してだけ関連があります。Windows 上では、フラッシュの方法は毎回 [async_unbuffered](#) で、変更する事はできません。

この変数の異なる値は InnoDB performance 上で著しい影響を持ちます。例えば、InnoDB データとログ ファイルが SAN 上に位置するいくつかのシステム上では、[innodb_flush_method](#) を [O_DIRECT](#) に設定する事は、3つの要因によってシンプルな [SELECT](#) ステートメントの性能を劣らせる可能性があるという事が発見されました。

- [innodb_force_recovery](#)

クラッシュ復旧モード。警告:この変数は、破損したデータベースからテーブルを捨てたいという緊急の場合のみ、0以降の値に設定しなければいけません!可能な値は1から6です。これらの値の意味は、InnoDB 復旧の強制で説明されています。安全策として、InnoDB はこの変数が0以上の時はそのデータへの変更を阻止します。

- innodb_lock_wait_timeout

InnoDB トランザクションがロールバックされる前に、ロックを待つ秒数でのタイムアウト。InnoDB は自動的にそれ自体のロックテーブル内でトランザクションデッドロックを検出し、トランザクションをロールバックします。InnoDB は LOCK TABLES ステートメントを利用してロックセットを通知します。デフォルトは50秒です。

- innodb_locks_unsafe_for_binlog

この変数は InnoDB サーチとインデックススキャン内でネクストキーロックをコントロールします。デフォルトによってこの変数は0(無効)であり、それはネクストキーロックが有効であると意味します。

通常、InnoDB は next-key locking と呼ばれるアルゴリズムを利用します。InnoDB は、それがテーブルインデックスを検索やスキャンする時に、遭遇した全てのインデックスレコード上で共有または専用ロックを設定する、という方法で行レベルロックを実行します。従って、行レベルロックは実際はインデックスレコードロックであるという事になります。InnoDB がインデックスレコード上で設定するロックは、そのインデックスレコードに先行する「ギャップ」にも影響を与えます。もしユーザがインデックス内のレコード R 上に共有または専用ロックを持っていたら、別のユーザはインデックスの順番で R の直前に新しいインデックスレコードを挿入する事はできません。この変数を有効にすると、InnoDB が検索やインデックススキャン内でネクストキーロックを利用しないよう働きかけます。ネクストキーロックは外部キー制約と複製キーチェックを保証する為にはまだ利用されます。この変数を有効にすると、ファントムを引き起こす可能性がある事に注意してください。後で選択した行内のいくつかのカラムを更新するつもりで、100よりも大きい値の識別子を持つ child テーブルから全ての子供を読み、ロックしたいと仮定します:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

id カラム上にインデックスがあると仮定してください。id が100以上の最初のレコードから、そのインデックスをクエリがスキャンします。もしインデックスレコード上に設定されたロックがギャップに挿入された物をロックしなければ、別のクライアントがテーブルに新しい行を挿入する事ができます。もし同じトランザクション内で同じ SELECT を実行すると、クエリから返された結果セット内に新しい行を見つける事ができます。これは、もしデータベースに新しい項目が追加されると、InnoDB はシリアリザビリティを保証しないという事も意味します。従って、もしこの変数が有効になると、InnoDB は最高の分離レベル READ COMMITTED を保証します。(コンフリクトシリアリザビリティは保証されたままです。)

この変数を有効にすると、追加効果があります:UPDATE や DELETE 内の InnoDB は、更新や削除を行う行だけをロックします。このおかげでデッドロックの可能性が大幅に低くなりますが、それでもまだ起こります。この変数を有効にしても、UPDATE のような操作が別の似た操作(別の UPDATE のような)を追い越す事は、たとえそれが別の行に影響を与えるとしても許されていない事に注意してください。このテーブルから始まる、次の例を検討してみてください:

```
CREATE TABLE A(A INT NOT NULL, B INT) ENGINE = InnoDB;
INSERT INTO A VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

1つのクライアントがこれらのステートメントを実行すると仮定してください:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 5 WHERE B = 3;
```

そして、別のクライアントが、最初のクライアントの後にこれらのステートメントを実行すると仮定してください:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 4 WHERE B = 2;
```

この場合、2つ目の UPDATE は、最初の UPDATE のコミットがロールバックを待つ必要があります。最初の UPDATE は行(2, 3)上に専用ロックを持ち、2つ目の UPDATE も行をスキャンしている間に同じ行に専用ロックを得ようとしませんが、それはできません。これは、2つの UPDATE のうち最初の物が行に専用ロックを得て、その行が結果セットに属しているかどうかを決める為に起こります。もしそうでなければ、それは innodb_locks_unsafe_for_binlog 変数が有効になった時に、不必要なロックを解除します。

従って、InnoDB は次のように UPDATE 1を実行します:

```
x-lock(1,2)
unlock(1,2)
x-lock(2,3)
update(2,3) to (2,5)
```

```
x-lock(3,2)
unlock(3,2)
x-lock(4,3)
update(4,3) to (4,5)
x-lock(5,2)
unlock(5,2)
```

InnoDB は UPDATE 2を次のように実行します:

```
x-lock(1,2)
update(1,2) to (1,4)
x-lock(2,3) - wait for query one to commit or rollback
```

- `innodb_log_archive`

InnoDB アーカイブ ファイルをログするかどうか。この変数は歴史的理由により存在していますが、利用はされていません。バックアップからの復旧は MySQL がそれ自身のログ ファイルを利用して行っていますので、InnoDB ログ ファイルをアーカイブに保管する必要はありません。この変数のデフォルトは0です。

- `innodb_log_buffer_size`

InnoDB がディスク上のログ ファイルに書き込む為に使用するバッファのバイトでのサイズ。実用的な値の範囲は 1MB から 8MB です。デフォルトは 1MB です。大きいログ バッファは、トランザクション コミットの前にディスクにログを書き込む必要なく、大きいトランザクションが起動する事を許容します。従って、もし大きいトランザクションを持っていたら、ログ ファイルを大きくしておく事でディスク I/O を節約する事ができます。

- `innodb_log_file_size`

ログ グループ内のそれぞれの長いファイルのバイトでのサイズ。ログ ファイルの結合したサイズは 32ビット コンピュータ上で 4GB 以下でなければいけません。デフォルトは 5MB です。実用的な値は、N がグループ内のログ ファイル数だとして、バッファ プールのサイズの 1MB から 1/N-th です。値が大きいほど、ディスク I/O を節約し、バッファ プール内で必要とされるチェックポイント フラッシュ活動は少なくなります。しかし、ログ ファイルが大きいという事はクラッシュした時の復旧のスピードが遅いという事も意味します。

- `innodb_log_files_in_group`

ログ グループ内のログ ファイル数。InnoDB はファイルに輪状に書き込みをします。デフォルト(そして推奨)は2です。

- `innodb_log_group_home_dir`

InnoDB ログ ファイルへのディレクトリ パス。もし InnoDB ログ変数を何も指定しなければ、デフォルトで MySQL データ ディレクトリ内に `ib_logfile0` と `ib_logfile1` という名前の2つの 5MB ファイルを作成します。

- `innodb_max_dirty_pages_pct`

これは 0 から 100 の範囲の間の整数です。デフォルトは 90 です。InnoDB 内の主スレッドは、ダーティ (まだ書き込まれていない) ページの割合がこの値を超えないようにバッファ プールからページを書くように試みます。

- `innodb_max_purge_lag`

この変数は、消去操作が遅れている時に(マルチバージョンの実装 参照) INSERT、UPDATE そして DELETE 操作をどのように遅らせるかをコントロールします。この変数のデフォルト値は 0 で、これは遅れは無いという事を意味します。

InnoDB トランザクション システムは UPDATE か DELETE 操作によって削除マークが付けられたインデックス レコードを持つトランザクションのリストを保持します。このリストの長さを `purge_lag` にして下さい。purge_lag が `innodb_max_purge_lag` を上回る時、各 INSERT、UPDATE そして DELETE 操作は $((\text{purge\_lag}/\text{innodb\_max\_purge\_lag}) \times 10) - 5$ ミリ秒遅れます。遅れは消去バッチの最初に、10秒ごとに計算されます。もし消去される行を知る事ができる、古い一貫した読み取りビューの為に消去が起動しなかったら、その操作は遅れません。

トランザクション サイズがたったの 100 バイトと小さく、テーブル内に消去されていない行を 100MB 許容できると仮定した時、問題を引き起こす可能性のある作業負荷の典型的な設定は 100 万でしょう。

- `innodb_mirrored_log_groups`

データベースの為に残すログ グループの同一コピー数。現在は、この値は1に設定しなければいけません。

- [innodb_open_files](#)

この変数は InnoDB 内で複数のテーブルスペースを利用する場合のみ関連があります。それは InnoDB が同時にオープンしておける .ibd ファイルの最大数を指定します。最小値は10です。デフォルトは300です。

.ibd ファイルに利用されるファイル記述子は、InnoDB に対しての物のみです。それらは、`--open-files-limit` サーバオプションによって指定された物からは独立していて、テーブル キャッシュの操作に影響を与えません。

- [innodb_rollback_on_timeout](#)

MySQL 5.1 内で、InnoDB はトランザクション タイムアウト上で最後のステートメントだけをロールバックします。このオプションが与えられると、トランザクション タイムアウトは InnoDB がトランザクション全体を異常終了し、ロールバックするよう働きかけます。(MySQL 4.1と同じ動作です。)この変数は、MySQL 5.1.15で追加されました。

- [innodb_support_xa](#)

ON か1(デフォルト)に設定されると、この変数は InnoDB が XA トランザクション内の二相コミット サポートを有効にします。[innodb_support_xa](#) を有効にすると、トランザクションの準備でディスク フラッシュが余計に起こります。XA を利用する事を気にしないのであれば、この変数を OFF か0に設定してこれを無効にする事ができ、ディスク フラッシュの数を減らし、InnoDB 操作性能を向上させる事ができます。

- [innodb_sync_spin_loops](#)

スレッドが、サスペンドされる前に InnoDB ミューテックスが開放されるのを待つ回数。

- [innodb_table_locks](#)

もし `AUTOCOMMIT=0`、InnoDB が `LOCK TABLES` を支持すると、MySQL は全てのスレッドがそれら全てのロックをテーブルにリリースするまで `LOCK TABLE .. WRITE` から戻りません。[innodb_table_locks](#) のデフォルト値は1です。それはもし `AUTOCOMMIT=0` なら `LOCK TABLES` は InnoDB がテーブルを内部的にロックするよう働きかける事を意味します。

- [innodb_thread_concurrency](#)

InnoDB は、この変数から与えられた制限よりも少ない、またはそれと同等の制限の InnoDB 内部に多くの OS スレッドを一斉に保存しようと試みます。性能に関する問題を持ち、多くのスレッドがセマフォを待っているという事が `SHOW ENGINE INNODB STATUS` によって明らかにされたのなら、スレッド「thrashing」を持ち、この変数を低くまたは高く設定するよう試みる必要があります。もしたくさんのプロセッサとディスクがあるコンピュータをお持ちであれば、それを有効に活用する為に値を高く設定する事もできます。推奨値はお使いのシステムのプロセッサとディスク数の合計値です。

この変数の範囲は0から1000です。20以上の値は無限並行処理として読み取られます。無限というのは、並行チェックが無効になり、ミューテックスを獲得、リリースする事で発生するであろう、多量の負荷を防ぐという意味です。

MySQL 5.1.11以前はデフォルト値は20で、5.1.11以降は8となっています。

- [innodb_thread_sleep_delay](#)

InnoDB スレッドは InnoDB の列に加わるまでに、マイクロ秒で何秒間スリープ状態にあるか。デフォルト値は10,000です。0の値ではスリープ状態にはなりません。

- [sync_binlog](#)

もし変数値が正数であれば、MySQL サーバはバイナリ ログへの毎 [sync_binlog](#) 書き込みごとに、ディスク (`fdatasync()`)にそのバイナリ ログを同期化します。オート コミット モードでは、各ステートメントにつきバイナリ ログへの書き込みが1つあり、そうでなければ各トランザクションにつき1つの書き込みがあると覚えて置いてください。デフォルトは、ディスクへの同期化を行わない0です。クラッシュしてしまった場合には、バイナリ ログから最大1つのステートメントかトランザクションが失われてしまう為、1の値が一番安全な値です。しかしこれは同

時に、一番スピードが遅い物になります。(ディスクが、同期化の作業を大変速くする事ができる、バッテリーで起動するキャッシュを搭載していない限り)

InnoDB テーブルスペースを作成する

必要な InnoDB 設定パラメータを含む事ができるように、MySQLをインストールし、オプション ファイルを編集したと仮定してください。MySQL を起動する前に、InnoDB データ ファイルとログ ファイルの為に指定したディレクトリが存在する事、そしてMySQL サーバがそれらのディレクトリにアクセスする権利がある事を確認しなければいけません。InnoDB はファイルだけを作成し、ディレクトリは作成しません。データとログ ファイルの領域が充分である事も確認してください。

InnoDB が有効な状態でサーバを初めて起動する時には、MySQL サーバ `mysqld` は `mysqld_safe` ラッパからや、Windows サービスとしてではなく、コマンド プロンプトから起動させるのが一番良いです。コマンド プロンプトから起動する時、`mysqld` が何をプリントするか、また何が起きているかが分かります。Unix 上では、ただ `mysqld` を呼び出して下さい。Windows 上では、`--console` オプションを利用してください。

オプション ファイル内で初めて InnoDB を設定した後 MySQL サーバを起動する時、InnoDB はデータ ファイルとログ ファイルを作成し、次のような物をプリントします:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

この時点で InnoDB はテーブルスペースとログ ファイルを初期化しました。`mysql` のように、通常の MySQL クライアント プログラムを利用して MySQL サーバに接続する事ができます。MySQL サーバを `mysqladmin shutdown` を利用して終了する時、アウトプットは次のようになります:

```
010321 18:33:34 mysqld: Normal shutdown
010321 18:33:34 mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

データ ファイルとログ ディレクトリを見ると、そこに作成されたファイルを確認する事ができます。MySQL が再起動する時、データ ファイルとログ ファイルは既に作成されているので、アウトプットはさらにブリーフな物になっています:

```
InnoDB: Started
mysqld: ready for connections
```

もし `innodb_file_per_table` オプションを `my.cnf` に追加すると、InnoDB は、`.frm` ファイルが作成されたのと同じ MySQL データベース ディレクトリ内の `.ibd` ファイル内に各テーブルを格納します。詳しくは [Per-Table テーブルスペースを利用する](#) を参照してください。

InnoDB 初期化問題の扱い

もしファイル操作の最中に InnoDB が OS エラーをプリントすると、通常はその問題は次のうちどれかになります:

- InnoDB データファイル ディレクトリか InnoDB ログ ディレクトリを作成しなかった。
- `mysqld` がそれらのディレクトリ内にファイルを作成するアクセス権を持っていない。
- `mysqld` は正しい `my.cnf` か `my.ini` オプション ファイルを読み込む事ができず、その結果指定したオプションを見る事ができません。
- ディスクが一杯か、ディスク割当量を超えました。
- 指定したデータ ファイルと同じ名前のサブディレクトリを作成したので、その名前をファイル名として利用する事はできません。
- `innodb_data_home_dir` か `innodb_data_file_path` 値内に構文エラーがあります。

InnoDB がそのテーブルスペースかログ ファイルを初期化しようとした時に何かが失敗すると、InnoDB によって作成されたファイル全てを削除しなければいけません。これは、全ての `ibdata` ファイルと全ての `ib_logfile` ファイルの事です。既にいくつかの InnoDB テーブルを作成していた場合、MySQL データベース ディレクトリからも、それらのテーブルの対応する `.frm` ファイル (もし複数のテーブルスペースを利用していたら全ての `.ibd` ファイルも) を削除してください。そして、InnoDB データベースの作成にもう一度挑戦できます。何が起きているのか確認できるように、MySQL サーバをコマンド プロンプトから起動するのが一番良いです。

InnoDB テーブルの作成と利用

InnoDB テーブルを作成する為には、`CREATE TABLE` ステートメント内で `ENGINE = InnoDB` オプションを指定してください:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
```

このステートメントは、`my.cnf` 内で指定したデータ ファイルで構成されている InnoDB テーブルスペース内のカラム `a` 上でテーブルとインデックスを作成します。さらに、MySQL は MySQL データベース ディレクトリ下の `test` ディレクトリ内でファイル `customers.frm` を作成します。内部的に、InnoDB はそれ自体のデータ ディレクトリのテーブルにエントリを追加します。そのエントリはデータベース名を含んでいます。例えば、もし `test` が `customers` テーブルが作成されたデータベースであれば、エントリは `'test/customers'` の為の物になります。これは、他のいくつかのデータベース内で、同名 `customers` のテーブルを作成する事ができ、そしてそのテーブル名は InnoDB 内で衝突しないという事を意味します。

InnoDB テーブルに `SHOW TABLE STATUS` ステートメントを発行する事で、InnoDB テーブルスペース内の空き領域の量をクエリする事ができます。`SHOW TABLE STATUS` のアウトプット内の `Comment` セクション内に現れるテーブルスペース内の空き領域の量。例:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

`SHOW` が InnoDB テーブルの為に表示する統計は単なる概算であるという事に注意してください。それらは SQL 最適化の中で利用されます。しかしテーブルやインデックスの、準備されていたバイトでのサイズは正確です。

異なる API と共に InnoDB 内でトランザクションをどのように利用するか

デフォルトによって、MySQL サーバに接続する各クライアントが、実行すると全ての SQL ステートメントを自動的にコミットするオート コミットが有効な状態で開始します。複数ステートメント トランザクションを利用する為に、SQL ステートメント `SET AUTOCOMMIT = 0` を利用してオートコミットをオフにし、トランザクションをコミットまたはロールバックする為に `COMMIT` と `ROLLBACK` を利用する事ができます。オートコミットをオンの状態のままにしておきたければ、トランザクションを `START TRANSACTION` と、`COMMIT` か `ROLLBACK` のどちらかで囲む事ができます。次の例は2つのトランザクションを表しています。最初の物はコミットされ、2つ目の物はロールバックされています。

```
shell> mysql test

mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
-> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
```



```
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+-----+-----+
| A   | B   |
+-----+-----+
| 10 | Heikki |
+-----+-----+
1 row in set (0.00 sec)
mysql>
```

PHP、Perl DBI、JDBC、ODBC、または MySQL のスタンダード C 呼び出しインターフェースのような API 内では、**COMMIT** のようなトランザクション コントロール ステートメントを **SELECT** や **INSERT** のような別の SQL ステートメントのような文字列として、MySQL サーバに送る事ができます。いくつかの API は、別々の特別トランザクション コミットやロールバック機能または方法も提供します。

MyISAM テーブルを InnoDB に変換する

重要:mysql データベース(**user** や **host** のような)内の MySQL システム テーブルを **InnoDB** タイプに変換しないでください。これはサポートされていない操作です。システム テーブルは必ず **MyISAM** タイプの物でなければいけません。

もし全ての(非システムの)テーブルを **InnoDB** テーブルとして作成したければ、サーバ オプション ファイルの **[mysqld]** セクションにライン **default-storage-engine=innodb** を追加するだけでよいです。

InnoDB は、**MyISAM** ストレージ エンジンがするのと同じように、インデックスを別々に作成する為の特別な最適化を行いません。従って、テーブルをエクスポート、インポートしたり、後でインデックスを作成したりはしません。テーブルを **InnoDB** に変換する一番早い方法は、**InnoDB** テーブルに直接挿入する事です。それは、**ALTER TABLE ... ENGINE=INNODB** を利用する、または同一定義を利用して空の **InnoDB** テーブルを作成し、**INSERT INTO ... SELECT * FROM ...** を利用して行を挿入するという事です。

もし2番目のキー上に **UNIQUE** 制限があったら、インポート操作の最中に一時的に一意性チェックを切り、テーブルインポートのスピードを上げる事ができます:

```
SET UNIQUE_CHECKS=0;... import operation ...
SET UNIQUE_CHECKS=1;
```

大きいテーブルに対しては、**InnoDB** が2番目のインデックス レコードをバッチとして書く為にそれ自身の挿入バッファを利用する事ができるので、この作業をするとディスク I/O を大幅に節約する事ができます。データが複製キーを含んでいない事を必ず確認してください。**UNIQUE_CHECKS** はストレージエンジンが複製キーを無視する事を許可しますが、それを要求はしません。

挿入の操作性をあげる為には、大きいテーブルを細かく分けて挿入するのが良いでしょう:

```
INSERT INTO newtable SELECT * FROM oldtable
WHERE yourkey > something AND yourkey <= somethingelse;
```

全てのレコードが挿入された後で、テーブルをリネームする事ができます。

大きいテーブルの変換の最中に、ディスク I/O を減らす為に **InnoDB** バッファ プールのサイズを増やす必要があります。しかし、物質的メモリの80% 以上は利用しないでください。**InnoDB** ログ ファイルのサイズを増やす事もできます。

テーブルスペースを一杯にしないように注意してください:**InnoDB** テーブルは **MyISAM** テーブルよりも多くのディスク領域を必要とします。もし **ALTER TABLE** 操作で領域を使い切ってしまうと、それはロールバックを始め、それがディスクに頼っている場合何時間も時間がかかります。挿入には、**InnoDB** はバッチ内のインデックスに2つ目のインデックス レコードをマージする為に挿入バッファを利用します。それでディスク I/O を大幅に節約する事ができます。ロールバックにはそのような構造は利用されず、挿入の30倍の時間がかかります。

ロールバックが暴走した場合は、データベースに貴重なデータがなければ、膨大なディスク I/O の完了を待つよりも、データベース プロセスを強制終了したほうが良いでしょう完全な手順に関しては、**InnoDB 復旧の強制** を参照してください。

AUTO_INCREMENT カラムが InnoDB 内でどのように機能するか

もし InnoDB テーブルに **AUTO_INCREMENT** カラムを指定すると、InnoDB データ ディレクトリ内のテーブル ハンドルは、カラムに新しい値を割り当てるのに利用される自動インクリメント カウンタと呼ばれる特別なカウンタを含みます。このカウンタは、ディスク上には格納されず、主メモリ内だけに格納されます。

InnoDB は、**ai_col** を名づけた **AUTO_INCREMENT** カラムを含むテーブル **T** に自動インクリメント カウンタを初期化する為に、次のアルゴリズムを利用します:サーバの起動の後で、テーブル **T** への最初の挿入をする為に、InnoDB はこのステートメントと同等な物を実行します:

```
SELECT MAX(ai_col) FROM T FOR UPDATE;
```

InnoDB はステートメントによって値が取り出された物によってインクリメントし、それをカラムとテーブルの自動インクリメント カウンタに割り当てます。もしテーブルが空だったら、InnoDB は値 **1** を利用します。もしユーザがテーブル **T** の為のアウトプットを表示する **SHOW TABLE STATUS** ステートメントを呼び出し、自動インクリメントカウンタがまだ初期化されていなかったら、InnoDB は値を初期化するがインクリメントはせず、そしてそれを後で挿入に利用する為に格納します。この初期化はテーブル上で通常の専用ロック読み込みを利用し、そのロックはトランザクションの最後まで続くという事に注意してください。

InnoDB は、作成されたばかりのテーブルの為に自動インクリメント カウンタを初期化するのと同じ手順に従います。

自動インクリメント カウンタが初期化された後、もしユーザが **AUTO_INCREMENT** カラムの値を明示的に指定しなければ、InnoDB はカウンタを1でインクリメントしカラムに新しい値を割り当てます。もしユーザがカラム値を明示的に指定する行を挿入し、それが現在のカウンタ値よりも大きければ、カウンタは指定されたカラムに設定されます。

もしカウンタを利用して生成された数値を持つトランザクションをロールバックすると、**AUTO_INCREMENT** に割り当てられた値のシーケンス内のギャップに気がつくでしょう。

もしユーザが **NULL** か **0** を **INSERT** 内の **AUTO_INCREMENT** カラムに指定すると、InnoDB は、値が指定されず、新しい値も生成されていないかのように行を扱います。

自動インクリメント構造の性能は、もしユーザがカラムにマイナスの値を割り当てたり、もし値が指定した整数タイプ内に格納する事ができる最大値を上回っていたりすると、定義できません。

自動インクリメント カウンタにアクセスする時、InnoDB は、トランザクションの最後までではなく、現在の SQL ステートメントの最後まで続く、特別なテーブルレベル **AUTO-INC** ロックを利用します。**AUTO_INCREMENT** カラムを含んでいるテーブルへの挿入の並行処理を向上させる為に、特別ロック リリース戦略が紹介されました。それにもかかわらず、2つのトランザクションは **AUTO-INC** ロックが長時間保持されればパフォーマンス インパクトを与える事ができる **AUTO-INC** ロックを同じテーブル上で同時に持つ事ができません。これは、1つのテーブルから全ての行を別のテーブルに挿入する **INSERT INTO t1 ... SELECT ... FROM t2** のようなステートメントのような場合の事です。

InnoDB はサーバが起動している限り、メモリ内の自動インクリメント カウンタを利用します。サーバが停止し再起動した時、先ほど説明があったように、InnoDB は、テーブルへの最初の **INSERT** に対する各テーブルのカウンタを再初期化します。

InnoDB は、初期カウンタ値を設定したり、現在のカウンタ値を変更する為に、**CREATE TABLE** と **ALTER TABLE** ステートメント内の **AUTO_INCREMENT = N** テーブル オプションをサポートします。このセクションの最初の方で説明があったとおり、このオプションの効果はサーバの再起動によって無くなってしまいます。

FOREIGN KEY 制約

InnoDB は 外部キー制約もまたサポートします。InnoDB 内の外部キー制約定義の構文は次のようになります:

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

外部キー定義には次のような条件があります:

- 両方のテーブルは InnoDB テーブルである必要があり、それらは **TEMPORARY** テーブルではいけません。
- 参照表の中では、外部キーカラムが同じ順番で first カラムとしてリストされているインデックスが存在する必要があります。もしそのようなインデックスが無ければ、自動的に参照表上に作成されます。
- 参照表の中では、参照カラムが同じ順番で first カラムとしてリストされているインデックスが存在する必要があります。
- 外部キー カラム上のインデックス プリフィックスはサポートされていません。この1つの結論は、それらのカラム上のインデックスは常にプリフィックス長を含む必要がある為、**BLOB** と **TEXT** カラムを外部キー内に含む事ができないという事です。
- もし **CONSTRAINT symbol** 条項が与えられると、**symbol** 値はデータベース上で固有である必要があります。もし条項が与えられなければ、InnoDB は名前を自動的に作成します。

もし親テーブル内に適合する候補キー値が無ければ、InnoDB は子テーブル内に外部キー値を作成しようとする **INSERT** か **UPDATE** 操作を拒絶します。子テーブル内にいくつかの適合する行を持つ親テーブル内で、候補キー値を更新または削除しようとする **UPDATE** や **DELETE** 操作に対して InnoDB が取るアクションは、**FOREIGN KEY** 条項の **ON UPDATE** と **ON DELETE** サブ条項を利用して指定された referential action 上で依存しています。ユーザが親テーブルから行を削除または更新しようとして、子テーブル内に1つ以上の適合する行がある時、InnoDB は取るべきアクションを考慮して5つのオプションをサポートします:

- **CASCADE**: 親テーブルから行を削除または更新し、子テーブル内で自動的に適合行を削除または更新します。**ON DELETE CASCADE** と **ON UPDATE CASCADE** の両方がサポートされています。2つのテーブルの間で、親テーブル内、または子テーブル内で同じカラム上に機能するいくつかの **ON UPDATE CASCADE** 条項を定義するべきではありません。
- **SET NULL**: 親テーブルから行を削除または更新し、子テーブル内で外部キー カラムを **NULL** に設定します。これは外部キー カラムが指定された **NOT NULL** 修飾子を持たない時だけ有効です。 **ON DELETE SET NULL** と **ON UPDATE SET NULL** 条項の両方がサポートされています。
- **NO ACTION**: スタンダード SQL 内で、**NO ACTION** は、もし参照表内に関連する外部キーがあれば主キー値を削除または更新しようとする事は許容されていないという意味で、no action を意味します。InnoDB は親テーブルの削除または更新操作を拒否します。
- **RESTRICT**: 親テーブルの削除または更新操作を拒否します。**NO ACTION** と **RESTRICT** は **ON DELETE** か **ON UPDATE** 条項を省略する事と同じです。(いくつかのデータベース システムが据え置きチェックを持ち、**NO ACTION** が据え置きチェックです。MySQL 内では、外部キー制約は即座に確認されるので、**NO ACTION** と **RESTRICT** は同じです。)
- **SET DEFAULT**: このアクションはパーサによって認識されますが、InnoDB は **ON DELETE SET DEFAULT** か **ON UPDATE SET DEFAULT** 条項を含むテーブル定義を拒否します。

InnoDB がテーブル内で外部キー制約をサポートする事に注意してください。これらのような場合、「子テーブル レコード」は本当に同じテーブル内で依存レコードを参照します。

InnoDB は、外部キー チェックが速くなり、テーブル スキャンを必要としないよう、外部キーと参照キー上にインデックスを要求します。外部キー上のインデックスは自動的に作成されます。これは、いくつかの古いバージョン内での、インデックスが明示的に作成される必要があり、そうでなければ外部キー制約の作成が失敗する、という物とは対照的です。

タイプ変換をせずに比較できるよう、外部キーと参照キー内の対応するカラムは InnoDB 内に類似内部データ タイプを持つ必要があります。整数タイプのサイズとサインは同じである必要があります。文字列タイプの長さは同じである必要はありません。もし **SET NULL** アクションを指定したら、子テーブル内のカラムを **NOT NULL** として宣言していない事を確認してください。

もし MySQL が **CREATE TABLE** ステートメントからエラー番号1005を報告し、そのエラーメッセージがエラー150を参照していたら、外部キー制約が正しく形作られていない為にテーブル作成は失敗します。同じように、もし **ALTER TABLE** が失敗し、それがエラー150を参照していたら、それは変更したテーブルに対して外部キー制約が間違っている形作られるという意味になります。サーバ内に一番新しい InnoDB 外部キー エラーの詳細説明を表示する為に **SHOW ENGINE INNODB STATUS** を利用する事ができます。

注意: InnoDB は **NULL** カラムを含む外部キーや参照キー上で外部キー制約を確認しません。

注意:トリガは現在、転送された外部キー アクションによって有効化されません。

内部 InnoDB カラムの名前と一致するカラム名を持つテーブルを作成する事はできません。(DB_ROW_ID、DB_TRX_ID、DB_ROLL_PTR そして DB_MIX_ID を含む)MySQL 5.1.10以前のバージョン内ではこれはクラッシュの原因となり、5.1.10からはサーバがエラー1005を報告し、エラーメッセージ内で `errno -1` を参照します。

SQL スタンダードからの逸脱:InnoDB は同じ参照キー値を持つ親テーブル内にいくつかの行があると、外部キーチェック内で同じキー値を持つ別の親行がまるで存在しないかのように機能します。例えば、もし `RESTRICT` タイプ制約を定義し、いくつかの親行を持つ子行があれば、InnoDB はそれらの親行の削除を許可しません。

InnoDB は、外部キー制約に対応するインデックス内のレコードに基づいた、縦型アルゴリズムを通して転送操作を行います。

SQL スタンダードからの逸脱:非 `UNIQUE` キーを参照する `FOREIGN KEY` 制約はスタンダード SQL ではありません。それはスタンダード SQL への InnoDB 拡張子です。

SQL スタンダードからの逸脱:もし `ON UPDATE CASCADE` か `ON UPDATE SET NULL` が転送の最中に既に更新された 同じテーブルの更新を反復すると、それは `RESTRICT` のように機能します。これは、自己参照型 `ON UPDATE CASCADE` か `ON UPDATE SET NULL` 操作を利用する事ができないという意味です。これは転送更新の結果に起きる無限ループを防ぐ為の物です。反対に、自己参照型 `ON DELETE SET NULL` は、自己参照型 `ON DELETE CASCADE` と同様可能です。転送操作は15レベルより深くネスト化される事はないでしょう。

SQL スタンダードからの逸脱:通常の MySQL のように、挿入、削除、または多くの行の更新を行う SQL ステートメント内では、InnoDB は `UNIQUE` と `FOREIGN KEY` 制約を行ごとに行います。SQL スタンダードによると、デフォルト動作は据え置きチェックでなければいけません。それは、SQL ステートメント全体 が処理された後に制約の確認だけが行われるという事です。InnoDB が据え置き制約チェックを実装するまでは、外部キーを通してそれ自身を参照するレコードを削除するというような、いくつかの操作を行う事が不可能になります。

ここに、単一カラム外部キーを通して `parent` と `child` テーブルを関連させるシンプルな例があります:

```
CREATE TABLE parent (id INT NOT NULL,
                     PRIMARY KEY (id))
ENGINE=INNODB;
CREATE TABLE child (id INT, parent_id INT,
                    INDEX par_ind (parent_id),
                    FOREIGN KEY (parent_id) REFERENCES parent(id)
                    ON DELETE CASCADE)
ENGINE=INNODB;
```

`product_order` テーブルが別の2つのテーブルに外部キーを持つ、さらに複雑な例。1つの外部キーが `product` テーブル内の2段インデックスに参照をつけます。その他の物は `customer` テーブル内で単一カラム インデックスに参照をつけます:

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) ENGINE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                       PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                             product_category INT NOT NULL,
                             product_id INT NOT NULL,
                             customer_id INT NOT NULL,
                             PRIMARY KEY(no),
                             INDEX (product_category, product_id),
                             FOREIGN KEY (product_category, product_id)
                             REFERENCES product(category, id)
                             ON UPDATE CASCADE ON DELETE RESTRICT,
                             INDEX (customer_id),
                             FOREIGN KEY (customer_id)
                             REFERENCES customer(id)) ENGINE=INNODB;
```

InnoDB は `ALTER TABLE` を利用してテーブルに新しい外部キー制約を追加する事を許容します:

```
ALTER TABLE tbl_name
ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
```

```
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

要求されたインデックスを最初に作成する事を忘れないでください。ALTER TABLE を利用して、自己参照型外部キー制約をテーブルに追加する事もできます。

InnoDB は外部キーをドロップする為の ALTER TABLE の利用もサポートします。

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

もし外部キーを作成した時に FOREIGN KEY 条項が CONSTRAINT 名を含んでいたら、外部キーをドロップする為にその名前を参照する事ができます。そうでなければ、fk_symbol 値は外部キーが作成された時に InnoDB によって内部的に生成されます。外部キーをドロップしたい時にシンボル値を見つけるには、SHOW CREATE TABLE ステートメントを利用してください。例:

```
mysql> SHOW CREATE TABLE ibtest11c\G
***** 1. row *****
Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default "",
  `C` varchar(175) default NULL,
  PRIMARY KEY (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`,`D`)
REFERENCES `ibtest11a` (`A`,`D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`,`C`)
REFERENCES `ibtest11a` (`B`,`C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

単一 ALTER TABLE ステートメントの別々の条項の中に外部キーを追加したりドロップしたりはできません。別々のステートメントが要求されます。

InnoDB パーサは、FOREIGN KEY ... REFERENCES ... 条項内のテーブルとカラム識別子がバックフォート内で参照される事を許容します。(あるいは、もし ANSI_QUOTES SQL モードが有効であれば二重引用符を利用する事もできます。)InnoDB パーサは、lower_case_table_names システム変数の設定も考慮します。

InnoDB はテーブルの外部キー定義を SHOW CREATE TABLE ステートメントのアウトプットの一部として返します:

```
SHOW CREATE TABLE tbl_name;
```

mysqldump はダンプ ファイルのテーブルの正しい定義も作成し、外部キーの事も忘れません。

次のようにテーブルの外部キー制約を表示する事もできます:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

外部キー制約はアウトプットの Comment カラム内にリストされています。

外部キー チェックを行っている時、InnoDB はそれが見なければいけない子または親レコード上に共有行レベル ロックを設定します。InnoDB は直ちに外部キー制約を確認します。その確認はトランザクション コミットに据え置きされません。

外部キー関係を持つテーブルのダンプ ファイルの再ロードを簡単にする為に、mysqldump は FOREIGN_KEY_CHECKS を 0 に設定する為に自動的にダンプ アウトプット内にステートメントを含みます。これは、ダンプが再ロードされた時にテーブルが特定の順番で再ロードされなければならないという問題を防ぎます。この変数をマニュアルで設定する事も可能です:

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
mysql> SOURCE dump_file_name;
mysql> SET FOREIGN_KEY_CHECKS = 1;
```


これは、もしダンプファイルが外部キーに対して正しい順番でオーダされていないテーブルを含んでいたら、テーブルをどんな順番でインポートしてもよいと許容します。これはインポート操作のスピードも上げます。[FOREIGN_KEY_CHECKS](#) を0に設定する事は、[LOAD DATA](#) と [ALTER TABLE](#) 操作の最中に外部キー制約を無視する為にも役に立ちます。しかし、[FOREIGN_KEY_CHECKS=0](#) であったとしても、InnoDB は、カラムが非適合カラム タイプの参照をつける外部キー制約の作成を許容しません。

InnoDB は、[SET FOREIGN_KEY_CHECKS=0](#) を行わない限り、[FOREIGN KEY](#) 制約によって参照を付けられたテーブルをドロップする事を許容しません。テーブルをドロップする時、その作成ステートメント内で定義された制約もまたドロップされます。

それは、もしドロップされたテーブルを再作成すると、それに参照をつける外部キー制約と同一の定義を持つはずで、それは右側のカラム名とタイプを持ち、先に述べたように参照キー上にインデックスを持つはずで、もしそれが満たされなければ、MySQL はエラー番号1005を返し、エラー メッセージ内で `errno 150` を参照します。

InnoDB と MySQL 複製

MySQL 複製は、[MyISAM](#) に対して機能するのと同じように [InnoDB](#) テーブルに機能します。スレーブ上のストレージ エンジンがマスタ上の元のストレージ エンジンと同じではない場合での方法で複製を利用する事も可能です。例えば、スレーブ上の [MyISAM](#) テーブルに、マスタ上の [InnoDB](#) テーブルへの修正を複製する事ができます。

マスタに新しいスレーブを設定するには、[InnoDB](#) テーブルの `.frm` ファイルと同様に、[InnoDB](#) テーブル スペースのコピーとログ ファイルのコピーを作成し、そのコピーをスレーブに移動させなければいけません。もし `innodb_file_per_table` 変数が有効であれば、`.ibd` ファイルもコピーする必要があります。これを行う為の正しい手順については [InnoDB データベースのバックアップと復旧](#) を参照して下さい。

もしマスタが既存スレーブを閉じる事ができるのであれば、[InnoDB](#) テーブル スペースとログ ファイルの完全なバックアップを取り、それをスレーブの設定の為に利用する事ができます。サーバを停止させずに新しいスレーブを作成するには、非フリー(商業用) [InnoDB Hot Backup ツール](#) を利用する事もできます。

[MyISAM](#) テーブルに対してだけ機能する [LOAD TABLE FROM MASTER](#) ステートメントを利用して [InnoDB](#) に複製を設定する事はできません。2つ可能な回避方法があります:

- マスタ上のテーブルをダンプし、ダンプ ファイルをスレーブ内にインポートしてください。
- [LOAD TABLE tbl_name FROM MASTER](#) を利用して複製を設定する前に、[ALTER TABLE tbl_name ENGINE=MyISAM](#) をマスタ上で利用し、そして後でマスタ テーブルを [InnoDB](#) に変換する為に [ALTER TABLE](#) を利用してください。しかし、定義が損失するのでこれは外部キー制約があるテーブルには利用しないで下さい。

マスタ上で失敗するトランザクションは複製に全く影響を与えません。MySQL 複製は、MySQL がデータを変更する SQL ステートメントを書き込むバイナリ ログに基づいています。失敗するトランザクション(例えば、外部キー違反の為、またはロールバックされる為)はバイナリ ログに書き込まないので、スレーブに送られません。詳しくは [START TRANSACTION](#)、[COMMIT](#)、そして [ROLLBACK](#) 構文 を参照してください。

InnoDB データとログ ファイルの追加と削除

このセクションでは、[InnoDB](#) テーブル スペースがスペースを使いきってしまった、ログ ファイルのサイズを変更したい時に何が出来るか説明しています。

[InnoDB](#) テーブル スペースのサイズを増やす一番簡単な方法は、最初からこれを自動拡大として設定する事です。テーブル スペース定義内の最後のデータ ファイルの `autoextend` 属性を指定してください。すると [InnoDB](#) は領域を使い切ってしまった時、そのファイルのサイズを自動的に8MB インクリメント増やします。インクリメントサイズは、MBで計られる `innodb_autoextend_increment` システム変数の値を設定する事で変更できます。

または、別のデータ ファイルを追加する事でテーブル スペースのサイズを増やす事ができます。これを行う為には、MySQL サーバを閉じ、`innodb_data_file_path` の最後に新しいデータ ファイルを追加する為にテーブル スペース設定を変更し、そしてサーバを再起動してください。

もし最後のデータ ファイルがキーワード `autoextend` で定義されていたら、テーブル スペースの再設定の手順は、最後のデータ ファイルがどのサイズまで成長するかを考慮する必要があります。データ ファイルのサイズを求め、それを 1024×1024 bytes (= 1MB) の倍数の最近値まで丸め、そして丸めたサイズを `innodb_data_file_path` 内で明示的に指定してください。すると別のデータ ファイルを追加する事ができます。`innodb_data_file_path` 内の最後のデータ ファイルだけが自動拡大として指定できるという事を覚えて置いてください。

ひとつの例として、テーブル スペースが1つだけ自動拡大データ ファイル `ibdata1` を持っていると仮定してください:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

このデータ ファイルが、時間をかけて988MB まで成長したと仮定してください。ここに、元のデータ ファイルを非自動拡大に変更し、別の自動拡大データ ファイルを追加した後の設定ラインがあります:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M:/disk2/ibdata2:50M:autoextend
```

テーブル スペース設定に新しいファイルを追加する時には、それが存在していない事を確認してください。InnoDB はサーバを再起動する時にファイルを作成し、初期化します。

現在、データ ファイルをテーブルスペースから削除する事はできません。テーブル スペースのサイズを小さくするには、この手順を利用してください:

1. 全ての InnoDB テーブルをダンプする為に `mysqldump` を利用してください。
2. サーバを停止してください。
3. 全ての存在するテーブルスペース ファイルを削除してください。
4. 新しいテーブルスペースを設定してください。
5. サーバを再起動してください。
6. ダンプ ファイルをインポートしてください。

InnoDB ログ ファイルの数やサイズを変更したければ、次の指示に従ってください。利用する手順は `innodb_fast_shutdown` の値によって決まります:

- もし `innodb_fast_shutdown` が2に設定されなければ:MySQL サーバを停止し、エラー無しでシャットダウンした事を確認する必要があります。(ログ内に未処理のトランザクションの情報が無い事を保証する為)シャットダウンの際に何かが起きた場合、テーブルスペースを復旧する為に必要になるので、古いログ ファイルを安全な場所にコピーしておいてください。古いログ ファイルをログ ファイル ディレクトリから削除し、ログ ファイル設定を変更する為に `my.cnf` を編集し、MySQL サーバを再起動してください。`mysqld` はログ ファイルが存在しない事を確認し、新しいものを作成している事を告げます。
- もし `innodb_fast_shutdown` が2に設定されると:サーバをシャットダウンし、`innodb_fast_shutdown` を1に設定し、サーバを再起動してください。サーバは復旧を許可されます。そして、サーバをもう一度シャットダウンし、InnoDB ログ ファイル サイズを変更する為に前出の項目で説明されている手順に従わなければいけません。`innodb_fast_shutdown` を2に設定し直し、サーバを再起動してください。

InnoDB データベースのバックアップと復旧

安全なデータベース管理の鍵は定期的にバックアップを取る事です。

InnoDB Hot Backup は InnoDB データベースが起動している最中にバックアップを取る事ができるオンライン バックアップ ツールです。InnoDB Hot Backup はデータベースをシャットダウンする必要が無く、ロックの設定も無く、通常のデータベースの処理を邪魔する事ありません。

もし MySQL サーバをシャットダウンする事ができるなら、テーブルを管理する為に InnoDB によって利用される全てのファイルで構成されているバイナリ バックアップを作成する事ができます。次の手順に従って下さい:

1. MySQL サーバをシャットダウンし、エラーが発生していない事を確認してください。
2. 全てのデータ ファイルを(`ibdata` ファイルと `.ibd` ファイル) 安全な場所にコピーしてください。
3. 全ての `ib_logfile` ファイルを安全な場所にコピーしてください。
4. `my.cnf` 設定ファイルを安全な場所にコピーしてください。
5. InnoDB テーブルの全ての `.frm` ファイルを安全な場所にコピーしてください。

複製は **InnoDB** テーブルと共に機能するので、ハイ アベイラビリティを必要とするデータベース サイトにデータベースのコピーを保管する為に、MySQL 複製性能を利用する事ができます。

今説明したようにバイナリ バックアップを作成する事に追加して、**mysqldump** を利用してテーブルのダンプを定期的に作成する必要があります。これは、バイナリ ファイルは気づかない内に破損する事があるからです。ダンプされたテーブルは人間が読解可能なテキスト ファイル内に格納されるので、テーブルの破損を見つける事は簡単になります。また、フォーマットが単純な為、深刻なデータ破損の可能性は小さいです。**mysqldump** は、別のクライアントをロックアウトせずに一貫性のあるスナップショットを作る為に利用できる **--single-transaction** オプションも持ちます

InnoDB データベースを今説明したばかりのバイナリ バックアップから現在まで復旧できるようにする為には、バイナリ ログがオンの状態で MySQL サーバを起動させる必要があります。すると、ポイント イン タイムの復旧を達成する為にバックアップ データベースにバイナリ ログを適応する事ができます:

```
mysqlbinlog yourhostname-bin.123 | mysql
```

MySQL サーバのクラッシュから復旧する為のたった一つの要求事項は、再起動させる事です。**InnoDB** は自動的にログを確認し、データベースの前進を現在まで実行します。**InnoDB** はクラッシュした時に存在していなかった、コミットされていないトランザクションを自動的にロールバックします。復旧の最中に、**mysqld** は次のようなアウトプットを表示します:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

もしデータベースが破損したり、ディスクが失敗したら、バックアップから復旧作業を行う必要があります。破損が起きた場合、まず最初に破損されていないバックアップを見つけなければいけません。ベース バックアップを復旧した後、バックアップが作成された後に実行された変更を格納する為に、**mysqlbinlog** と **mysql** を利用してバイナリ ログ ファイルから復旧を行ってください。

場合によっては、1つか複数の破損したテーブルをダンプ、ドロップ、または再作成するだけで十分な事も有ります。もちろん **CHECK TABLE** が全ての破損を検出する事はできませんが、テーブルが破損したかどうかを確認する為に **CHECK TABLE** SQL ステートメントを利用する事ができます。テーブルスペース ファイル内のファイル領域管理のインテグリティを確認する為に、**innodb_tablespace_monitor** を利用する事ができます。

場合によっては、明白なデータベース ページの破損は、OSがそれ自体のファイル キャッシュを破損している為に起きていて、ディスク上のデータは無傷な事があります。まず最初にコンピュータを再起動するのが一番良いでしょう。それを行う事で、データベース破損のように見えていたエラーを排除する事ができます。

InnoDB 復旧の強制

もしデータベース ページが破損したら、**SELECT INTO OUTFILE** を利用してデータベースからテーブルをダンプしたいかもしれません。通常、この方法で取得されたデータは無傷です。そうだとすると、破損によって **SELECT * FROM tbl_name** ステートメントや **InnoDB** バックグラウンド操作がクラッシュしたりアサートしたり、または **InnoDB** 前進復旧がクラッシュしたり、という事が起こります。しかし、バックグラウンドの操作を防いでいる間に、テーブルをダンプする事ができるように **InnoDB** ストレージ エンジンの起動を強制する事ができます。例えば、サーバを再起動する前に、オプション ファイルの **[mysqld]** セクションに次のラインを追加する事ができます:

```
[mysqld]
```

```
innodb_force_recovery = 4
```

`innodb_force_recovery` のゼロではない許容値が続きます。大きい数字は小さい数字の全ての予防策を含んでいます。もし最大4のオプション値を利用してテーブルをダンプする事ができれば、破損した独立ページ上のいくつかのデータが失われるだけなので、比較的に安全です。データベース ページは既に廃止された状態で残されるので、6の値はさらに徹底的であり、Bツリーやその他のデータベース構造に更なる破損を引き起こす可能性があります。

- 1 (`SRV_FORCE_IGNORE_CORRUPT`)

破損ページを検出したとしてもサーバを起動させてください。テーブルをダンプする助けになるので、`SELECT * FROM tbl_name` が破損したインデックス レコードとページを飛び越えるようにして下さい。

- 2 (`SRV_FORCE_NO_BACKGROUND`)

主スレッドが起動するのを防いで下さい。もし消去操作の最中にクラッシュが起きそうであれば、この復旧値はそれを防ぎます。

- 3 (`SRV_FORCE_NO_TRX_UNDO`)

復旧後にトランザクション ロールバックを起動しないでください。

- 4 (`SRV_FORCE_NO_IBUF_MERGE`)

挿入バッファ マージ操作も避けてください。もしそれらがクラッシュしそうであれば、行わないでください。テーブル統計を計算しないでください。

- 5 (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

データベースを起動する時に取り消しログを見ないで下さい:InnoDB は不完全なトランザクションもコミットしたように扱います。

- 6 (`SRV_FORCE_NO_LOG_REDO`)

復旧と共にログ前進を接続内で行わないでください。

それらをダンプする為にテーブルから `SELECT` する事ができ、または強制復旧が利用されたとしてもテーブルを `DROP` か `CREATE` する事ができます。もし与えられたテーブルがロールバック上でクラッシュを引き起こしていると知ったら、それをドロップする事ができます。大量の失敗インポートや `ALTER TABLE` によって引き起こされた暴走ロールバックを停止する為にこれを利用する事ができます。ロールバックせずにデータベースを立ち上げる為に `mysqld` 処理を停止し、`innodb_force_recovery` を 3 に設定し、そして暴走ロールバックを引き起こしているテーブルを `DROP` する事ができます。

データベースはそれ以外の場合にゼロ以外の値の `innodb_force_recovery` と共に利用するべきではありません。`innodb_force_recovery` が0よりも大きい場合、安全の為、InnoDB はユーザが `INSERT`、`UPDATE`、または `DELETE` 操作を行うのを防ぎます。

チェックポイント

InnoDB は「fuzzy」チェックポイントとして知られているチェックポイント性能を実装します。InnoDB は小さいバッチ内のバッファ プールから変更されたデータベース ページをフラッシュします。チェックポイント処理の最中にユーザ SQL ステートメントの処理を実際に停止させる、バッファ プールを単一バッチ内でフラッシュする必要はありません。

クラッシュ復旧の最中に、InnoDB はログ ファイルに書き込まれたチェックポイント ラベルを探します。それは、ラベルの前のデータベースへの全ての変更がデータベースのディスク イメージ内に存在する事を知っています。そして、InnoDB はデータベースにログされた変更を適用しながら、チェックポイントから前方にログ ファイルをスキップします。

InnoDB は交代制でそのログファイルに書き込みをします。バッファ プール内のデータベース ページがディスク上のイメージと異なるようにコミットされた全ての変更は、InnoDB が復旧を行わなければならない場合の為にログ ファイル内で有効である必要があります。これは、InnoDB がログ ファイルを再利用し始めた時、ディスク上のデータベースのイメージが、InnoDB が再利用しようとしているログ ファイル内にログされた変更を確実に含むという事を意味します。言い換えると、InnoDB はチェックポイントを作成する必要がある、変更されたデータベース ページをディスクにフラッシュする事を含んでいる事が多いです。

前出の説明の中で、なぜログ ファイルをととても大きくする事がチェックポイントの中でディスク I/O を救うかも知れないのかが説明されています。ログ ファイル全体の大きさをバッファ プールと同じ、またはそれよりも大きく設定する事は意味を持つ事が多いです。大きいログ ファイルを利用する事の欠点は、データベースに適応させるログされた情報がより多くある為に、クラッシュ復旧に長時間かかるという事です。

InnoDB データベースを別のマシンに移動する

Windows 上では InnoDB はいつもデータベースとテーブル名を小文字で内部的に格納します。データベースを Unix から Windows に、または Windows から Unix にバイナリ フォーマットで移動するには、全てのテーブルとデータベース名を小文字で持つ必要があります。これを行う簡単な方法は、データベースやテーブルを作成する前に `my.cnf` や `my.ini` ファイルの `[mysqld]` セクションに次のラインを追加する事です:

```
[mysqld]
lower_case_table_names=1
```

MyISAM データ ファイルのように、InnoDB データとログ ファイルは同じ浮動小数点数フォーマットを持つ全てのプラットフォーム上でバイナリ互換性があります。InnoDB データベースのバックアップと復旧 内にリストされている、全ての関連のあるファイルをコピーするだけで InnoDB データベースを移動する事ができます。もし浮動小数点フォーマットが異なっても、テーブル内で `FLOAT` か `DOUBLE` データタイプを利用していなければ、手順は同じです: 関連のあるファイルをコピーしてください。もしフォーマットが異なりテーブルが浮動小数点データを含んでいたら、1つのマシン上でテーブルをダンプする為に `mysqldump` を利用し、そして別のマシンにダンプ ファイルをインポートしなければいけません。

性能を向上させる為のひとつの方法は、インポート トランザクションが生成する大きいロールバック セグメントの為にテーブルスペースが充分な領域を持っていると仮定して、データをインポートする時に自動コミットモードをオフにする事です。テーブル全体か、テーブルのセグメントをインポートした後にコミットを行ってください。

InnoDB トランザクション モデルとロック

InnoDB トランザクション モデル内のゴールは、マルチ バージョン データベースの優れた性質を、従来の二相ロックと合体させる事です。InnoDB は、行レベルでロックを行い、デフォルトではクエリを Oracle 式の非ロックの一貫した読み取りとして実行します。InnoDB のロック テーブルは領域効率の高い方法で格納される為、ロック エスカレーションは不要です: 一般には、複数のユーザがデータベースのあらゆるレコードまたはレコードのランダムなサブセットをロックする事ができ、InnoDB でメモリ不足が発生する事ありません。

InnoDB ロック モード

InnoDB は2つのタイプのロックがあるスタンダード行レベル ロックを実装します:

- 共有(S)ロックはトランザクションが行を読む事を許容します(タプル)。
- 専用(X)ロックはトランザクションが行を更新、削除する事を許容します。

もしトランザクション `T1` が共有(S)ロックをタプル `t` 上で保持していたら、

- `t` 上の `S` ロックの為にいくつかの独特なトランザクション `T2` からのリクエストは直ちに認められます。結果として、`T1` と `T2` の両方は `t` 上で `S` ロックを保持します。
- `t` 上の `X` ロックの為にいくつかの独特なトランザクション `T2` からのリクエストは直ちに認められます。

もしトランザクション `T1` が独特な(X)ロックをタプル `t` 上で保持していたら、その時は `t` 上のロックに対する独特なトランザクション `T2` からのリクエストは直ちに認められません。代わりに、トランザクション `T2` はトランザクション `T1` がタプル `t` 上でそのロックをリリースするのを待たなければいけません。

さらに、InnoDB はレコード ロックの共存とテーブル全体のロックを許容する 複数粒度ロック をサポートします。複数粒度レベルでのロックを実用的にする為に、インテンション ロックと呼ばれるロックの追加タイプが利用されます。インテンション ロックは InnoDB 内のテーブル ロックです。インテンション ロックの裏にある考えは、後でトランザクションが、そのテーブル内でどのタイプ(共有か専用か)のロックを行の為に要求するのかを指示するという事です。InnoDB 内で利用されるインテンション ロックには2つのタイプがあります。(トランザクション `T` がテーブル `R` 上で指示されたタイプのロックをリクエストしたと仮定してください。):

- 共有インテンション(IS): トランザクション `T` は `R` 内の独立行上に `S` ロックを設定する予定です。

- 専用インテンション(**IX**): トランザクション **T** はそれらの行上に **X** ロックを設定する予定です。

インテンション ロック プロトコルは次のようになります:

- 与えられたトランザクションは与えられた行上で **S** ロックを得る前に、まずその行を含んでいるテーブル上に **IS** か、またはさらに強いロックを得る必要があります。
- 与えられたトランザクションは与えられた行上で **X** ロックを得る前に、まずその行を含んでいるテーブル上に **IX** ロックを得る必要があります。

これらのルールは ロック タイプ変換互換性マトリックス を用いて便利に要約することができます:

	X	IX	S	IS
X	対立	対立	対立	対立
IX	対立	互換性あり	対立	互換性あり
S	対立	対立	互換性あり	互換性あり
IS	対立	互換性あり	互換性あり	互換性あり

既存ロックと互換性があれば、リクエストしているトランザクションにロックが与えられます。既存ロックと対立すれば、リクエストしているトランザクションにロックは与えられません。トランザクションは、既存の対立中のロックがリリースされるまで待ちます。もしロックのリクエストが既存ロックと対立する為にデッドロックが起り、そのロックが与えられないとしたら、エラーが発生します。

従って、インテンション ロックはフル テーブル リクエスト以外は何もブロックしません。(例えば **LOCK TABLES ... WRITE**) **IX** と **IS** ロックの主な目的は、誰かが行をロックしている、またはテーブル内の行をロックしようとしている、という事です。

次の例は、ロック リクエストがデッドロックを引き起こす時にどのようにエラーが発生するかを表しています。この例には、**A** と **B** の2つのクライアントが登場します。

最初に、クライアント **A** が行を1つ含むテーブルを作成し、トランザクションを開始します。トランザクション内で、**A** は共有モードで選択した行に **S** ロックを獲得します:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+-----+
| i |
+-----+
| 1 |
+-----+
1 row in set (0.10 sec)
```

次に、クライアント **B** がトランザクションを開始し、テーブルから行を削除しようとします:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

削除作業は **X** ロックを必要とします。クライアント **A** が保持している **S** ロックと互換性が無い為にそのロックは与えられず、そのリクエストは行とクライアント **B** のロック リクエストの列に並びます。

最後に、クライアント **A** もテーブルから行を削除しようとします:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

クライアント A は行を削除する為に X ロックが必要なので、ここでデッドロックが発生します。しかし、クライアント B が既に X ロックへのリクエストを持ち、またその S ロックをクライアント A がリリースするのを待っている為に、ロック リクエストは与えられません。B による X ロックのリクエストの為に、A によって保持されている S ロックを X ロックにアップグレードする事もできません。結果として、InnoDB はクライアント A に対してエラーを生成し、そのロックをリリースします。その時点で、クライアント B のロック リクエストが与えられ、B はテーブルから行を削除します。

InnoDB と AUTOCOMMIT

InnoDB 内では、全てのユーザ行動はトランザクション内で起こります。もし自動コミット モードが有効なら、各 SQL ステートメントはそれ自体の上に単一トランザクションを形作ります。MySQL はデフォルトで自動コミットを有効にして新しい接続を始めます。

もし自動コミット モードが `SET AUTOCOMMIT = 0` を利用してオフにされたら、ユーザは常にトランザクションを開いていると考える事ができます。SQL `COMMIT` か `ROLLBACK` ステートメントは現在のトランザクションを終了し新しい物を開始します。`COMMIT` は、現在のトランザクション内で行われた変更は永続的であり、別のユーザからも見る事ができるという事を意味します。`ROLLBACK` ステートメントは反対に、現在のトランザクションによって行われた全ての変更をキャンセルします。両ステートメントは、現在のトランザクションの最中に設定された全ての InnoDB ロックをリリースします。

もしその接続の自動コミットが有効であれば、ユーザは明示的な `START TRANSACTION` か `BEGIN` ステートメントを利用して複合ステートメント トランザクションを開始する事でそれを実行し、また `COMMIT` か `ROLLBACK` を利用して終了する事ができます。

InnoDB と TRANSACTION ISOLATION LEVEL

SQL:1992 トランザクション分離レベルの観点では、InnoDB デフォルトは `REPEATABLE READ` です。InnoDB は SQL スタンダードによって説明されている4つの全てのトランザクション分離レベルを提供します。コマンドライン上、またはオプション ファイル内で `--transaction-isolation` オプションを利用する事で、全ての接続にデフォルトの分離レベルを設定する事ができます。例えば、オプション ファイルの `[mysqld]` セクション内で次のようにオプションを設定する事ができます:

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                        | REPEATABLE-READ | SERIALIZABLE}
```

ユーザは `SET TRANSACTION` ステートメントを利用して単一セッションや全ての新しい接続の分離レベルを変更する事ができます。その構文は次のようになります:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED | READ COMMITTED
     | REPEATABLE READ | SERIALIZABLE}
```

`--transaction-isolation` オプションのレベル名内にはハイフンがありますが、`SET TRANSACTION` ステートメントには無い事に注意してください。

デフォルト動作は、次の (まだ始まっていない) トランザクションの分離レベルを設定する事です。もし `GLOBAL` キーワードを利用すると、そのステートメントはその時点以降に作成された全ての新しい接続に対して、デフォルトのトランザクション レベルをグローバルに設定します。(既に存在する接続には設定しません。)これには `SUPER` 権限が必要です。`SESSION` キーワードの利用は、現在の接続上で今後行われる全てのトランザクションに対して、デフォルト トランザクション レベルを設定します。

全てのクライアントは自由にセッションの分離レベル(トランザクションの最中だとしても)、または次のトランザクションの分離レベルを変更する事ができます。

次のステートメントを利用して `tx_isolation` システム変数の値を確認する事で、グローバル、またセッション トランザクションの分離レベルを決定する事ができます:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

行レベル ロック内では、InnoDB はネクスト キー ロックを利用します。これは、インデックス レコード以外に、InnoDB が、別のユーザによってインデックス レコードの直前に挿入される事を防ぐ為に、インデックス レコードに先行する「ギャップ」をロックする事もできるという事を意味します。ネクスト キー ロックは、インデックス

レコードとその前のギャップをロックするロックを参照します。ギャップ ロックは、ただ単に、いくつかのインデックスレコードの前にギャップをロックするだけのロックを参照します。

InnoDB 内の各分離レベルに関する詳細説明は次の物です:

- **READ UNCOMMITTED**

SELECT ステートメントは非ロックの方法で実行されますが、レコードの以前のバージョンが利用される事もあるでしょう。従って、この分離レベルを利用したこのような読み込みは一貫性がありません。これは「ダーティ リード」とも呼ばれます。そうでなければ、この分離レベルは **READ COMMITTED** のように機能します。

- **READ COMMITTED**

若干 Oracle に似ている分離レベル。全ての **SELECT ... FOR UPDATE** と **SELECT ... LOCK IN SHARE MODE** ステートメントは、インデックス レコードだけをロックしそれらの前のギャップはロックしませんので、ロックされたレコードの隣に新しいレコードの自由挿入を許容します。固有検索条件を持つ固有インデックスを利用した **UPDATE** と **DELETE** ステートメントは、検出されたインデックスレコードだけをロックし、その前のギャップはロックしません。値域タイプの **UPDATE** と **DELETE** ステートメント内では、InnoDB はネクスト キーがギャップロックを設定し、別のユーザからの値域によって変換されたギャップへの挿入をブロックする必要があります。これは、「phantom rows」が MySQL 複製と復旧が機能する為にブロックされなければいけない為になります。

一貫した読み取りは、Oracle 内と同じように機能します:同じトランザクション内でも、各一貫した読み取りはそれ自体の新鮮なスナップショットを設定し、読み取ります。詳しくは **一貫非ロック読み取り** を参照してください。

- **REPEATABLE READ**

これは InnoDB のデフォルト分離レベルです。固有検索条件を持つ固有インデックスを利用する **SELECT ... FOR UPDATE**、**SELECT ... LOCK IN SHARE MODE**、**UPDATE**、そして **DELETE** ステートメントは、前にあるギャップではなく、検索したインデックスレコードのみをロックします。その他の検索条件を利用して、これらの操作はネクスト キーがギャップロックでインデックス範囲をスキャンしながらネクスト キー ロックを採用し、別のユーザによる新しい挿入をブロックします。

一貫した読み取りの中に、**READ COMMITTED** 分離レベルとの重要な違いがあります:同一トランザクション内の全ての一貫した読み取りは、最初の読み取りで確立された同じスナップショットを読み取ります。このしきりとは、もし同じトランザクション内でいくつかの単純な **SELECT** ステートメントを発行すると、これらの **SELECT** ステートメントはお互いに対しても一貫性を持つという事を意味します。詳しくは **一貫非ロック読み取り** を参照してください。

- **SERIALIZABLE**

このレベルは **REPEATABLE READ** と似ていますが、InnoDB は暗黙的に全ての単純な **SELECT** ステートメントを **SELECT ... LOCK IN SHARE MODE** にコミットします。

一貫非ロック読み取り

一貫した読み取りとは、InnoDBがそのマルチ バージョニング機能を使用して、ある時点でのデータベースのスナップショットをクエリに提示する事を意味します。クエリには、その時点より前にコミットされたトランザクションによる変更のみが示され、その時点より後のトランザクションまたはコミットされていないトランザクションによる変更は示されません。例外として、クエリを発行したトランザクション自体による変更はクエリに示されます。このルールに対する例外によって次のような事が起こると覚えておいてください:テーブル内のいくつかの行を更新すると、**SELECT** は他の行の古いバージョンを確認すると同時に、更新された行の最新のバージョンを確認します。もし別のユーザが同時に同じテーブルを更新すると、今までとは違う状態のテーブルをデータベース内で確認するかもしれないという例外があるかもしれません。

デフォルトの **REPEATABLE READ** 分離レベルで起動しているなら、同じトランザクション内の全ての一貫した読み取りは、そのトランザクション内の最初の読み取りで確立されたスナップショットを読み取ります。現在のトランザクションをコミットしその後新しいクエリを発行する事で、クエリにより新鮮なスナップショットを得る事ができます。

InnoDB が **READ COMMITTED** と **REPEATABLE READ** 分離レベル内で **SELECT** ステートメントを処理する中で、一貫した読み取りはデフォルトのモードです。一貫した読み取りはそれがアクセスするテーブル上に一切ロックを設定しないので、別のユーザは、そのテーブル上で一貫した読み取りが行われているのと同時にそれらのテーブルを自由に変更する事ができます。

一貫した読み取りは **DROP TABLE** と **ALTER TABLE** 全体には機能しない事に注意してください。MySQL がドロップされたテーブルを利用する事ができず、InnoDB がそのテーブルを破壊する為、一貫した読み取りは **DROP TABLE** 全体には機能しません。**ALTER TABLE** は元テーブルのテンポラリ コピーを作成し、それができた時に元テーブルを破壊する事で機能する為、一貫した読み取りは **ALTER TABLE** 全体には機能しません。トランザクション内で一貫した読み取りを再発行する時、新しいテーブル内の行はトランザクションのスナップショットが撮られた時に存在していなかった為に見る事ができません。

SELECT ... FOR UPDATE と SELECT ... LOCK IN SHARE MODE ロック読み取り

場合によっては、一貫した読み取りは便利ではありません。例えば、テーブル **child** に新しい行を追加したければ、子供がテーブル **parent** に親を持っている事を確認するでしょう。次の例は、応用コード内でどのように参照整合性を実装するのかを表しています。

テーブル **parent** を読み取る為に一貫した読み取りを利用し、実際にテーブル内に子供の親を確認したと仮定してください。テーブル **child** に子供行を安全に追加する事ができますか?別のユーザが知らない間にテーブル **parent** から親行を削除する可能性があるので、答えはノーです。

この解決法は、**LOCK IN SHARE MODE** を利用して **SELECT** をロック モードで実行する事です:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

共有モードで読み取りを行うというのは、まず最新の有効なデータを読み取り、そして読んだ行に共有モードを設定するという意味です。共有モード ロックは、読み取った行が別の人によって更新されたり、削除されたりする事を防ぎます。また、もし最新データが別のクライアント接続のコミットされていないトランザクションに属していたら、そのトランザクションがコミットされるまで待ちます。先行クエリが親 'Jones' を返すのを確認した後、**child** テーブルに子レコードを安全に追加し、トランザクションをコミットする事ができます。

別の例を見てみましょう:テーブル **child** に追加された各子供に固有識別子を割り当てる為に使用する整数カウンタフィールドが、テーブル **child_codes** 内にあります。カウンタの現在値を読む為に、一貫した読み取りや、共有モード読み取りを利用する事は、そのデータベースの2ユーザが同じカウンタ値を確認する可能性があり、またその2ユーザが同じ識別子を利用してテーブルに子供を追加しようとするすると複製キーエラーが発生する為、良い考えとは言えません。

もし2ユーザがカウンタを同時に読むと、少なくとも1ユーザはカウンタを更新しようとする時にデッドロックになってしまう為、**LOCK IN SHARE MODE** はこの場合良い解決法とはいえません。

この場合、カウンタの読み取りとインクリメントを実装する為の良い方法が2つあります:(1) カウンタを1でインクリメントする事で更新し、その後にだけ読み取る、または、(2) ロックモード **FOR UPDATE** を利用してまずカウンタを読み取り、その後にインクリメントする。後者の方法は、次のように実装できます:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A **SELECT ... FOR UPDATE** は、読み取る各行上に専用ロックを設定し、最新の有効データを読み取ります。従って、それは SQL **UPDATE** が行上に設定する物と同じロックを設定します。

前出の例は、ただ単に **SELECT ... FOR UPDATE** がどのように機能するかを表す例です。MySQL 内では、固有識別子生成する特定のタスクは、実際にはテーブルへの単一アクセスの利用だけで達成する事ができます:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

SELECT ステートメントはただ単に識別子情報を検索するだけです。(現在の接続特有の物)これはテーブルにアクセスしません。

IN SHARE MODE と **FOR UPDATE** 読み取りによって設定されたロックは、トランザクションがコミットされたりロールバックされたりした時にリリースされます。

ネクスト キー ロック:ファントムの問題を防ぐ

行レベル ロック内では、InnoDB は ネクスト キー ロック と呼ばれるアルゴリズムを利用します。InnoDB は、それがテーブルのインデックスを検索やスキャンする時に、遭遇したインデックス レコード上で共有、または専用ロックを設定する、という方法で行レベル ロックを実行します。従って、行レベル ロックは実際はインデックス レコードロックであるという事になります。

InnoDB がインデックスレコード上で設定するロックは、そのインデックスレコードの前の「ギャップ」にも影響を与えます。もしユーザがインデックス内のレコード **R** 上に共有または専用ロックを持っていたら、別のユーザはインデックスの順番で **R** の直前に新しいインデックスレコードを挿入する事はできません。このようなギャップのロックは、一般的に「ファントムの問題」と呼ばれる物を防ぐ為に行われます。後で選択した行内のいくつかのカラムを更新するつもりで、100よりも大きい値の識別子を持つ **child** テーブルから全ての子供を読み、ロックしたいと仮定します:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

id カラム上にインデックスがあると仮定してください。クエリは、**id** が100以上の最初のレコードから、そのインデックスをスキャンします。もしインデックスレコード上に設定されたロックがギャップに挿入された物をロックしなければ、その一方で新しい行がテーブルに挿入されるでしょう。もし同じトランザクション内で同じ **SELECT** を実行すると、クエリから返された結果セット内に新しい行を見つける事ができます。これは、トランザクションの分離原理に反しています:トランザクションは、読み取ったデータをトランザクションの最中に変更させない為に起動する必要があります。行セットをデータ項目であるとみなすと、新しい「ファントム」の子供はこの分離原理に違反します。

When InnoDB がインデックスをスキャンする時、インデックス内の最後のレコードの後のギャップをロックする事もできます。それは前出の例の中で起きています:InnoDB によって設定されるロックは、**id** が100以上になるテーブルへの挿入を防ぎます。

アプリケーション内に一意性チェックを実装する為にネクストキーロックを利用する事ができます:共有モードでデータを読み取り、挿入しようとする行に重複が見られなければ、行を確実に挿入できます。また、読み取り中は対象となる行の後続の行にネクストキーロックが設定されて、第三者による重複行の挿入を防ぎます。このように、ネクストキーロックによって、テーブル内に存在しない物を「ロック」する事ができます。

InnoDB 内での一貫した読み取りの例

デフォルトの **REPEATABLE READ** 分離レベルで起動していると仮定して下さい。一貫した読み取り(通常の **SELECT** ステートメント)では、InnoDBは、クエリがデータベースを参照する時の基準となるタイムポイントをトランザクションに与えます。こうして、タイムポイントが割り当てられた後に、他のトランザクションが行を削除してコミットしたとしても、一度読み取った内容は変わりません。挿入と更新も同じように扱われます。

割り当てられたタイムポイントを先に進めるには、トランザクションをコミットし、新たな **SELECT** を実行します。

これは、マルチバージョン並行処理制御と呼ばれています。

	User A	User B
	SET AUTOCOMMIT=0;	SET AUTOCOMMIT=0;
time		
	SELECT * FROM t;	
	empty set	
		INSERT INTO t VALUES (1, 2);
v	SELECT * FROM t;	
	empty set	
		COMMIT;
	SELECT * FROM t;	
	empty set	
	COMMIT;	
	SELECT * FROM t;	

	1 2	

	1 row in set	

この例の中では、ユーザ A は、ユーザ A と B の両方が挿入をコミットした時だけ、B によって挿入された行を確認する事ができ、それによってタイムポイントは B のコミットよりも先に進みます。

データベースの「最新の」状態を確認したければ、**READ COMMITTED** 分離レベルがロック読み取りのどちらかを利用しなければいけません:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

InnoDB 内で各種 SQL ステートメントによって設定されるロック

ロックする読み取り、**UPDATE**、または **DELETE** は通常、SQL ステートメントの処理の中でスキャンされる全てのインデックスレコード上にレコードロックを設定します。行を排除するステートメント内に **WHERE** 条件があればそれは問題ではありません。InnoDB は正確な **WHERE** 条件を記憶しませんが、どのインデックス範囲がスキャンされたのかは分かっています。レコードロックは通常、レコードの前に「ギャップ」への挿入も速やかにブロックするネクストキーロックです。

もし設定されるロックが専用であれば、InnoDB は常にクラスタ化されたインデックスレコードの検索と、それに対するロックの設定もします。

もしご自分のステートメントに適応したインデックスがなく、MySQL がステートメントを処理する為にテーブル全体をスキャンしなければいけないなら、テーブルの全ての行がロックされ、それはその代わりにテーブルへの別のユーザによる全ての挿入をブロックします。クエリが不必要にたくさんの行をスキャンする必要がなくなるように、よいインデックスを作成する事が重要です。

InnoDB は次のように特定のロックタイプを設定します:

- **SELECT ... FROM** は一貫した読み取りであり、データベースのスナップショットを読み取り、トランザクションの分離レベルが **SERIALIZABLE** に設定されなければロックは設定しません。これは、**SERIALIZABLE** レベルに対して、直面するインデックスレコード上に共有ネクストキーロックを設定します。
- **SELECT ... FROM ... LOCK IN SHARE MODE** は、その読み取りが直面する全てのインデックスレコード上に共有ネクストキーロックを設定します。
- **SELECT ... FROM ... FOR UPDATE** は、その読み取りが直面する全てのインデックスレコード上に専用共有ネクストキーロックを設定します。
- **INSERT INTO ... VALUES (...)** は挿入された行上に専用ロックを設定します。このロックはネクストキーロックではなく、挿入された行の前のギャップに別のユーザが挿入する事を防ぎます。もし複製キーエラーが発生すると、複製インデックスレコード上の共有ロックが設定されます。
- テーブル上であらかじめ指定された **AUTO_INCREMENT** カラムを初期化している間、InnoDB は **AUTO_INCREMENT** カラムと関係しているインデックスの最後に専用ロックを設定します。

自動インクリメントカウンタにアクセスする時、InnoDB は、トランザクション全体の最後までではなく、現在の SQL ステートメントの最後まで続く、特別なテーブルロックモード **AUTO-INC** を利用します。**AUTO-INC** テーブルロックが行われている間は、別のクライアントはテーブルに挿入ができない事に注意してください。InnoDB と **AUTOCOMMIT** を参照してください。

InnoDB は、ロックを設定せずに、あらかじめ初期化された **AUTO_INCREMENT** カラムの値をフェッチします。

- **INSERT INTO T SELECT ... FROM S WHERE ...** は **T** に挿入された各行上に専用(ネクストキーではない)ロックを設定します。InnoDB は、**innodb_locks_unsafe_for_binlog** が有効でなければ共有ネクストキーロックを **S** に設定し、その場合それは **S** に対して一貫した読み取りとしての検索を行います。InnoDB は後者の場合にロックを設定する必要があります:バックアップからの前進復旧では、全ての SQL ステートメントはそれが元々行われたのと全く同じ方法で実行されなければいけません。
- **CREATE TABLE ... SELECT ...** は、前出の項目のように、**SELECT** を一貫した読み取りとして、または共有ロックを利用して実行します。
- **REPLACE** は、もし固有キーにコリジョンがなければ挿入と同じように行われます。反対に、専用ネクストキーロックは更新されなければいけない行上に置かれます。
- **UPDATE ... WHERE ...** は、検索が直面する全てのレコード上に専用ネクストキーロックを設定します。
- **DELETE FROM ... WHERE ...** は、検索が直面する全てのレコード上に専用ネクストキーロックを設定します。
- もし **FOREIGN KEY** 制約がテーブル上で定義されると、確認される制約条件を要求する全ての挿入、更新、または削除が、制約を確認する為に参照するレコード上に共有レコードレベルロックを設定します。InnoDB も、制約が失敗する場合に備えてこれらのロックを設定します。

- **LOCK TABLES** はテーブル ロックを設定しますが、これはこれらのロックを設定する InnoDB レイヤより上位の MySQL レイヤです。InnoDB は `innodb_table_locks=1` (デフォルト)と `AUTOCOMMIT=0` であればテーブル ロックを認識しており、また InnoDB より上位の MySQL レイヤは行レベル ロックを識別します。そうでなければ、InnoDB の自動デッドロック検出は、そのようなテーブル ロックが関連しているデッドロックを検出する事はできません。また、上位の MySQL レイヤは行レベル ロックを識別しないので、別のユーザが現在行レベル ロックを持っているテーブル上にテーブル ロックを得る事が可能です。しかし、**デッドロックの検出とロールバック** で説明されているように、これはトランザクション インテグリティを危険にさらしたりはしません。InnoDB テーブル上の制約 もご参照ください。

暗黙的なトランザクション コミットとロールバック

デフォルトにより、MySQL は自動コミット モードが有効な状態で各クライアント接続を起動します。自動コミットが有効な時、もしステートメントがエラーを返さなければ MySQL は各 SQL ステートメントの後にコミットを行います。もし SQL ステートメントがエラーを返したら、コミットやロールバック性能はそのエラーによって決まります。詳しくは [InnoDB エラー処理](#) を参照してください。

もし自動コミット モードがオフで、最後のトランザクションを明示的にコミットせずに接続を閉じると、MySQL はそのトランザクションをロールバックします。

次の各ステートメント(そしてそれらの同義語)は、まるでステートメントを実行する前に **COMMIT** を行ったかのように、暗黙にトランザクションを終了します。

- **ALTER FUNCTION**、**ALTER PROCEDURE**、**ALTER TABLE**、**BEGIN**、**CREATE DATABASE**、**CREATE FUNCTION**、**CREATE INDEX**、**CREATE PROCEDURE**、**CREATE TABLE**、**DROP DATABASE**、**DROP FUNCTION**、**DROP INDEX**、**DROP PROCEDURE**、**DROP TABLE**、**LOAD DATA INFILE LOCK TABLES**、**RENAME TABLE**、**SET AUTOCOMMIT=1**、**START TRANSACTION**、**TRUNCATE TABLE**、**UNLOCK TABLES**
- MySQL 5.1.3 から、**ALTER VIEW**、**CREATE TRIGGER**、**CREATE USER**、**CREATE VIEW**、**DROP TRIGGER**、**DROP USER**、**DROP VIEW**、そして **RENAME USER** ステートメントは暗黙的なコミットを引き起こすようになりました。
- **UNLOCK TABLES** は、もしテーブルが現在 **LOCK TABLES** でロックされていたらトランザクションを行います。これは、**FLUSH TABLES WITH READ LOCK** ステートメントがテーブル レベル ロックを取得しない為、これに続く **UNLOCK TABLES** に対しては行われません。
- InnoDB 内の **CREATE TABLE** ステートメントは単一トランザクションとして生成されます。これは、ユーザからの **ROLLBACK** はトランザクションの最中にユーザが作成した **CREATE TABLE** ステートメントを解除しないという意味です。
- **CREATE TABLE** と **DROP TABLE** は、もし **TEMPORARY** キーワードが利用されたらトランザクションを実行しません。(これは、コミットを引き起こさない **CREATE INDEX** のようなテンポラリ テーブルへのその他の操作には当てはまりません。)
- MySQL 5.1.11 以前では、**LOAD DATA INFILE** は全てのストレージ エンジンに対して暗黙的なコミットを引き起こしました。MySQL 5.1.12 からは、**NDB** ストレージエンジンを利用しているテーブルに対してだけ暗黙的なコミットを引き起こすようになりました。更なる情報については、バグ #11151を参照してください。

トランザクションはネスト化されません。これは、**START TRANSACTION** ステートメントかその同義語の1つを発行する時点でのトランザクションに対して実行される、暗黙的な **COMMIT** の結果です。

トランザクションが **ACTIVE** 状態の間は、暗黙的なコミットを引き起こすステートメントは XA トランザクションでは利用する事はできません。

デッドロックの検出とロールバック

InnoDB は自動的にトランザクションのデッドロックを検出し、デッドロックを破壊する為にトランザクションをロールバックします。InnoDB は、トランザクションのサイズが挿入、更新、または削除された行数によって決定される小さいトランザクションを選んでロールバックしようとします。

InnoDB は `innodb_table_locks=1` (デフォルト)と `AUTOCOMMIT=0` であればテーブル ロックを認識しており、またそれより上位の MySQL レイヤは行レベル ロックを識別します。そうでなければ、InnoDB は MySQL **LOCK TABLES**

ステートメントによるテーブル ロック セットや InnoDB 以外のストレージ エンジンによるロック セットが関連しているデッドロックを検出する事ができません。innodb_lock_wait_timeout システム変数の値を設定する事によって、これらの状況を解決しなければいけません。

InnoDB がトランザクションの完全なロールバックを実行する時、トランザクションによって設定される全てのロックはリリースされます。しかし、もし単一 SQL ステートメントだけがエラーの結果ロールバックされると、ステートメントによって設定されたいくつかのロックは維持されるかもしれません。これは、InnoDB が、後でどの行がどのステートメントによって設定されたのかという事を確認する事ができないようなフォーマットで行ロックを格納する為に起こります。

デッドロックにどのように対処するか

デッドロックはトランザクション データベースの中ではよく知られている問題ですが、ある特定のトランザクションを全く起動できないほど頻繁に起きる訳ではないのならば危険では有りません。通常は、トランザクションがデッドロックの為にロールバックされたらそれを再発行できる準備が常にできているように、アプリケーションを書き込まなければいけません。

InnoDB は自動行レベル ロックを利用します。単一行を挿入または削除したばかりのトランザクションの場合でもデッドロックを得る事ができます。これは、これらの操作は実際は「アトミック」ではないからです。それらは自動的に挿入または削除された行の(可能であればいくつかの)インデックス レコード上にロックを設定します。

次のテクニックを利用して、デッドロックに対処し、それらの発生の可能性を減らす事ができます:

- **SHOW ENGINE INNODB STATUS** を利用して最新のデッドロックの原因を究明してください。それで、アプリケーションがデッドロックを防ぐ様に調整する事ができます。
- トランザクションがデッドロックのせいで失敗したら再発行できるように常に準備しておいてください。デッドロックは危険ではありません。もう一度やってみてください。
- トランザクションを頻繁にコミットしてください。小さいトランザクションはコリジョンの傾向が少ないです。
- もしロック読み取り(**SELECT ... FOR UPDATE** or **... LOCK IN SHARE MODE**)を利用しているなら、**READ COMMITTED** のような低分離レベルを利用するように試みてください。
- 決まった順番でテーブルと行にアクセスしてください。するとトランザクションは明確な列になりデッドロックしません。
- テーブルに適切なインデックスを追加してください。するとクエリがスキャンしなければいけないインデックス レコードが減り、その結果ロックの設定が減ります。MySQL サーバが、クエリにとってどのインデックスが最適だと認識するのかを究明する為に **EXPLAIN SELECT** を利用してください。
- ロックの利用を少なくしてください。もし **SELECT** が古いスナップショットからデータを返す事を許容できるなら、それに条項 **FOR UPDATE** か **LOCK IN SHARE MODE** を追加しないでください。同じトランザクション内のもそれぞれの一貫した読み取りは、それ自体の新鮮なスナップショットから読み取りをするので、**READ COMMITTED** 分離レベルを利用する事は良い事です。
- もし他に方法がなければ、テーブル レベル ロックを利用してトランザクションを直列化してください。**LOCK TABLES** を InnoDB テーブルのようなトランザクション テーブルと共に利用する正しい方法は、**AUTOCOMMIT = 0** を設定し、トランザクションを明示的にコミットするまでは **UNLOCK TABLES** をコールしないという方法です。例えば、もしテーブル **t1** に書き込み、テーブル **t2** から読み取る必要があれば、これを行う事ができます:

```
SET AUTOCOMMIT=0;
LOCK TABLES t1 WRITE, t2 READ, ..., ... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

テーブル **child_codes** 内にテーブル レベル ロックはトランザクションの列を整え、デッドロックを防ぎます。

- トランザクションを直列化する別の方法は、単一行だけを含む補助「セマフォ」テーブルを作成する事です。各トランザクションが別のテーブルにアクセスする前にその行を更新させてください。そうすると、全てのトランザクションは連続で起こります。直列化ロックは行レベル ロックなので、InnoDB インスタント デッドロック検出アルゴリズムもこの場合機能するという事に注意してください。MySQL テーブル レベル ロックでは、デッドロックを解決する為にタイムアウト法を利用しなければいけません。

- `LOCK TABLES` コマンドを利用するアプリケーション内では、`AUTOCOMMIT=1` であれば MySQL は InnoDB データベース ロックを設定しません。

InnoDB パフォーマンス チューニング ヒント

- InnoDB 内では、長い `PRIMARY KEY` を持つと、その値が全てのセカンダリ インデックス レコードを利用して格納される為、ディスク領域の無駄遣いになります。(詳しくは [InnoDB テーブルとインデックス構造](#) をご確認ください。) もし主キーが長かったら、`AUTO_INCREMENT` カラムを主キーとして作成してください。
- もし Unix `top` ツールか、Windows タスク マネージャが、作業負荷 CPU 使用率が 70% 以下であると表示したら、その作業負荷はおそらくディスクに頼っているでしょう。トランザクション コミットをたくさん作りすぎているか、バッファ プールが小さすぎるという事でしょう。バッファ プールを大きく作成する事も良いですが、物質的メモリの 80% 以上に設定しないでください。
- 複数の変更を1つのトランザクションにまとめてください。InnoDB は、もしトランザクションがデータベースに変更を行うなら、各トランザクション コミットの際にディスクにログをフラッシュしなければいけません。もしディスクが OS を「欺かなければ」、ディスクの回転速度は一般的に最大167 回転/秒で、コミット数も1秒につき167th に制限されます。
- クラッシュが発生した時にいくつかの最新のコミットされたトランザクションの損失を受け入れる事ができるなら、`innodb_flush_log_at_trx_commit` パラメータを0に設定する事ができます。フラッシュが保証されていないくても、InnoDB は1秒に1回ログをフラッシュします。
- バッファ プールと同じ大きさまでログ ファイルを大きくしてください。InnoDB がログ ファイルを一杯に書き込むと、それはチェックポイント内でバッファ プールの変更された内容をディスクに書き込まなければいけません。小さいログ ファイルは多くの不必要なディスク書き込みを引き起こします。大きいログ ファイルの欠点は、復旧時間が長いという事です。
- ログ バッファもとても長く作成してください。(約 8MB)
- もし可変長文字列を格納していたり、カラムが `NULL` 値をたくさん含んでいたら、`CHAR` の代わりに `VARCHAR` データ タイプを利用してください。`CHAR(N)` カラムは文字列が短かったりその値が `NULL` だとしても、データを格納する為にいつも `N` 文字を取ります。小さいテーブルはバッファ プール内によりフィットし、ディスク I/O を減らします。

`row_format=compact` (MySQL 5.1 内のデフォルト InnoDB レコードフォーマット) と、`utf8` や `sjis` のような可変長文字セットを利用する時、`CHAR(N)` は最低でも `N` バイト分の変数領域を占有します。

- GNU/Linux と Unix のいくつかのバージョンでは、Unix `fsync()` コール(InnoDB がデフォルトで利用する物)を利用してファイルをディスクにフラッシュする方法やそれと似た方法は、スピードが大変遅いです。もしデータベースの書き込み性能に満足していなければ、`O_DSYNC` に `innodb_flush_method` パラメータを設定してみるのが良いかもしれません。ほとんどのシステム上で `O_DSYNC` のスピードは遅いかもしれませんが、お使いの物はそうではないかもしれません。
- InnoDB ストレージ エンジンを x86_64 アーキテクチャ(AMD Opteron)の Solaris 10で利用する時、`forcedirectio` オプションを利用して、InnoDB に関連するファイルを格納するのに利用されるファイル システムをマウントする事が重要です。(Solaris 10/x86_64 のデフォルトはこのオプションを利用 しません。) `forcedirectio` 利用に失敗すると、このプラットフォーム上での InnoDB のスピードと性能の深刻な劣化を引き起こします。

Solaris 2.6 以降のリリース版と全てのプラットフォーム(sparc/x86/x64/amd64)で、大きい `innodb_buffer_pool_size` 値と共に InnoDB ストレージ エンジンを利用する時、未加工デバイスや別々のディレクト I/O UFS ファイル システム(マウント オプション `forcedirectio` を利用。 `mount_ufs(1M)` を参照)上に InnoDB データ ファイルとログ ファイルを置く事で、大幅な性能向上を実現する事ができます。Veritas ファイル システム VxFS ユーザは、マウント オプション `convosync=direct` を利用しなければいけません。

MyISAM テーブルに対する物などのようなその他 MySQL データ ファイルはディレクト I/O ファイル システム上に置くべきではありません。実行ファイルやライブラリは、ディレクト I/O ファイル システム上に置いてはいけません。

- InnoDB にデータをインポートする時、MySQL が自動コミットを持っていると各挿入ごとにディスクへのログ フラッシュが要求されるので、自動コミットを持っていない事を確認して下さい。インポート操作の最中に自動コミットを無効にするには、それを `SET AUTOCOMMIT` と `COMMIT` ステートメントで囲んで下さい:


```
SET AUTOCOMMIT=0;... SQL import statements ...
COMMIT;
```

もし `mysqldump` オプション `--opt` を利用すれば、`SET AUTOCOMMIT` と `COMMIT` ステートメントで囲まなくても InnoDB テーブル内にすばやくインポートできるダンプ ファイルを得る事ができます。

- 大量挿入の大きいロールバックに気をつけてください: InnoDB は挿入時にディスク I/O を節約する為に挿入バッファを利用しますが、対応するロールバック内ではそのような仕組みは利用されません。ディスクに頼ったロールバックを実行するには、それと対応する挿入操作の30倍の時間がかかります。データベース処理を停止しても、ロールバックはサーバ起動の際にもう一度起動するので意味がありません。暴走ロールバックを無くす唯一の方法は、ロールバックが CPU に頼り処理が速くなるようにバッファ プールを増やす事、または特別な方法を利用する事です。詳しくは [InnoDB 復旧の強制](#) を参照してください。
- その他のディスクに頼った大きい操作にも気をつけてください。テーブルを空にするには `DROP TABLE` と `CREATE TABLE` を利用し、`DELETE FROM tbl_name` は利用しないでください。
- もし行をたくさん挿入したいのであれば、クライアントとサーバ間の伝達オーバーヘッドを減らす為に 複数行 `INSERT` 構文を利用してください:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

この方法は、InnoDB テーブルだけではなく、全てのテーブルへの挿入に有効なヒントです。

- もし2番目のキー上に `UNIQUE` 制限があったら、インポート操作の最中に一時的に一意性チェックを切り、テーブル インポートのスピードを上げる事ができます:

```
SET UNIQUE_CHECKS=0;... import operation ...
SET UNIQUE_CHECKS=1;
```

大きいテーブルに対しては、InnoDB が2番目のインデックスレコードをバッチ内に書く為にそれ自身の挿入バッファを利用する事ができるので、この作業をするとディスク I/O を大幅に節約する事ができます。データが複製キーを含んでいない事を必ず確認してください。 `UNIQUE_CHECKS` はストレージエンジンが複製キーを無視する事を許可しますが、それを要求はしません。

- もしテーブル内に `FOREIGN KEY` 制約があったら、インポート セッションの持続時間に対して外部キー チェックを切る事でテーブル インポートのスピードを早める事ができます:

```
SET FOREIGN_KEY_CHECKS=0;... import operation ...
SET FOREIGN_KEY_CHECKS=1;
```

大きいテーブルに対しては、これでディスク I/O を大幅に節約する事ができます。

- もし頻繁には更新されないテーブルに自動更新クエリを持っていたら、次のクエリ キャッシュを利用してください:

```
[mysqld]
query_cache_type = ON
query_cache_size = 10M
```

- MyISAM とは違い、InnoDB はそのテーブル内にインデックス濃度を格納しません。代わりに、InnoDB は、起動してから初めてアクセスするテーブルに対して自動更新を算出します。多数のテーブルがあると、この操作はかなり時間がかかります。重要なのは初期テーブル起動操作なので、後ほど利用する時に備えてテーブルを「暖める」為に、`SELECT 1 FROM tbl_name LIMIT 1` のようなステートメントを発行する事で起動後に速やかにこれを利用した方が良いでしょう。

MySQL Enterprise

#####MySQL ##### http://
www.jp.mysql.com/products/enterprise/advisors.html #####

SHOW ENGINE INNODB STATUS と InnoDB モニタ

InnoDB は InnoDB 内部の状態についての情報をプリントする InnoDB モニタを含んでいます。ご自分の SQL クライアントにスタンダード InnoDB モニタのアウトプットをフェッチする為に、いつでも `SHOW ENGINE INNODB STATUS` SQL ステートメントを利用する事ができます。この情報は性能調整をするうえで役立ちます。(もし `mysql` インタラクティブ SQL クライアントを利用しているなら、`\G` を通常のセミコロン ステートメント ターミネータと

置き換えれば、アウトプットはより読みやすくなります。)InnoDB ロック モードの説明に関しては、[InnoDB ロックモード](#) を参照してください。

```
mysql> SHOW ENGINE INNODB STATUS\G
```

InnoDB モニタを利用する別の方法は、それらが `mysqld` サーバのスタンダードアウトプットにデータを定期的に書き込む事を許可する事です。この場合、アウトプットがクライアントに送られる事はありません。スイッチが入ると、InnoDB モニタは大体15秒毎にデータをプリントします。サーバアウトプットは通常 MySQL データディレクトリ内の `.err` ログに導かれます。このデータは性能調整をするうえで役立ちます。Windows 上では、もしアウトプットをエラーログではなくウィンドウに導きたければ、`--console` オプションを利用して、コンソールウィンドウ内のコマンドプロンプトからサーバを起動しなければいけません。

モニタアウトプットは次のタイプの情報を含んでいます:

- 各アクティブトランザクションによって保持されるテーブルとレコードロック
- トランザクションのロック取得待ち
- スレッドのセマフォ待ち
- 保留中のファイル I/O リクエスト
- バッファプール統計
- ほとんどのシステム上で InnoDB のメインスレッドのページおよび挿入バッファマージ活動

スタンダード InnoDB モニタに、`mysqld` のスタンダードアウトプットへの書き込みをさせる為、次の SQL ステートメントを利用してください:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

次のステートメントを発行する事でモニタを停止する事ができます:

```
DROP TABLE innodb_monitor;
```

`CREATE TABLE` 構文は、MySQL の SQL パーサを通してコマンドを InnoDB エンジンに渡す手段に過ぎません:唯一の問題となるのは、テーブル名 `innodb_monitor` と、InnoDB テーブルです。テーブルの構造は InnoDB モニタとまったく無関係です。サーバを一度シャットダウンして、再度サーバを起動しても、モニタは自動的に起動しません。再びモニタを起動するには、まずモニタテーブルをドロップし、そして新しい `CREATE TABLE` ステートメントを発行しなければいけません。(この構文は今後のリリース版で変更される可能性があります。)

`innodb_lock_monitor` を似たような方法で利用する事ができます。これは、大量のロック情報の提供もするという事以外、`innodb_monitor` と同じです。別々の `innodb_tablespace_monitor` は、テーブルスペース内に存在し、テーブルスペース割り当てデータ構造を認証する、作成されたファイルセグメントのリストをプリントします。さらに、InnoDB 内部データディレクトリの内容をプリントする事ができる `innodb_table_monitor` もあります。

InnoDB モニタアウトプットの例:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Status:
=====
030709 13:00:59 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 18 seconds
=====
SEMAPHORES
=====
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the
semaphore: X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 108462, OS waits 37964; RW-excl spins 681824, OS waits
375485
```

LATEST DETECTED DEADLOCK

TRANSACTIONS

841


```

Buffer pool hit rate 999 / 1000
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
-----
END OF INNODB MONITOR OUTPUT
=====

```

アウトプットの注意点:

- もし **TRANSACTIONS** セクションがロック待ちを報告したら、アプリケーションでロックが競合している可能性があります。このアウトプットから、トランザクションのデッドロックの原因を追跡する事もできます。
- **e.SEMAPHORES** セクションには、セマフォを待っているスレッドと、スレッドがスピンまたは相互排他ロック (mutex) や 読み書きロックでの待機を必要とした回数に関する統計情報が報告されます。多数のスレッドがセマフォを待っている場合は、ディスク I/O または **InnoDB** 内部の競合が原因となっている可能性があります。競合の原因としては、多数のクエリを並行して処理しているか、OSでのスレッドのスケジューリングに問題がある事が考えられます。 **innodb_thread_concurrency** をデフォルト値よりも小さく設定すると、このような状況を救う事ができます。
- **BUFFER POOL AND MEMORY** セクションは読み込み、書き込みされたページの統計を提供します。これらの数字から、クエリが現在いくつのデータ ファイル I/O 操作を行っているかを計算する事ができます。
- **ROW OPERATIONS** セクションはメインスレッドが何をしているのかを表します。

InnoDB は、バッファ オーバ フローの可能性を避ける為に、 **stdout** や固定サイズ メモリ バッファではなく、 **stderr** やファイルに診断アウトプットを送ります。副作用として、15秒ごとに MySQL データ ディレクトリ内のス データス ファイルに **SHOW ENGINE INNODB STATUS** のアウトプットが書き込まれます。ファイルの名前は **innodb_status.pid** で、 **pid** はサーバ プロセス ID です。 **InnoDB** は通常のシャットダウンの為にファイルを削除します。もし異常なシャットダウンが起きたら、これらのステータス ファイルが存在する可能性があります、それはマニュアルで削除する必要があります。それらを削除する前に、異常シャットダウンの原因に関する有益な情報を含んでいるかどうかを確認した方が良いでしょう。 **innodb_status.pid** ファイルは設定オプション **innodb_status_file=1** が設定された場合のみ作成されます。

マルチバージョンの実装

InnoDB がマルチバージョン ストレージ エンジンなので、それはテーブル スペース内に古いバージョンの行についての情報を保管しておく必要があります。この情報は、rollback segment (Oracle内の類似データ構造の後)と呼ばれるデータ構造内に格納されます。

InnoDB は内部的にデータベース内に格納された各行に2つのフィールドを追加します。6バイト フィールドは、行を挿入または更新した最後のトランザクションに対して、トランザクション識別子を指示します。また、行内の特別ビットが削除されたとマークするように設定されている点で、削除は内部的に更新として扱われます。各行は、ロール ポインタと呼ばれる7バイトのフィールドも含んでいます。そのロール ポインタは、ロールバック セグメントに書かれた取り消しログレコードを指し示します。もし行が更新されると、取り消しログレコードは行の内容が更新される前に、それを再構築する為に必要な情報を含みます。

InnoDB はトランザクション ロールバック内で必要とされた取り消し操作を実行する為に、ロールバック セグメント内の情報を利用します。それはまた、一貫した読み取りに対する行のこれまでのバージョンを構築する為の情報も利用します。

ロールバック セグメント内の取り消しログは挿入と更新取り消しログに分割されます。挿入取り消しログはトランザクション ロールバックの中でだけ必要であり、トランザクションがコミットしたらすぐに廃棄できます。更新取り消しログも一貫した読み取りの中で利用されますが、それらはトランザクションがなくなり、それに対して、一貫した読み取りの中でデータベース行のこれまでのバージョンを構築する為の更新取り消しログ内の情報を必要とするスナップショットを **InnoDB** が割り当てた後でだけ廃棄できます。

一貫した読み取りだけを発行するトランザクションを含み、トランザクションを定期的にコミットする事を覚えておく必要があります。そうでなければ、 **InnoDB** は更新取り消しログからデータを廃棄する事ができず、ロールバック セグメントが大きく成長しすぎてテーブルスペースを一杯にしてしまいます。

ロールバック セグメント内の取り消しログ レコードの物質的サイズは、一般的にはそれに対応する挿入された、または更新された行よりも小さいです。ロールバック セグメントに必要な領域を計算する為にこの情報を利用する事ができます。

InnoDB マルチ バージョン スキーム内では、行はSQL ステートメントで削除しても、データベースから即座に物理的に削除されません。InnoDB が削除の為に書かれた更新取り消しログ レコードを廃棄する事ができる時だけ、それはデータベースからそれに対応する行とそのインデックスを物理的に削除する事もできます。この削除操作はパージと呼ばれる大変早い操作で、通常削除を行った SQL ステートメントと同じ時間順をとります。

ユーザがテーブル内で大体同じくらいの比率で小さめのバッチの行を挿入、削除するというようなシナリオでは、パージ スレッドが遅れをとり始め、そしてテーブルが大きくなり続け、全てがディスクに頼った状態になり操作がとて遅くなる可能性があります。テーブルがたったの10MBの有効データしか持っていないとしても、たくさんの「停止した」行が10GBを占めるほどにまで大きくなる事があります。そのような場合は、新しい行操作を抑制し、パージ スレッドにもっと多くのリソースを割り当てるのが良いでしょう。innodb_max_purge_lag システム変数は、まさにこの目的の為に存在します。詳細については、[InnoDB 起動オプションとシステム変数](#) をご参照ください。

InnoDB テーブルとインデックス構造

MySQL は .frm ファイル内のテーブルのデータ ディレクトリ情報をデータベース ディレクトリに格納します。これは全ての MySQL ストレージ エンジンに言える事です。しかし全ての InnoDB テーブルもテーブルスペースの内側にある InnoDB 内部データ ディレクトリ内にそれ自体のエントリを持っています。MySQL がテーブルやデータベースをドロップする時、それは .frm ファイルと InnoDB データ ディレクトリ内の対応するエントリの両方を削除する必要があります。これが、単に .frm ファイルを移動するだけで InnoDB テーブルをデータベース間で移動する事ができない理由です。

全ての InnoDB テーブルは、行のデータが格納されている clustered index と呼ばれる特別なインデックスを持っています。もし PRIMARY KEY をテーブル上で定義したら、主キーのインデックスは集合インデックスになります。

もしテーブルに PRIMARY KEY を定義しなければ、MySQL は主キーとして NOT NULL カラムだけを持つ最初の UNIQUE インデックスを選択し、InnoDB がそれを集合インデックスとして利用します。もしテーブル内にそのようなインデックスがなければ、InnoDB は、行が InnoDB がそのようなテーブル内の行に割り当てた行 ID によってオーダされる集合インデックスを内部的に生成します。行 ID は、新しい行が挿入されると単調に増加する6バイトのフィールドです。従って、行 ID によってオーダされた行は物理的に挿入順になっています。

行データはインデックス サーチが導く物と同じページ上にあるので、集合インデックスを通しての行へのアクセスは速いです。テーブルが大きいと、集合インデックス構造は従来の解決法と比較して、ディスク I/O を節約する事が多いです。(多くのデータベース システムでは、データの格納はインデックス レコードからの別のページを利用しています。)

InnoDB では、非集合インデックス(セカンダリ インデックスとも呼ばれる)内のレコードは、行に対して主キー値も含んでいます。InnoDB は、この主キー値を集合インデックスから行を検索するのに利用します。もし主キーが長いと、セカンダリ インデックスがより多くの領域を利用する事に注意して下さい。

InnoDB は、短い方の文字列の残りの長さが領域で詰められたかのように扱われるように、長さの異なる CHAR と VARCHAR 文字列を比較します

インデックスの物理構造

全ての InnoDB インデックスは、インデックスのレコードがツリーのリーフページに格納されるB ツリーです。インデックス ページのデフォルト サイズは16 KB です。新しいレコードが挿入されると、InnoDB はページの1/16 を、将来のインデックス レコードの挿入や更新に備えて空けようとします。

インデックス レコードがシーケンシャル (昇順または降順) に挿入されると、インデックスページの約15/16 までがいっぱいになります。レコードがランダムに挿入された場合は、ページの1/2から15/16までがいっぱいになります。インデックス ページの使用容量が1/2未満になると、InnoDB はインデックス ツリーを縮小してページを解放しようとします。

挿入バッファ

データベース アプリケーションでは、主キーが固有の識別子であり、新しい行が主キーの昇順で挿入される事が一般的です。したがって、集合インデックスへの挿入では、ディスクからのランダムな読み取りを必要としません。

一方、セカンダリ インデックスは通常固有ではなく、セカンダリ インデックスへの挿入は比較的ランダムな順番で行われます。この為、InnoDB で特別な構造が使用される事なく、多数のランダムなディスク I/O が発生します。

固有でないセカンダリ インデックスにインデックス レコードが挿入される場合は、セカンダリ インデックス ページがすでにバッファ プール内にあるかどうか InnoDB によってチェックされます。すでにある場合は、InnoDB によってインデックス ページに直接レコードが挿入されます。バッファ プール内にインデックス ページがなかった場合は、InnoDB によって特別な挿入バッファ構造にレコードが挿入されます。挿入バッファは、その全体がバッファ プール内に収まるように小さくしてある為、このバッファへの挿入はきわめて高速です。

挿入バッファは、データベース内のセカンダリ インデックス ツリーに定期的にマージされます。インデックス ツリーの同じページ上で複数の挿入をマージする事で、ディスク I/O を削減できます。挿入バッファによってテーブルへの挿入速度が最大15倍に高められる事が測定されています。

挿入バッファ マージは、挿入トランザクションがコミットされた 後 まで発生し続けるでしょう。実際、これはサーバがシャットダウンし、再起動する後まで発生し続けます。(InnoDB 復旧の強制 を参照してください。)

挿入バッファ マージは、多くのセカンダリ インデックスが更新される必要があり、多くの行が挿入された時に、何時間かかる可能性があります。この間はディスク I/O が増加してしまいます。それにより、ディスク操作をたくさん行うようなクエリはスローダウンしてしまいます。この他の主なバックグラウンド I/O スレッドは、パージスレッドです。(マルチバージョンの実装 を参照してください。)

適応ハッシュ インデックス

データベースのほぼ全体がメイン メモリ内に収まる場合に、そのデータベースで最も速くクエリを実行するには、ハッシュ インデックスを使用します。InnoDB には、テーブルに定義されたインデックスで実行される検索を監視する構造があります。ハッシュ インデックスの構築がクエリにとって有益であると InnoDB が判断した場合は、自動的にそのインデックスが構築されます。

ただし、ハッシュ インデックスは常にテーブルに存在する B ツリー インデックスを基に構築されるので注意してください。InnoDB は、B ツリー インデックスに対して InnoDB が検出した検索パターンに応じて、任意の長さの B ツリーに定義されたキーの先頭部分に、ハッシュ インデックスを構築できます。ハッシュ インデックスは部分的であってもかまいません。つまり、B ツリー インデックス全体をバッファ プールにキャッシュする必要はありません。InnoDB は、頻繁にアクセスされるインデックス ページへの要求に応じてハッシュ インデックスを構築します。

ある意味では、柔軟なハッシュ インデックスの構造を利用して、InnoDB が十分に余裕のあるメイン メモリに適応する事で、メイン メモリ データベースの構造に近づいています。

物理的なレコード構造

InnoDB テーブルの物理的なレコード構造は、テーブルが作成された時に指定された行フォーマットによって決まります。MySQL 5.1 ではデフォルトで InnoDB が COMPACT フォーマットを利用しますが、MySQL の古いバージョンとの互換性を保持する為には REDUNDANT フォーマットが有効です。

InnoDB ROW_FORMAT=REDUNDANT テーブル内のレコードは、次の特徴を持っています：

- 各インデックス レコードは6バイトのヘッダを含んでいます。このヘッダは、連続するレコードをリンクする為と、行レベル ロックで使用される。
- 集合インデックス内のレコードには、すべてのユーザ定義カラムのフィールドが含まれます。これに加えて、トランザクション ID 用の6バイトのフィールドと、ロール ポインタ用の7バイトのフィールドが1つずつ含まれています。
- ユーザがテーブルに主キーを定義していない場合は、集合インデックスの各レコードは6バイトの行 ID フィールドも含みます。
- セカンダリ インデックスの各レコードには、集合インデックス キーに対して定義されたすべてのフィールドも含まれます。
- レコードには、そのレコードの各フィールドへのポインタも含まれます。レコード内のフィールド長の合計が128 バイト未満の場合はポインタが1バイト、128バイト以上の場合はポインタが2バイトになります。これらのポインタの配列はレコード ディレクトリと呼ばれます。これらのポインタが指し示すエリアはレコードのデータ部分と呼ばれます。

- InnoDB は内部的に固定長フォーマットの **CHAR(10)** のような固定長文字カラムを格納します。InnoDB は **VARCHAR** カラムから後続領域を切り捨てます。
- SQL **NULL** 値はレコード デイレクトリ内で1か2バイトを蓄えておきます。それ以外に、SQL **NULL** 値は可変長カラム内に格納されるとレコードのデータ部分にゼロ バイトを蓄えます。それは固定長カラム内でレコードのデータ部分内にカラムの固定長を蓄えます。**NULL** 値に固定領域を蓄える目的は、そうする事でインデックス ページの崩壊を起こさずに **NULL** からのカラムを非 **NULL** 値に、更新する事ができるという事です。

InnoDB **ROW_FORMAT=COMPACT** テーブル内のレコードには次の特徴があります:

- 各インデックス レコードは可変長ヘッダに先導される5バイトのヘッダを含んでいます。このヘッダは、連続するレコードをリンクする為と、行レベルロックで使用されます。
- レコードヘッダは **NULL** カラムを指示する為にビット ベクタを含んでいます。ビット ベクタは $(n\_nullable+7)/8$ バイトを占めています。**NULL** カラムがこのベクタ内のビット以外の領域を占める事はありません。
- 各非 **NULL** 可変長フィールドに対して、レコードヘッダは1か2バイトのカラム長を含みます。カラムの一部が外部的に格納されたり、最大長が255バイトを超える、または実際の長さが127バイトを超えたりしなければ2バイトだけ必要になります。
- レコードヘッダにはカラムのデータ内容が続きます。**NULL** のカラムは省略されます。
- 集合インデックス内のレコードには、すべてのユーザ定義カラムのフィールドが含まれます。これに加えて、トランザクション ID 用の6バイトのフィールドと、ロール ポインタ用の7バイトのフィールドが1つずつ含まれています。
- ユーザがテーブルに主キーを定義していない場合は、集合インデックスの各レコードは6バイトの行 ID フィールドも含みます。
- セカンダリ インデックスの各レコードには、集合インデックス キーに対して定義されたすべてのフィールドも含まれます。
- InnoDB は内部的に固定長フォーマットの **CHAR(10)** のような固定長、固定幅文字カラムを格納します。InnoDB は **VARCHAR** カラムから後続領域を切り捨てます。
- InnoDB は、後続領域を切り取る事で内部的に UTF-8 **CHAR(n)** カラムを n バイトで格納しようとしています。**ROW_FORMAT=REDUNDANT** 内では、そのようなカラムは $3*n$ バイトを占めます。最小領域 n を蓄える目的は、これによって多くの場合、インデックス ページの崩壊を起こさずにカラムの更新ができるからです。

InnoDB ファイル領域の管理とディスク I/O

InnoDB ディスク I/O

InnoDB は、擬似非同期 I/O を使用します。InnoDB は、多数の I/O スレッドを作成して、先読みなどの I/O 操作に対応します。

InnoDB には2つの先読みヒューリスティックがあります:

- シーケンシャル先読みでは、InnoDB がテーブル スペース内のセグメントへのアクセス パターンがシーケンシャルである事に 気づくと、I/O システムにデータベース ページの読み取りバッチをあらかじめ連絡します。
- 任意の先読みでは、InnoDB がテーブル スペース内のいくつかのスペースがバッファ プールに完全に読み取られている最中である事に 気づくと、I/O システムに残りの読み取りを連絡します。

InnoDB は 二重書き込み と呼ばれる新しいファイル フラッシュ テクニックを利用します。これは、OS のクラッシュや停電後の復旧に安全性を追加し、また **fsync()** オペレーションの必要性を削減する事で ほとんどの種類の Unix の性能を向上させます。

二重書き込みとは、データ ファイルにページを書き込む前に、InnoDB が最初にそれらを二重書き込みバッファと呼ばれる隣接するテーブルスペース エリアに書き込む事を意味します。二重書き込みバッファへの書き込みとフラッシュが完了した後に InnoDB はデータ ファイル内の正しい位置にページを書き込みます。もし OS がページ書き込みの最中にクラッシュしたら、InnoDB は復旧の最中に二重書き込みバッファからページの有効なコピーを見つける事ができます。

ファイル領域管理

設定ファイルに定義するデータ ファイルから、InnoDB のテーブル スペースが構成されます。これらのファイルは、単純に連結されてテーブルスペースになります。ストライピングは使用されません。現時点では、テーブルスペースのどの位置にテーブルが割り当てられるかを定義できません。しかし、新たに作成されるテーブルスペース内では、InnoDB が最初のデータ ファイルから領域を割り当てます。

テーブルスペースは、デフォルト サイズが16 KB のデータベース ページで構成されます。これらのページは、64個の連続するページから成るエクステントにグループ化されます。InnoDB では、テーブルスペース内部の「files」をセグメントと呼びます。これは実際には多くのテーブルスペース セグメントを含んでいる為、「ロールバック セグメント」という名前は、多少誤解を招くおそれがあります。

InnoDB では、各インデックスに2つのセグメントが割り当てられます。1つは B ツリーの非リーフ ノード用、もう1つはリーフ ノード用です。これには、データを含んでいるリーフ ノードで連続性を高める意図があります。

テーブルスペース内でセグメントが大きくなると、InnoDB はそのセグメントに最初の32ページを個別に割り当てます。InnoDB はその後、エクステント全体をセグメントに割り当て始めます。InnoDB では、データの連続性を確保する為に、大きなセグメントに一度に最大4つのエクステントを追加できます。

テーブルスペースには、他のページのビットマップを含んだページがある為、InnoDB テーブルスペース内のいくつかのエクステントは、全体としてではなく個別のページとしてのみセグメントに割り当てる事ができます。

SHOW TABLE STATUS ステートメントを発行してテーブルスペース内の空き領域を照会すると、InnoDB からテーブルスペース内の完全に空いているエクステントが報告されます。InnoDB は、常にいくつかのエクステントをクリーンアップとその他の内部的な用途の為に確保しており、これらのエクステントは空き領域に含まれません。

テーブルからデータを削除すると、InnoDB によって対応する B ツリー インデックスが縮小されます。これによって、他のユーザが開放された領域を利用できるようになるかどうかは、削除のパターンがテーブルスペースの個々のページやエクステントを開放するかどうかによって異なります。テーブルを破棄したり、またはテーブルからすべての行を削除すると、他のユーザに確実に領域が解放されますが、削除された行は、トランザクション ロールバックまたは一貫した読み取りでそのレコードが必要なくなった後のバージ操作で初めて物理的に削除されるという事に注意してください (詳しくは [マルチバージョンの実装](#) を参照してください。)

テーブルのデフラグメント化

テーブルのインデックスでランダムな挿入または削除が行われると、インデックスがフラグメント化される事があります。フラグメント化とは、ディスクでのインデックス ページの物理的な順序が、ページでのレコードのインデックス順とかけ離れている事、またはインデックスに割り当てられた64ページのブロック内に多数の未使用ページがある事を意味します。

フラグメント化の兆候は、テーブルが「必要とする」以上の領域を取るという事です。それがどの程度なのかという事を正確に決めるのは困難です。全ての InnoDB データとインデックスは B ツリー内に格納され、それらの充てん比は50% から100% で異なっています。

その他のフラグメント化の兆候は、このようなテーブルスキャンに「必要」以上の時間がかかるという事です。

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

(前出のクエリの中で、SQL オプチマイザはセカンダリ インデックスではなく、集合インデックスのスキャンの中で「欺かれて」います。)ほとんどのディスクが1秒に付き10から50 MB で読み込む事ができ、それでテーブル スキャンがどれくらいのスピードで起動しなければいけないかを概算します。

それは「null」ALTER TABLE 操作を定期的に実行すればインデックス スキャンの速度を上げる事ができます:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

それによって MySQL はテーブルを再構成します。デフラグ操作を行う別の方法は、テーブルをテキスト ファイルにダンプし、テーブルをドロップし、そしてそれをダンプ ファイルから再ロードする為に [mysqldump](#) を利用する事です。

インデックスへの挿入が常に昇順で行われ、レコードが必ず末尾から削除される場合は、InnoDB のファイル領域管理アルゴリズムによってインデックスのフラグメント化が発生しない事が保証されます。

InnoDB エラー処理

InnoDB でのエラー処理は、必ずしも SQL スタンダードに明記されている通りではありません。スタンダードによると、SQL ステートメントでエラーが発生した場合は、その SQL ステートメントでロールバックを実行するように記述されています。InnoDB では、ステートメントの一部のみ、またはトランザクション全体がロールバックされる事があります。次の項目は、InnoDB でのエラー処理の仕様を説明しています:

- テーブルスペース内でファイル領域を使い果たすと、MySQL の `Table is full` エラーが発生し、InnoDB が SQL ステートメントをロールバックします。
- トランザクション デッドロックが発生すると、InnoDB がトランザクション全体をロールバックします。ロック待ちタイムアウトが起きた場合は、InnoDB は最新の SQL ステートメントだけをロールバックします。

トランザクション ロールバックが、デッドロックやロック待ちタイムアウトによって引き起こされると、それはトランザクション内のステートメントの効果をキャンセルします。しかし、トランザクション開始ステートメントが `START TRANSACTION` か `BEGIN` ステートメントであると、ロールバックはこのステートメントをキャンセルしません。さらなる SQL は、暗黙のコミットを引き起こす `COMMIT`、`ROLLBACK`、またはいくつかの SQL ステートメントが発生するまでの間トランザクションの一部になります。

- ステートメント内で `IGNORE` オプションを指定しなければ、複製キー エラーは SQL ステートメントをロールバックします。
- `row too long error` は SQL ステートメントをロールバックします。
- その他のエラーは主に MySQL のコード レイヤ(InnoDB ストレージ エンジン レベルの上)によって検出され、対応する SQL ステートメントがロールバックされます。ロックは単一 SQL ステートメントのロールバック内でリリースされません。

暗黙のロールバックの最中に、明示的な `ROLLBACK` SQL コマンドの実行の最中と同じように、`SHOW PROCESSLIST` は関連する接続の `State` カラム内で `Rolling back` を表示します。

InnoDB エラーコード

次にある物は、今後直面するであろう主な InnoDB 特有エラーの非消耗リストと、それらがなぜ起きるのか、そしてその問題をどのように解決するのかについての情報です。

- `1005 (ER_CANT_CREATE_TABLE)`

テーブルを作成できません。もしエラー メッセージが `errno 150`を参照していたら、外部キー制約が正しく形成されなかった為にテーブル作成は失敗しました。もしエラー メッセージが `errno -1`を参照していたら、テーブルが内部 InnoDB テーブルの名前と一致するカラム名を含んでいる為にテーブル作成はおそらく失敗したでしょう。

- `1016 (ER_CANT_OPEN_FILE)`

InnoDB テーブルに対する `.frm` ファイルが存在しているとしても、InnoDB データ ファイルから InnoDB を見つける事はできません。詳しくは [トラブルシューティング InnoDB データ ディクショナリ操作](#) を参照してください。

- `1114 (ER_RECORD_FILE_FULL)`

InnoDB はテーブルスペース内で空き領域を使い果たしました。新しいデータ ファイルを追加する為にテーブルスペースを再設定しなければいけません。

- `1205 (ER_LOCK_WAIT_TIMEOUT)`

ロック待ちタイムアウトが失効しました。トランザクションがロールバックされました。

- `1213 (ER_LOCK_DEADLOCK)`

トランザクション デッドロック。トランザクションを返さなければいけません。

- `1216 (ER_NO_REFERENCED_ROW)`

行を追加しようとしているのに親行がなく、外部キー制約が失敗します。まず親行を追加しなければいけません。

- [1217 \(ER_ROW_IS_REFERENCED\)](#)

子供を持つ親行を削除しようとしていて、外部キー制約が失敗します。子供を先に削除する必要があります。

OS エラー コード

OS エラー番号の意味をプリントする為に、MySQL ディストリビューションに付属している [perror](#) プログラムを利用してください。

次のテーブルにはいくつかの主な Linux システム エラー コードが紹介されています。完全なリストに関しては、[Linux ソースコード](#) を参照してください。

- [1 \(EPERM\)](#)
許可されていない操作
- [2 \(ENOENT\)](#)
そのようなファイルやディレクトリは存在しない
- [3 \(ESRCH\)](#)
そのような処理は存在しない
- [4 \(EINTR\)](#)
妨害されたシステム コール
- [5 \(EIO\)](#)
I/O エラー
- [6 \(ENXIO\)](#)
そのようなデバイスやアドレスは存在しない
- [7 \(E2BIG\)](#)
Arg リストが大きすぎる
- [8 \(ENOEXEC\)](#)
Exec フォーマット エラー
- [9 \(EBADF\)](#)
不良ファイル番号
- [10 \(ECHILD\)](#)
子供の処理がない
- [11 \(EAGAIN\)](#)
再度試してください
- [12 \(ENOMEM\)](#)
メモリ不足
- [13 \(EACCES\)](#)
許可が却下された
- [14 \(EFAULT\)](#)

不良アドレス

- 15 (ENOTBLK)
ブロック デバイスが要求された
- 16 (EBUSY)
デバイスがリソースがビジー
- 17 (EEXIST)
ファイルが存在する
- 18 (EXDEV)
クロス デバイス リンク
- 19 (ENODEV)
そのようなデバイスは存在しない
- 20 (ENOTDIR)
ディレクトリではない
- 21 (EISDIR)
ディレクトリである
- 22 (EINVAL)
無効引数
- 23 (ENFILE)
ファイル テーブル オーバーフロー
- 24 (EMFILE)
オープン ファイルが多すぎる
- 25 (ENOTTY)
デバイスの不適切な ioctl
- 26 (ETXTBSY)
テキスト ファイルがビジー
- 27 (EFBIG)
ファイルが大きすぎる
- 28 (ENOSPC)
デバイスに領域が残っていない
- 29 (ESPIPE)
不正シーク
- 30 (EROFS)
読み込み専用ファイル システム

- 31 (EMLINK)

リンクが多すぎる

次のテーブルにはいくつかの主な Windows システム エラー コードが紹介されています。完全なリストに関しては [Microsoft ウェブ サイト](#) を参照してください。

- 1 (ERROR_INVALID_FUNCTION)

不正関数です。

- 2 (ERROR_FILE_NOT_FOUND)

システムは指定されたファイルを見つける事ができません。

- 3 (ERROR_PATH_NOT_FOUND)

システムは指定されたパスを見つける事ができません。

- 4 (ERROR_TOO_MANY_OPEN_FILES)

システムはファイルを開く事ができません。

- 5 (ERROR_ACCESS_DENIED)

アクセスは却下されました。

- 6 (ERROR_INVALID_HANDLE)

ハンドルが無効です。

- 7 (ERROR_ARENA_TRASHED)

ストレージ コントロール ブロックは破壊されました。

- 8 (ERROR_NOT_ENOUGH_MEMORY)

.このコマンドを処理する十分なストレージがありません。

- 9 (ERROR_INVALID_BLOCK)

ストレージ コントロール ブロック アドレスが無効です。

- 10 (ERROR_BAD_ENVIRONMENT)

環境が不適切です。

- 11 (ERROR_BAD_FORMAT)

不適切なフォーマットによってプログラムのロードを行おうとしました。

- 12 (ERROR_INVALID_ACCESS)

アクセス コードが不正です。

- 13 (ERROR_INVALID_DATA)

データが無効です。

- 14 (ERROR_OUTOFMEMORY)

.この操作を完了する十分なストレージがありません。

- 15 (ERROR_INVALID_DRIVE)

システムは指定されたドライブを見つける事ができません。

- 16 (ERROR_CURRENT_DIRECTORY)
ディレクトリを削除できません。
- 17 (ERROR_NOT_SAME_DEVICE)
システムはファイルを別のディスク ドライブに移動できません。
- 18 (ERROR_NO_MORE_FILES)
これ以上ファイルはありません。
- 19 (ERROR_WRITE_PROTECT)
メディアは書き込み保護されています。
- 20 (ERROR_BAD_UNIT)
システムは指定されたデバイスを見つける事ができません。
- 21 (ERROR_NOT_READY)
デバイスの準備ができていません。
- 22 (ERROR_BAD_COMMAND)
デバイスはコマンドを認識しません。
- 23 (ERROR_CRC)
データ エラー (サイクリック リダンダンシー チェック)
- 24 (ERROR_BAD_LENGTH)
プログラムがコマンドを発行しましたがコマンド長が不適切です。
- 25 (ERROR_SEEK)
ドライブはディスク上に特定のエリアやトラックをロケートできません。
- 26 (ERROR_NOT_DOS_DISK)
特定のディスクやディスクットにアクセスできません。
- 27 (ERROR_SECTOR_NOT_FOUND)
ドライブはリクエストされたセクタを見つける事ができません。
- 28 (ERROR_OUT_OF_PAPER)
プリンタの紙切れです。
- 29 (ERROR_WRITE_FAULT)
システムは指定されたデバイスに書き込む事ができません。
- 30 (ERROR_READ_FAULT)
システムは指定されたデバイスから読み込む事ができません。
- 31 (ERROR_GEN_FAILURE)
システムに添付されたデバイスが機能していません。
- 32 (ERROR_SHARING_VIOLATION)
別の処理で利用されている為ファイルにアクセスできません。

- [33 \(ERROR_LOCK_VIOLATION\)](#)
別の処理がファイルの一部をロックした為にファイルにアクセスできません。
- [34 \(ERROR_WRONG_DISK\)](#)
不正なディスクがドライブ内にあります。挿入 %2 (ボリューム通し番号:%3) ドライブ内に %1
- [36 \(ERROR_SHARING_BUFFER_EXCEEDED\)](#)
共有の為に開かれたファイルが多すぎます。
- [38 \(ERROR_HANDLE_EOF\)](#)
ファイルの最後に到達しました。
- [39 \(ERROR_HANDLE_DISK_FULL\)](#)
ディスクが一杯です。
- [87 \(ERROR_INVALID_PARAMETER\)](#)
パラメータが不正です。(もしこのエラーが Windows で起き、サーバ オプション ファイル内で `innodb_file_per_table` を有効にしたら、ライン `innodb_flush_method=unbuffered` をファイルにも追加してください。)
- [112 \(ERROR_DISK_FULL\)](#)
ディスクが一杯です。
- [123 \(ERROR_INVALID_NAME\)](#)
ファイル名、ディレクトリ名、またはボリューム ラベル構文が不適切です。
- [1450 \(ERROR_NO_SYSTEM_RESOURCES\)](#)
要求されたサービスを完了する為のシステム リソースが充分ではありません。

InnoDB テーブル上の制約

- 警告:MySQL システムテーブルを `mysql` データベースの中で `MyISAM` から `InnoDB` テーブルに変換しないでください!これはサポートされていない操作です。もしこれをしてしまうと、バックアップから古いシステム テーブルを復旧するか、`mysql_install_db` スクリプトを利用してそれらを再生成するまで MySQL は再起動しません。
- テーブルが1000 以上のカラムを含む事はできません。
- 内部的な最大キー長は3500バイトですが、MySQL 自体はそれを1024バイトに制限しています。
- `VARCHAR`、`BLOB` そして `TEXT` カラム以外の最大行長は、データベース ページの半分よりも少し短いです。これは、最大行長は約8000バイトであるという事です。`LOB` と `LONGTEXT` カラムは4GB 以下である必要があり、`BLOB` と `TEXT` カラムを含んだ合計行長は4GB 以下でなければいけません。`InnoDB` が行内の `VARCHAR`、`BLOB`、または `TEXT` カラムの最初の768バイトを格納し、残りは別のページに格納されます。
- `InnoDB` は内部的にサイズ65535以上の行をサポートしますが、65535以上のサイズの `VARCHAR` カラムを含む行を定義する事はできません:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
-> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
-> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```
- いくつかの古い OS では、ファイルは2GB 以下でなければいけません。これは `InnoDB` 自体の制限ではありませんが、もし大きいテーブルスペースを要求すると、1つではなく複数の小さいデータ ファイルを利用してそれを設定するか、または大きいデータ ファイルをファイルしなければいけません。

- 結合した InnoDB ログ ファイルのサイズは4GB 以下でなければいけません。
- 最小テーブルスペース サイズは10MB です。最大テーブルスペース サイズは40億データベース ページ(64TB)です。これはテーブルにとっても最大サイズです。
- InnoDB テーブルは FULLTEXT インデックスをサポートしません。
- InnoDB テーブルは空間タイプはサポートしますが、そのインデックスはサポートしません。
- ANALYZE TABLE は、各インデックス ツリーにランダムにダイブし、インデックス濃度概算をそれに応じて更新する事で、インデックス濃度(SHOW INDEX アウトプットの Cardinality カラム内に表示されるように)を決定します。これらは単なる概算である為、ANALYZE TABLE を繰り返す事で別の数値が導かれる事があると覚えておいてください。これによって ANALYZE TABLE の InnoDB テーブル上での速度は速くなりますが、全ての行を考慮する訳ではないので100% 正確とは言えません。

MySQL はインデックス濃度概算を結合最適化でしか利用しません。いくつかの結合が正しい方法で最適化されなければ、ANALYZE TABLE を利用してみると良いでしょう。ANALYZE TABLE が特定のテーブルに十分な値を発行しなかった場合、特定のインデックスの利用を強制する為にクエリと FORCE INDEX を共に利用するか、または MySQL がテーブル スキャンよりもインデックス検索を好む事を保証する為に max_seeks_for_key システム変数を設定する事ができます。システム変数と Optimizer-Related Issues を参照して下さい。

- SHOW TABLE STATUS はテーブルが確保した物理サイズ以外、InnoDB テーブルに、正確な統計を与えません。行カウントは SQL 最適化で利用される単なる概算です。
- InnoDB はテーブル内の行の内部カウントを保持しません。(実際は、マルチ バージョンの為、少々複雑になります。)SELECT COUNT(*) FROM t ステートメントを処理する為に、InnoDB はテーブルのインデックスをスキャンする必要があり、それはもしインデックスが完全にバッファ プールの中に無いのであれば時間がかかります。速いカウントの為に、自分で作成したカウンタ テーブルを利用し、そのカウンタが行う挿入と削除に従ってアップリケーションを更新させなければいけません。テーブルが頻繁に変更されないのであれば、MySQL クエリ キャッシュを利用するのが良い解決法です。もし行カウントの概算で充分であれば、SHOW TABLE STATUS を利用する事もできます。詳しくは InnoDB パフォーマンス チューニング ヒント を参照してください。
- Windows 上では InnoDB はいつもデータベースとテーブル名を小文字で内部的に格納します。Unix から Windows に、または Windows から Unix にデータベースをバイナリ フォーマットで移動するには、データベースとテーブルを作成する時に必ず明示的に小文字の名前を利用する必要があります。
- AUTO_INCREMENT カラムに対しては、テーブルにインデックスを常に定義する必要があり、そしてそのインデックスは AUTO_INCREMENT カラムだけを含んでいなければいけません。MyISAM テーブル内では、AUTO_INCREMENT カラムは複合カラム インデックスの一部であるかもしれません。
- テーブル上であらかじめ指定された AUTO_INCREMENT カラムを初期化している間、InnoDB は AUTO_INCREMENT カラムと関係しているインデックスの最後に専用ロックを設定します。

自動インクリメント カウンタにアクセスする時、InnoDB は、トランザクション全体の最後までではなく、現在の SQL ステートメントの最後まで続く、特別なテーブル ロック モード AUTO-INC を利用します。AUTO-INC テーブル ロックが行われている間は、別のクライアントはテーブルに挿入ができない事に注意してください。InnoDB と AUTOCOMMIT を参照してください。

- MySQL サーバを再起動する時、InnoDB は AUTO_INCREMENT カラムの為に生成されたが、格納はされなかった古い値を再利用するかもしれません。(それは、ロールバックされた古いトランザクション内で生成された値です。)
- AUTO_INCREMENT カラムが値を使い果たした時、InnoDB は BIGINT を -9223372036854775808 に、そして BIGINT UNSIGNED を 1 に切り上げます。しかし、BIGINT 値は64ビットあるので、もし1秒に100万行挿入しようとする、BIGINT がその上限に達するまで3百万年ほどかかるという事を覚えておいてください。その他の全ての整数タイプ カラムを利用すると、複製キー エラーが発生します。これは最も一般的な MySQL 性能であり、特定のストレージ エンジンに関する事ではないので、MyISAM の機能の仕方と似ています。
- DELETE FROM tbl_name はテーブルを再生成しませんが、その代わりに全ての行を1つ1つ削除します。
- 状況によっては、InnoDB テーブルの TRUNCATE tbl_name は DELETE FROM tbl_name にマップされ、AUTO_INCREMENT カウンタをリセットしません。詳しくは TRUNCATE 構文 を参照してください。
- MySQL 5.1 では、もし innodb_table_locks=1 (デフォルト)であれば、MySQL LOCK TABLES 操作は各テーブルに2つのロックを取得します。MySQL レイヤのテーブル ロックに加えて、それは InnoDB テーブル ロックも取得しま

す。MySQL の古いバージョンは InnoDB テーブルロックを取得しませんでした。innodb_table_locks=0 を設定する事で古いバージョンを選択する事ができます。もし InnoDB テーブル ロックが取得されなければ、テーブルのいくつかのレコードが別のトランザクションによってロックされなくても LOCK TABLES が完了します。

- トランザクションによって保持される全ての InnoDB ロックは、トランザクションがコミットされた時が異常終了した時にリリースされます。従って、取得された InnoDB テーブル ロックは直ちにリリースされるので、LOCK TABLES を AUTOCOMMIT=1 モードの InnoDB テーブル上で呼び出す意味はありません。
- トランザクションの最中にさらにテーブルをロックするのが有効な場合があります。残念ながら、MySQL 内の LOCK TABLES は暗黙の COMMIT と UNLOCK TABLES を実行します。LOCK TABLES の InnoDB 変異形は、トランザクションの最中で実行できるように作られています。
- 複製スレーブ サーバを設定する為の LOAD TABLE FROM MASTER ステートメントは InnoDB テーブルには機能しません。次善策は、マスタ上でテーブルを MyISAM に変更し、それをロードし、その後マスタ テーブルを再度 InnoDB に戻すという方法です。もしテーブルが、外部キーなどのような InnoDB 特有の特徴を利用していたら、これは行わないでください。
- InnoDB 内のデフォルト データベース ページ サイズは16KB です。コードを再コンパイルする事で、8KB から64KB の範囲の値に設定する事ができます。univ.i ソース ファイル内で UNIV_PAGE_SIZE と UNIV_PAGE_SIZE_SHIFT の値を更新しなければいけません。
- トリガは現在、転送された外部キー アクションによって有効化されません。

InnoDB トラブルシューティング

次の一般的なガイドラインは、トラブルシューティング InnoDB 問題に適応します:

- 操作に失敗したり、バグの疑いがある時は、.err のサフィックスを持つデータ ディレクトリ内のファイルである、MySQL サーバ エラー ログを確認する必要があります。
- トラブルシューティングを行う時は、通常、mysqld_safe ラッパを通したり、Windows サービスとしてではなく、コマンド プロンプトから MySQL サーバを起動するのが一番良い方法です。そうする事で mysqld がコンソールに何をプリントするのかを確認でき、何が起きているのかをよく知る事ができます。Windows 上では、アウトプットをコンソール ウィンドウに導く為に --console オプションを利用してサーバを起動させる必要があります。
- 問題についての情報を得る為には InnoDB モニタを利用してください。(SHOW ENGINE INNODB STATUS と InnoDB モニタ を参照してください。)もしその問題が性能に関連していたり、サーバがハングアップしているようであれば、InnoDB 内部の状態に関する情報をプリントする為に innodb_monitor を利用しなければいけません。もしその問題がロックに関連していたら、innodb_lock_monitor を利用してください。もしその問題がテーブル作成や別のデータ ディレクトリ操作に関連していたら、InnoDB 内部データ ディクショナリの内容をプリントする為に innodb_table_monitor を利用してください。
- もしテーブルが破損した疑いがあれば、そのテーブル上で CHECK TABLE を起動してください。

MySQL Enterprise

```
MySQL #####InnoDB #####
##### http://www-jp.mysql.com/products/enterprise/advisors.html
#####
```

トラブルシューティング InnoDB データ ディクショナリ操作

テーブルに関する特定の問題点は、MySQL サーバが、データベース ディレクトリ内に格納する .frm ファイル内にデータ ディクショナリ情報を保存する一方、InnoDB もまたテーブルスペース ファイル内にあるそれ自体のデータ ディクショナリに情報を格納するという事です。もし .frm ファイルを移動させたり、サーバがデータ ディクショナリ操作の最中にクラッシュしたりすると、.frm ファイルの場所は、InnoDB 内部データ ディクショナリ内に記録された場所と同期しなくなってしまうかもしれません。

同期していないデータ ディクショナリの兆候は、CREATE TABLE ステートメントが失敗する事です。もしこれが起こったら、サーバのエラー ログを確認する必要があります。もしログが、テーブルは既に InnoDB 内部データディクショナリ内に存在すると報告すると、対応する .frm ファイルを持たない InnoDB テーブルスペース ファイル内に孤立テーブルを持つという事になります。エラー メッセージはこのようなになります:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

エラー メッセージで指示された方法に従えば、孤立テーブルをドロップする事ができます。もしまだ **DROP TABLE** を無事にに利用できないのであれば、その問題は **mysql** クライアント内での名前の完了に原因があるかもしれません。この問題を解決するには、**--skip-auto-rehash** オプションを利用して **mysql** クライアントを開始し、もう一度 **DROP TABLE** を実行してみてください。(名前の完了がオンになっていると、**mysql** は今説明したような問題が存在する時に失敗する、テーブル名のリストを構築しようとしします。)

同期していないデータ ディクショナリのその他の兆候は、**.InnoDB** ファイルを開く事ができないエラーを MySQL がプリントする事です:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

エラー ログ内に次のようなメッセージが表示されます:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

これは **InnoDB** 内に、対応するテーブルを持たない孤立した **.frm** ファイルがある事を意味します。孤立した **.frm** ファイルは、マニュアルで削除する事でドロップできます。

もし MySQL が **ALTER TABLE** 操作の最中でクラッシュしたら、**InnoDB** テーブルスペースの中に孤立したテンポラリテーブルができてしまうかもしれません。**innodb_table_monitor** を利用して、**#sql-...** が誰の名前なのかが表示されたテーブルのリストを確認する事ができます。名前をバックフォートで囲めば、文字 **#** を名前に含んでいるテーブル上で SQL ステートメントを実行する事ができます。したがって、先ほど説明した方法で、そのような孤立テーブルを他の孤立テーブルと同じようにドロップする事ができます。Unix シェル内でファイルをコピーしたりリネームしたりするには、もしファイル名が **#** を含んでいたら、ファイル名を二重引用符で囲まなければいけない事を覚えておいてください。

MERGE ストレージエンジン

MRG_MyISAM エンジンとしても知られている **MERGE** ストレージエンジンは、一つの物として使用する事ができる同一の **MyISAM** テーブルの集まりです。「同一の」というのは、全てのテーブルが同一のカラムとインデックス情報を持つという意味です。カラムのリストされている順番が違っていたり、カラムが完全に一致していなかったり、インデックスの順番が違っていたりすると **MyISAM** テーブルをマージする事はできません。しかし、全ての **MyISAM** テーブルは **myisampack** で圧縮する事ができます。詳しくは **myisampack — 圧縮された、読み取り専用 MyISAM テーブルを作成する** を参照してください。**AVG_ROW_LENGTH**、**MAX_ROWS**、または **PACK_KEYS** などのようなテーブルオプションの違いは問題ではありません。

MERGE テーブルを作成する時、MySQL はディスク上に二つファイルを作成します。そのファイル名はテーブル名で始まり、ファイルタイプを指示する拡張子が付きます。**.frm** ファイルはテーブルフォーマットを格納し、**.MRG** ファイルは一つの物として使用されるべきテーブルの名前を含んでいます。それらのテーブルは、**MERGE** テーブルそのものと同じデータベースになくなくてはならないという訳ではありません。

MERGE テーブル上では、**SELECT**、**DELETE**、**UPDATE**、そして **INSERT** を利用する事ができます。**MERGE** テーブルにマップする **MyISAM** テーブル上に、**SELECT**、**UPDATE**、そして **DELETE** 権限を持たなければいけません。

注記

MERGE テーブルの利用は、次のセキュリティに関する問題を引き起こします。ユーザが **MyISAM** テーブル **t** にアクセスする事ができる時、そのユーザは **t** にアクセスする事ができる **MERGE** テーブル **m** を作成する事ができます。しかし、もしユーザの **t** に対する特権

が後で破棄されるなら、`m`にアクセスする事で`t`にアクセスし続ける事ができます。もしこの作業が好ましくなければ、`MERGE` ストレージエンジンを無効にする為に、新しい--`skip-merge` オプションを使ってサーバをスタートさせる事ができます。このオプションは MySQL 5.1.12.以降で利用できます。

`MERGE` テーブルを`DROP`する時、`MERGE` 仕様だけが削除されます。基礎となるテーブルは影響を受けません。

`MERGE` テーブルを作成するには、どの `MyISAM` テーブルを一つの物として利用したいかを示す`UNION=(list-of-tables)` 条項を指定しなければいけません。`MERGE` テーブルに、`UNION` リストの最初か最後のテーブルに位置する為の挿入が必要であれば、自由に `INSERT_METHOD` オプションを指定する事ができます。テーブルの最初か最後に挿入されるように、`FIRST` か `LAST` 値をそれぞれ使用してください。`INSERT_METHOD` オプションを指定しない場合や、`NO`の値で指定した場合は、`MERGE` テーブルに行を挿入しようとしてもエラーが発生します。

次の例は、`MERGE` テーブルの作成方法を紹介しています。

```
mysql> CREATE TABLE t1 (
-> a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
-> a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
-> a INT NOT NULL AUTO_INCREMENT,
-> message CHAR(20), INDEX(a))
-> ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

基礎となっている `MyISAM` テーブルの中で、`a` カラムは `PRIMARY KEY` としてインデックスされていますが、`MERGE` テーブルの中ではそうではない事に注意してください。`MERGE` テーブルは、基礎となるテーブルに対して一意性を実行することができないので、インデックスはされるのですが、それは `PRIMARY KEY` としてではないのです。

In MySQL 5.1.15 とそれ以降のバージョンでは、`MERGE` テーブルの一部であるテーブルが開かれる時に次のチェックが行われます。もし一つでも適合性チェックに失敗したら、そのテーブルを開く事はできません。それぞれのテーブルに適用される適合性チェックは次のような物です。

- `MERGE` テーブルとまったく同じカラム数がなければいけません。
- `MERGE` テーブルのカラムの順番は、基礎となるテーブルのカラムの順番と一致しなければいけません。
- さらに、親 `MERGE` テーブルと、基礎となるテーブルのカラム仕様がそれぞれ比較されます。それぞれのカラムに対して、MySQL は次のような事を確認します。
 - 基礎となるテーブルのカラムタイプは、`MERGE` テーブルのカラムタイプと等しい。
 - 基礎となるテーブルのカラムの長さは、`MERGE` テーブルのカラムの長さと同じ。
 - 基礎となるテーブルのカラムと `MERGE` テーブルのカラムは `NULL` となり得る。
- 基礎となるテーブルは、少なくともマージテーブルと同量のキーを持つ必要がある。基礎となるテーブルは、`MERGE` よりも多量のキーを持つ事はできません。
- それぞれのキーに対して
 - 基礎となるテーブルのキータイプがマージテーブルのキータイプと等しいかどうかチェックして下さい。
 - 基礎となるテーブルのキー定義中のキーパーツ(例 合成キー内の複数カラム)数が、マージテーブルのキー定義中のキーパーツ数を等しいかどうかチェックして下さい。
 - それぞれのキーパーツに対して
 - キーパーツの長さが等しいかどうか確認して下さい。
 - キーパーツタイプが等しいかどうか確認して下さい。
 - キーパーツの言語が等しいかどうか確認して下さい。

- キーパーツが **NULL** になり得るかどうか確認してください。

MERGE テーブルを作成した後、テーブルグループに対してまとめて機能するクエリを発行する事ができます。

```
mysql> SELECT * FROM total;
```

```
+---+-----+
| a | message |
+---+-----+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+-----+
```

MERGE テーブルを再度別の**MyISAM**テーブルのグループに対してマップするには、次の方法の一つを利用する事ができます。

- **MERGE** テーブルを**DROP**して、もう一度作成してください。
- 基礎となるテーブルのリストを変更する為に**ALTER TABLE tbl_name UNION=(...)** を利用して下さい。

MERGE テーブルは、次のような問題を解決するのに役立ちます。

- ログテーブルを簡単に処理する事ができます。例えば、違う月のデータを別々のテーブルに入力し、**myisampack**を利用してそれらを圧縮し、そしてそれらを一つの物として利用する為に**MERGE**テーブルを作成する事ができます。
- スピードを上げる事ができます。大きいリードオンリーのテーブルを、いくつかの基準に基づいて分割し、個々のテーブルを別々のディスク上に置く事ができます。このような場合は、**MERGE** テーブルを利用した方が、大きいテーブルを利用するより早く処理ができます。
- より有効な検索を行う事ができます。何を探しているかが明らかであれば、いくつかのクエリを分割されたテーブルのうちの一つだけで検索し、それ以外の物に対しては**MERGE** テーブルを使用します。共通のテーブル式を利用する、別々の**MERGE** テーブルをいくつも持つ事もできます。
- より有効な検索を行う事ができます。一つの大きなテーブルを修正するよりも、**MERGE** テーブルに位置づけられた個々のテーブルを修正するほうが簡単です。
- 瞬時にいくつものテーブルを一つの物として位置づけます。**MERGE** テーブルは個々のテーブルのインデックスを利用するので、それ自体のインデックスを整備する必要がありません。その結果、**MERGE** テーブルの集まりは、作成や最位置づけを大変 速いスピードで行うことができます。(インデックスが何も作成されていないとしても、**MERGE** テーブルを作成する時はインデックス定義を指定しなければいけない事を覚えておいてください。)
- 要求に応じて作成した大きなテーブルにテーブル式がある場合、それらの代わりに要求に応じた**MERGE** テーブルを作成する必要があります。この方が、ディスクの場所を節約し、すばやく処理する事ができます。
- オペレーティングシステムのファイルサイズ制限を上回ります。一つ一つの **MyISAM** テーブルはこの制限に制約されますが、**MyISAM** テーブルの集まりは制約されません。
- その一つのテーブルに位置づける**MERGE** テーブルを定義する事によって、**MyISAM** テーブルに仮名や同義語を作成する事ができます。この作業を行う事によって特に顕著なインパクトは現れません。(個々の読み取りに対していくつかの間接的なコールや**memcpy()**コールがあるだけです。)

MERGE テーブルの不都合な点は次のような物です。

- **MERGE** テーブルに対して、同一の**MyISAM** テーブルしか利用する事ができません。
- **MERGE** テーブルの中で、いくつもの**MyISAM** フィーチャーを利用する事はできません。例えば、**MERGE** テーブル上で**FULLTEXT** インデックスを作成する事はできません。(もちろん、基礎となる**MyISAM** テーブル上に**FULLTEXT** インデックスを作成する事はできますが、全文検索で **MERGE** テーブルを検索する事はできません。)
- もし **MERGE** テーブルがテンポラリーでない場合、基礎となる全ての **MyISAM** テーブルもそうでなければいけません。もし **MERGE** テーブルがテンポラリーであれば、**MyISAM** テーブルはテンポラリー、テンポラリーでない物の両方が混在した物になり得ます。

- **MERGE** テーブルはより多くのファイルディスクリプタを利用します。もし、10個のクライアントが10個のテーブルに位置づける**MERGE**テーブルを利用する場合、サーバーは $(10 \times 10) + 10$ 個のファイルディスクリプタを利用します。(10個のクライアントに対して10個のデータファイルディスクリプター、そして、10個のインデックスファイルディスクリプターがクライアントの間で共有されます。)
- キーの読み込みが遅いです。キーを読み込む時、**MERGE** ストレージエンジンは、与えられたキーにどれが一番近いかを確認するために、全ての基礎となるテーブルに対して読み込みを行う必要があります。次のキーを読み込む時、それを見つける為に**MERGE** ストレージエンジンは、読み込みバッファを検索する必要があります。一つのキーバッファを使い切った時に限り、ストレージエンジンは次のキーブロックを読み込む必要があります。この為に `eq_ref` 検索の**MERGE** キーが遅くなりますが、`ref` 検索ほど遅くはありません。`eq_ref` と `ref` の詳細については**EXPLAIN**を使用して、クエリを最適化することをご参照ください。

追加情報

- **MERGE** ストレージエンジンを専門に扱うフォーラムがあります。<http://forums.mysql.com/list.php?93>。

MERGE テーブルの問題点

既に分かっている **MERGE** テーブルの問題点は次のような物です。

- **MERGE** テーブルを別のストレージエンジンに変える為に、**ALTER TABLE** テーブルを利用すると、基礎となるテーブルへの位置付けが失われます。その代わりに、基礎となる **MyISAM** テーブルのレコードが変化したテーブルにコピーされ、そしてそれは指定されたストレージエンジンを利用します。
- **REPLACE** は機能しません。
- **MERGE** テーブルは領域確保をサポートしません。これは、**MERGE** テーブルも、**MERGE** テーブルの基礎となる **MyISAM** テーブルも領域確保ができないという事です。
- 開いている**MERGE** テーブルにマップされた全てのテーブルに対して、**WHERE** 条項、**REPAIR TABLE**、**TRUNCATE TABLE**、**OPTIMIZE TABLE**、また **ANALYZE TABLE**がない **DROP TABLE**、**ALTER TABLE**、**DELETE**を使う事はできません。それをするに**MERGE** テーブルは元のテーブルを参照する事があるので、好ましくない結果をもたらす可能性があります。このような事を防ぐのに一番簡単なのは、**FLUSH TABLES** ステートメントを事前に発行する事によって、全ての **MERGE** テーブルを閉じておくという方法です。
- **MERGE** ストレージエンジンのテーブル位置づけはMySQLの上位レイヤーから隠れているので、**MERGE** テーブルによって使用されている**DROP TABLE** はWindowsでは機能しません。Windowsでは開いているファイルを削除する事はできないので、まず **MERGE** テーブルをフラッシュするか(**FLUSH TABLES**を使う)、テーブルをドロップする前に**MERGE** テーブルをドロップする必要があります。
- **MERGE** テーブルは、そのテーブル全体に対して一意性制約を保持する事ができません。**INSERT**を行う時、データは最初、または最後の **MyISAM** テーブルに入ります。(`INSERT_METHOD` オプションの値による。)MySQLは **MyISAM** テーブルの中では一意の値はそのまま残る事を保障しますが、全てのテーブルのコレクションの中ではそうではありません。
- MySQL 5.1.15以降のバージョンでは、アクセスされた時に**MyISAM** テーブルと **MERGE** テーブルの定義が確認されます。(例 `SELECT` か `INSERT` ステートメントの一部として)。そのように確認する事によって、テーブル定義と親**MERGE** テーブル定義がカラムの順番、タイプ、サイズ、そして関連するインデックスなどに関して一致する事を保障します。テーブル間で違いがあれば、エラーが発生し、ステートメントは失敗します。

テーブルが開かれた時にこのようなチェックが行われるので、カラム変更、カラムの順番やエンジンの入れ替え等を含む定義の変更によってステートメントが失敗します。

MySQL 5.1.14 とそれ以前のバージョンにて

- **MERGE** テーブルを修正、または作成する時、基礎となるテーブルが**MyISAM** テーブル内に存在し、それらが同一の構造である事を保証する為の確認は行われません。**MERGE** テーブルが使用される時、MySQL は全てのマップされたテーブルの行の長さが等しいかどうかを確認しますが、これは絶対間違いの無い物ではありません。異なる**MyISAM**テーブルから **MERGE** テーブルを作成すると、未知の問題に直面する可能性が高いです。
- 同じように、**MyISAM** テーブル以外から **MERGE** テーブルを作成したり、または基礎となるテーブルを**MyISAM** テーブル以外にドロップしたり変更したりすると、後で**MERGE** テーブルを使用する時までエラーは発生しません。

- 基礎となる **MyISAM** テーブルは、**MERGE** テーブルが作成される時には存在する必要がないので、全ての基礎となるテーブルが配置されるまで**MERGE** テーブルを使用しなければ、テーブルをどんな順番で作成しても構いません。また、**MERGE** テーブルが与えられた時間内に使用されない事を保障できるのであれば、基礎となるテーブルのバックアップ、リストア、変更、ドロップ、または再作成などのメンテナンスを行う事ができます。基礎となるテーブルに対して作業を行っている間に、それらを除外する為に **MERGE** テーブルを一時的に再定義する必要はありません。
- **MERGE** テーブル内と、その基礎となるテーブルのインデックスの順番は同一でなければいけません。もし**MERGE** テーブル内で使われるテーブルに**UNIQUE** インデックスを追加する為に**ALTER TABLE**を使用し、また**MERGE** テーブルに一意ではないインデックスを追加するために**ALTER TABLE** を利用するのなら、基礎となるテーブルの中に既に一意ではないインデックスがあれば、テーブルのインデックス順は異なります。(これは、重複キーの速やかな検出を促進する為に**ALTER TABLE** が **UNIQUE** インデックスを一意でないインデックスの前に置くからです。) その結果、そのようなインデックスを持つテーブルに対するクエリは予想外の結果をもたらします。
- もし **ERROR 1017 (HY000): Can't find file: 'mm.MRG' (errno: 2)** と同じようなエラーメッセージが出たら、それは通常いくつかのベーステーブルが**MyISAM**ストレージエンジンを使用していない事を表します。全てのテーブルが**MyISAM**である事を確認してください。
- **MyISAM**にもあるように、**MERGE** テーブルにも 2^{32} (~4.295E+09)行の制限があります。その為、この制限を越える複数の**MyISAM** テーブルをマージする事はできません。しかし、MySQLを **--with-big-tables** オプションで作成すると、行の制限は $(2^{32})^2$ (1.844E+19) 行に増加します。詳しくは**典型的な configure オプション**を参照してください。MySQL 5.0.4バージョンより、全ての標準バイナリはこのオプションで作成されます。

MEMORY (HEAP) ストレージエンジン

MEMORY ストレージエンジンはメモリ上に情報を格納するテーブルを作成します。従来、これらは **HEAP** テーブルと呼ばれていましたが、現在 **MEMORY** テーブルへ名称変更されています。ただし、下位互換性があるため **HEAP** も引き続きサポートします。

各 **MEMORY** テーブルは一つのディスクファイルと関連付けられています。ファイルネームはテーブルの名前で始まり、テーブル定義を格納することを示す為に拡張子 **.frm** をつけます。

MEMORY テーブル作成を明確に指示したい場合は、**ENGINE** テーブルオプションを指定します。

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

その名前からもわかるように、**MEMORY** テーブルはメモリ上に格納されます。ハッシュインデックスを使用し、処理が非常に高速で、テンポラリーテーブルを作成するのに大変便利です。ただし、サーバがクラッシュすると、この **MEMORY** テーブルに格納されたすべてのデータが失われます。テーブル自体は、定義がディスク上に **.frm** ファイルで格納されているので引き続き存在しますが、サーバが再起動したときにはデータは全て失われています。

ここに例として挙げるのは、**MEMORY** テーブルの作成・利用・破棄の方法です。

```
mysql> CREATE TABLE test ENGINE=MEMORY
-> SELECT ip,SUM(downloads) AS down
-> FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY テーブルには次のような特徴があります。

- **MEMORY** テーブルは小さなブロックに割り当てられており、100% 動的ハッシュを使用しています。オーバーフローエリア、または追加の入力スペースは必要ありません。フリーリスト用の余分な領域も必要ありません。削除されたデータはリンクリストに保存され、テーブルに新たにデータを入力する際に再利用されます。**MEMORY** テーブルでは、ハッシュテーブルでよく見られる削除 + 挿入の問題も起こりません。
- **MEMORY** テーブルは、テーブルあたり最大 3 2 インデックスまで、インデックスあたり 1 6 列、インデックスの最大幅は 5 0 0 バイトです。
- The **MEMORY** ストレージエンジンは **HASH** と **BTREE** 両方のインデックスを使って実行します。ここに記すように **USING** 節を追加することによりどちらのインデックスであるかを明確にすることが出来ます。

```
CREATE TABLE lookup
(id INT, INDEX USING HASH (id))
```



```
ENGINE = MEMORY;
CREATE TABLE lookup
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

B-Treeインデックスとハッシュインデックスの一般的特徴は [MySQLにおけるインデックスの使用](#) に記載されています。

- **MEMORY** テーブルに、一意でないキーを使用出来ます (ハッシュテーブルにはあまり見られない機能です)。
- **MEMORY** テーブル上に重複キーをもつハッシュインデックスがある場合(作成されるインデックスは同じ値を持つことが多い)、キーの値に影響を与えるテーブルアップデートと、全ての消去は非常に処理が遅くなります。処理速度がどの程度落ちるかは、重複の度合いに比例します (或いは、インデックス濃度に反比例します。)。 **BTREE** インデックス を使用すれば、この問題は生じません。
- インデックスを張ったカラムが **NULL** 値を含みます。
- **MEMORY** テーブルが固定長のレコードフォーマットを使用しています。
- **MEMORY** テーブルは **BLOB** や **TEXT** カラムをサポートしません。
- **MEMORY** は **AUTO_INCREMENT** カラムのサポートもしています。
- **INSERT DELAYED** のスレッドを **MEMORY** テーブルで使用出来ます。詳しくは [INSERT DELAYED 構文](#) を参照してください。
- **MEMORY** テーブルは全てのクライアント間で共有されます(他の **TEMPORARY** でないテーブルと同様)。
- **MEMORY** テーブルのデータはメモリ内に格納されますが、**MEMORY** テーブルは、クエリ実行中サーバが作成するテンポラリーテーブルとその情報を共有します。しかしながら、**MEMORY** テーブルの種類が異なる二つのテーブルは格納変換の対象となりませんが、一方テンポラリーテーブルについては:
 - テンポラリーテーブルのサイズが大きくなりすぎると、サーバは自動的にテーブル形式を変換してディスクに格納します。最大サイズはシステム変数 `tmp_table_size` から求められます。
 - **MEMORY** テーブルは決してディスク上のテーブルに変換されないようにします。誤った操作を行わないために、`max_heap_table_size` システム変数をセットして **MEMORY** テーブルの最大サイズを設定することが出来ます。個々のテーブルに関しては、**CREATE TABLE** ステートメントの中で **MAX_ROWS** のオプションを指定することも出来ます。
- サーバには、同時に動作する全ての **MEMORY** テーブルを維持するための十分なメモリが必要です。
- **MEMORY** テーブルの不要になったメモリを開放するには、**DELETE** または **TRUNCATE TABLE** を実行するか、**DROP TABLE** を指定してテーブルをまとめて削除します。
- MySQLサーバ起動時に **MEMORY** テーブルにデータを投入したい場合は、`--init-file` オプションを指定します。例えば、ファイルに **INSERT INTO ... SELECT** または **LOAD DATA INFILE** iなどのステートメントを使用し、固定データソースからテーブルを取り込むことが出来ます。[コマンド オプション](#)、[LOAD DATA INFILE 構文](#) 参照。
- レプリケーションを使用している場合、シャットダウン、再起動後のマスタサーバの **MEMORY** テーブルは空になっています。ところが、スレーブはこれを認識しないため、そこにある情報は古いものとなってしまいます。マスタ起動後、マスタ上の**MEMORY** テーブルを初めて使う際、**DELETE** ステートメントがマスタのバイナリログに自動的に書き込まれます。このようにして、スレーブをマスタに再び一致させます。ここで注意することは、この方法によっても、マスタ再起動からその後のテーブル使用時までのインターバル中、スレーブは依然として古いデータを保存し続けているということです。ただし、マスタ起動時 **MEMORY** テーブルにデータを投入する際 `--init-file` オプションを使用することにより、このインターバルタイムはゼロになります。
- **MEMORY** テーブル上、一行に必要なメモリは以下の表現で算出できます。

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) × 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) × 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

ALIGN() は行の長さを **char** ポインタサイズのちょうど倍数にするための数式です。`sizeof(char*)` は 32ビットマシンでは 4、64ビットマシンでは 8 です。

追加情報

- MEMORY ストレージエンジンを専門に扱うフォーラムがあります。 <http://forums.mysql.com/list.php?92>.

EXAMPLE ストレージエンジン

EXAMPLE ストレージエンジンはスタブで実装されており、何の機能も持ちません。このエンジンは、MySQL ソースコードの中で新しいストレージエンジンを作成する方法を説明するための、見本の役割を果たしています。それ自体は、ソフトウェア開発者向のものです。

ソースからMySQLを構築し EXAMPLE ストレージエンジンの機能を有効にするには、`--with-example-storage-engine` オプションの `configure` コマンドを呼び出します。

EXAMPLE エンジンのソースを調べるには、MySQL ソースディストリビューションの `storage/example` ディレクトリを検索します。

EXAMPLE テーブルを作成すると、サーバーはデータベースのディレクトリ上にテーブル形式のファイルを作成します。ファイルはテーブル名から始まり `.frm` 拡張子が付きます。これ以外のファイルは作成されません。このテーブルにはデータは格納されません。検索しても結果は得られません。

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000):Table storage engine for 'test' doesn't » have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

EXAMPLE ストレージエンジンはインデックスをサポートしません。

FEDERATED ストレージエンジン

FEDERATED ストレージエンジンは、レプリケーションやクラスターテクノロジーを使わずに、ローカルサーバ上のリモートMySQLデータベースからデータにアクセスすることを可能にします。FEDERATED テーブルを利用する時、ローカルサーバ上のクエリはリモート(federated)テーブル上で自動的に実行されます。ローカルテーブル上にはデータは何も格納されません。

ソースからMySQLを構築し FEDERATED ストレージエンジンの機能を有効にするには、`--with-federated-storage-engine` オプションの `configure` コマンドを呼び出します。

FEDERATED エンジンのソースを調べるには、MySQL ソースディストリビューションの `sql` ディレクトリを検索します。

FEDERATED ストレージエンジン概要

MyISAMや CSV または InnoDBなどのような標準的なストレージエンジンを利用してテーブルを作成する時、それはテーブル定義とその関連データによって構成されます。FEDERATED テーブルを作成する時、テーブル定義は同じですが、データの物理記憶はリモートサーバ上で処理されます。

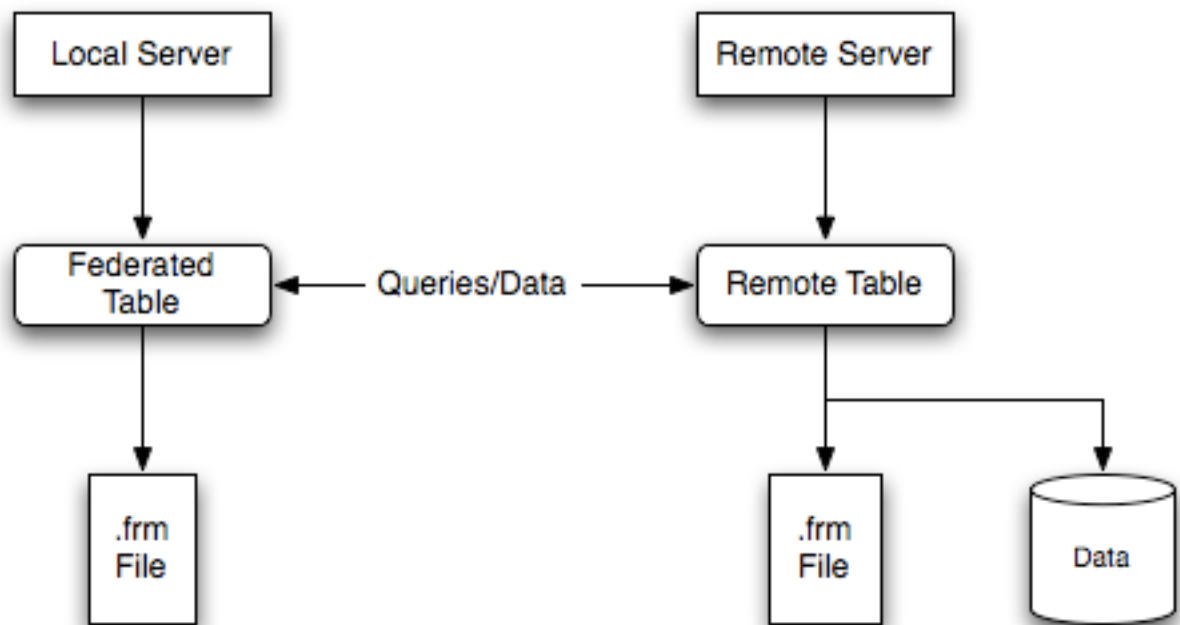
FEDERATED テーブルは2つの要素で成り立っています。

- データベースのあるリモートサーバ、(`.frm` ファイルに格納されている)テーブル定義や関連テーブルのテーブルによって成り立っているものであると言い換える事もできます。リモートテーブルのテーブルタイプは MyISAM や InnoDB を含むリモート `mysqld` サーバにサポートされている全てのタイプと言えるかもしれません。
- リモートサーバ上のテーブル定義とマッチするデータベースを持つ ローカルサーバ テーブル定義は `.frm` ファイルの中に格納されています。しかし、ローカルサーバにはデータファイルはありません。その代わりに、テーブル定義はリモートテーブルを指し示すコネクション文字列を含んでいます。

ローカルサーバ上の FEDERATED テーブルでクエリやステートメントを実行する時、通常は情報をローカルデータファイルから挿入、アップデート、または削除する操作が、その代わりにリモートサーバに送られ、そこでデータファイルをアップデートしたり、適合する行を戻したりします。

FEDERATED テーブルセットアップの基本的な構造は 図13.2 「FEDERATED テーブル構造」に示されています。

図13.2 FEDERATED テーブル構造



ローカルサーバ(SQLステートメントが実行される所)とリモートサーバ(データが物質的に保管される所)間での情報の流れは次のようになっています。

1. SQLコールが局所的に発行される
2. MySQL ハンドラ API (ハンドラフォーマットのデータ)
3. MySQL クライアント API (SQLコールに変換されたデータ)
4. リモートデータベース -> MySQL クライアント API
5. 結果があればそれをハンドラフォーマットに変換する
6. ハンドラ API -> 結果行、または行に影響された部分的な計算

FEDERATED テーブルの作成方法

FEDERATED テーブルを作成するには、これらのステップに従わなければいけません。

1. リモートサーバ上にテーブルを作成します。または、`SHOW CREATE TABLE` ステートメントを利用したりして、存在するテーブルのテーブル定義のノートを作成します。
2. ローカルサーバ上に同一のテーブル定義を使ってテーブルを作成しますが、そのときローカルテーブルをリモートテーブルにリンクさせるための接続情報を追加します。

例えば、リモートサーバ上に次のようなテーブルを作成することができます。

```
CREATE TABLE test_table (  
  id INT(20) NOT NULL AUTO_INCREMENT,  
  name VARCHAR(32) NOT NULL DEFAULT "",  
  other INT(20) NOT NULL DEFAULT '0',  
  PRIMARY KEY (id),  
  INDEX name (name),
```

```
INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1;
```

リモートテーブルに連合されたローカルテーブルを作成するには、二つのオプションがあります。ローカルテーブルを作成し、[CONNECTION](#)を利用して、リモートテーブルに接続するために使う接続文字列(サーバ名とログインパスワードを含んでいる)を指定したり、または、[CREATE SERVER](#) ステートメントを利用して既に作成済の接続を利用する事ができます。

注記

ローカルテーブルを作成する時、`must` はリモートテーブルと同一の定義を持ちます。

CONNECTIONを利用して FEDERATED テーブルを作成する

最初の方法を使うには、[CREATE TABLE](#) ステートメントのエンジンタイプの後ろに [CONNECTION](#) 文字列を指定する必要があります。例:

```
CREATE TABLE federated_table (
  id INT(20) NOT NULL AUTO_INCREMENT,
  name VARCHAR(32) NOT NULL DEFAULT "",
  other INT(20) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  INDEX name (name),
  INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

注記

[CONNECTION](#) はMySQLの以前のバージョンで使われた [COMMENT](#) を置き換えます。

[CONNECTION](#) 文字列は データを物質的に格納する為に利用されるテーブルを含む、リモートサーバに接続する為に要求される情報を含んでいます。接続文字列は、サーバ名、ログイン実績、ポート番号、そしてデータベースとテーブルの情報を指定します。例の中では、リモートテーブルはポート9306を利用しているサーバ `remote_host` 上にあります。名前とポート番号は、リモートテーブルとして利用するリモートMySQLサーバインスタンスのホスト名(またはIPアドレス)とポート番号に適合する必要があります。

接続文字列のフォーマットは次のようになります。

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

場所:

- `scheme` — は認識された接続プロトコルです。この時点では、`mysql` だけが `scheme` 値としてサポートされています。
- `user_name` — 接続用のユーザ名です。このユーザはリモートサーバ上に作成されなければいけません。そして、リモートテーブル上で要求された処理([SELECT](#), [INSERT](#), [UPDATE](#) etc.)を行うために適切な権限を持つ必要があります。
- `password` — (任意) `username`に対応するパスワード
- `host_name` — リモートサーバのホスト名、またはIPアドレス
- `port_num` — (任意) リモートサーバのポート番号。デフォルトは3306です。
- `db_name` — リモートテーブルを保有するデータベースの名前。
- `table_name` — リモートテーブルの名前。ローカルテーブルとリモートテーブルの名前は一致する必要はありません。

接続文字列の例

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

CREATE SERVERを利用してFEDERATED テーブルを作成する

同じサーバ上にいくつかのFEDERATEDテーブルを作成する場合や、FEDERATEDテーブルの作成プロセスをシンプルにしたい場合には、**CONNECTION** 文字列の時と同じように、サーバ接続パラメータを定義するために **CREATE SERVER** ステートメントを利用する事ができます。

CREATE SERVER ステートメントのフォーマットは

```
CREATE SERVERserver_name
FOREIGN DATA WRAPPER wrapper_name
OPTIONS (option ...)
```

server_name は新しいFEDERATEDテーブルを作成する時、接続文字列として利用されます。

例えば、**CONNECTION** 文字列と同一のサーバ接続を作成するには

```
CONNECTION='mysql://root@remote_host:9306/federated/test_table';
```

以下のステートメントを利用する必要があります。

```
CREATE SERVER fedlink
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

この接続を利用した **FEDERATED** テーブルを作成するには、**CONNECTION**キーワードも使うのですが、**CREATE SERVER**ステートメントで使用した名前を指定してください。

```
CREATE TABLE test_table (
  id INT(20) NOT NULL AUTO_INCREMENT,
  name VARCHAR(32) NOT NULL DEFAULT "",
  other INT(20) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  INDEX name (name),
  INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1
CONNECTION='fedlink.test_table';
```

この例の中の接続名には、(**fedlink**) 接続名と、リンクされる(**test_table**)テーブル名が、ピリオドで区切られて含まれています。この方法で明確にテーブル名を指定しないと、代わりにローカルテーブルのテーブル名が使用されます。

CREATE SERVERについては **CREATE SERVER 構文**を参照してください。

CREATE SERVER ステートメントは、**CONNECTION**文字列と同じ引数を受け入れます。**CREATE SERVER** ステートメントは **mysql.servers** テーブルの中のレコードをアップデートします。接続文字列を利用する時、**CREATE SERVER**ステートメントのオプションと、**mysql.servers**テーブルカラムなど、テーブルのパラメータ情報を見てください。次にあるのが、参照用の**CONNECTION** 文字列のフォーマットです。

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

解説	CONNECTION 文字列	CREATE SERVER オプション	mysql.servers カラム
接続スキーム	scheme	wrapper_name	Wrapper
リモートユーザ	user_name	USER	Username
リモートパスワード	password	PASSWORD	Password
リモートホスト	host_name	HOST	Host
リモートポート	port_num	PORT	Port
リモートデータベース	db_name	DATABASE	Db

FEDERATED ストレージエンジン 注意とヒント

FEDERATED ストレージエンジンを使用する時は、次の点に注意する必要があります。

- FEDERATED テーブルは他のスレーブに複製されることもあります。しかし、リモートサーバに接続するためには、スレーブサーバが **CONNECTION** (または **mysql.servers** テーブルのレコード) で定義されたユーザ/パスワードの組み合わせを確実に利用できなければいけません。

次の項目は FEDERATED ストレージエンジンがサポートする、またはしない特徴を指定します。

- 最初のバージョンでは、リモートサーバはMySQLサーバでなければいけませんでしたが、FEDERATED によるその他のデータベースエンジンのサポートは将来追加される可能性があります。
- FEDERATED テーブルを指し示すリモートテーブルはFEDERATED テーブルを通してそこにアクセスしようと試みる前に既に存在していなければいけません。
- 一つのFEDERATED テーブルが他の物を指し示す事は可能ですが、その時ループを作成しないように気をつける必要があります。
- FEDERATED テーブル上で 大量挿入(例えば **INSERT INTO ... SELECT ...** ステートメント上で)などを行う時は、それぞれの選択されたレコードがfederated テーブル上で個別の**INSERT**ステートメントとして扱われるので、他のテーブルタイプで行う時と比べると動作は遅いです。
- トランザクションはサポートされていますが、現在は配信されたトランザクション(XA)サポートされていません。
- FEDERATED エンジンでは、リモートテーブルが変えられているかどうかを知る事はできません。それは、このテーブルはデータベース以外の物には書き込まれる事が無い、データファイルのように機能しなければいけないからです。ローカルテーブル内のデータのインテグリティは、リモートデータベースに何か変更があると破壊される事があります。
- FEDERATED ストレージエンジンは、**SELECT**、**INSERT**、**UPDATE**、**DELETE**、**TRUNCATE**、そしてインデックスをサポートします。**ALTER TABLE** や、**DROP TABLE**以外のデータ定義言語 ステートメントはサポートしません。現在のインプリメンテーションは、用意されたステートメントを利用しません。
- **CONNECTION** 文字列を利用する時、パスワードに '@' を用いることはできません。この制限は、サーバ接続を作成する時に**CREATE SERVER**を利用すれば避ける事ができます。
- **INSERT_ID** と **TIMESTAMP** オプションは、データ提供者に対して用意されている物ではありません。
- FEDERATED テーブルに対して発行された**DROP TABLE** ステートメントは、ローカルテーブルは除去しますが、リモートテーブルは除去しません。
- インプリメンテーションは、**SELECT**、**INSERT**、**UPDATE**、そして **DELETE**を利用しますが、**HANDLER**は利用しません。
- FEDERATED テーブルはクエリキャッシュとは機能しません。
- ユーザー定義領域確保はFEDERATED テーブルの為にはサポートされていません。MySQL 5.1.15からは、そのようなテーブルを作成する事は不可能になりました。

これらの制限のうちのいくつかは、FEDERATED ハンドラの今後のバージョンで撤廃されるでしょう。

FEDERATED ストレージエンジンリソース

次の追加リソースをFEDERATED ストレージエンジンに利用する事ができます。

- FEDERATED ストレージエンジンを専門に扱うフォーラムがあります。<http://forums.mysql.com/list.php?105>

ARCHIVE ストレージエンジン

ARCHIVE ストレージエンジンは、小さい容量の中に大量のデータをインデックスなしで記憶しておく為に使われます。

ARCHIVE ストレージエンジンはMySQLの2極分布に含まれています。ソースからMySQLを構築し、このストレージエンジンの機能を有効にするには、`--with-archive-storage-engine` オプションの `configure` コマンドを呼び出します。

ARCHIVE エンジンのソースを調べるには、MySQL ソースディストリビューションの `storage/archive` ディレクトリを検索します。

このステートメントを使って **ARCHIVE** ストレージエンジンが有効かどうかを調べる事が出来ます。

```
mysql> SHOW VARIABLES LIKE 'have_archive';
```

ARCHIVE テーブルを作成すると、サーバはデータベースのディレクトリ上にテーブル形式のファイルを作成します。ファイルはテーブル名から始まり `.frm` 拡張子が付きます。ストレージエンジンはその他のファイルを作成し、それら全てはテーブル名で始まる名前を持ちます。データとメタデータファイルはそれぞれ `.ARZ` と `.ARM` という拡張子を持ちます。最適化作業の最中に `.ARN` ファイルが現れる事があります。

ARCHIVE エンジンは **INSERT** と **SELECT** をサポートしますが、**DELETE** と **REPLACE** そして **UPDATE** はサポートしません。また、基本的に **ORDER BY** オペレーション、**BLOB** カラム、そして空間データタイプ ([MySQL Spatial Data Types](#) を参照) 以外の全てをサポートします。**ARCHIVE** エンジンは低レベルロッキングを使用します。

MySQL 5.1.6にもあるように **ARCHIVE** エンジンは **AUTO_INCREMENT** カラム属性をサポートします。**AUTO_INCREMENT** カラムは固有、または固有で無いインデックスの両方を持つ事が出来ます。それ以外のコラムにインデックスを作成しようとしてもエラーが発生します。**ARCHIVE** エンジンは、**CREATE TABLE** や **ALTER TABLE** ステートメントの中の **AUTO_INCREMENT** テーブルオプションを新しいテーブルの初期シーケンス値を示す為や、既に存在するテーブルのシーケンス値を再設定する時に、それぞれをサポートします。

MySQL 5.1.6にあるように、**ARCHIVE** エンジンは、**BLOB** カラムが要求されていない時や読み取り中にスキャンがパスした時はそれらを無視します。以前は次の二つのステートメントは同等のコストでしたが、5.1.6では2つ目が1つ目よりも大変効率的になりました。

```
SELECT a, b, blob_col FROM archive_table;
SELECT a, b FROM archive_table;
```

保存: 挿入されると行は圧縮されます。**ARCHIVE** エンジンは **zlib** ロスレスデータ圧縮を使用しています (<http://www.zlib.net/> を参照)。 **OPTIMIZE TABLE** テーブルを分析し、それをもっと小さいフォーマットにまとめる為に (**OPTIMIZE TABLE** を利用する理由に関しては、このセクションの後半を参照して下さい。) このエンジンは **CHECK TABLE** もサポートしています。次のような何種類かの挿入方法が使用されています。

- **INSERT** ステートメントが行を圧縮バッファに押し入れ、そのバッファは必要に応じてフラッシュします。バッファへの挿入はロックによって保護されます。**SELECT** は、**INSERT DELAYED** (必要に応じてフラッシュする) だけが挿入された場合を除いて、強制的にフラッシュを発生させます。詳しくは **INSERT DELAYED** 構文を参照してください。
- 大量挿入は、他の挿入が同時に行われなければ、完了後にだけ見えるようになります。**SELECT** は、ロード中に通常挿入が行われない限り、大量挿入をフラッシュする事はありません。

取り出し: 取り出しの際、要求によって行が解凍されます。行キャッシュはありません。**SELECT** 操作によって完全なテーブルスキャンを実行する事が出来ます。**SELECT** 操作は、現在いくつの行が有効かを見つけその行数を読み取ります。**SELECT** 操作は一貫した読み取りとして実行されます。大量挿入または遅延挿入が使用された場合以外は、挿入の最中の多くの **SELECT** ステートメントが圧縮の質を低下させる事があると覚えておいて下さい。圧縮の質を高めるには、**OPTIMIZE TABLE** か **REPAIR TABLE** を使う事が出来ます。**SHOW TABLE STATUS** によって報告される **ARCHIVE** テーブルの行数はいつも正確です。**OPTIMIZE TABLE** 構文、**REPAIR TABLE** 構文、**SHOW TABLE STATUS** 構文を参照して下さい。

追加情報

- **ARCHIVE** ストレージエンジンを専門に扱うフォーラムがあります。 <http://forums.mysql.com/list.php?112>.

CSV ストレージエンジン

CSV ストレージエンジンはコンマ区切りの値を使ったフォーマットでデータをテキストファイルに保存します。

MySQLを作成する時、このストレージエンジンを使用するには、`configure` する為に `--with-csv-storage-engine` オプションを使います。

ソースからMySQLを構築し CSV ストレージエンジンの機能を有効にするには、`--with-csv-storage-engine` オプションの `configure` コマンドを呼び出します。

CSV エンジンのソースを調べるには、MySQL ソースディストリビューションの `storage/csv` ディレクトリを検索します。

CSV テーブルを作成すると、サーバはデータベースのディレクトリ上にテーブル形式のファイルを作成します。ファイルはテーブル名から始まり `.frm` 拡張子が付きます。このストレージエンジンはデータファイルも作成します。その名前はテーブル名で始まり `.CSV` という拡張子を持ちます。このデータファイルはシンプルなテキストファイルです。データをテーブルに保存する時、ストレージエンジンはコンマで区切られた値のフォーマットでそのデータをデータファイルに保存します。

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
+-----+-----+
| i | c |
+-----+-----+
| 1 | record one |
| 2 | record two |
+-----+-----+
2 rows in set (0.00 sec)
```

MySQL 5.1.9から、CSVテーブルを作成するとテーブルの状態とそのテーブル内の行数を記憶する、対応メタファイルも同時に作成されるようになりました。このファイルの名前は `CSM` という拡張子のついたテーブル名と同じです。

前回のステートメント実行時に作成されたデータベースディレクトリの中の `test.CSV` ファイルを分析すると、その中身はこうなっているでしょう。

```
"1","record one"
"2","record two"
```

このフォーマットはマイクロソフトエクセルやスターオフィスなどのような表計算アプリケーションによって読み取る事も、書き込む事も可能です。

CSVテーブル修正と確認

バージョン5.1.9で紹介する機能性

CSVストレージエンジンは、損傷したCSVテーブルを検証し、可能であればそれを修正する `CHECK` コマンドと `REPAIR` コマンドをサポートします。

`CHECK` コマンドを実行する時には、正しいフィールド区切り、エスケープフィールド（一致した引用符や欠落した引用符）、テーブル定義や対応CSVメタファイルとの比較フィールド数などを探す事によって、そのCSVファイルの正当性チェックが行われます。最初に発見された不正な行がエラーを報告します。正当なテーブルをチェックする事によって、下記のようなアウトプットが作成されます。

```
mysql> check table csvtest;
+-----+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest | check | status | OK |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

破損したテーブルを抑制するとフォルトが戻ります。

```
mysql> check table csvtest;
+-----+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest | check | error | Corrupt |
```



```
+-----+-----+-----+
1 row in set (0.01 sec)
```

もし抑制に失敗するとそのテーブルにはクラッシュ（破損）の印が付きます。一度クラッシュの印がつくと、次に **CHECK** を実行した時や **SELECT** 命令を実行した時に、そのテーブルは自動的に修正されます。破損状態や新しい状態は **CHECK** を実行した時に表示されます。

```
mysql> check table csvtest;
+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+
| test.csvtest | check | warning | Table is marked as crashed |
| test.csvtest | check | status | OK |
+-----+-----+-----+
2 rows in set (0.08 sec)
```

テーブルを修正するには **REPAIR** を使う事が出来ます。この機能は存在するCSVデータから可能な限りの有効な行をコピーし、存在するCSVファイルと復元された行を置き換えます。破損データを越えた行は全て失われます。

```
mysql> repair table csvtest;
+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+
| test.csvtest | repair | status | OK |
+-----+-----+-----+
1 row in set (0.02 sec)
```

警告

修正の最中は、CSVファイルから最初のダメージを受けた行までだけが、新しいテーブルにコピーされるという事を覚えておいて下さい。それ以外の行は、有効であっても全て除去されます。

CSV制限

重要**CSV** ストレージエンジンはインデックスをサポートしません。

CSV ストレージエンジンを使ったテーブルの領域確保はサポートされません。MySQL 5.1.12からは、分割された**CSV** テーブルを作成する事は不可能になりました。（Bug #19307を参照して下さい。）

BLACKHOLE ストレージエンジン

BLACKHOLE ストレージエンジンは データを受け入れますがそれを格納せずに捨ててしまう「black hole」として機能します。検索しても結果は得られません。

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

ソースからMySQLを構築し **BLACKHOLE** ストレージエンジンの機能を有効にするには、`--with-blackhole-storage-engine` オプションの `configure` コマンドを呼び出します。

BLACKHOLE エンジンのソースを調べるには、MySQL ソースディストリビューションの `sql` ディレクトリを検索します。

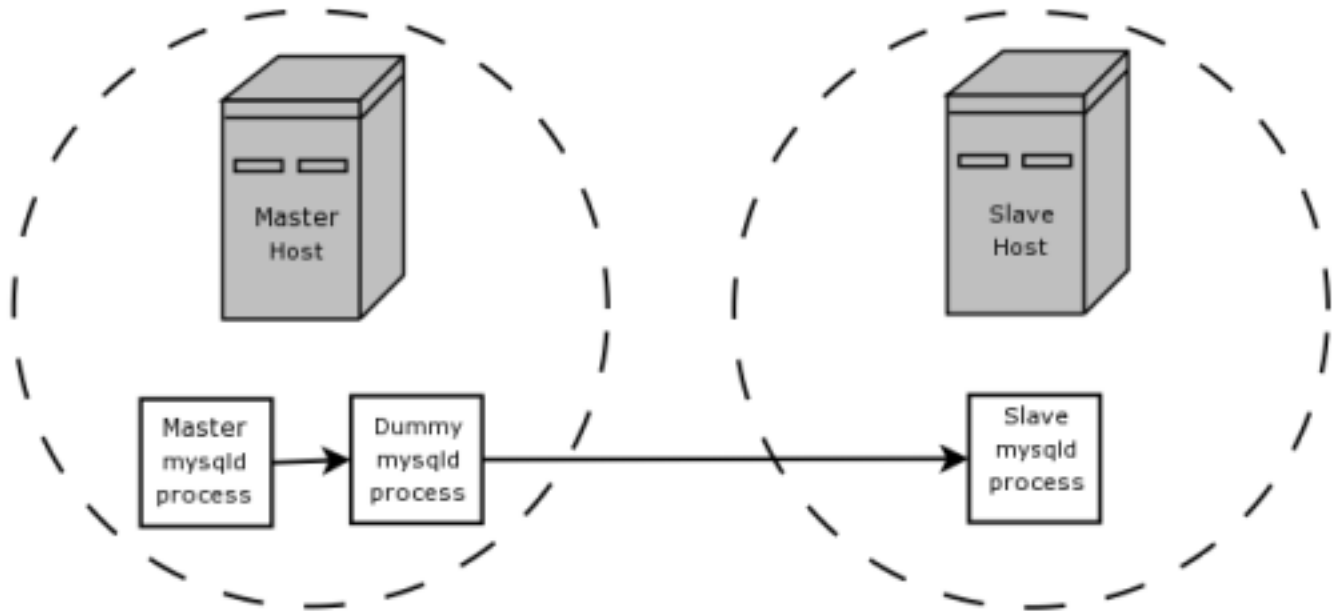
BLACKHOLE テーブルを作成すると、サーバーはデータベースのディレクトリ上にテーブル形式のファイルを作成します。ファイルはテーブル名から始まり `.frm` 拡張子が付きます。テーブルに関連する他のファイルはありません。

BLACKHOLE ストレージエンジンは全てのインデックスをサポートします。それは、インデックスデクラレーションをテーブル定義に含む事ができるという事を意味します。

このステートメントを使って **BLACKHOLE** ストレージエンジンが有効かどうかを調べる事が出来ます。

```
mysql> SHOW VARIABLES LIKE 'have_blackhole_engine';
```

BLACKHOLE テーブルへ挿入してもデータは格納されませんが、 バイナリログが有効な場合、SQLステートメントはログされます。(そしてスレーブサーバーに複製されます。)これはリピータやフィルタメカニズムに有用です。例えば、アプリケーションがスレーブサイドフィルタルールを要求したとします。しかし、スレーブに全てのバイナリログデータを転送すると、すぐにトラフィックが混み合ってしまいます。そのような場合は、次に描かれているようにマスタホスト上に、デフォルトストレージエンジンが**BLACKHOLE**である「dummy」スレーブプロセスをセットアップする事が可能です。



マスタはそれ自体のバイナリログに書き込みを行います。「dummy」 **mysql** プロセスは、希望の **replicate-do-*** と **replicate-ignore-*** ルールの組み合わせを適応させスレーブとして機能し、それ自体の新しくフィルタされたバイナリログを書きます。(詳しくは [レプリケーションのオプションと変数](#) をご確認ください。) このフィルタされたログはスレーブに提供されます。

ダミープロセスはデータの保存は行いませんので、レプリケーションマスタホストに **mysql** プロセスを追加で実行させると、プロセスオーバーヘッドを被る事があります。このタイプのセットアップは追加のレプリケーションスレーブを使って繰り返す事ができます。

BLACKHOLE テーブルの **INSERT** トリガは予想通りに機能します。しかし、**BLACKHOLE** テーブルはデータの格納ができないので **UPDATE** と **DELETE** トリガは有効ではありません。行がないので、トリガ定義中の **FOR ANY ROW** 節は適用しません。

その他の **BLACKHOLE** ストレージエンジンの利用方法は次のようなものです。

- ダンプファイル構文の検証
- バイナリログの有無にかかわらず **BLACKHOLE** を利用して性能を比較する事によって、バイナリログからのオーバーヘッド計算が可能になりました。
- **BLACKHOLE** は基本的に「no-op」ストレージエンジンなので、ストレージエンジン自体には関係ない性能障害を見つけることができます。

MySQL 5.1.4にあるように、実行されたトランザクションはバイナリログに書き込まれるが、ロールバックトランザクションは書き込まれない、という意味では **BLACKHOLE** エンジンはトランザクション認識型です。

第14章 MySQL Cluster

目次

MySQL Cluster の概要	873
基本的な MySQL Cluster のコンセプト	875
MySQL Cluster ノード、ノード グループ、レプリカ、およびパーティション	876
簡単なマルチ コンピューターの手引き	878
ハードウェア、ソフトウェア、およびネットワークの構築	880
マルチ コンピュータのインストール	881
マルチ コンピュータの設定	883
最初の起動	885
サンプル データのローディングとクエリの実行	886
安全なシャットダウンと再起動	889
MySQL Cluster の設定	889
ソースコードによる MySQL Cluster の構築	890
ソフトウェアのインストール	890
MySQL Cluster の簡単なテストの設定	890
設定ファイル	892
クラスタ設定パラメータの概要	917
ローカル チェックポイントのパラメータの設定	923
MySQL Cluster のアップグレードおよびダウンロード	924
クラスタのローリング再起動の実行	924
クラスタのアップグレードおよびダウングレードの互換性	926
MySQL Cluster のプロセス管理	929
MySQL Cluster のための MySQL Server プロセスの使用法	929
ndbd、ストレージ エンジン ノード プロセス	930
ndb_mgmd、マネジメント サーバードプロセス	931
ndb_mgm、マネジメント クライアント プロセス	932
MySQL Cluster プロセスのコマンド オプション	932
MySQL Cluster の管理	935
MySQL Cluster 起動フェーズ	935
マネジメント クライアントのコマンド	937
MySQL Cluster で生成されたイベント レポート	937
単一ユーザーモード	944
MySQL Cluster のオンライン バックアップ	944
クラスタ バックアップの概念	944
バックアップを作成するためのマネジメント クライアントの使用	945
クラスタのバックアップの復旧方法	946
クラスタ バックアップの設定	948
バックアップのトラブルシューティング	949
クラスタ ユーティリティ プログラム	949
ndb_config — NDB 設定情報の抽出	950
ndb_delete_all — NDB テーブルからのすべての行を削除する	952
ndb_desc — NDB テーブルの説明	953
ndb_drop_index	954
ndb_drop_table	954
ndb_error_reporter	955
ndb_print_backup_file	955
ndb_print_schema_file	955
ndb_print_sys_file	956
ndb_select_all	956
ndb_select_count	958
ndb_show_tables	958
ndb_size.pl — NDBCluster サイズ仕様エスティメーター	959
ndb_waiter	961
MySQL Cluster レプリケーション	962

略語と記号	963
仮定条件と一般要件	964
既知の問題	964
レプリケーション スキーマおよびテーブル	965
レプリケーションにクラスタを準備する	967
レプリケーションの開始 (シングル レプリケーション チャンネル)	968
2 つのレプリケーション チャンネルを使用する	969
MySQL Cluster にフェールオーバーを導入する	970
MySQL Cluster のレプリケーションによるバックアップ	970
MySQL Cluster ディスク データ ストレージ	975
MySQL Cluster での高速インターコネクトを使用する	978
SCI ソケットを使用するための MySQL Cluster の設定	978
Cluster インターコネクトの理解する	981
MySQL Cluster の既知の制限	983
MySQL Cluster 開発ロードマップ	988
MySQL 5.1 における MySQL Cluster の変更点	988
MySQL Cluster の用語	989

コンパイル時の不手際のため、MySQL 5.1.12のバイナリ配布にはNDBクラスタやパーティショニングは含まれていませんでした。ご不便をお掛けし恐縮です。バージョン5.1.14.へ更新してください。ソースからコンパイルする場合には、`--with-ndbcluster`、`--with-partition`オプションとともに`configure`を実行して下さい。

MySQL Cluster は MySQL の高可用性、高冗長性バージョンで分散型コンピュータ環境に採用されています。MySQL Cluster はクラスタで数台の MySQL サーバーを動作させるための **NDB Cluster** ストレージ エンジンを使用しています。このストレージ エンジンは MySQL 5.1 バイナリ リリースおよび最新の Linux 分散型互換の RPM で利用できます。

MySQL Cluster は現在利用可能で以下のプラットフォームでサポートされています。

- Linux : x86、AMD64、EMT64、s/390、PPC、Alpha、SPARC、UltraSparc
- Solaris:SPARC、UltraSparc、x86、AMD64、EMT64
- BSD (FreeBSD、NetBSD、OpenBSD):x86、AMD64、EMT64、PPC
- Mac OS X:PPC
- HP-UX:PA-RISC
- Tru64:Alpha
- OpenVMS:Alpha
- IRIX:MIPS
- Novell Netware:x86
- QNX Neutrino:x86
- SCO OpenServer、OpenUnix、UnixWare:x86

特定のオペレーティング システムのバージョンの組み合わせ、オペレーティング システムの流通、およびハードウェア プラットフォームでの MySQL Cluster の適格なサポート レベルについては、MySQL AB のウェブ サイトの MySQL のサポートチームが維持している<http://www.mysql.com/support/supportedplatforms/cluster.html> を参照してください。

MySQL Cluster は現在 Microsoft Windows ではサポートされて いません。弊社では MySQL がサポートする Windows を含むすべてのオペレーティング システムでクラスタを利用できるように目指しており、ここに掲載した情報を開発の進捗に応じて更新して参ります。

本章では現在進行中の作業について説明し、その内容は MySQL Cluster の開発の進捗に応じて改訂します。MySQL Cluster に関する詳細は MySQL AB のウェブサイト <http://www.mysql.com/products/cluster/> を参照してください。

その他のリソース

- クラスタに関するよく出る質問に関する回答は [MySQL 5.1 FAQ — MySQL Cluster](#) に掲載してあります。
- MySQL Cluster のメーリング リスト : <http://lists.mysql.com/cluster>.
- MySQL Cluster フォーラム : <http://forums.mysql.com/list.php?25>.
- 多くの MySQL Cluster ユーザーや MySQL Cluster の開発者の中にはクラスタの経験をブログし、それらの経験を [PlanetMySQL](#) で共有できるようになっています。
- MySQL Cluster を初めてお使いになる場合には、弊社の開発者ゾーンの記事 [How to set up a MySQL Cluster for two servers](#) が役に立つかと思います。

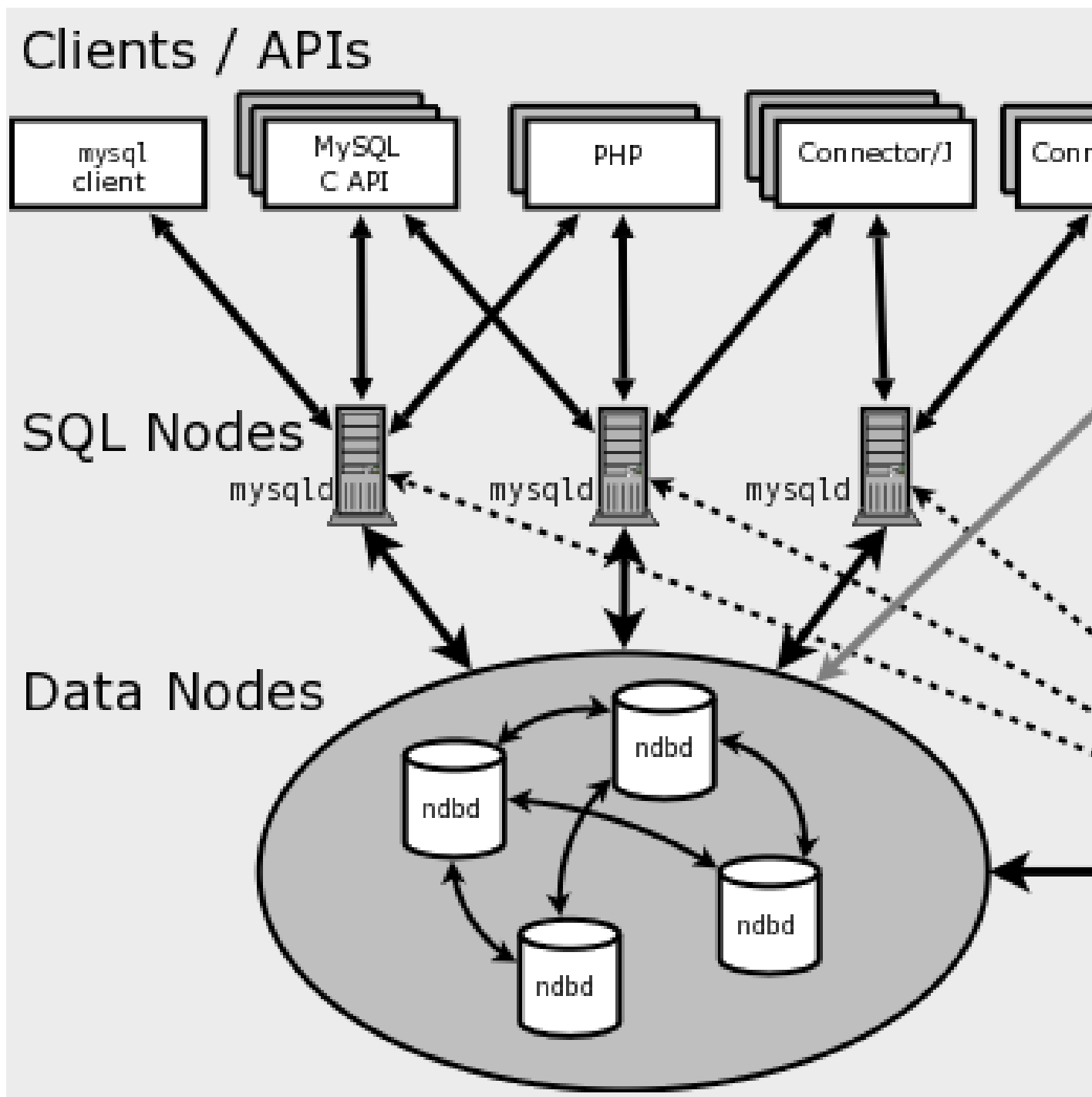
MySQL Cluster の概要

MySQL Cluster は非共有システム (shared-nothing system) での in-memory データベースのクラスタを可能にするテクノロジーです。非共有システムのアーキテクチャでは非常に廉価なハードウェアで最低限のハードウェアあるいはソフトウェアの特殊仕様でシステムを構築できます。

MySQL Cluster を使用することで一極集中型不具合 (シングル ポイント オブ ファイリユア) を回避できます。これを実現するため、各コンポーネントはそれぞれ自身のメモリとディスクを持ち、ネットワーク共有、ネットワークファイルシステム、および SAN などの共有ストレージに使用は推奨もしくはサポートしていません。

MySQL Cluster は **NDB** と呼ばれる in-memory のクラスタ ストレージ エンジンで標準の MySQL サーバーを統合しています。弊社の説明資料では、**NDB** という用語はストレージ エンジンに特化した設定を意味し、そこでは「MySQL Cluster」は MySQL および **NDB** ストレージ エンジンの組み合わせを意味しています。

MySQL Cluster は MySQL サーバー、データノード、マネジメント サーバー、および (多分に) 特定のデータ アクセス プログラムを含む 1つあるいはそれ以上のプロセスをそれぞれ動作させるコンピュータの組み合わせで構築されます。以下にクラスタでのこれらのコンポーネントの関係を示します。



これらのすべてのプログラムは一緒に動作して MySQL Cluster を構築します。データが **NDB Cluster** ストレージエンジンに保持されると、テーブルはデータノードに保持されます。それらのテーブルはクラスタのすべての他の MySQL サーバーから直接アクセスできます。このように、クラスタに保持された給料計算のアプリケーションでは、一つのアプリケーションで一人の社員の給料を更新すると、このデータをクエリする他のすべての MySQL サーバーで瞬時のこの変更を表示できます。

MySQL Cluster のデータノードに保持されたデータはミラーされます。クラスタはトランザクションの状態を見失うことによる少数のトランザクションの失敗以外影響を受けずに個々のデータノードの不具合を処理します。なぜなら、トランザクション関係のアプリケーションはトランザクション処理がその目的にあるため、これは問題の根源にはなり得ないからです。

基本的な MySQL Cluster のコンセプト

NDB は高可用性およびデータ堅牢性を提供する in-memory のストレージ エンジンです。

NDB ストレージ エンジン是一定のフェールオーバーおよび負荷分散の環境で構築できますが、ストレージ エンジン をクラスタレベルで始めるのが無難です。MySQL Cluster の **NDB** ストレージ エンジンは完全なデータセットを含み、クラスタ内の他のデータのみに依存します。

MySQL Cluster のクラスタ部分は現在 MySQL サーバーとは個別に設定されています。MySQL Cluster では、クラスタの各部を **ノード** と呼んでいます。

注:多くの説明書では、用語の「ノード」はコンピュータに意味に使われていますが、MySQL Cluster の説明ではノードはプロセスを意味します。一台のコンピュータでのノードは幾つも操作できるので、弊社では クラスタ ホスト を用語に使用しています。

(しかし、MySQL は現在は生産環境の設定ではまだ一台のコンピューターで複数のデータノードをサポートしていませんのでその点ご注意ください。詳細は、[Issues exclusive to MySQL Cluster \[986\]](#)を参照してください。)

クラスタ ノードは 3 種類あり、最小の MySQL Cluster の設定では、最低 3 台のノードを使用し、以下の種類になります。

- マネジメント ノード (MGM ノード) : この種のノードはの役割は MySQL Cluster 内の他のノードを管理し、設定データなどの機能を実行し、ノードを起動あるいは停止したりバックアップなどを行います。この種のノードは他のノードの設定を管理するため、この種のノードは他のノードより先に起動する必要があります。MGM ノードはコマンド `ndb_mgmd` で起動します。
- データノード:この種のノードはクラスタのデータを保持します。レプリカにフラグメントを乗算した分の多くのデータノードがあります。例えば、2 つのレプリカがあれば、各レプリカには 2 つのフラグメントがあるため、4 つのデータノードが必要になります。1 つ以上のレプリカを持つ必要はありません。データノードはコマンド `ndbd` で起動します。
- SQL ノード:これはクラスタ データにアクセスするノードです。MySQL Cluster の場合、SQL ノードは **NDB Cluster** ストレージ エンジンを使用した従来の MySQL サーバーです。SQL ノードは一般的には `mysqld --ndbcluster` または、`my.cnf` に追加した `ndbcluster` オプションとで `mysqld` を使用して起動します。

SQL ノードは実際は単に API ノードの特化版でクラスタ データにアクセスするアプリケーションを意味します。API ノードの一つの例としてはクラスタのバックアップの保持に使用される `ndb_restore` ユーティリティがあります。**NDB API** を使用してそのようなアプリケーションを記述できます。

重要生産環境では 3 台のノード設定の使用を期待するのは現実的ではありません。そのような設定は冗長性に欠け、MySQL Cluster の高可用性の機能を活かすには、複数のデータノードおよび SQL ノードを使用する必要があります。複数のマネジメント ノードの使用も強くお勧めします。

MySQL Cluster のノード、ノードグループ、レプリカ、およびパーティションの関係に関する概説は、[MySQL Cluster ノード、ノードグループ、レプリカ、およびパーティション](#)を参照してください。

クラスタの構築にはクラスタの各個々のノードの設定およびノード間の個々の通信リンクの設定が含まれます。MySQL Cluster は現在データノードがプロセッサの性能、メモリ スペース、および帯域において均一になるよう開発しています。さらに、一元管理の設定を提供するために、クラスタのすべての設定データは全体として一つの設定ファイルに格納されています。

マネジメント サーバー (MGM ノード) はクラスタの設定ファイルおよびクラスタ ログを管理します。クラスタの各ノードはマネジメント サーバーから設定データを取り出し、マネジメント サーバーが常駐する場所を決める方法を要求します。データノードで何か珍しいイベントが発生すると、ノードはこれらのイベントの情報をマネジメント サーバーに伝送し、マネジメント サーバーがその情報をクラスタ ログに書き込みます。

さらに、クラスタのクライアント プロセスおよびアプリケーションはどんな数でも利用できます。これらには 2 種類あります。

- 標準の MySQL クライアント:標準 (非クラスタ) の MySQL 以外の違いは MySQL Cluster でもありません。換言すれば、MySQL Cluster は PHP、Perl、C、C++、Java、Python、Ruby など記述された既存の MySQL アプリケーションからアクセスできます。

- マネジメント クライアント:これらのクライアントはマネジメント サーバーと接続してノードを厳かに始動・停止し、メッセージの追跡 (デバッグ バージョンのみ) を始動・停止し、ノードのバージョンおよび状態を表示し、バックアップを始動・停止するなどのコマンドを提供します。

MySQL Cluster ノード、ノード グループ、レプリカ、およびパーティション

この項では MySQL Cluster によるストレージ用のデータの分割または複製方法について説明します。

この件で特に理解していただきたいのは以下のコンセプトで、簡単な定義と共にここに一覧にまとめました。

- (データ) ノード:`ndbd` プロセスはレプリカ を保持します。つまり そのノードが構成要素であるノード グループに割り当てられたパーティション (以下参照) のコピーです。

各データノードは個別のコンピューターに配置される必要があります。1 台のコンピューターで複数の `ndbd` プロセスをホストできますが、そのような設定はサポートしていません。

用語の「ノード」や「データノード」のは一般的には `ndbd` プロセスを意味する場合にはどちらも使用できますが、マネジメント (MGM) ノード (`ndb_mgmd` プロセス) および SQL ノード (`mysqld` プロセス) はこの説明ではそのように指定されます。

- ノード グループ:ノードグループは 1 つ以上のノードで構成され、パーティション、あるいはレプリカ (次のアイテム参照) のセットを保持します。

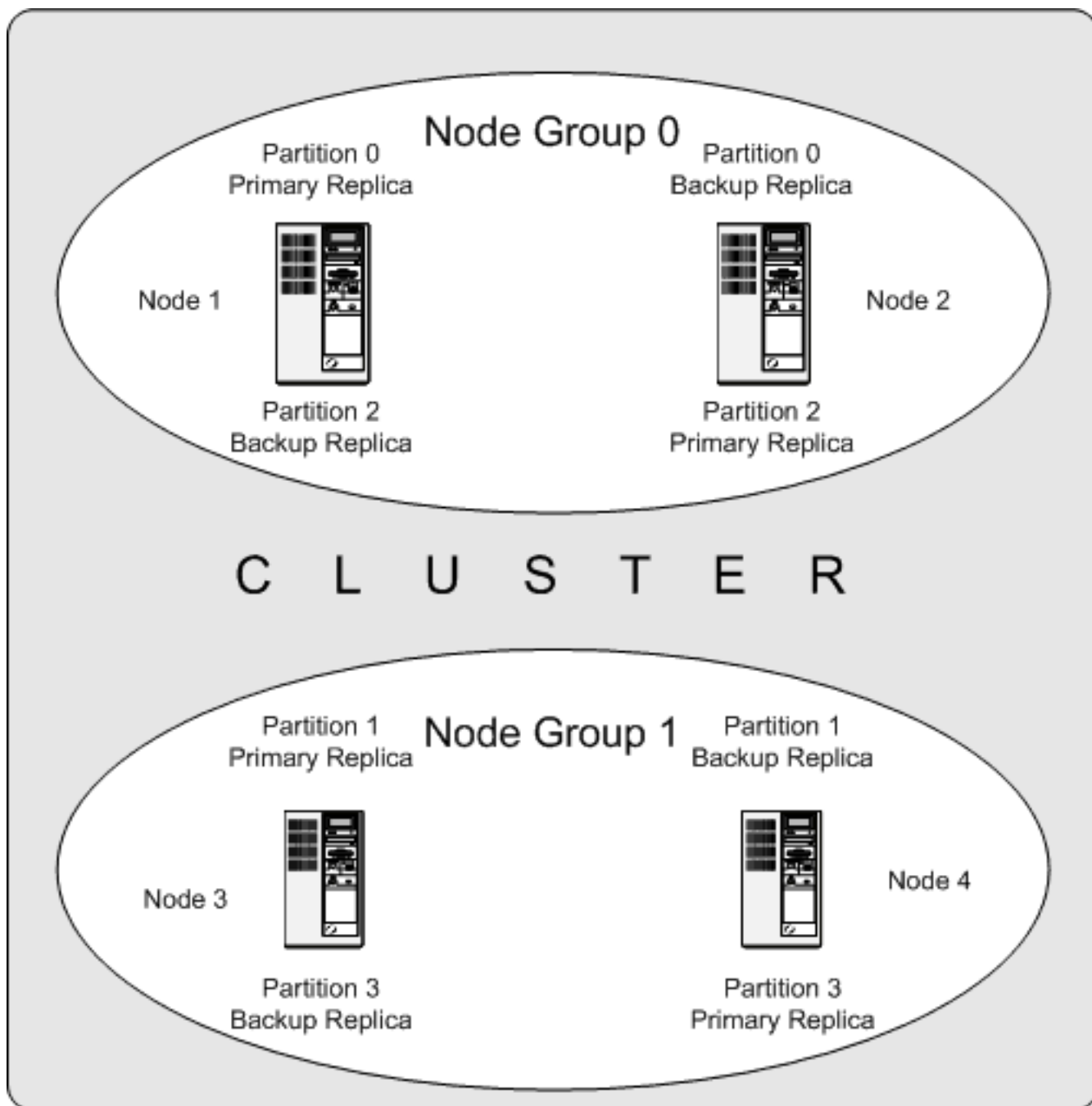
注:現在は、クラスタのすべてのノード グループは同数のノードを持つ必要があります。

- パーティション:これはクラスタで保持されているデータの一部です。クラスタのノードのパーティション数ほどのクラスタのパーティション数があります。各ノードはクラスタで利用できるノードに割り当てられた少なくともパーティションのコピー 1 つ (つまり、少なくとも 1 つのレプリカ) を維持する役目があります。

レプリカはすべて 1 つのノードに属します。ノードはいくつかのレプリカを保持 (通常保持する) できます。

- レプリカ:これはクラスタのパーティションのコピーです。ノード グループの各ノードはレプリカを 1 つ保存します。パーティション レプリカと呼ばれる場合もあります。ノードグループ毎のレプリカ数はノードの数に一致します。

以下はそれぞれ 2 つのノードを持つ 2 つのノード グループを配置した 4 つのデータノードを持つ MySQL Cluster の図です。ノード 1 およびノード 2 はノード グループ 0 に属し、ノード 3 とノード 4 はノード グループ 1 に属します。ここではデータ (`ndbd`) ノードのみ示しています。実際に使用されるクラスタではクラスタの管理に `ndb_mgmd` プロセスとクラスタに保持されているデータにアクセスするための少なくとも 1 つの SQL ノードが必要ですが、分かり易くするためにここでは省略しています。

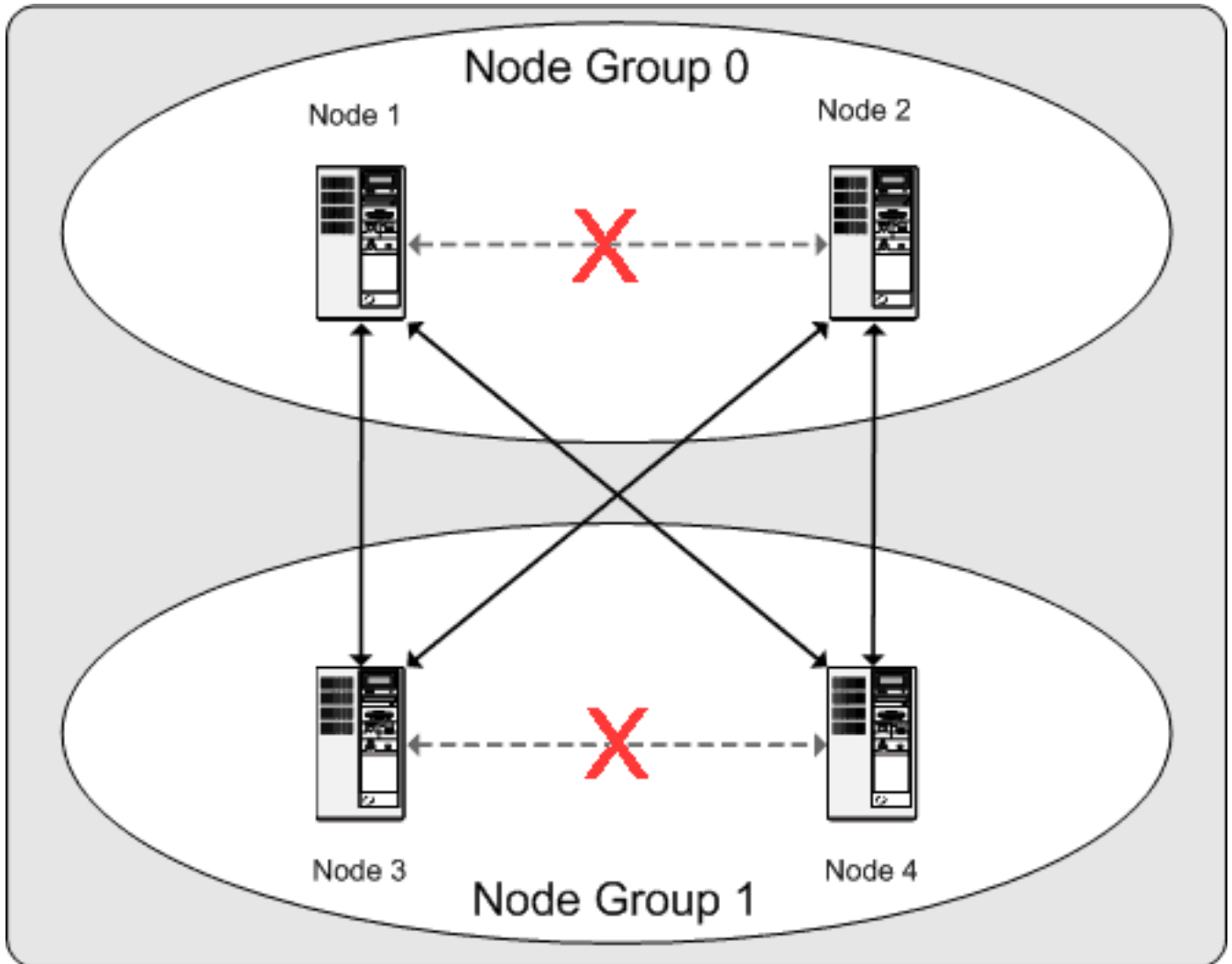


クラスタが保持するデータは 0、1、2、3 の番号を付いた 4 つのパーティションに分割されます。各パーティションは同じノードグループの複数のコピーに保持されます。パーティションは交互にノードグループに保持されます。

- パーティション 0 はノードグループ 0 に保持されます。プライマリ レプリカ (プライマリ コピー) はノード 1 に保持されます。バックアップ レプリカ (パーティションのバックアップ コピー) はノード 2 に保持されます。
- パーティション 1 は別のノードグループ (ノードグループ 1) に保持されます。このパーティションのプライマリ レプリカはノード 3、そのバックアップ レプリカはノード 4 の保持されます。
- パーティション 2 はノードグループ 0 に保持されます。しかし、2 つのレプリカの配置はパーティション 0 と逆になります。パーティション 2 では、プライマリ レプリカはノード 2 に保持され、バックアップはノード 1 に保持されます。

- パーティション 3 はノードグループ 1 に保存され、その 2 つのレプリカの配置はパーティション 1 のそれと逆になります。つまり、プライマリレプリカはノード 4 に配置され、バックアップはノード 3 に配置されます。

MySQL Cluster の継続的なオペレーションに関してこの意味するところは以下になります。クラスタで使用される各ノードグループが動作している少なくとも 1 つのノードを持つ限り、クラスタはすべてのデータの完全なコピーを持ち実行可能であり続けます。これを次の図に示します。



この例では、クラスタがそれぞれ 2 つのノードを持つ 2 つのノードグループで構成されている場合、少なくともノードグループ 0 に 1 つのノードおよび少なくともノードグループ 1 の 1 つのノードの組み合わせでクラスタを「有効」に維持します (図の矢印で示した部分) ことができます。しかし、どちらかのノードグループの両方のノードが失敗した場合、残りの 2 つのノードでは不十分 (X の印の付いた矢印) です。どちらの場合も、クラスタは全体のパーティションを失いすべてのクラスタのデータの完全なセットのアクセスを提供できなくなります。

簡単なマルチ コンピューターの手引き

この項は「手引書」で MySQL Cluster の計画、インストール、設定、および運営に関して説明します。その一方で [MySQL Cluster の設定](#) の例では様々なクラスタの構築および設定について詳しく説明します。ここで説明するガイドラインおよびプロシージャを実行することによって可用性およびデータの安全に必要な 最低の条件を満たした運用可能な MySQL Cluster を構築できます。

この項ではハードウェアおよびソフトウェア要件；ネットワークの問題、MySQL Cluster のインストール、設定の問題、起動、停止、およびクラスタの再起動；サンプル データベースのローディング、並びにクエリの実行について説明します。

この手引書は以下の仮定条件に基づいています。

ノード	IP アドレス
マネジメント (MGM) ノード	192.168.0.10
MySQL サーバー (SQL) ノード	192.168.0.20
データ (NDBD) ノード "A"	192.168.0.30
データ (NDBD) ノード "B"	192.168.0.40

ndb_mgmd
ndb_mgm

192.168.0.10

Management Server
(MGM Node)

mysqld --ndb-cluster

192.168.0.20

MySQL Server
(SQL Node)

Switch

ndbd

192.168.0.30

Data Node "A"
(NDBD Node)

ndbd

192.168.0.40

Data Node "B"
(NDBD Node)

(Network)

2. このシナリオの各ホストはインテル ベースのデスクトップ PC で、標準設定の不必要なサービスの要らないディスクにインストールした一般的な Linux 分散型で行っています。標準の TCP/IP ネットワーク機能を備えた

コアの OS で十分です。簡略化するために、すべてのホストのファイルシステムは全く同じものを使用しています。ファイルシステムが同じでない場合、以下の指示に従う必要があります。

3. 標準の 100 Mbps あるいは 1 ギガビットの Ethernet カードをそのカードの適切なドライバを使用して各マシンにインストールしています。4 台のすべてのホストはスイッチなどの標準の Ethernet ネットワーク機器を使用して接続しています。(すべてのマシンは同じスループットのネットワーク カードを使用する必要があります。つまり、クラスタの 4 台のすべてのマシンは 100 Mbps カードあるいは 4 台のすべてのマシンは 1 Gbps カードを使用する必要があります。)MySQL Cluster は 100 Mbps のネットワークで動作します。しかし、ギガビットの Ethernet を使用するとパフォーマンスが向上します。

MySQL Cluster は 100 Mbps 以下のスループットのネットワークで使用されるようには設計されておりませんのでこの点ご留意願います。このため (とりわけ)、MySQL Cluster をインターネットなどの公共のネットワークを使用しても多分うまく行かないし、またまた推奨していません。

4. サンプルのデータ用に、弊社では [world](#) データベースを使用します。それは MySQL AB のウェブ サイトからダウンロードできます。このデータベースは比較的スペースが小さいので、弊社では各マシンは 256MB RAM あれば、オペレーティング システム、ホスト NDB プロセス、および (データノード用) データベースの保持に十分であると考えています。

この手引書では Linux オペレーティング システムにを採り上げていますが、ここでの説明およびプロシージャは他のサポートしているオペレーティング システムにも転用できます。弊社ではお客様が既にネットワーク機能を備えたオペレーティング システムの最低限のインストールおよび設定をご存知で、必要に応じてこの件に関する支援をどこからでも得られるものであるという前提で説明しています。

次項では MySQL Cluster のハードウェア、ソフトウェア、およびネットワーク要件についてさらに突っ込んだ説明をします。([ハードウェア、ソフトウェア、およびネットワークの構築](#) 参照。)

ハードウェア、ソフトウェア、およびネットワークの構築

MySQL Cluster の利点の 1 つは通常のハードウェアで動作し、すべての生きたデータ ストレージは in-memory で行われるので、大容量の RAM 以外にこの点に関し特別な仕様を必要としないということです。(これはディスク データ テーブルには適用されません。— この点に関する詳細は [MySQL Cluster ディスク データ ストレージ](#) を参照してください。)当然のことながら、マルチの高速 CPU はパフォーマンスを強化します。他のクラスタ プロセスに必要なメモリは比較的小さくて済みます。

クラスタに必要なソフトウェア要件もまた穏やかなものです。MySQL Cluster をサポートするホストのオペレーティング システムには特別なモジュール、サービス、アプリケーション、あるいは設定の必要はありません。サポートしているオペレーティング システムには、標準のインストールで十分です。MySQL のソフトウェア要件は簡素です。今必要なものはクラスタをサポートした量産用の SQL 5.1 のリリースだけです。単にクラスタを使用するためにだけにお客様ご自身で MySQL をコンパイルする必要はありません。この手引書では、MySQL ソフトウェア ダウンロード ページ <http://dev.mysql.com/downloads/> で入手可能なお客様のオペレーティング システムに適したサーバー バイナリを使用しているものとしてこの説明を続けます。

ノード間通信には、クラスタは TCP/IP ネットワークを標準のトポロジでサポートしており、各ホストに必要な最低条件は標準の 100 Mbps Ethernet カード、それにクラスタ全体の接続を提供するスイッチ、ハブ、あるいはルータです。弊社では以下の理由により MySQL Cluster が非クラスタ マシンを共有していない個別のサブネットで行われることを強くお勧めします。

- セキュリティ:クラスタ ノード間の通信は暗号化あるいはシールドされていません。MySQL Cluster 内での伝送保護の唯一の方法はお客様のクラスタを保護されたネットワークで運用することです。MySQL Cluster をウェブ アプリケーションに使用する場合には、クラスタは必ずファイアウォールの内側に常駐させ、ネットワークの De-Militarized (DMZ) ゾーンあるいはそのような場所に常駐させないでください。
- 効率:MySQL Cluster をプライベートあるいは保護されたネットワークに設定することでクラスタのホスト間の帯域を排他的に使用できます。個別のスイッチを MySQL Cluster の使用することでクラスタのデータへの無許可のアクセスから保護するだけでなく、ネットワーク上の他のコンピュータ間の伝送による干渉からクラスタ ノードをシールドできます。信頼性を向上させるには、デュアルのスイッチおよびデュアルのカードを使用して一極集中型不具合 (シングル ポイントのファイリユア) からネットワークを守ることが出来ます。多くのデバイス ドライバがそのような通信リンクのフェールオーバーをサポートしています。

MySQL Cluster に高速のスケラブル コヒーラント インターフェース (SCI) を使用することもできます。これは必要条件ではありません。このプロトコルおよびその MySQL Cluster での使用方法については [MySQL Cluster での高速インターコネクトを使用する](#) を参照してください。

マルチ コンピュータのインストール

データあるいは SQL ノードを実行する各 MySQL Cluster のホスト コンピュータには MySQL サーババイナリをインストールする必要があります。マネジメント ノードには MySQL サーババイナリをインストールする必要はありませんが、MGM サーバデーモンおよびクライアント バイナリ (それぞれ `ndb_mgmd` および `ndb_mgm`) をインストールする必要があります。この項では各種のクラスタ ノードに適切なバイナリをインストールするために必要なステップについて説明します。

MySQL AB ではクラスタをサポートするコンパイル済みのバイナリを提供していますので、通常これらをお客様ご自身でコンパイルする必要はありません。ですから、各クラスタ ホストのインストール プロセスの最初のステップはファイル `mysql-5.1.15-beta-pc-linux-gnu-i686.tar.gz` を [MySQL downloads area](#) からダウンロードすることです。それを各マシン `/var/tmp` ディレクトリに配置したものと想定します。(カスタムのバイナリが必要な場合には、[開発ソース ツリーからのインストール](#) を参照してください。)

RPM も 32 ビットおよび 64 ビットの Linux プラットフォームに利用できます。MySQL Cluster 1 台につき 4 つの RPM が必要です。

- サーバ RPM (例えば、[MySQL-server-5.1.15-beta-0.glibc23.i386.rpm](#)) は、MySQL サーバの動作に必要なコア ファイルを提供します。
- サーバ/Max RPM (例えば、[MySQL-Max-5.1.15-beta-0.glibc23.i386.rpm](#)) は、MySQL サーバにクラスタをサポートしたバイナリを提供します。
- NDB Cluster - ストレージ エンジン RPM (例えば、[MySQL-ndb-storage-5.1.15-beta-0.glibc23.i386.rpm](#)) は、MySQL Cluster にデータノード バイナリ (`ndbd`) を提供します。
- NDB Cluster - ストレージ エンジン マネジメント RPM (例えば、[MySQL-ndb-management-5.1.15-beta-0.glibc23.i386.rpm](#)) は MySQL Cluster にマネジメント サーババイナリ (`ndb_mgmd`) を提供します。

さらに、NDB Cluster を取得する必要があります - ストレージ エンジンの基本ツール RPM (例えば、[MySQL-ndb-tools-5.1.15-beta-0.glibc23.i386.rpm](#)) は、MySQL Cluster で使用するいくつかの有用なアプリケーションを提供します。これらのうちで重要なものは MySQL Cluster マネジメント クライアント (`ndb_mgm`) です。NDB Cluster - ストレージ エンジン予備ツール RPM (例えば、[MySQL-ndb-extra-5.1.15-beta-0.glibc23.i386.rpm](#)) は、いくつかの追加テストおよびモニタリングのプログラムが含まれていますが、MySQL Cluster のインストールには必要ありません。(これらの追加プログラムの詳細に関しては、[クラスタ ユーティリティ プログラム](#) を参照してください。)

RPM ファイル名 (`5.1.15-beta` として表示) の MySQL のバージョン番号は実際に使用するバージョンによって変わります。インストールするすべてのクラスタ RPM の MySQL バージョン番号が同じであることが非常に重要です。`glibc` バージョン番号 (使用している場合 — `glibc23` として表示)、およびアーキテクチャ名 (`i386` として表示) は RPM をインストールするマシンに適したものであることが必要です。

MySQL AB 供給の RPM を使用した MySQL をインストールに関する一般情報は、[Linux に MySQL をインストールする](#) を参照してください。

RPM のパッケージをインストールした後、さらに [マルチ コンピュータの設定](#) で説明するクラスタを設定する必要があります。

注:インストールを完了しても、まだどのバイナリも起動しないでください。すべてのノードの設定が終了した段階で、起動の仕方を説明します。

データおよび SQL ノードのインストール — `.tar.gz` バイナリ

データあるいは SQL ノードのホスト用の各マシンで、システム `root` ユーザーとして以下のステップを踏みます。

1. `/etc/passwd` および `/etc/group` ファイル (あるいはユーザーおよびグループを管理のためのお客様のオペレーティング システムで提供されたツール) をチェックし、`mysql` グループおよび `mysql` ユーザーがシステムに既に用意されているか確認します。OS のディストリビューションの中にはこれらをオペレーティング システムのインストール プログラムとして作成している場合もあります。それらがまだ作成されていない場合、新たに `mysql` ユーザーグループを作成し、`mysql` ユーザーをこのグループに追加します。

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

`useradd` および `groupadd` の文法は Unix のバージョンによって多少異なる場合があります、また `adduser` および `addgroup` などの別な名前を使用している場合もあります。

2. ダウンロードしたファイルを含むディレクトリにロケーションを変更し、アーカイブを開いて、`mysql` に symlink を作成します。実際のファイル名およびディレクトリ名は MySQL のバージョン番号によって異なる場合があります。

```
shell> cd /var/tmp
shell> tar -C /usr/local -xzf mysql-5.1.15-beta-pc-linux-gnu-i686.tar.gz
shell> ln -s /usr/local/mysql-5.1.15-beta-pc-linux-gnu-i686 /usr/local/mysql
```

3. ロケーションを `mysql` ディレクトリに変更し、供給されたスクリプトを実行してシステムのデータベースを作成します。

```
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
```

4. MySQL サーバーおよびデータ ディレクトリに必要な許可を設定します。

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

データノードをホストしている各マシンのデータ ディレクトリは `/usr/local/mysql/data` であることを確認します。マネジメント ノードを設定する際にこの情報が必要になります。([マルチ コンピュータの設定](#) 参照。)

5. 適切なディレクトリに MySQL 起動スクリプトをコピーし、実行できる状態にし、オペレーティング システムがブートしたときに起動できるように設定します。

```
shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

(起動スクリプトのディレクトリはオペレーティング システムおよびバージョンによって異なる場合があります— 例えば、いくつかの Linux ディストリビューションの場合、それは `/etc/init.d` となります。)

ここでは起動スクリプトのリンクの作成に Red Hat の `chkconfig` を使用します。Debian の `update-rc.d` などお客様のオペレーティング システムおよびディストリビューションでこの目的に適切と思われるものを使用します。

データノードおよび SQL ノードが常駐する各マシンで前に説明したステップを個別に実行する必要がありますので忘れずに実行してください。

SQL ノードのインストール — RPM ファイル

クラスタ SQL ノードをホストに使用する各マシンで、以下のコマンドをシステムのルート ユーザーとして実行して MySQL Max RPM をインストールし、RPM に表示された名前を必要に応じて MySQL AB のウェブサイトからダウンロードした RPM に一致する名前に置き換えます。

```
shell> rpm -Uvh MySQL-server-5.1.15-beta-0.glibc23.i386.rpm
shell> rpm -Uvh MySQL-Max-5.1.15-beta-0.glibc23.i386.rpm
```

これによりインストールに必要なすべての MySQL サポート ファイルおよび MySQL サーバーバイナリ (`mysqld`) を `/usr/sbin` ディレクトリにインストールします。これにより `mysql.server` および `mysqld_safe` 起動スクリプトもまた `/usr/share/mysql` および `/usr/bin` にそれぞれインストールします。RPM インストーラが一般的な設定の操作を (必要に応じて `mysql` ユーザーおよびグループの作成) を自動的に実行します。

データノードのインストール — RPM ファイル

クラスタのデータノードをホストするコンピュータには NDB Cluster - ストレージ エンジン RPM のみをインストールする必要があります。インストールするには、この RPM をデータノードのホストにコピーし、以下のコマンドをシステム ルートのユーザーとして実行し、RPM に表示された名前を必要に応じて MySQL AB のウェブサイトからダウンロードした RPM に一致する名前に置き換えます。

```
shell> rpm -Uvh MySQL-ndb-storage-5.1.15-beta-0.glibc23.i386.rpm
```

前のコマンドで MySQL Cluster データノード バイナリ (`ndbd`) を `/usr/sbin` ディレクトリにインストールします。

マネジメント ノードのインストール — [.tar.gz](#) バイナリ

(MGM) ノードのインストールには [mysqld](#) バイナリをインストールする必要はありません。MGM サーバーとクライアントにのみバイナリが必要で、それはダウンロードしたアーカイブにあります。再度このファイルを [/var/tmp](#) に配置したか確認します。

システム [root](#) (つまり、[sudo](#)、[su root](#)、あるいはお客様のシステムで相当するものを一時的にシステムの管理者のアカウント権限として) として、以下のステップを実行して [ndb_mgmd](#) および [ndb_mgm](#) を Cluster マネジメントノードのホストにインストールします。

1. ロケーションを [/var/tmp](#) ディレクトリに変更して、[ndb_mgm](#) および [ndb_mgmd](#) をアーカイブから [/usr/local/bin](#) などの適切なディレクトリに抽出します。

```
shell> cd /var/tmp
shell> tar -zxvf mysql-5.1.15-beta-pc-linux-gnu-i686.tar.gz
shell> cd mysql-5.1.15-beta-pc-linux-gnu-i686
shell> cp bin/ndb_mgm* /usr/local/bin
```

(ダウンロードしたアーカイブを開いたときに作成されたディレクトリとそれが含んでいるファイルを [/var/tmp](#) から、[ndb_mgm](#) および [ndb_mgmd](#) が実行可能なディレクトリにコピーされたら削除できます。

2. ファイルをコピーしたディレクトリにロケーションを変更し、次にその両方を実行出来るようにします。

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

マネジメント ノードのインストール — RPM ファイル

MySQL Cluster マネジメント サーバーをインストールするには、NDB Cluster - ストレージ エンジン マネジメント RPM のみを使用する必要があります。この RPM をマネジメント ノードをホストするコンピュータにコピーし、次にそれを以下のコマンドをシステム ルートのユーザー (RPM に表示された名前を必要に応じて MySQL AB ウェブサイトからダウンロードしたストレージ エンジン マネジメント RPM に一致する名前に置き換えます) としてインストールします。

```
shell> rpm -Uvh MySQL-ndb-management-5.1.15-beta-0.glibc23.i386.rpm
```

これによりマネジメント サーバーバイナリ ([ndb_mgmd](#)) を [/usr/sbin](#) ディレクトリにインストールします。

また ストレージ エンジン基本ツール RPM で供給された [NDB](#) マネジメント クライアントをインストールする必要があります。この RPM をマネジメント ノードと同じコンピュータにコピーし、次にそれを以下のコマンドをシステム ルートのユーザー (再度、RPM に表示された名前を必要に応じて MySQL AB ウェブ サイトからダウンロードしたストレージ エンジン基本ツール RPM に一致する名前に置き換えます) としてインストールします。

```
shell> rpm -Uvh MySQL-ndb-tools-5.1.15-beta-0.glibc23.i386.rpm
```

ストレージ エンジンの基本ツール RPM は MySQL Cluster マネジメント クライアント ([ndb_mgm](#)) を [/usr/bin](#) ディレクトリにインストールします。

[マルチ コンピュータの設定](#) で、弊社の参考クラスタですべてのノードに設定ファイルを作成します。

マルチ コンピュータの設定

4 ノード、4 ホスト MySQL Cluster には ノード毎に 1 つずつ 4 つの設定ファイルを書く必要があります。

- 各データ ノードあるいは SQL ノードは 2 種類の情報を提供する [my.cnf](#) ファイルが必要です。ノードに MGM ノードの所在を知らせる接続文字列、および NDB モードでこのホスト (データノードをホストしているマシン) の MySQL サーバーに実行を告げる行。

接続文字列に関する詳細は、[クラスタの 接続文字列](#) を参照してください。

- マネジメント ノードは維持するレプリカの数量、各データ ノードのデータおよびインデックスのメモリ容量、データ ノードをどこで探すか、各データ ノードのディスクのデータの保存場所、どこで SQL ノードを探すかなどを告げる [config.ini](#) ファイルが必要とします。

ストレージおよび SQL ノードの設定

データ ノードが必要とする `my.cnf` ファイルはかなり簡素です。設定ファイルは `/etc` ディレクトリにあり、テキスト エディタで編集できることが必要です。(ない場合にはファイルを作成します。)例えば:

```
shell> vi /etc/my.cnf
```

ここではファイルを作成するために `vi` が使用しますが、どのテキスト エディタも同様に機能する必要があります。

弊社の模範設定での各データノードおよび SQL ノードに対し `my.cnf` は以下のようになります。

```
# Options for mysqld process:
[MYSQLD]
ndbcluster                # run NDB storage engine
ndb-connectstring=192.168.0.10 # location of management server

# Options for ndbd process:
[MYSQL_CLUSTER]
ndb-connectstring=192.168.0.10 # location of management server
```

上記の情報を入力後、このファイルを保存しテキスト エディタを終了します。これをデータノード「A」、データノード「B」、および SQL ノードをホストしているマシンに行います。

重要前述のように `mysqld` プロセスを `my.cnf` の `[ MYSQLD ]` にある `ndbcluster` および `ndb-connectstring` パラメータで実行すると、`CREATE TABLE` あるいは `ALTER TABLE` ステートメントをクラスタを実際に起動するまで実行できなくなります。または、これらのステートメントはエラーが表示されて失敗します。これは設計によります。

マネジメント ノードの設定

MGM ノードを設定する最初のステップは、設定ファイルを格納するディレクトリを作成し、次にファイルそのものを作成します。例えば (`root` として実行する):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

弊社の見本設定では、`config.ini` ファイルは以下のようになります。

```
# Options affecting ndbd processes on all data nodes:
[NDBD DEFAULT]
NoOfReplicas=2 # Number of replicas
DataMemory=80M # How much memory to allocate for data storage
IndexMemory=18M # How much memory to allocate for index storage
                # For DataMemory and IndexMemory, we have used the
                # default values. Since the "world" database takes up
                # only about 500KB, this should be more than enough for
                # this example Cluster setup.

# TCP/IP options:
[TCP DEFAULT]
portnumber=2202 # This the default; however, you can use any
                # port that is free for all the hosts in cluster
                # Note: It is recommended beginning with MySQL 5.0 that
                # you do not specify the portnumber at all and simply allow
                # the default value to be used instead

# Management process options:
[NDB_MGMD]
hostname=192.168.0.10 # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster # Directory for MGM node log files

# Options for data node "A":
[NDBD]
                # (one [NDBD] section per data node)
hostname=192.168.0.30 # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's data files

# Options for data node "B":
[NDBD]
```

```
hostname=192.168.0.40      # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's data files

# SQL node options:
[MYSQLD]
hostname=192.168.0.20      # Hostname or IP address
                           # (additional mysqld connections can be
                           # specified for this node for various
                           # purposes such as running ndb_restore)
```

(注:world データベースは <http://dev.mysql.com/doc/> の「例題設定」のリストからダウンロードできます。)

すべての設定ファイルを作成してこれらの最低限のオプションを指定すると、クラスタを実行する準備が整い、すべてのプロセスが動作していることを検証できます。これをどのように行うか [最初の起動](#) で説明します。

利用できる MySQL Cluster の設定パラメータおよびその使用方法の詳細については [設定ファイル](#)、および [MySQL Cluster の設定](#) を参照してください。バックアップに関する MySQL Cluster の設定については、[クラスタ バックアップの設定](#) を参照してください。

注:クラスタ マネジメント ノードのデフォルトのポートは 1186で、データ ノードのデフォルトのポートは 2202 です。MySQL 5.0.3 以降ではこの制限が解除されており、クラスタはすでにフリーになっているものから自動的にデータ ノードにポートを割り当てます。

最初の起動

設定後のクラスタの稼働はそれほど難しくありません。各クラスタ ノードはクラスタが常駐するホストで個別に起動します。マネジメント ノードを最初に起動し、次にデータノード、最後に SQL ノードを起動します。

1. マネジメント ホストでシステム シェルから以下のコマンドを発行し、MGM ノード プロセスを実行します。

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

`ndb_mgmd` に `-f` あるいは `--config-file` オプションを使用して設定ファイルの場所を知らせる必要があります。(詳細は [ndb_mgmd](#)、[マネジメント サーバードプロセス](#) を参照してください。)

2. 各データ ノードのホストで、このコマンドを実行して最初の `ndbd` プロセスを始めます。

```
shell> ndbd --initial
```

`--initial` パラメータのみを `ndbd` を最初に実行するとき、あるいはバックアップ/復旧作業あるいは設定変更後に起動したときに使用することが非常に重要です。これは `--initial` オプションによってノードがリカバリのログファイルを含みリカバリに必要な `ndbd` インスタンスで以前作成したファイルを削除するからです。

この例外は `--initial` は ディスク データ ファイルは削除しないということです。クラスタの最初の再起動を実行する必要がある場合、既存のディスク データのログファイルおよびデータ ファイルを手動で削除する必要があります。

3. SQL ノードが常駐するクラスタのホストに MySQL をインストールするために RPM ファイルを使用した場合には、SQL ノードで MySQL サーバードプロセスを実行するには供給された起動スクリプトを使用 (する必要があります) できます。

すべてが上手くいってクラスタが正しく設定できたら、この段階でクラスタを使用できます。`ndb_mgm` マネジメント ノード クライアントを起動してこれをテストできます。その出力は以下になる必要があります。使用している MySQL のバージョンによって出力が多少異なる場合もあります。

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.30 (Version: 5.1.15-beta, Nodegroup: 0, Master)
```



```
id=3 @192.168.0.40 (Version: 5.1.15-beta, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.10 (Version: 5.1.15-beta)

[mysqld(SQL)] 1 node(s)
id=4 (Version: 5.1.15-beta)
```

注:SQL ノードはここでは [\[mysqld\(API\)\]](#) として使用しています。これは完全に通常で、[mysqld](#) プロセスがクラスタ API ノードの役割を果たしていることを表しています。

この段階でMySQL クラスタのデータベース、テーブル、およびデータを使用できる準備ができました。簡単な説明については、[サンプル データのローディングとクエリの実行](#) を参照してください。

サンプル データのローディングとクエリの実行

MySQL Cluster のデータの作業はクラスタ無しの MySQL で行うのとそれほど違いはありません。考慮すべき点が 2 点あります。

- クラスタでコピーするテーブルには、[NDB Cluster](#) ストレージ エンジンを使用する必要があります。これを指定するには [ENGINE=NDB](#) あるいは [ENGINE=NDBCLUSTER](#) テーブル オプションを使用します。テーブルを作成する際にこのオプションを追加できます。

```
CREATE TABLE tbl_name ( ... ) ENGINE=NDBCLUSTER;
```

代わりに、異なるストレージ エンジンを使用している既存のテーブルに対して、[ALTER TABLE](#) を使用してテーブルを変更して [NDB Cluster](#) を使用できます。

```
ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
```

- 各 [NDB](#) テーブルにはプライマリ キーが必要です。ユーザーがテーブルを作成したときにプライマリ キーを定義しなかった場合、[NDB Cluster](#) ストレージ エンジンが自動的に非表示のテーブルを生成します。(注:この非表示のテーブルは他のテーブル インデックスと同じスペースを使用します。メモリ不足のためにこれらの自動的に作成されたインデックスを使用する際に問題に遭遇することは普通ではありません。)

[mysqldump](#) の出力を使用して既存のデータベースからテーブルをインポートする場合、テキスト エディタの SQL スクリプトを開きテーブルの作成ステートメントに [ENGINE](#) オプションを追加します。あるいは既存の [ENGINE](#) (あるいは [TYPE](#)) オプションを置き換えます。MySQL Cluster をサポートしていない別の MySQL サーバーに [world](#) サンプルのデータベースを持っていて、[City](#) テーブルをエクスポートするとします。

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

その結果の [city_table.sql](#) ファイルはテーブルの作成ステートメント (およびテーブル データをインポートするために必要な [INSERT](#) ステートメント) を含みます。

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default "",
  `CountryCode` char(3) NOT NULL default "",
  `District` char(20) NOT NULL default "",
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);(remaining INSERT statements omitted)
```

MySQL が [NDB](#) ストレージ エンジンがこのテーブルに使用していることを確認する必要があります。この確認の方法は 2 つあります。その 1 つは Cluster データベースにインポートする前 before にテーブルの定義を変更することです。例として [City](#) テーブルを使用し、以下のように定義の [ENGINE](#) オプションを変更します。

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
```



```
`Name` char(35) NOT NULL default "",
`CountryCode` char(3) NOT NULL default "",
`District` char(20) NOT NULL default "",
`Population` int(11) NOT NULL default '0',
PRIMARY KEY (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabul',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);(remaining INSERT statements omitted)
```

クラスタ テーブルの一部になる各テーブルの定義にこれを行う必要があります。これを行うための最も簡単な方法は定義を含むファイルに検索および置き換えを実行し、`TYPE=engine_name` あるいは `ENGINE=engine_name` のすべてのインスタンスを `ENGINE=NDBCLUSTER` で置き換えることです。ファイルの変更を希望しない場合には、変更していないファイルを使用してテーブルを作成し、次に `ALTER TABLE` を使用してストレージ エンジンを変更します。この項の後で詳細を説明します。

クラスタの SQL ノードで `world` と呼ばれるデータベースを既に作成したとして、`mysql` コマンドライン クライアントを使用して `city_table.sql` を読み込み、そして次にいつもの方法で相当するテーブルを作成して移植します。

```
shell> mysql world < city_table.sql
```

前述のコマンドを SQL ノードを実行している (この場合、IP アドレスが `192.168.0.20` のマシンで) ホストで忘れないで実行することが非常に重要です。

SQL ノードで `world` 全体のデータベースのコピーを作成するには、非クラスタ サーバーの `mysqldump` を使用してデータベースをファイル名 `world.sql` にエクスポートします。例えば、`/tmp` のディレクトリで行います。次にテーブルの定義を今説明したように変更し、以下のようにクラスタの SQL ノードにそのファイルをインポートします。

```
shell> mysql world < /tmp/world.sql
```

ファイルを別のロケーションに保存した場合、前述の説明をしかるべく調整します。

MySQL の `NDB Cluster` が 5.1 データベースのオートディスカバリーをサポートしていないことを覚えておくことが重要です。(MySQL Cluster の既知の制限 参照。)このことは、データ ノードで `world` データベースおよびそのテーブルを作成したら、`CREATE SCHEMA world` ステートメントを発行する必要があることを意味しています。

`SELECT` クエリを SQL ノードで実行することは MySQL サーバーの他のインスタンスで実行するのと違いはありません。。コマンドラインからクエリを実行するには、最初に MySQL Monitor にいつもの方法 (`root` パスワードを `Enter password:` プロンプトで指定します) でログインします。

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.1.15-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

弊社では単純に MySQL サーバーの `root` アカウントを使用していますが、お客様は MySQL サーバーのインストールの際は堅固な `root` パスワードの設定を含む標準的なセキュリティ対策を講じているものと想定しています。詳細は [最初の MySQL アカウントの確保](#) を参照してください。

クラスタ ノードがお互いにアクセスするときに MySQL 権限システムを使用しないことを考慮する必要があります。MySQL ユーザーアカウント (`root` アカウントを含む) の設定あるいは変更によって、ノード間のインターラク션に影響を及ぼさず、SQL ノードにアクセスするアプリケーションにのみ影響を及ぼします。

SQL スクリプトのインポートに先立ちテーブル定義の `ENGINE` 節を変更しなかった場合、この時点で以下のステートメントを実行する必要があります。

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

データベースを選択してそのデータベースのテーブルに対する **SELECT** クエリの実行も既存の MySQL Monitor と同様に通常の方法で実現できます。

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----+-----+
| Name   | Population |
+-----+-----+
| Bombay | 10500000   |
| Seoul  | 9981619    |
| São Paulo | 9968485   |
| Shanghai | 9696300   |
| Jakarta | 9604900   |
+-----+-----+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

MySQL を使用しているアプリケーションは標準の API で NDB テーブルにアクセスします。お客様のアプリケーションが、MGM あるいはデータノードではなく SQL ノードにアクセスすることを覚えておくことが重要です。この簡単な例ではネットワーク上の Web サーバーで動作している PHP 5 の **mysqli** 拡張を使用した場合のようにどのように **SELECT** ステートメントを実行するかを示しています。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
# connect to SQL node:
$link = new mysqli('192.168.0.20', 'root', 'root_password', 'world');
# parameters for mysqli constructor are:
# host, user, password, database

if( mysqli_connect_errno() )
  die("Connect failed: " . mysqli_connect_error());

$query = "SELECT Name, Population
  FROM City
  ORDER BY Population DESC
  LIMIT 5";

# if no errors...
if( $result = $link->query($query) )
{
  ?>
<table border="1" width="40%" cellpadding="4" cellspacing="1">
  <tbody>
    <tr>
      <th width="10%">City</th>
      <th>Population</th>
    </tr>
    <?
    # then display the results...
    while($row = $result->fetch_object())
      printf("<tr>\n  <td align='center'>%s</td><td>%d</td>\n</tr>\n",
        $row->Name, $row->Population);
  ?>
  </tbody>
</table>
<?
# ...and verify the number of rows that were retrieved
printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
}
else
  # otherwise, tell us what went wrong
```

```
echo mysqli_error();

# free the result set and the mysqli connection object
$result->close();
$link->close();
?>
</body>
</html>
```

Web サーバー上で実行されているプロセスが SQL ノードの IP アドレスにアクセスできるものとします。

同様に、MySQL C API、Perl-DBI、Python-mysql、あるいは MySQL AB の自身のコネクタを使用してデータ定義および操作のタスクを MySQL で行うのと同様に実行できます。

安全なシャットダウンと再起動

クラスタをシャットダウンするには、MGM ノードをホストしているマシンのシェルに以下のコマンドを入力します。

```
shell> ndb_mgm -e shutdown
```

この `-e` オプションがコマンドをシェルから `ndb_mgm` クライアントに渡すために使用されます。[コマンドラインにおけるオプションの使用](#) 参照。そのコマンドが `ndb_mgm`、`ndb_mgmd`、および他の `ndbd` プロセスを終了させます。SQL ノードは `mysqladmin shutdown` および他の方法でシャットダウンできます。

クラスタを再起動するには、これらのコマンドを使用します。

- マネジメント ホスト ([192.168.0.10](#) サンプル設定):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- 各データ ノードのホスト ([192.168.0.30](#) および [192.168.0.40](#)):

```
shell> ndbd
```

NDBD ノードを通常に再起動する際にはこのコマンドを `--initial` オプションで実行しないことを覚えておいてください。

- SQL ホスト ([192.168.0.20](#)):

```
shell> mysql &
```

クラスタのバックアップを取るには [バックアップを作成するためのマネジメント クライアントの使用](#) を参照してください。

クラスタをバックアップから復旧するには `ndb_restore` コマンドを使用する必要があります。これは [クラスタのバックアップの復旧方法](#) で説明しています。

MySQL Cluster の設定の詳細は [MySQL Cluster の設定](#) で入手できます。

MySQL Cluster の設定

MySQL Cluster を構成する MySQL サーバーが通常（非クラスタ）の MySQL サーバーと 1 点異なります。それは [NDB Cluster](#) ストレージ エンジンを搭載していることです。このエンジンは単に [NDB](#) とも言われ、その 2 つの名前は同じ意味です。

リソースの不必要な割り当てを避けるために、サーバーはデフォルトで [NDB](#) ストレージ エンジンを無効に設定しています。[NDB](#) を有効にするには、サーバーの `my.cnf` 設定ファイルを変更するか、またはサーバーを `--ndbcluster` オプションで起動します。

MySQL サーバーはクラスタの構成要素の一部ですので、クラスタの設定データを取得するには MGM ノードへのアクセス方法を知る必要があります。デフォルトでは MGM ノードを `localhost` で探します。しかし、そのロケーションの場所を指定する必要がある場合には、`my.cnf`、あるいは MySQL サーバーのコマンドラインでできます。[NDB](#) スト

レージ エンジンを使用する前に、所望のデータ ノードはもとより少なくとも MGM ノードを 1 つ動作できるようにしておく必要があります。

ソースコードによる MySQL Cluster の構築

NDB、つまりクラスタ ストレージ エンジンはバイナリのディストリビューションで Linux、Mac OS X、および Solaris で利用できます。現在 Windows を含む MySQL でサポートするすべてのオペレーティング システムでクラスタを動作できるように作業しています。

ソース tarball あるいは MySQL 5.1 BitKeeper ツリーからの構築を選択する場合、`--with-ndbcluster` オプションを `configure` を実行する際必ず使用してください。BUILD/compile-pentium-max ビルド スクリプトを使用することもできます。このスクリプトは OpenSSL が含まれているため、構築を成功裏に終わるにはオープン SSL を使用するかあるいは取得する必要があります。あるいは、`compile-pentium-max` を変更して要件を外します。勿論、ご自身のバイナリにコンパイルするには標準の手順に説明書に従い、次に通常のテストおよびインストール プロシーダを実行します。開発ソース ツリーからのインストール 参照。

ソフトウェアのインストール

次の数項で MySQL のインストール方法には習熟されると思いますので、ここでは MySQL Cluster の設定および非クラスタの MySQL の違いについて説明します。(後者について詳細は、2章MySQL のインストールと更新 を参照してください。)

すべてのマネジメント ノードおよびデータノードを最初に起動しているとクラスタの設定が非常に簡単だということが分かります。これは設定の中でもっとの時間にかかる部分です。`my.cnf` ファイルの編集はとても簡単です。この項では非クラスタの MySQL の設定との違いについてのみ説明します。

MySQL Cluster の簡単なテストの設定

基本に習熟して頂くために、最も簡単な MySQL Cluster の実用面の設定について説明します。これが終了すると、本章の関連する項で提供された情報に従ってお客様のご所望の設定が出来るようになります。

最初に、`/var/lib/mysql-cluster` のような設定ディレクトリを作成する必要があります。それを作成するには以下のコマンドをシステム `root` ユーザーとして実行します。

```
shell> mkdir /var/lib/mysql-cluster
```

このディレクトリで以下の情報を含む `config.ini` と呼ばれるファイルを作成します。必要に応じてお客様のシステムに適切な値を `HostName` および `DataDir` に入力します。

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: DB, API, and MGM are aliases for NDBD, MYSQLD, and NDB_MGMD
# respectively. DB and API are deprecated and should not be used in new
# installations.
[NDBD DEFAULT]
NoOfReplicas= 1

[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
[TCP DEFAULT]

[NDB_MGMD]
HostName= myhost.example.com

[NDBD]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[MYSQLD]
```

```
[MYSQLD]
[MYSQLD]
```

ndb_mgmd この段階でマネジメント サーバーを起動できます。デフォルトでは **config.ini** ファイルを現在動作しているディレクトリから読み込もうとしますので、ファイルが存在するディレクトリに変更して、**ndb_mgmd** を起動します。

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

次に **ndbd** を実行してシングルのデータ ノードを起動します。**ndbd** を所定のデータノードにまさに初めて起動するには、以下のに示す **--initial** オプションを使用する必要があります。

```
shell> ndbd --initial
```

その後の **ndbd** の起動では、普通は **--initial** オプションを省きたいと思うでしょう

```
shell> ndbd
```

--initial をその後の再起動で省く理由はこのオプションでは **ndbd** がこのデータノードの既存のすべてのデータおよびログ ファイルを削除し、新たにそれらを作成するからです。**--initial** を最初の **ndbd** 起動以外に使用しないこの規則の例外はそれをクラスタを起動するときに使用し、新しいデータ ノードを追加した後にバックアップから保存することです。

デフォルトでは、**ndbd** はマネジメント サーバーをポート 1186 の **localhost** で探します。

注:MySQL をバイナリの tarball からインストールした場合には、**ndb_mgmd** および **ndbd** サーバーのパスを明示的に指定する必要があります。(通常、これらは **/usr/local/mysql/bin** にあります。)

最後に、ロケーションを MySQL データ ディレクトリ (通常 **/var/lib/mysql** あるいは **/usr/local/mysql/data**) に変更し、**my.cnf** ファイルが NDB ストレージ エンジンに起動に必要なオプションが含まれているか確認します。

```
[mysqld]
ndbcluster
```

この段階で MySQL サーバーを従来通りに起動できます。

```
shell> mysqld_safe --user=mysql &
```

MySQL サーバーが適切に動作しているか確認するために少し待ちます。**mysql ended** との通知が表示された場合には、サーバーの **.err** ファイルをチェックして何が間違っているか調べます。

ここまですべてが問題なく動作した場合、この段階でクラスタを使用して起動できます。サーバーに接続して **NDBCLUSTER** ストレージ エンジンが有効であることを確認します。

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.1.15-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
***** 12. row *****
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
***** 13. row *****
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

前述の参考例の出力で表示された行番号はサーバーの設定によってお客様のシステムに表示されたものとは異なる場合があります。

NDBCLUSTER テーブルの作成

```

shell> mysql
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
***** 1. row *****
      Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

```

お客様のノードが適切に設定されているか確認するには、マネジメント クライアントを起動します。

```
shell> ndb_mgm
```

クラスタのステータスのレポートを取得するにはマネジメント クライアントの **SHOW** コマンドを使用します。

```

NDB> SHOW
Cluster Configuration
-----
[ndbd(NDB)] 1 node(s)
id=2 @127.0.0.1 (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @127.0.0.1 (Version: 3.5.3)

[mysqld(API)] 3 node(s)
id=3 @127.0.0.1 (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)

```

この段階で、実働可能な MySQL Cluster の設定を完了しました。ここで **ENGINE=NDBCLUSTER** あるいはその別名 **ENGINE=NDB** で作成されたテーブルを使用してクラスタのデータを保存できます。

設定ファイル

MySQL を設定には 2 つのファイルの作業が伴います。

- **my.cnf**: すべての MySQL Cluster 実行ファイルにオプションを指定します。このファイルは MySQL のこれまでの説明で既に習熟していると思いますが、クラスタで使用されている各実行ファイルからアクセスできる必要があります。
- **config.ini**: このファイルは MySQL Cluster マネジメント サーバーによってのみ読み込まれ、クラスタで使用されているすべてのプロセスにそこに含まれる情報を配布します。**config.ini** にはクラスタに使用されている各ノードの説明が含まれます。これにはデータ ノードの設定パラメータおよびクラスタのすべてのノード間接続の設定パラメータが含まれています。このファイルに表示されるセクションを素早く参照したり、各セクションにどんな種類の設定パラメータが含まれているかを調べるには [config.ini File \[894\]](#) を参照してください。

弊社では継続的にクラスタの設定を改善しており、また現在このプロセスを簡素化する作業に携わっています。弊社では以前のバージョンとの互換性維持に努めていますが、互換性の無い変更が行われる場合もあります。そのような場合には弊社では変更によって以前のバージョンとの互換性が無くなる旨を事前niクラスタのユーザーに連絡するつもりです。お客様がそのような変更気付いても弊社で文書化していない場合、[質問またはバグの報告](#) にある指示に従ってそれを MySQL のバグ データベースに報告をお願いします。

基本的な設定例

MySQL Cluster をサポートするには **my.cnf** を以下の例に基づいて更新する必要があります。ここに示すオプションを **config.ini** ファイルに使用したオプションと混同しないように注意してください。実行ファイルを実行する際にコマンドラインでこれらのパラメータを指定することもできます。

```

# my.cnf
# example additions to my.cnf for MySQL Cluster

```



```
# (valid in MySQL 5.1)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(接続文字列に関する詳細は、[クラスタの 接続文字列](#) を参照してください。)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

重要前述のように `mysqld` プロセスを `my.cnf` ファイルの `[MYSQLD]` にある `ndbcluster` および `ndb-connectstring` パラメータで実行すると、`CREATE TABLE` あるいは `ALTER TABLE` ステートメントをクラスタを実際に始めるまでは実行できなくなります。または、これらのステートメントはエラーが表示されて失敗します。これは設計によります。

クラスタの `my.cnf` ファイルの個別の `[mysql_cluster]` セクションを使用して設定を読み込んですべての実行ファイルで使用することもできます。

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

`my.cnf` ファイルで設定できる他の `NDB` 変数については [システム変数](#) を参照してください。

設定ファイルはデフォルトで `config.ini` の名前になっています。設定ファイルは起動時に `ndb_mgmd` で読み込まれどこにでも配置できます。そのロケーションと名前は `ndb.mgmd` のコマンドラインにある `--config-file=path_name` を使用して指定できます。設定ファイルが指定されていない場合、`ndb_mgmd` がデフォルトで現在の動作中のディレクトリにある `config.ini` ファイルを読み込もうとします。

現在設定ファイルは INI フォーマットで、最初にセクションの見出しのあるセクション (角括弧で括られている) で構成され、その後に適切なパラメータ名や値が続きます。標準の INI フォーマットと違うところはパラメータ名や値がコロン (':') および等号記号('=') で分離できることです。もう一箇所違うところはセクションがセクション名で独自に認識されないことです。その代わり、独自のセクション (同じ種類の異なる 2 つのノードなど) はセクション内のパラメータとして指定された独自の ID で認識されます。

デフォルト値は殆どのパラメータに定義され、`config.ini` で指定することもできます。デフォルト値のセクションを作成するには、単に単語 `DEFAULT` をセクション名に追加するだけです。例えば、`[NDBD]` セクションは特定のデータノードに適用されるパラメータを含み、`[NDBD DEFAULT]` セクションはすべてのデータノードに適用されるパラメータを含みます。すべてのデータノードが同じデータのメモリ容量を使用するものと仮定します。それらをすべて設定するには、`DataMemory` ラインを含む `[NDBD DEFAULT]` セクションを作成してデータのメモリ容量を指定します。

最低の設定でも、設定ファイルはクラスタに使用されるコンピュータやノードを定義し、それらのコンピュータのこれらのノードが属します。1 台のマネジメント サーバー、2 台のデータノードおよび 2 台の MySQL サーバーで構成した簡素な設定ファイルの例をここに示します。

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com
```

```
[NDBD DEFAULT]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster
```

```
[NDB_MGMD]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster
```

```
[NDBD]
HostName= ndbd_2.mysql.com
```

```
[NDBD]
HostName= ndbd_3.mysql.com
```

```
[MYSQLD]
[MYSQLD]
HostName= mysqld_5.mysql.com
```

各ノードはそれぞれ独自のセクションを [config.ini](#) に持つ必要があります。例えば、このクラスタには 2 台のデータノードがあるので、前述の設定ファイルにはこれらのノードを定義する 2 つの [\[NDBD\]](#) セクションがあります。

[config.ini](#) ファイルのセクション

以下のリストに示すように、[config.ini](#) 設定ファイルで利用できる異なる 6 つのセクションがあります。

- [\[COMPUTER\]](#): クラスタのホストを定義します。これは実際の MySQL クラスタの設定には必要ありませんが、大きなクラスタを設定する際に使用すると便利です。詳細については、[クラスタ コンピュータの定義](#) をご参照してください。
- [\[NDBD\]](#): クラスタ データノード (ndbd プロセス) の定義します。詳細は、[Defining Data Nodes](#) を参照してください。
- [\[MYSQLD\]](#): クラスタ MySQL サーバーノード (SQL または API ノードとも呼ばれている) を定義します。SQL ノード設定の説明については [SQL および他の API ノードの定義](#) を参照してください。
- [\[MGM\]](#) あるいは [\[NDB_MGMD\]](#): クラスタ マネジメント サーバー (MGM) ノードを定義します。MGM ノードの設定に関する情報は [マネジメント サーバーの定義](#) を参照してください。
- [\[TCP\]](#): TCP/IP がデフォルトの接続プロトコルの場合のクラスタ ノード間の TCP/IP 接続を定義します。通常、[\[TCP\]](#) あるいは [\[TCP DEFAULT\]](#) セクションは、クラスタが自動的にこれを行うので MySQL Cluster の設定には必要はありませんが、クラスタにより提供されたデフォルトをオーバーライドする際に必要になる場合があります。利用できる TCP/IP の設定パラメータおよびその使用方法については [Cluster TCP/IP Connections](#) を参照してください。([直接接続を使用した TCP/IP の接続](#) にもこの TCP/IP に関する情報を載せてあります。)
- [\[SHM\]](#): ノード間の共有メモリの接続を定義します。MySQL 5.1 では、それはデフォルトで有効になっていますが現在はまだ試験段階です。SHM のインターコネクトの説明に付いては [共有メモリ接続](#) を参照してください。
- [\[SCI\]](#): クラスタノード間のスケーラブル コヒーレント インターフェースの接続を定義します。そのような接続に MySQL Cluster のデистриビューションには含まれていないが、自由に入手できるソフトウェアと特定のハードウェアが必要です。SCI のインターコネクトに関する詳細は [SCI トランスポート接続](#) を参照してください。

各セクションに対して [DEFAULT](#) の値を定義できます。すべてのクラスタのパラメータは重要なパラメータで、[my.cnf](#) あるいは [my.ini](#) ファイルで指定されたパラメータとは異なります。

クラスタの [接続文字列](#)

MySQL Cluster マネジメント サーバー ([ndb_mgmd](#)) を除いて、MySQL Cluster の一部を構成する各ノードはマネジメント サーバーのロケーションをポイントする接続文字列 が必要です。この接続文字列はマネジメント サーバーへの接続の確立およびクラスタのノードの役割に基づいた他のタスクの実行に使用されます。接続文字列の構文は以下のようになります。

```
<connectstring> :=
  [<nodeid-specification>,<host-specification>,<host-specification>]

<nodeid-specification> := node_id

<host-specification> := host_name[:port_num]
```

`node_id` は 1 より大きい整数で `config.ini` のノードを認識します。`host_name` は文字列で有効なインターネットのホスト名あるいは IP アドレスを表します。`port_num` は整数で TCP/IP ポート番号を意味します。

```
example 1 (long): "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short): "myhost1"
```

すべてのノードは `localhost:1186` を他に無い場合デフォルトの接続文字列として使用します。`port_num` が接続文字列に無い場合、デフォルトのポートは 1186 です。このポートはこの目的のために IANA に割り当てられているので、常にネットワークで利用できる状態でなければなりません (詳細は <http://www.iana.org/assignments/port-numbers> 参照)。

複数の `<host-specification>` 値を入力すると、いくつかの冗長マネジメント サーバーを指定できます。クラスタのノードは指定された順序で各ホストのマネジメントサーバーに接続が確立されるまで接続を試みます。

接続を指定する多くの異なる方法があります。

- 各実行ファイルにはそれ自身のコマンドラインのオプションがあり、起動時にマネジメント サーバーを指定します。(それぞれの実行ファイルについてはその説明書を参照してください。)
- 接続文字列をマネジメント サーバー `my.cnf` ファイルの `[mysql_cluster]` セクションに置くことで接続文字列をクラスタのすべてのノードに一度に設定することもできます。
- 以前のバージョンへの互換性について打ては、同じ構文を使用して他の 2 つのオプションが利用できます。
 1. `NDB_CONNECTSTRING` 環境変数を設定して接続文字列を含みます。
 2. 各実行ファイルに接続文字列を書きそれを `Ndb.cfg` 名のテキスト ファイルに入れこのファイルを実行ファイルの起動ディレクトリに入れます。

しかし、これらは現在あまり利用されていないため新しいインストールには使用されません。

接続文字列の指定で推奨している方法は接続文字列をコマンドラインに設定するかあるいは各実行ファイルの `my.cnf` ファイルで設定します。

クラスタ コンピュータの定義

コンピュータ `[COMPUTER]` セクションはシステムの各ノードのホスト名を定義する必要がなくなるだけで特に重要ではありません。以下のパラメータはすべて必要です。

- `Id`

これは整数値で、設定ファイルのホスト コンピュータを参照するために使用します。この ID はノード ID とは同じではありません。

- `HostName`

これはコンピュータのホスト名あるいは IP アドレスです。

マネジメント サーバーの定義

`[NDB_MGMD]` セクションはマネジメント サーバーの振る舞いを設定するために使用します。`[MGM]` が別名として使用されます。この 2 つのセクション名は同じです。以下のリストのすべてのパラメータはオプションで省略されるとデフォルトの値になります。注:`ExecuteOnComputer` または `HostName` パラメータのどちらも存在しない場合、デフォルトの値 `localhost` がその両方の値に使用されます。

-

Id

クラスタの各ノードにはそれぞれ一意の ID があり、1 ~ 63 の整数値で表されます。この ID はすべての内部のクラスタ メッセージでノードを示すために使用されます。

- ### ExecuteOnComputer

これは `config.ini` ファイルの `[COMPUTER]` セクションで定義されたコンピュータの中の 1 台のコンピュータの `id` セットの意味します。

- ### PortNumber

これはポート番号でこれによりマネジメント サーバーが設定要求およびマネジメント コマンドを受け取ります。

- ### HostName

このパラメータを指定するとマネジメント ノードが常駐するコンピュータのホスト名を定義します。`localhost` 以外のホスト名を指定するには、このパラメータあるいは `ExecuteOnComputer` のいずれかが必要です。

- ### LogDestination

このパラメータはクラスタのログインの情報をどこに送るかを指定します。この点に関しては `— FILE` をデフォルトとして `CONSOLE`、`SYSLOG`— および `FILE` の 3 つのオプションがあります。

- `CONSOLE` はログを `stdout` に出力します。

```
CONSOLE
```

- `SYSLOG` はログを `syslog` ファシリティに送ります。可能な値はこれら `auth`、`authpriv`、`cron`、`daemon`、`ftp`、`kern`、`lpr`、`mail`、`news`、`syslog`、`user`、`uucp`、`local0`、`local1`、`local2`、`local3`、`local4`、`local5`、`local6` あるいは `local7` の内の 1 つです。

注:すべてのファシリティが必ずしもすべてのオペレーティング システムでサポートされる必要はありません。

```
SYSLOG:facility=syslog
```

- `FILE` クラスタのログ出力を同じマシンの通常のファイル送ります。以下の値を指定できます。

- `filename`:ログ ファイルの名前です。
- `maxsize`:ファイルがロールオーバーして新しいファイルに切り替わる前の最大サイズ (バイト表示)。これが起こると、古いログ ファイルはファイル名に `.N` が付いたファイル名に変わります。`N` はこの名前でまだ使用されていない次の番号になります。
- `maxfiles`:ログ ファイルの最大数です。

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

`FILE` パラメータのデフォルトの値は `FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6` です。`node_id` はノードの ID です。

以下に示すようにセミコロンで区切って複数のログ デスティネーションを指定できます。

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- ### ArbitrationRank

このパラメータはどのノードがアービトレーターとしての役割を果たすかを定義します。MGM ノードおよび SQL ノードのみがアービトレーターになれます。`ArbitrationRank` は以下の値の 1 つを取ることができます。

- **0:** このノードはアービトレーターとしては使用されません。
- **1:** このノードは優先度が高く、つまり、低い優先度のノードに対してアービトレーターとしての優先されます。
- **2:** は優先度の低いノードを意味し、優先度の高いノードがその目的に利用できないときにのみアービトレーターとして使用されます。

通常、マネジメント サーバーはその [ArbitrationRank](#) で 1 (デフォルトの値) でアービトレーターとして設定され、SQL ノードは 0 に設定されます。

- [ArbitrationDelay](#)

マネジメント サーバーのアービトレーションの要求への応答をミリ秒の数値で遅延させる整数値です。デフォルトではこの値は 0 です。通常はこの値を変更する必要はありません。

- [DataDir](#)

これはマネジメント サーバーの出力ファイルを格納するディレクトリを指定します。これらのファイルはクラスタ ログ ファイル、プロセス出力ファイル、およびデーモンのプロセス ID (PID) ファイルを含んでいます。(ログファイルでは、このロケーションはこの項で以前説明したように [FILE](#) パラメータを [LogDestination](#) に設定すると書き換えられます。

このパラメータのデフォルトの値はディレクトリで、その中に [ndb_mgmd](#) があります。

Defining Data Nodes

[\[NDBD\]](#) および [\[NDBD DEFAULT\]](#) セクションはクラスタのデータノードの振る舞いを設定するために使用されます。バッファ サイズ、プール サイズ、タイムアウトなどを管理する多くのパラメータがあります。唯一必須のパラメータは:

- [ExecuteOnComputer](#) あるいは [HostName](#) のいずれかは、[\[NDBD\]](#) セクションで定義される必要があります。
- パラメータ [NoOfReplicas](#) はそれがすべてのクラスタ データ ノードに共通なため、[\[NDBD DEFAULT\]](#) セクションで定義する必要があります。

ほとんどのデータ ノードのパラメータは [\[NDBD DEFAULT\]](#) セクションで設定されます。ローカル値を設定できる明示的に指定されたパラメータのみが [\[NDBD\]](#) セクションで変更できます。存在する場合、[HostName](#)、[Id](#) および [ExecuteOnComputer](#) はローカルの [\[NDBD\]](#) セクションで定義され、[config.ini](#) の他のセクションでは定義されません。換言すれば、これらのパラメータは 1 つのデータ ノード固有のものです。

メモリの使用およびバッファ サイズに影響を及ぼすパラメータは、[K](#)、[M](#)、あるいは [G](#) を 1024、1024×1024、あるいは 1024×1024×1024 の単位を示す接尾辞として使用できます。(例えば、[100K](#) は 100 × 1024 = 102400 の意味です。)パラメータ名および値は現在ケース センシティブです。

データ ノードの認識

[Id](#) 値 (つまり、データ ノードの識別子) はノードが起動されたときあるいは設定ファイルでコマンドラインに割り当てることができます。

- [Id](#)

これはすべてのクラスタの内部メッセージのノードのアドレスとして使用されるノード ID です。これは 1 ~ 63 までの整数です。クラスタの各ノードは一意的 ID を持つ必要があります。

- [ExecuteOnComputer](#)

これは [\[COMPUTER\]](#) セクションで定義されたコンピュータの [Id](#) セットです。

- [HostName](#)

このパラメータを指定するとデータ ノードが常駐するコンピュータのホスト名を定義します。[localhost](#) 以外のホスト名を指定するには、このパラメータあるいは [ExecuteOnComputer](#) のいずれかが必要です。

- [ServerPort](#) (旧式)

クラスタの各ノードは他のノードに接続するためにポートを使用しています。このポートはまた接続設定段階の非 TCP トランスポーターに使用されています。デフォルトのポートは同じコンピュータ上の 2 つのノードが同じポート番号を受信しないようにダイナミックに割り当てられているため、通常このパラメータの値を指定する必要はありません。

- [NoOfReplicas](#)

このグローバル パラメータは [\[NDBD DEFAULT\]](#) セクションでのみ設定され、クラスタに保持された各テーブルのレプリカを番号を定義します。このパラメータはまたノード グループのサイズを指定します。ノード グループはすべて同じ情報を保持した一連のノードです。

ノード グループは明示的に形成されます。最初のノード グループは最も低いノード ID を持つデータ ノードのセットで形成され、次のノード グループは次に最も低いノード ID のデータ セットで形成されます。参考例として、[NoOfReplicas](#) が 2 に設定された 4 つのデータ ノードがあるとしたします。その 4 つのデータ ノードのノード ID を 2, 3, 4 および 5 とします。すると最初のノード グループはノードの 2 および 3 から形成され、2 番目のノード グループのノードは 4 と 5 になります。同じノード グループのノードは同じコンピュータには使用しないようにクラスタを設定する必要があります。というのは、1 つのハードウェアの不具合がクラスタ全体のクラッシュにつながるからです。

ノード ID が提供されていない場合、データノードの順序はノード グループの決定要素になります。明示の割り当てがされるされないに拘わらず、それらはマネジメント クライアントの [SHOW](#) ステートメントの出力に表示されます。

[NoOfReplicas](#) にはデフォルトの値はありません。最大の可能な値は 4 です。

重要このパラメータの値はクラスタのデータ ノード数に同等に分けられる必要があります。例えば、2 つのデータ ノードがあるとしたと、[NoOfReplicas](#) は 1 あるいは 2 のいずれかに同じで、2/3 および 2/4 は両方とも機能的な値になります。4 つのデータ ノードがあるとしたと、[NoOfReplicas](#) は 1、2、あるいは 4 に同じになります。

- [DataDir](#)

このパラメータはトレース ファイル、pid ファイルおよびエラーファイルを格納するディレクトリを指定します。

- [FileSystemPath](#)

このパラメータはメタデータ、REDO ログ、UNDO ログおよびデータ ファイルに作成されたすべてのファイルを格納するディレクトリを指定します。デフォルトは [DataDir](#) で指定されたディレクトリです。注:このディレクトリは [ndbd](#) プロセスが実行される前に存在する必要があります。

MySQL Cluster の推奨されるディレクトリの階層には [/var/lib/mysql-cluster](#) が含まれます。そこでノードのファイルシステムのディレクトリが作成されます。このサブディレクトリの名前にはノード ID が含まれます。例えば、ノード ID が 2 の場合、このサブディレクトリの名前は [ndb_2_fs](#) となります。

- [BackupDataDir](#)

このパラメータはバックアップを格納するディレクトリを指定します。省略された場合、デフォルトのバックアップ ロケーションは [FileSystemPath](#) パラメータで指定されたロケーションの下に [BACKUP](#) の名前のディレクトリになります。(上記参照。)

データメモリ、インデックスメモリ、および文字列メモリ

[DataMemory](#) および [IndexMemory](#) は [\[NDBD\]](#) パラメータで、実際のレコードおよびそれらのインデックスを保持するメモリ シグメントのサイズを指定します。これらの値を設定する際に、[DataMemory](#) および [IndexMemory](#) がどのよ

うに使われるかを知っておくことが重要です。なぜなら、それらはクラスタの実際の使用を反映して更新される必要があるからです。

- **DataMemory**

このパラメータはデータベースのレコードを保持するスペース (バイト表示) を定義します。この値で指定される全体量はメモリで割り当てられます。ですからマシンにそれを収容できる十分な物理メモリがあることが非常に重要です。

DataMemory で割り当てられたメモリは実際のレコードおよびインデックスの保持に使用されます。各レコードには 16 バイトのオーバーヘッドがあります。各レコードはそれが 32KB ページで 128 バイト ページのオーバーヘッドに保持されるのでさらに使用できないスペースが増えます。(下参照)。各レコードは 1 ページにしか保存されないで、各ページ毎に使用できないスペース少しずつあります。

MySQL 5.1 の可変サイズ テーブル属性により、データは **DataMemory** から割り当てられた個別のデータページに保持されます。可変長レコードには 4 バイトのオーバーヘッドの固定サイズ部分を使用し可変サイズ部分を参照します。可変サイズ部分には 2 バイトのオーバーヘッドと属性毎に 2 バイトのオーバーヘッドがあります。

最大のレコード サイズは現在 8052 バイトです。

DataMemory で定義されたメモリ スペースは、レコード毎に 10 バイトを使用する順序付けされたインデックスの保持にも使用されます。各テーブル行は順序づけされたインデックスで表されます。ユーザー間での共通のエラーはすべてのインデックスは **IndexMemory** で割り当てられたメモリに保持されるためと想定できるが、これがすべてではありません。プライマリ キーと一意のハッシュ インデックスのみがこのメモリを使用します。順序付けされたインデックスは **DataMemory** で割り当てられたメモリを使用します。しかし、プライマリ キーあるいは一意のハッシュ インデックスを作成すると インデックス作成ステートメントの **USING HASH** を指定しない限り同じキーで順序付けされたインデックスも作成されます。これはマネジメント クライアントの **ndb_desc -d db_name table_name** を実行してと検証できます。

DataMemory で割り当てられたメモリ スペースはテーブル フラグメントに割り当てられた 32KB のページで構成されます。各テーブルはクラスタにデータ ノードがあるため通常同じ数のフラグメントにパーティションされます。このように、各ノードに対して、**NoOfReplicas** で設定された同数のフラグメントがあります。

ページが割り当てられると、テーブルを削除する以外にフリーページのプールに戻ることは現在できません。(このことは **DataMemory** ページが一度所定のテーブルに割り当てられると他のテーブルで使用できないということの意味しています。)すべてのレコードが他の生きたノードから空のパーティションに挿入されるため、ノードのリカバリを実行するとパーティションを圧縮します。

DataMemory メモリ スペースはまた UNDO 情報を含んでいます。更新毎に、変更されないレコードのコピーが **DataMemory** で割り当てられます。順序付けされたテーブル インデックスに各コピーの参照があります。一意のハッシュ インデックスは一意のインデックスのカラムが更新されたときのみ更新されます。その場合、インデックス テーブルでの新しいエントリが挿入されその挿入によって古いエントリは削除されます。このため、クラスタを使用したアプリケーションの大きなトランザクションを扱うのに十分なメモリを割り当てることも重要です。いずれにしても、いくつかの大きなトランザクションを実行することは、以下の理由によって多くの小さなトランザクションを実行するのに対して優位性がある訳ではありません。

- 大きなトランザクションは小さなトランザクションより早い訳ではない
- 大きなトランザクションは失われるオペレーション数が増えるので、トランザクションが失敗した場合には繰返す必要がある
- 大きなトランザクションは多くのメモリを使用する

DataMemory のデフォルトの値は 80MB です。最小は 1 MB です。最大サイズはありませんが、現実的には制限に達したときプロセスがスワップしないように最大サイズを決める必要があります。制限はマシンで利用できる物理 RAM の容量およびオペレーティング システムがプロセスの実行に必要な容量によって決まります。32 ビットのオペレーティング システムは一般的にはプロセス毎では 2-4GB です。64 ビットのオペレーティング システムはさらに多くのメモリを使用できます。この理由により大きなデータベースの場合、64 ビットのオペレーティング システムの使用が望まれます。

- **IndexMemory**

このメモリは MySQL のハッシュ インデックスに使用されるストレージ量を管理します。ハッシュ インデックスは常にプライマリ キーのインデックス、独自のインデックス、および独自の制約に使用されます。プライマリ キーおよび独自のインデックスを定義する際、2 つのインデックスが作成され、その 1 つがすべての tuple アクセスおよびロックの取扱いに使用されるハッシュのインデックスです。それはまた独自の制約の強化にも使用されます。

ハッシュ インデックスのサイズはレコード毎に 25 バイトで、それにプライマリ キーのサイズが加わります。32 バイト以上のプライマリ キーには別に 8 バイト追加されます。

`IndexMemory` のデフォルトの値は 18MB です。最低は 1MB です。

StringMemory

このパラメータは例えばテーブル名などの文字列に使用されるメモリ容量に割り当てを決定し、`config.ini` の `[NDBD]` あるいは `[NDBD DEFAULT]` セクションで指定されます。0 および 100 の間の値は最大のデフォルトの値のパーセントで、テーブル数、最大のテーブル名のサイズ、最大の `.FRM` ファイル、`MaxNoOfTriggers`、最大のカラム名のサイズ、および最大のデフォルトのカラムの値などを含む多くの要素に基づいて算出されます。一般的には 1000 のテーブルを持つ MySQL Cluster の場合の最大のデフォルト値はおおよそ 5 MB にすると安全です。

100 より大きい値はバイト数を意味します。

デフォルトの値が 5 ー の場合、つまりデフォルトの最大の 5 パーセントあるいはおおよそ 5 KB です。(これが MySQL Cluster の以前のバージョンからの変更点です。)

ほとんどの環境で、そのデフォルト値で十分ですが、非常に大きなクラスタ テーブル (1000 あるいはそれ以上) の場合、エラー 773 が出る場合があります。文字列メモリで、`StringMemory config` のパラメータを変更してください。恒久的なエラー：スキーマのエラー で、この場合この値を「増やします。25 (25 パーセント) でも良いでしょう。これですべてのしかも最も極端な場合のエラーの再発を妨げる必要があります。

以下の例でテーブルにメモリがどのように使用され手いるか説明します。このテーブルの定義を考慮します。

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

各レコードには 12 バイトのデータと 12 バイトのオーバーヘッドがあります。無効なカラムを無くすと 4 バイトのオーバーヘッドを節約できます。さらに、カラム `a` と `b` にレコード毎におおよそ 10 バイト使用する順序付けされた 2 つのインデックスがあります。ベース テーブルにレコード毎におおよそ 29 バイト使用するプライマリ キーのハッシュ インデックスがあります。独自の制約はプライマリ キーとして `b` およびカラムとして `a` を持つ個別のテーブルにより課されます。この他のテーブルは `example` テーブルでさらにレコード毎に 29 バイトのインデックス メモリおよびレコード データに 8 バイトおよびオーバーヘッドに 12 バイト使用します。

このように、100 万のレコードでは、プライマリ キーと独自の制約のハッシュ インデックス処理に 58MB のインデックス メモリが必要です。さらに、ベース テーブルと独自のインデックス テーブル、および 2 つの順序付けされたインデックス テーブルのレコードに 64MB 必要です。

この様にハッシュ インデックスはかなりのメモリ スペースを必要としますが、その代わりに高速のデータ アクセスを提供します。それらはまた独自の制約を処理するために MySQL で使用されています。

現在、パーティション アルゴリズムはハッシュのみで順序付けされたインデックスは各ノードに対してローカルです。このように、順序付けされたインデックスは一般的には独自の制約の処理には使用できません。

`IndexMemory` および `DataMemory` の重要な点は、各ノード グループのデータベース サイズの合計はすべてのデータ メモリおよびすべてのインデックス メモリの合計であるということです。各ノード グループはレプリケート (複製) された情報の保持に使用されますので、2 つのレプリカを持つ 4 つのノードがあれば、2 つのノード グループがあることになります。このように、利用可能なデータ メモリの合計は各データ ノードに対して $2 \times \text{DataMemory}$ です。

`DataMemory` と `IndexMemory` を全てのノードに対して同じ値で設定するよう強くお勧めします。クラスタではデータの配布はすべてのノードで同一ですので、各ノードの最大利用可能スペースはクラスタで一番小さいノード スペースより大きくなることはできません。

[DataMemory](#) と [IndexMemory](#) は変更できますが、これらのいずれかを少なくすることは危険で、そうすることによってノードあるいは MySQL Cluster 全体がメモリ スペースの不足によって再起動できなくなります。これらの値を増やすことは容認できますが、そのようなアップグレードをする場合にはソフトウェアのアップグレードと同じ方法で、つまり設定ファイルのアップグレード、次に各データ ノードを順番に再起動してからマネジメント サーバーを再起動するようお勧めします。

アップグレードによって使用できるインデックス メモリの量は増えません。挿入は直ぐできます。しかし、行はトランザクションが実施されるまで実際は削除されません。

トランザクション パラメータ

これから説明する次の 3 つの [\[NDBD\]](#) パラメータは重要です。なぜなら、それらはシステムが処理する並列トランザクション数およびトランザクションのサイズに影響を及ぼすからです。[MaxNoOfConcurrentTransactions](#) はノードで可能な並列トランザクション数を設定します。[MaxNoOfConcurrentOperations](#) は更新段階およびあるいは同時ロック時のレコード数を設定します。

これらのパラメータはどちらも (特に [MaxNoOfConcurrentOperations](#) は) 特定の値をしかもデフォルトの値を使用しないで設定する ユーザーにとっては目標値になると思われます。デフォルトの値はこれらの値が過剰なメモリを使用しないように確認するために、小さなトランザクションを使用してシステムに設定されます。

- [MaxNoOfConcurrentTransactions](#)

クラスタのアクティブなそれぞれトランザクションはクラスタ ノードの 1 つにレコードを持つ必要があります。協調的なトランザクションはノード間で実行されます。クラスタのトランザクション レコードの合計数はクラスタの所定のノードのトランザクション数にノードを乗算した数になります。

トランザクション レコードは個々の MySQL サーバーに割り当てられます。通常は、少なくともクラスタのテーブルを使用している少なくとも 1 つのトランザクション レコードが接続毎に割り当てられます。このため、クラスタのトランザクション レコード数がクラスタの すべての MySQL サーバーに同時接続している数よりも多いことを確認する必要があります。

このパラメータは全てのクラスタ ノードに対し同じ値を設定する必要があります。

このパラメータを変更することは安全ではなく、変更することによってクラスタがクラッシュする場合があります。ノードがクラッシュすると、ノードの 1 つ (実際は一番最後までクラッシュしないで残ったノード) がクラッシュしたときにクラッシュしたノードで進行中のすべてのトランザクションの状態に戻します。ですからこのノードが失敗したノードの出来るだけ多くのレコードを持っていることが重要です。

デフォルトの値は 4096 です。

- [MaxNoOfConcurrentOperations](#)

このパラメータの値をトランザクション数やサイズに基づいて調整することはいいい考えです。少数のオペレーションをそれぞれあまり多くのレコードを使用しないでトランザクションを実行するときには、このパラメータを高く設定する必要はありません。多くのレコードを含む大きなトランザクションを実行するするときにはこのパラメータを高く設定する必要があります。

レコードは各トランザクション毎に記録され、トランザクション コーディネーターおよび実際の更新が行われるノードでクラスタのデータを更新します。このレコードにはロールバック、ロック キュー、およびその他の目的のための UNDO レコードの検索に必要なステート情報が含まれます。

このパラメータはクラスタのデータ ノード数で除算した、トランザクションで同時に更新されるレコードの数に設定する必要があります。例えば、4 つのデータ ノードを持ち 1,000,000 の同時更新をトランザクションで処理するクラスタでは、この値を $1000000 / 4 = 250000$ に設定する必要があります。

ロックを設定するクエリの読み込みでもオペレーション レコードが作成されます。ノードへの配布が完全でない場合にそれに対処するためにいくつかの予備のスペースが個々のノード内で割り当てられます。

クエリが独自のハッシュ インデックスを使用する場合、トランザクションで実際にレコード毎に使用される 2 つのオペレーション レコードがあります。最初のレコードはインデックス テーブルの読み込みを行い、2 番目はベース テーブルのオペレーションを処理します。

デフォルトの値は 32768 です。

このパラメータは個別に設定される 2 つの値を扱います。これらの最初の値はトランザクション コーディネーターに配置するトランザクション レコード数を指定します。2 番目の値はデータベースに対してローカルにするオペレーション レコード数を指定します。

8 つのノードを使用したクラスタで実行される非常に大きなトランザクションのトランザクション コーディネーターにはトランザクションでの読み込み、更新、削除に相当するオペレーション レコードが必要です。しかし、オペレーション レコードは 8 つのノードすべてで使用されます。このように、システムを 1 つの非常に大きなトランザクションに設定する必要がある場合には、2 つの部分の個別に設定するほうが良いでしょう。[MaxNoOfConcurrentOperations](#) はトランザクション コーディネーターのノード部分のオペレーション レコード数の算出に使用されます。

オペレーション レコードにはメモリ要件を考慮に入れることも重要です。これらはレコード毎に約 1KB 使用します。

- [MaxNoOfLocalOperations](#)

デフォルトでは、このパラメータは $1.1 \times \text{MaxNoOfConcurrentOperations}$ で算出されます。これはトランザクションがそれほど大きくない多くのトランザクションを同時に行うシステムに向いています。一度に非常に大きなトランザクションを扱う必要がある場合には、このパラメータを明示的に指定してデフォルト値をオーバーライドするのがいいでしょう。

トランザクションのテンポラリー ストレージ

次の [\[NDBD\]](#) パラメータ セットはクラスタ トランザクションの一部のステートメントを実行する際のテンポラリーのストレージを決定するために使用されます。すべてのレコードがステートメントが完了しクラスタが実行あるいはローカルバックを待っているときにリリースされます。

これらのパラメータのデフォルト値で殆どの状況をカバーします。しかし、多くの行数やオペレーションが絡むトランザクションのサポートが必要なユーザーはシステムの並列効果を高めるためにこれらの値を増やす必要がある場合があります。一方、トランザクション数が少ないアプリケーションのユーザーはメモリを節約するためにその値を下げるができます。

- [MaxNoOfConcurrentIndexOperations](#)

独自のハッシュ インデックスを使用したクエリでは、オペレーション レコードの別のテンポラリー セットがクエリの実行フェーズで使用されます。このパラメータセットはレコードのプール サイズを設定します。このように、レコードはクエリの一部を実行中にのみ割り当てられます。この部分が実行されるとすぐ、レコードがリリースされます。失敗や実行の処理に必要なステートは通常オペレーション レコードで扱われ、プール サイズはパラメータ [MaxNoOfConcurrentOperations](#) で設定されます。

このパラメータのデフォルトの値は 8192 です。ごく稀に独自のハッシュ インデックスを使用した極端に高い並列仕様においてはこの値を上げる必要があります。DBA がクラスタに高度な並列が要求されないことを確認できる場合、小さい値が可能でメモリを節約できます。

- [MaxNoOfFiredTriggers](#)

[MaxNoOfFiredTriggers](#) のデフォルトの値は 4000 です。これで殆どの状態に十分です。DBA がクラスタの並列仕様が高くないと確認できた場合、場合によっては値を下げることもできます。

独自のハッシュ インデックスに影響を及ぼすオペレーションが実行されるとレコードが作成されます。独自のハッシュ インデックスでテーブルにレコードを挿入あるいは削除するあるいは独自のハッシュ インデックスの一部の列を更新するとインデックス テーブルの挿入や削除が無効になります。その結果のレコードは完了するためにオペレーションが無効にした元のオペレーションを待つ間にこのインデックス テーブルのオペレーションを代わりにするために使用されます。このオペレーションは短命ですが、それでも一連の独自のハッシュ インデックスを含むベース テーブルで多くの並列書き込みオペレーションに対応するプールで多数のレコードを必要とします。

- [TransactionBufferMemory](#)

このパラメータで影響を受けたメモリはインデックス テーブルの更新や独自にインデックスを読み込むときに無効になったオペレーションの追跡に使用されます。このメモリはこれらのオペレーションのキーおよびカラムの情報を保持するために使用されます。このパラメータの値をデフォルトの値からの変更を必要とするケースは非常に稀です。

[TransactionBufferMemory](#) のデフォルトの値は 1MB です

通常の読み込み書き込みのオペレーションは同様のバッファを 1 つ使用します。その使用はもっと短命です。コンパイル時間のパラメータ [ZATTRBUF_FILESIZE](#) ([ndb/src/kernel/blocks/Dbtc/Dbtc.hpp](#) に表示) は 4000 × 128 バイト (500KB) に設定します。キー情報の同様のバッファは、[ZDATABUF_FILESIZE](#) ([Dbtc.hpp](#) に表示) は 4000 × 16 = 62.5KB のバッファ スペースを含みます。[Dbtc](#) はトランザクションのコーディネーションを扱うモジュールです。

スキャンとバッファリング

[Dblqh](#) モジュール ([ndb/src/kernel/blocks/Dblqh/Dblqh.hpp](#) にあります) に読み込みと更新に影響を及ぼす追加の [\[NDBD\]](#) パラメータがあります。これらはデフォルトで 10000 × 128 バイト (1250KB) および [ZDATABUF_FILE_SIZE](#)、デフォルトで 10000*16 バイト (およそ 156KB) のバッファ スペースに設定された [ZATTRINBUF_FILESIZE](#) を含みます。現在までのところ、これらのコンパイル時間制限を増やすべきだという弊社のユーザーおよび弊社の広範なテストでの結果ありません。

- [MaxNoOfConcurrentScans](#)

このパラメータはクラスタで実行される並列スキャン数の管理のために使用されます。各トランザクション コーディネーターはこのパラメータに定義された数の並列スキャンを処理します。各スキャンのクエリは並列の全てのパーティションをスキャンすることで実行できます。各パーティション スキャンはパーティションがあるノードのスキャン レコード、このパラメータの値であるレコード数にノード数を乗算したレコード数を使用します。クラスタは [MaxNoOfConcurrentScans](#) スキャンをクラスタの全てのノードと同時に維持する必要があります。

スキャンは実際には 2 つのケースで実行されます。この最初のケースはクエリを扱うハッシュあるいは順序付けされたインデックスが存在しないとき、クエリがテーブルのフル スキャンを実行することで実行されます。2 番目のケースはクエリをサポートするハッシュ インデックスが無くて順序付けされたインデックスがある場合にスキャンが実行されます。順序付けされたインデックスを使用するということは並列の範囲スキャンを実行することを意味します。その順序はローカルのパーティションにのみ維持されるので、すべてのパーティションにインデックスのスキャンが行う必要があります。

[MaxNoOfConcurrentScans](#) のデフォルトの値は 256 です。最大値は 500 です。

このパラメータはトランザクションのコーディネーターでの可能なスキャン数を指定します。ローカル スキャン数が提供されていない場合、[MaxNoOfConcurrentScans](#) およびシステムのデータ ノード数の積によって計算されます。

- [MaxNoOfLocalScans](#)

多くのスキャンが完全に並列化されない場合にローカルのスキャン レコード数を指定します。

- [BatchSizePerLocalScan](#)

このパラメータは多くの同時スキャン オペレーションを扱うロック レコード数の計算に使用されます。

デフォルトの値は 64 です。この値は SQL ノードで定義された [ScanBatchSize](#) と強い関連があります。

- [LongMessageBuffer](#)

これは内部のバッファで個々のノード内およびノード間でメッセージを渡すために使用されます。これを変更する必要は殆ど考えられませんが、設定はできます。デフォルトでは 1MB に設定されます。

ロギングとチェックポイント

これらの [\[NDBD\]](#) パラメータはログおよびチェックポイントの振る舞いを管理します。

- [NoOfFragmentLogFiles](#)

このパラメータはノードの REDO (やり直し) ログ ファイル数を設定し、この様に REDO ログにスペースが割り当てられます。REDO ログ ファイルはリングに環状に構成されますので、そのセットの最初および最後のログ ファイル (それぞれ「頭」 および「尻尾」 ログ ファイルとも呼ばれる) が一致しないように設定することが非常に重要です。これらが互いにあまりにも近づくと、新しいログ レコードのスペースが足りないためにノードが更新に関わるすべてのトランザクションを中断させる場合があります。

REDO ログ ファイルはそのログ レコードが挿入されてから 3 回のローカル チェックポイントが完了するまで削除できません。チェックポイントの頻度はこの章の別の場所で説明したように、それ自身の一連の設定パラメータで決定されます。

これらのパラメータの相互作用およびその設定については [ローカル チェックポイントのパラメータの設定](#) で説明しています。

デフォルトのパラメータ値は 8 ですので 8 セットの 4 16MB ファイルで合計 512MB になります。換言すれば、REDO ログ スペースは 64MB のブロックに割り当てられる必要があります。非常に多くの更新が要求される場合には、REDO ログに十分なスペースを提供するには [NoOfFragmentLogFiles](#) の値は 300 あるいはそれ以上に高く設定する必要があります。

チェックポイントが遅く、データベースへの書き込み数がログ ファイルが一杯になりログの尻尾がリカバリの悪化なしにカットできなくなるほど多い場合、すべての更新トランザクションは内部のエラーコード 410 ([Out of log file space temporarily](#)) によって中断されます。この状態はチェックポイントが完了しログの尻尾が前進できるようになるまで続きます。

重要このパラメータは「稼働中」には変更できません。--initial を使用してノードを再起動する必要があります。この値をクラスタの稼働中の変更を希望する場合には、動作中のノードを再起動します。

- [MaxNoOfOpenFiles](#)

このパラメータはオープン ファイルに内部スレッド割り当て上限を設定します。このパラメータの変更が必要な状況が発生した場合にはバグ として報告をお願いします。

デフォルトの値は 40 です。

- [MaxNoOfSavedMessages](#)

このパラメータは古い値が書き換えられるまでのトレース ファイルの最大数を設定します。トレース ファイルは、どのような理由であれ、ノードがクラッシュすると生成されます。

デフォルトは 25 トレース ファイルです。

Metadata Objects

次の [\[NDBD\]](#) パラメータ セットはメタデータ オブジェクトのプール サイズを定義し、インデックス、イベント、およびクラスタ間のレプリケーションに使用される属性、テーブル、インデックス、およびトリガ オブジェクトの最大数の定義に使用されます。これらはクラスタへの単なる「助言」で、ここに指定されない値は以下のデフォルトの値になります。

- [MaxNoOfAttributes](#)

クラスタで定義される属性数を定義します。

デフォルトの値は 1000 で、最大の可能な値は 32 です。最大は 4294967039。すべてのメタデータはサーバーで完全にレプリケート (複製) されるため各属性はノード毎に約 200 バイトのストレージを使用します。

[MaxNoOfAttributes](#) を設定する前に、将来実行を希望するであろう [ALTER TABLE](#) ステートメントを事前に用意することが重要です。これは クラスタ テーブルで [ALTER TABLE](#) を実行中に、元のテーブル 3 倍の属性が使用されるからです例えば、テーブルが 100 の属性を必要し、その変更を後で希望する場合、[MaxNoOfAttributes](#) の値を 300 に設定する必要があります。希望するすべてのテーブルを問題無く作成できるとして、経験則では念のために

一番大きなテーブルで 2 倍の属性を [MaxNoOfAttributes](#) に追加します。パラメータを設定した後に実際の [ALTER TABLE](#) を試してこの数字が十分であるか検証できます。その数字でうまくいかない場合、[MaxNoOfAttributes](#) を元の値の数倍に増やしてももう一度試してみます。

- [MaxNoOfTables](#)

各テーブル、独自のハッシュ インデックス、および順序付けられたインデックスにテーブル オブジェクトを割り当てられます。このパラメータは全体としてクラスタにテーブル オブジェクトの最大数を設定します。

[BLOB](#) のデータ タイプを持つ各属性に対して、ほとんどの [BLOB](#) データを保持するために予備のテーブルが使用されます。これらのテーブルはテーブルの合計数を決める場合に考慮する必要があります。

このパラメータのデフォルトの値は 128 です。最小は 8 で最大は 1600 です。各テーブル オブジェクトはおおよそノード毎に 20KB を使用します。

- [MaxNoOfOrderedIndexes](#)

クラスタの順番付けされたインデックスに対し、何にインデックスするかおよびそのストレージ セグメントを記述したオブジェクトが 1 つ割り当てられます。デフォルトでは、そのように定義された各インデックスはまた順番付けされたインデックスを定義します。それぞれの独自のインデックスおよびプライマリ キーは順序付けされたインデックスおよびハッシュ インデックスの両方を持っています。

このパラメータのデフォルトの値は 128 です。各オブジェクトはノード毎におおよそ 10KB のデータを使用します。

- [MaxNoOfUniqueHashIndexes](#)

プライマリ キー以外の各独自のインデックスに対して、独自のキーをインデックスの付いたテーブルのプライマリ キーマップする特別なテーブルが割り当てられます。デフォルトでは、順序付けされたインデックスもまた各独自のインデックスに定義されます。これを防ぐには、独自のインデックスを定義する際に [USING HASH](#) オプションを指定する必要があります。

デフォルトの値は 64 です。各インデックスはノード毎におおよそ 15KB 使用します。

- [MaxNoOfTriggers](#)

内部のトリガの更新、挿入、および削除が各独自のハッシュ インデックスに割り当てられます。(これは 3 つのトリガが各独自のハッシュ インデックスに作成されることを意味します。)しかし、順序付けされた インデックスにはシングルのトリガ オブジェクトのみ必要です。バックアップもまた 3 つのトリガ オブジェクトをクラスタの各通常のテーブルに使用します。

クラスタ間のレプリケーションもまた内部のトリガを使用します。

このパラメータはクラスタのトリガ オブジェクトの最大数を設定します。

デフォルトの値は 768 です。

- [MaxNoOfIndexes](#)

このパラメータは MySQL 5.1 は使用されない方向にあります。代わりに [MaxNoOfOrderedIndexes](#) および [MaxNoOfUniqueHashIndexes](#) を使用する必要があります。

このパラメータは独自のハッシュ インデックスのみで使用されます。このプールではクラスタで定義された各独自のインデックスに対しレコードが 1 つ必要です。

このパラメータのデフォルトの値は 128 です。

Boolean パラメータ

データ ノードの振る舞いも boolean 値に使用された [\[NDBD\]](#) パラメータ セットの影響を受けます。これらのパラメータはそれぞれ 1 あるいは Y に設定すると [TRUE](#)、および 0 あるいは N に設定すると [FALSE](#) を指定できます。

- ### LockPagesInMainMemory

Solaris および Linux を含む多くのオペレーティング システムで、プロセスをメモリにロックしてディスクへのスワップを回避することができます。これはクラスタのリアルタイム特性を保証するために使用されます。

MySQL 5.1.15 以降の場合、このパラメータは 0、1、あるいは 2 のいずれかの整数値を取ります。それぞれ以下の役割があります。

- 0: ロックを無効にします。これはデフォルトの値です。
- 1: メモリをプロセスに割り当てた後にロックを実行します。
- 2: メモリをプロセスに割り当てる前にロックを実行します。

以前は、このパラメータは Boolean でした。0 あるいは false はデフォルトに設定で、ロックを無効にしました。1 あるいは true はメモリが割り当てられたあとのプロセスのロックを有効にしました。重要MySQL 5.1.15 以降では、true あるいは false をもはやこのパラメータに使用できません。以前のバージョンからアップグレードする場合には、その値を 0、1、あるいは 2 に変更する必要があります。

- ### StopOnError

このパラメータはエラーが発生した場合に **ndbd** プロセスを終了するかあるいは自動的に再起動させるかを指定します。

この機能はデフォルトで有効になっています。

- ### Diskless

MySQL クラスタのテーブルを テーブルがディスクにチェックポイントされずロギングも発生しないDisklessに指定できます。そのようなテーブルは主メモリにのみ存在します。ディスク無しのテーブルを使用することによってそれらのテーブルのテーブルあるいはレコードのどちらもクラッシュの影響を受けないことを意味しています。しかし、ディスク無しモードを起動中に、ディスク無しコンピュータで **ndbd** を実行できます。

重要この機能によりクラスタ 全体 をディスク無しモードで稼働できます。

この機能を有効にすると、クラスタのオンラインバックアップは無効になります。さらに、クラスタの部分的な起動が出来なくなります。

Diskless はデフォルトで無効になっています。

- ### RestartOnErrorInsert

この機能はデバッグ バージョンを作成中にのみアクセスでき、そこでエラーをコードの個々のブロックにテストの一部として挿入できます。

この機能はデフォルトで無効になっています。

タイムアウト、インターバル、およびディスク ページングの管理

クラスタ データ ノードの様々な操作でのタイムアウトやインターバルを指定する多くの **[NDBD]** パラメータがあります。ほとんどのタイムアウトの値はミリ秒で指定されます。値がミリ秒でない場合にはその都度説明します。

- ### TimeBetweenWatchDogCheck

主スレッドの無限ループのある地点でのスタックを避けるために、「監視」スレッドが主スレッドをチェックします。このパラメータはチェック間のミリ秒数を指定します。プロセスが 3 回のチェックの後でも同じ状態が続くようであれば、監視スレッドがそれを停止します。

パラメータは試験用あるいはローカル条件を採用するために簡単に変更できます。それは特にノード単位で指定する意味もないのだがノード単位で指定できます。

デフォルトのタイムアウトは 4000 ミリセカンド (4 秒) です。

- [StartPartialTimeout](#)

このパラメータはクラスタの初期化ルーチンが起動される前のクラスタのすべてのノードが起動するまでの待機時間を指定します。このタイムアウトは必要に応じてクラスタの部分的起動を避けるために使用されます。

デフォルトの値は 30000 ミリセカンド(30 秒) です。0 でタイムアウトを無効にします。この場合クラスタはすべてのノードが利用できる段階で起動します。

- [StartPartitionedTimeout](#)

クラスタが [StartPartialTimeout](#) ミリセカンドの待機後に起動の用意が出来ても、まだパーティションの状態である場合、クラスタはこのタイムアウトが経過するまで待ちます。

デフォルトのタイムアウトは 60000 ミリセカンド (60 秒) です。

- [StartFailureTimeout](#)

データノードがこのパラメータで指定された時間内にその起動シーケンスを完了できない場合、そのノードの起動は失敗します。このパラメータを 0 (デフォルトの値) に設定するとデータ ノードのタイムアウトが適用されないことを意味します。

0 以外の値の場合、このパラメータはミリセカンドで測定されます。極端い大きな量のデータを含むデータ ノードには、このパラメータを大きくします。例えば、数ギガ バイトのデータを含むデータノードの場合、ノードの再起動を実行するには 10-15 分 (つまり、600000 から 1000000 ミリセカンド) が必要になります。

- [HeartbeatIntervalDbDb](#)

失敗したノードを見つける主な方法の 1 つにハートビートを使用する方法があります。このパラメータはハートビート信号の送受信の頻度を指定します。続けて 3 回ハートビート インターバルに失敗した場合、そのノードはデッドを宣言されます。このように、ハートビート メカニズムを使用した不具合発見の最大の回数はハートビート インターバルの 4 倍です。

デフォルトのハートビート インターバルは 1500 ミリセカンド (1.5 秒) です。このパラメータを大幅に変更したりまたはノード間で大きく違わないようにします。1 つのノードが 5000 ミリセカンドを使用しそれを監視するノードが 1000 ミリセカンドを使用した場合、明らかにノードは直ちにデッドを宣言されます。このパラメータはオンラインのソフトウェアのアップグレード時に変更できますが、ほんの小さな増分だけです。

- [HeartbeatIntervalDbApi](#)

各データノードはハートビート信号を各 MySQL サーバー (SQL ノード) に送って接続されていることを確認します。MySQL サーバーがハートビートを時間内に送信できない場合「デッド」を宣言され、その場合すべての実行中のトランザクションは完了してすべてのリソースがリリースされます。SQL ノードは以前の MySQL インスタンスで始められたすべての操作が完了するまで再接続できません。この測定の 3 回のハートビート基準は [HeartbeatIntervalDbDb](#) の説明と同じです。

デフォルトのインターバルは 1500 ミリセカンド (1.5 秒) です。インターバルは個々のデータ ノード別にすることも出来ます。というのは、各ノードは接続された MySQL サーバーを監視し、他のすべてのデータ ノードから独立しているからです。

- [TimeBetweenLocalCheckpoints](#)

このパラメータはその例外で新しいローカルチェックポイントの起動までの待機時間を設定しません。これはむしろ、ローカルチェックポイントは比較的少ない更新が実行されるクラスタでは実行されないことを確認するために

使用されます。更新回数が多い多くのクラスタでは、おそらく新しいローカル チェックポイントは前のチェックポイントが完了すると直ぐに開始されるからです。

前のローカル チェックポイントが開始されてから実行されたすべての書き込み操作のサイズが追加されます。このパラメータもその例外で 4 バイトの単語の基数 2 対数で指定されますので、デフォルト値の 20 は 4MB (4×2^{20}) の書き込み操作を意味し、21 は 8MB で、8 GB の書き込み操作に相当する最大値 31 まで続きます。

クラスタのすべての書き込み操作はが一緒に追加されます。`TimeBetweenLocalCheckpoints` を 6 あるいはそれ以下に設定することはローカルのチェックポイントが、クラスタの負荷に関係なく休みなく継続的に実行されることを意味します。

- `TimeBetweenGlobalCheckpoints`

トランザクションがコミットされると、データがミラーされているすべてのノードの主メモリでコミットされます。しかし、トランザクションのログレコードはそのコミットの一部としてディスクにフラッシュされません。この振る舞いの背後にある論法はトランザクションを少なくとも 2 台の自律型ホスト マシン上で成功裏に実行するには持続性に於ける合理的な基準を満たす必要があるからです。

最悪の場合—クラスタの完全なクラッシュ—でも適切に処理されることを確認することも重要です。これを保証するには、所定のインターバル内に実行されるすべてのトランザクションをグローバル チェックポイントに設定すると、それがディスクにフラッシュされコミットされたトランザクションのセットとして考慮されます。換言すれば、コミット プロセスの一部として、トランザクションがグローバル チェックポイントに組み入れられます。後でこのグループのログレコードがディスクにフラッシュされ、トランザクション グループ全体がクラスタのすべてのコンピュータ上のディスクでコミットされます。

このパラメータはグローバル チェックポイント間のインターバルを定義します。デフォルトは 2000 ミリセカンドです。

- `TimeBetweenInactiveTransactionAbortCheck`

タイムアウトの処理はこのパラメータで指定されたインターバル毎に各トランザクションのタイマーを一度チェックすることで実行されます。この様に、このパラメータを 1000 ミリセカンドに設定すると、すべてのトランザクションは 1 秒毎にタイムアウトをチェックします。

デフォルトの値は 1000 ミリセカンド (1 秒) です。

- `TransactionInactiveTimeout`

このパラメータはトランザクションが中断される前の同じトランザクションのオペレーション間に許容された最大の経過時間を表します。

このパラメータのデフォルトはゼロ (タイムアウト無し) です。トランザクションのロックをあまり長くしないことを確認する必要があるリアルタイムのデータベースでは、このパラメータは非常に小さい値に設定する必要があります。単位はミリセカンドです。

- `TransactionDeadlockDetectionTimeout`

ノードがトランザクションに関わるクエリを実行するとき、そのノードは継続する前にクラスタの他のノードが応答するまで待ちます。応答の失敗は以下の理由のいずれかで起こります。

- そのノードが「デッド」状態にある
- オペレーションがロック キューの状態にある
- アクションの実行をリクエストしたノードが過負荷にある

このタイムアウトのパラメータはトランザクション コーディネーターがトランザクションの中断にいたるまでの別のノードによるクエリを実行するまでの待機時間を示したもので、ノードの中断処理およびデッドロック検知の両方に重要です。その設定が高すぎるとデッドロックおよびノード中断を含む状態において望まない振る舞いを引き起こします。

デフォルトのタイムアウトは 1200 ミリセカンド (1.2 秒) です。

- [DiskSyncSize](#)

データをローカルのチェックポイント ファイルにフラッシュする前に保持するする最大バイト数です。

デフォルトの値は 4M (4 メガバイト) です。

このパラメータは MySQL 5.1.12 に追加されています。

- [DiskCheckpointSpeed](#)

ローカル チェックポイント中に転送されるデータ量をバイト/秒で表したものです。

デフォルトの値は 10M (10 メガバイト/秒) です。

このパラメータは MySQL 5.1.12 に追加されています。

- [DiskCheckpointSpeedInRestart](#)

ローカル チェックポイント中に再起動の操作の一部とし転送されるデータ量をバイト/秒で表したものです。

デフォルトの値は 100M (100 メガバイト/秒) です。

このパラメータは MySQL 5.1.12 に追加されています。

- [NoOfDiskPagesToDiskAfterRestartTUP](#)

ローカル チェックポイント実行中に、アルゴリズムがすべてのデータページをディスクにフラッシュします。軽減策なしで単にそのように素早く行うのは過剰な負荷をプロセッサ、ネットワーク、およびディスクにかける場合があります。書き込み速度を管理するために、このパラメータは 100 ミリセカンド毎の書き込みページ数を指定します。ここでは 1 「ページ」は 8KB に定義されています。このパラメータは 80KB/秒単位で指定され、[NoOfDiskPagesToDiskAfterRestartTUP](#) を 20 の値に設定するとローカル チェックポイント中のデータページでのディスクへの書き込み速度 1.6MB/秒を意味します。この値には UNDO ログ レコードのデータ ページへの書き込みが含まれています。つまり、このパラメータはデータ メモリの書き込み数の制限を扱っています。インデックス ページの UNDO ログ レコードはパラメータ [NoOfDiskPagesToDiskAfterRestartACC](#) で処理されます。(インデックス ページに関する情報 [IndexMemory](#) は入力を参照してください。)

要するに、このパラメータはローカル チェックポイントをどれだけ速く実施するかを指定します。それは [NoOfFragmentLogFiles](#)、[DataMemory](#)、および [IndexMemory](#) と一緒に使用されます。

このパラメータのインターラクシオンおよびそれらの適切な値の選択肢に関する情報は、[ローカル チェックポイントのパラメータの設定](#) を参照してください。

デフォルトの値は 40 (3.2MB/秒のデータ ページ) です。

注:このパラメータは MySQL 5.1.6 では使用頻度が下がっています。それはMySQL 5.1.12 およびそれ以降のバージョンは、[DiskCheckpointSpeed \[909\]](#) および [DiskSyncSize \[909\]](#) を使用しているからです。.

- [NoOfDiskPagesToDiskAfterRestartACC](#)

このパラメータは [NoOfDiskPagesToDiskAfterRestartTUP](#) と同じ単位を使用し同じ役割を果たしますが、インデックス メモリからインデックス ページの書き込み速度を制限します。

このパラメータのデフォルト値は 20 (1.6MB/秒 のインデックス メモリ ページ) 。

注:このパラメータは MySQL 5.1.6 では使用頻度が下がっています。それはMySQL 5.1.12 およびそれ以降のバージョンは、[DiskCheckpointSpeed \[909\]](#) および [DiskSyncSize \[909\]](#) を使用しているからです。.

- [NoOfDiskPagesToDiskDuringRestartTUP](#)

このパラメータは [NoOfDiskPagesToDiskAfterRestartTUP](#) および [NoOfDiskPagesToDiskAfterRestartACC](#) と同じ役割を果たしますが、ノードが再起動されて時にノードで実行されるローカルのチェックポイントに関してのみそのように振舞います。ローカル チェックポイントは常にすべてのノードの再起動の一部として実行されます。ノードの再起動中にはノードで実行される作業量が少ないため他の場合より高速でディスクに書き込みできます。

このパラメータはデータ メモリで書き込まれたページをカバーしています。

デフォルトの値は 40 (3.2MB/秒) です。

注:このパラメータは MySQL 5.1.6 では使用頻度が減っています。それはMySQL 5.1.12 およびそれ以降のバージョンは、[DiskCheckpointSpeedInRestart \[909\]](#) および [DiskSyncSize \[909\]](#) を使用しているからです。.

- [NoOfDiskPagesToDiskDuringRestartACC](#)

ノードの再起動時のローカル チェックポイント中にディスク書き込まれるインデックスのメモリ ページ数を管理します。

[NoOfDiskPagesToDiskAfterRestartTUP](#) および [NoOfDiskPagesToDiskAfterRestartACC](#) と同じで、このパラメータ値は 100 ミリセカンド (80KB/秒) で書き込まれた 8KB ページを表します。

デフォルトの値は 20 (1.6MB/秒) です。

注:このパラメータは MySQL 5.1.6 では使用頻度が減っています。。それは MySQL 5.1.12 およびそれ以降のバージョンは、[DiskCheckpointSpeedInRestart \[909\]](#) および [DiskSyncSize \[909\]](#) を使用しているからです。.

- [ArbitrationTimeout](#)

このパラメータはデータ ノードのアービトレーターからアービトレーション メッセージへの応答の待機時間を指定します。この待機時間を過ぎると、ネットワークは切断されたものとみなされます。

デフォルトの値は 1000 ミリセカンド (1 秒) です。

バッファリングとロギング

[RedoBuffer](#)

すべての更新はログに記録する必要があります。REDO ログはこれらの更新をシステムが再起動されたときにはいつでも実行できるようにします。NDB リカバリ アルゴリズムはデータの「ファジー」チェックポイントを UNDO ログと一緒に使用し、REDO ログを使用してすべての変更を修復ポイントまで再現します。

[RedoBuffer](#) は REDO ログが書き込まれるバッファのサイズを設定します。デフォルトでは 8MB です。このバッファは REDO ログのレコードをディスクに書き込む際のファイル システムへのフロント エンドとして使用されます。このバッファが小さすぎると、[NDB ストレージ エンジン](#)がエラーコード 1221 ([REDO log buffers overloaded](#) を発行します)。

最小値は 1MB です。ノードがディスク無しの状態で実行されると、ディスクの書き込みが [NDB ストレージ エンジン](#)のファイルシステム抽出レイヤによって「フェイク」されるため、これらのパラメータはペナルティ無しで最小値に設定されます。

重要ローリングの再起動時にこのパラメータを下げることは安全ではありません。

注:MySQL の以前のバージョンの [UndoIndexBuffer](#) および [UndoDataBuffer](#) パラメータは MySQL 5.1 ではもはや必要なくなりました (あったにしても)。

ログ メッセージの管理

クラスターの管理においては、様々なイベントに関して [stdout](#) に送られたログ メッセージ数を管理できることが非常に重要です。各イベント カテゴリでは、16 の実行可能なイベント レベル (0~15 番まで) があります。イベントのレポートを所定のイベント カテゴリのレベル 15 設定するとそのカテゴリのすべてのイベント レポートが [stdout](#) に送られます。0 に設定するとそのカテゴリにはイベント レポートが無いことを意味します。

デフォルトでは、起動メッセージのみが `stdout` に送られます。残りのイベントレポートレベルのデフォルトは 0 に設定されます。この理由はこれらのメッセージもまたマネジメント サーバーのクラスタ ログに送られるからです。

相似性のレベルをマネジメント クライアントに設定してどのイベント レベルをクラスタ ログでレコードするかを決めることができます。

- [LogLevelStartup](#)
プロセスの起動時に生成されたイベントのレポート レベルです。
デフォルトのレベルは 1 です。
- [LogLevelShutdown](#)
ノードの優雅なシャットダウンの一部として生成されたイベントのレポート レベルです。
デフォルトのレベルは 0 です。
- [LogLevelStatistic](#)
プライマリ キーの読み込み、更新数、挿入数、バッファ使用に関する情報など統計的なイベントのレポート レベルです。
デフォルトのレベル 0 です。
- [LogLevelCheckpoint](#)
ローカルおよびグローバル チェックポイントにより生成されたイベントのレポート レベル。
デフォルトのレベルは 0 です。
- [LogLevelNodeRestart](#)
ノード再起動時に生成されたイベントのレポート レベルです。
デフォルトのレベルは 0 です。
- [LogLevelConnection](#)
クラスタ ノード間の接続で生成されたイベントのレポート レベルです。
デフォルトのレベル 0 です。
- [LogLevelError](#)
全体としてクラスタによるエラーおよび警告により生成されたイベントのレポート レベルです。これらのエラーはノード障害に結びつくものではないが報告すべきだと思われるエラーです。
デフォルトのレベルは 0 です。
- [LogLevelInfo](#)
クラスタの一般的な状態に関する情報に生成されたイベントのレポート レベルです。
デフォルトのレベルは 0 です。

バックアップ パラメータ

この項で説明した[\[NDBD\]](#) パラメータはオンライン バックアップの実行用に確保するメモリ バッファを定義します。

- [BackupDataBufferSize](#)

バックアップの作成においては、データをディスクに転送する 2 種類のバッファがあります。バックアップのデータ バッファはノード テーブルのスキャンにより記録されたデータを書き込みむために使用されます。このバッファが [BackupWriteSize](#) (以下参照) で指定されたレベルまで書き込まれると、そのページはディスクに転送されます。データをディスクに書き込んでいる間に、バックアップ プロセスではスペースがなくなるまでこのバッファに書き込みを続けます。スペースが無くなると、バックアップ プロセスがスキャンを停止し、ディスクへに書き込みによってメモリが増えてスキャンを継続できるようになるまで待ちます。

デフォルトの値は 2MB です。

- [BackupLogBufferSize](#)

バックアップ ログのバッファはバックアップの実行中にそれがすべてのテーブルの書き込みのログ生成に使用されることを除いてバックアップ データ バッファと同様の役割を果たします。同様の原則が、バックアップ ログ バッファにスペースが無いときを除いてバックアップ データ バッファと同じようにこれらの ページの書き込みに適用されて、そのバックアップは失敗します。その理由により、バックアップ ログ バッファのサイズはバックアップ中の書き込みの負荷を処理するだけの十分な大きさが必要です。 [クラスタ バックアップの設定](#) 参照。

このパラメータのデフォルトの値は殆どアプリケーションに十分なものでなければなりません。実際のところ、バックアップの失敗はディスクへの書き込み速度が遅いよりはバックアップ ログ バッファが満杯になることによりよく起こります。ディスクのサブシステムがアプリケーションによる書き込み負荷対応するように設定されていない場合、クラスタは所定の作業を実行できなくなります。

クラスタ ノードの設定ではディスクあるいはネットワーク接続をボトルネックにするよりはプロセッサがボトルネックになるように構成するほうが望まれます。

デフォルトの値は 2MB です。

- [BackupMemory](#)

このパラメータは単純に [BackupDataBufferSize](#) と [BackupLogBufferSize](#) の合計です。

デフォルトの値は 2MB + 2MB = 4MB です。

重要[BackupDataBufferSize](#) および [BackupLogBufferSize](#) の合計が 4MB 以上の場合、このパラメータは [config.ini](#) ファイルでその合計として明示的に設定される必要があります。

- [BackupWriteSize](#)

このパラメータはバックアップ ログおよびバックアップ データ バッファによってディスクに書き込まれたメッセージのデフォルトサイズを指定します。

デフォルトの値は 32KB です。

- [BackupMaxWriteSize](#)

このパラメータはバックアップ ログおよびバックアップ データ バッファによってディスクに書き込まれたメッセージの最大サイズを指定します。

デフォルトの値は 256KB です。

重要これらのパラメータを指定する際は、以下の関係が真であることが必要です。そうでない場合、データ ノードは起動しません。

- [BackupDataBufferSize](#) >= [BackupWriteSize](#) + 188KB
- [BackupLogBufferSize](#) >= [BackupWriteSize](#) + 16KB
- [BackupMaxWriteSize](#) >= [BackupWriteSize](#)

SQL および他の API ノードの定義

`config.ini` ファイルの `[MYSQLD]` および `[API]` のセクションは MySQL サーバー (SQL ノード) およびクラスタ データへのアクセスに使用される他のアプリケーション (API ノード) を定義します。表示されたパラメータのどれも必要ありません。コンピュータ名およびホスト名の提供が無い場合、ホストのいずれかが SQL あるいは API ノードを使用できます。

一般的には、`[MYSQLD]` セクションは SQL インターフェースをクラスタに提供している MySQL サーバーを示すために使用され、`[API]` セクションはクラスタのデータにアクセスしている `mysqld` プロセスよりはアプリケーションに使用されますが、2つの名前は実際は同義語です。例えば、`[API]` セクションの SQL サーバーとしての役割を果たしている MySQL サーバーにパラメータを記入できます。

- `Id`

`Id` 値はすべてのクラスタの内部メッセージのノードを識別するために使用されます。それは 1 ~ 63 の整数で、クラスタ無いのすべてのノード ID に対して一意出なければなりません。

- `ExecuteOnComputer`

これは `config.ini` ファイルの `[COMPUTER]` セクションで定義されたコンピュータ (ホスト) の `id` セットのことを意味します。

- `HostName`

このパラメータを指定することで SQL ノード (API ノード) が常駐するコンピュータのホスト名を定義します。`localhost` 以外のホスト名を指定するには、このパラメータあるいは `ExecuteOnComputer` のいずれかが必要です。

- `ArbitrationRank`

このパラメータはどのノードがアービトラーターとしての役割を果たすかを定義します。MGM ノードおよび SQL ノードの両方がアービトラーターになれます。0 の値は所定のノードがアービトラーターとして使用されないことを意味し、1 の値はアービトラーターとしての優先度が高く、2 の値は優先度が低いことをいみます。通常の設定ではマネジメント サーバーをアービトラーターとして使用し、`ArbitrationRank` を 1 (デフォルト) に、すべての SQL ノードに対しては 0 に設定します。

- `ArbitrationDelay`

この値を 0 (デフォルト) 以外に設定するとアービトレーション要求に対するアービトラーターの応答は指定されたミリ秒数によって遅延されます。通常この値を変更する必要はありません。

- `BatchByteSize`

インデックスのフル テーブルあるいは範囲スキャンに翻訳されたクエリに対しては、レコードを適切なサイズで取り出すことがベスト パフォーマンスには重要です。適切なサイズをレコード数 (`BatchSize`) およびバイト (`BatchByteSize`) の両方で設定できます。実際のバッチ サイズは両方のパラメータで制限されます。

クエリが実行される速度はこのパラメータの設定によって 40% 以上変化します。今後のリリースでは、MySQL Server にクエリの種類に基づいた適切なバッチ サイズに関するパラメータの設定を取り入れます。

このパラメータはバイト数で測定され、そのデフォルトの値は 32KB です。

- `BatchSize`

このパラメータはレコード数で測定されそのデフォルトの設定値は 64 です。最大サイズは 992 です。

-

MaxScanBatchSize

バッチ サイズは各データ ノードから送られる各バッチのサイズです。多くのスキャンは MySQL Server を並列の多くのノードからデータを余り多く受け取らないようにするために並列で行われます。このパラメータはすべてのノードに対してバッチの合計を制限します。

このパラメータのデフォルトの値は 256KB です。最大のサイズは 16MB です。

ここに表示している [MySQL クライアントの SHOW STATUS](#) を使用して Cluster SQL として実行されている MySQL サーバーから情報を入手できます。

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_cluster_node_id | 5 |
| Ndb_config_from_host | 192.168.0.112 |
| Ndb_config_from_port | 1186 |
| Ndb_number_of_storage_nodes | 4 |
+-----+-----+
4 rows in set (0.02 sec)
```

これらの Cluster システムのステータス変数に関する情報は、[ステータス変数](#) を参照してください。

Cluster TCP/IP Connections

TCP/IP は MySQL Cluster の接続を確立するデフォルトのトランスポート メカニズムです。通常は Cluster が自動的に各ノード間、各データ ノード間および MySQL サーバーノード、および各データ ノード間さらには各データ ノードとマネジメント サーバーの接続を設定しますので接続を定義する必要はありません。(この規則に対する 1 つの例外については、[直接接続を使用した TCP/IP の接続](#) を参照)`config.ini` ファイルの `[TCP]` セクションでクラスタのノード間の TCP/IP 接続を明示的に定義しています。

デフォルトの接続パラメータをオーバーライドするときのみ接続を定義する必要があります。そのような場合、少なくとも `NodeId1`、`NodeId2`、および変更するパラメータを定義する必要があります。

これらのパラメータのデフォルトの値を `[TCP DEFAULT]` セクションで設定することで変更することもできます。

- [NodeId1](#)、[NodeId2](#)

2 つのノード間での接続を認識するには設定ファイルの `[TCP]` セクションでそれらのノード ID を提供します。これらは [SQL および他の API ノードの定義](#) の説明にあるようにそれぞれのこれらのノードに対して同じ `Id` 値をしています。

- [SendBufferMemory](#)

TCP トランスポーターはオペレーティング システムに送信呼び出しを実行する前にバッファを使用してすべてのメッセージを保存します。このバッファが 64KB になると、そのコンテンツを送信します。これらはまた一通りのメッセージが完了すると送信されます。一時的な過負荷状態を処理するために、大きな送信バッファを定義することも出来ます。送信バッファのデフォルトのサイズは 256KB です。

- [SendSignalId](#)

配布されたメッセージのダイアグラムを調べるには、各メッセージを認識する必要があります。このパラメータを `Y` に設定すると、メッセージ ID がネットワーク上に転送されます。この機能は生産ビルドでデフォルトによって無効にできます。`-debug` ビルドで有効にします。

- [Checksum](#)

このパラメータはブール パラメータ (`Y` あるいは `1` に設定して有効にする。無効にするには `N` あるいは `0`) に設定します。デフォルトでそれを無効に出来ます。有効にすると、すべてのメッセージのチェックサムが送信場ファに格納される前に計算されます。この機能によりメッセージが送信バッファで待機している間に、あるいは転送メカニズムで転化してないか確認します。

- **PortNumber** (OBSOLETE)
これは他のノードの接続の確認に使用されるポート番号を正式に指定します。このパラメータはもはや使用されません。
- **ReceiveBufferMemory**
データを TCP/IP ソケットから受信するときにバッファのサイズを指定するために使用されます。メモリを節約するとき以外に、このパラメータをそのデフォルトの値の 64KB からめったに変更する必要はありません。

直接接続を使用した TCP/IP の接続

データノード間の直接接続を使用してクラスタを設定するにはクラスタ `config.ini` の `[TCP]` セクションで接続されているデータノードの交差 IP アドレスを明示的に指定する必要があります。

以下の例では、それぞれマネジメント サーバー、SQL ノード、および 2 つのデータノードを持つ少なくとも 4 台のホストを持つクラスタを説明します。そのクラスタは全体として LAN の `172.23.72.*` サブネットに常駐します。通常のネットワーク接続に加えて、2 つのデータノードを標準の交差ケーブルを使用して直接接続し、以下の範囲の `1.1.0.*` の IP アドレスを使用して直接お互いに通信します。

```
# Management Server
[NDB_MGMD]
Id=1
HostName=172.23.72.20

# SQL Node
[MYSQLD]
Id=2
HostName=172.23.72.21

# Data Nodes
[NDBD]
Id=3
HostName=172.23.72.22

[NDBD]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[TCP]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

`HostNameN` パラメータ、そこでは `N` は整数で、TCP/IP の直接接続を指定する際にのみに使用されます。

データノード間の直接接続を使用することによってデータノードをスイッチ、ハブ、ルータなどの Ethernet デバイスを経由できるのでクラスタの全体的な効率が改善され、よってクラスタのレイテンシーを下げます。2 つ以上のデータノードでの直接接続の利点をこのように最大限に活用するには、同じノードグループの各データノードとそれぞれの他のデータノードを直接接続することが重要です。

共有メモリ接続

MySQL Cluster は共有メモリの転送を試みて可能であればそれを自動的に設定します。`config.ini` ファイルの `[SHM]` セクションでクラスタのノード間の共有メモリの接続を明示的に定義します。

共有メモリを接続方法として明示的に定義する際は、少なくとも `NodeId1`、`NodeId2` および `ShmKey` を定義する必要があります。すべての他のパラメータには殆どのケースでよく動作するデフォルトの値があります。

重要 SHM 機能は現在まだ実験的なものです。それは正式には これまでリリースされた 5.1 を含む MySQL シリーズではサポートされていません。このことはその使用はお客様自身の決断であるいは弊社のフリーのリソース (フォーラム、メールリスト) を使用してお客様の特定のケースに適切に使用できるか決める必要があります。

•

- [ShmKey](#)

共有メモリのセグメントを設定する場合には、整数として表されるノード ID を使用して個別に共有メモリのセグメントを通信に使用するために認識します。デフォルトの値はありません。

- [ShmSize](#)

各 SHM 接続には共有メモリのセグメントがあり、ノード間のメッセージは送信者のよって送られ受信者によって読まれます。このセグメントのサイズは [ShmSize](#) によって認識されます。デフォルトの値は 1MB です。

- [SendSignalId](#)

配布したメッセージのパスを辿るには、各メッセージに一意の識別子を付ける必要があります。このパラメータを Y に設定するとこれらのメッセージ ID をネットワーク上でも転送できます。この機能は生産ビルドでデフォルトによって無効にできます。-debug ビルドで有効にします。

- [Checksum](#)

このパラメータは boolean (Y/N) パラメータでデフォルトで無効にできます。有効にすると、すべてのメッセージのチェックサムが送信バッファ格納される前に計算されます。

この機能により送信バッファで待機中のメッセージの破損を防ぎます。またそれによって転送中のデータの破損をチェックすることもできます。

SCI トランスポート接続

config.ini ファイルの [SCI] でセクションで クラスタ ノード間の SCI (Scalable Coherent Interface) 接続を明示的に定義しています。MySQL での SCI トランスポートの使用は --with-ndb-sci=/your/path/to/SCI を使用して MySQL バイナリがビルドされたときのみサポートされています。path は最低の lib を含み SICI バイナリおよびヘッダーファイルを含む include ディレクトリを含んだディレクトリを指す必要があります。(SCI の詳細は [MySQL Cluster での高速インターコネクトを使用する](#) を参照してください。)

その他に、SCI には特別なハードウェアが必要です。

ndbd プロセス間の通信のときのみ SCI トランスポートを使用するよう強くお勧めします。また SCI トランスポートを使用する際は ndbd プロセスが有効であること確認します。このため、SCI トランスポートは少なくとも ndbd プロセスによる 2 つの専用の CPU を搭載したマシン上でのみ使用する必要があります。少なくとも ndbd プロセス毎に少なくとも 1 つの CPU を使用し、オペレーティング システムを動作させるための少なくとも 1 つの CPU 予備として残しておきます。

- [Host1ScId0](#)

これにより最初のクラスタ ノード (NodeId1 による識別) 上で SCI ノード ID を識別します。

- [Host1ScId1](#)

SCI トランスポートをノード間の個別のネットワークを使用する 2 枚の SCI カード間のフェールオーバーに設定できます。これによってノード ID および最初のノードに使用される 2 番目の SCI カードを認識します。

- [Host2ScId0](#)

これにより 2 番目のクラスタ ノード (NodeId2 により識別される) の SCI ノード ID を認識します。

- [Host2ScId1](#)

フェールオーバーを提供するために 2 つの SCI カードを使用する時は、このパラメータが 2 番目のノードに使用される 2 番目の SCI カードを認識します。

- [SharedBufferSize](#)

各 SCI トランスポートにはノード間の通信に使用される共有メモリのセグメントがあります。このセグメントのサイズをフォルト値の 1MB に設定すると殆どのアプリケーションに十分です。その値を小さく設定すると並列で多くの挿入を行う場合問題が出る場合があります。共有バッファが小さ過ぎると `ndbd` プロセスでクラッシュする場合があります。

- [SendLimit](#)

SCI メディアの前にある小さなバッファが SCI ネットワークにメッセージを転送する前にメッセージを保存します。デフォルトでは 8KB に設定されています。弊社のベンチマークでは 64KB でパフォーマンスはベストですが、16KB ではこの数パーセントにしか届きませんが、それを 8KB 以上にした場合でも殆ど利点はありませんでした。

- [SendSignalId](#)

配布したメッセージをトレースするには各メッセージを独自に認識する必要があります。このパラメータを `Y` に設定すると、メッセージ ID がネットワーク上に転送されます。この機能は生産ビルドでデフォルトによって無効にできます。-debug ビルドで有効にします。

- [Checksum](#)

このパラメータは boolean 値で、デフォルトで無効にできます。[Checksum](#) が有効にすると、チェックサムは送信バッファに格納される前にすべてのメッセージに対して計算されます。この機能により 2 番目の送信バッファで待機中のメッセージの破損を防ぎます。またそれによって転送中のデータの破損をチェックすることもできます。

クラスタ設定パラメータの概要

次の 3 項ではクラスタの機能を管理する `config.ini` ファイルに使用される MySQL Cluster の設定パラメータの概要テーブルを提供します。各テーブルでは、パラメータの種類およびその利用できるデフォルト、最小、および最大クラスタ ノードを含むプロセスの種類 (`ndbd`、`ndb_mgmd`、および `mysqld`) の 1 つのパラメータを説明します。

またどのような再起動 (ノードの再起動あるいはシステムの再起動) — が必要で、所定の設定パラメータの値を変更するために `--initial` — の再起動しなければならないかどうかについて述べます。このパラメータは各テーブルの Restart Type カラムにあり、リストにある値の 1 つを含んでいます。

- **N**:ノードの再起動
- **IN**:最初のノードの再起動
- **S**:システムの再起動
- **IS**:最初のシステムの再起動

ノードを再起動あるいは最初のノードの再起動には、すべてのクラスタのデータノードが順番に再起動 (ルーリング再起動とも言う) する必要があります。**N** あるいは **IN** の印の付いたクラスタの設定パラメータををオンラインで — 変更できます。つまり、クラスタをシャットダウンしないで — 変更できます。最初のノードの再起動をするには各 `ndbd` プロセスを `--initial` オプションで再起動する必要があります。

システムの再起動にはシステムを完全にシャットダウンしクラスタ全体を再起動します。最初のシステムの再起動する際はクラスタのバックアップを取り、シャットダウンした後にクラスタのファイルシステムを消去し、次に再起動の後にバックアップを保存します。

どのクラスタを再起動する場合でも、クラスタの更新された設定パラメータの値を読ませるために、すべてのクラスタ マネジメント サーバーを再起動する必要があります。

重要数値クラスタのパラメータは一般的には問題なく上げることができます。しかし、それらを上げる場合には比較的に小さな増分で調整しながら徐々に上げるのがよいでしょう。しかし、そのようなパラメータの値を下げる場合には — 特にメモリの使用およびディスク スペース — に関わるものは簡単に変更しないで、慎重に計画してテストをした上で変更することをお勧めします。さらに、簡単なノードの再起動を使用して上げられるメモリおよびディスクの使用に関するパラメータ一般的には最初のノードの再起動を下げる必要があります。

なぜなら、これらのパラメータの中には 1 つ以上のクラスタの設定に使用されるものがあり、1 つ以上のテーブルに使用されている場合があるからです。

(これらのテーブルの最大値として使用される 4294967039 —— は $2^{32} - 2^8 - 1$ と等価です。)

データノードの設定パラメータ

以下のテーブルは MySQL Cluster のデータノードを設定する `config.ini` ファイルの `[NDBD]` あるいは `[NDB_DEFAULT]` セクションで使用されるパラメータに関する情報を提供します。これらのパラメータの詳細な説明および補足情報に関しては、[Defining Data Nodes](#) を参照してください。

再起動のタイプ カラム値

- **N**: ノードの再起動
- **IN**: 最初のノードの再起動
- **S**: システムの再起動
- **IS**: 最初のシステムの再起動

これらの略語の説明は [クラスタ設定パラメータの概要](#) を参照してください。

パラメータ名	タイプ/ 単位	デフォルト値	最小値	最大値	再起 動の タイ プ
ArbitrationTimeout [910]	ミリ セカ ンド	1000	10	4294967039	N
BackupDataBufferSize [912]	バイ ト	2M	0	4294967039	N
BackupDataDir [898]	文字 列	FileSystemPath/ BACKUP	N/A	N/A	IN
BackupLogBufferSize [912]	バイ ト	2M	0	4294967039	N
BackupMemory [912]	バイ ト	4M	0	4294967039	N
BackupWriteSize [912]	バイ ト	32K	2K	4294967039	N
BackupMaxWriteSize [912]	バイ ト	256K	2K	4294967039	N
BatchSizePerLocalScan [903]	整数	64	1	992	N
DataDir [898]	文字 列	/var/lib/mysql-cluster	N/A	N/A	IN
DataMemory [899]	バイ ト	80M	1M	1024G (利用できるシ ステムの RAM および IndexMemory のサイ ズによる)	N
Diskless [906]	true false (1 0)	0	0	1	IS
ExecuteOnComputer [897]	整数				
FileSystemPath [898]	文字 列	DataDir に指定された 値	N/A	N/A	IN

HeartbeatIntervalDbApi [907]	ミリ セカ ンド	1500	100	4294967039	N
HeartbeatIntervalDbDb [907]	ミリ セカ ンド	1500	10	4294967039	N
HostName [897]	文字 列	localhost	N/A	N/A	S
Id [897]	整数	なし	1	63	N
IndexMemory [899]	バイ ト	18M	1M	1024G (利用できるシ ステムの RAM および DataMemory のサイズ による)	N
LockPagesInMainMemory [906]	MySQL 5.1.15の 場 合: 整 数; 以 前 の: true false (1 0)		0	1	N
LogLevelCheckpoint [911]	整数	0	0	15	IN
LogLevelConnection [911]	整数	0	0	15	N
LogLevelError [911]	整数	0	0	15	N
LogLevelInfo [911]	整数	0	0	15	N
LogLevelNodeRestart [911]	整数	0	0	15	N
LogLevelShutdown [911]	整数	0	0	15	N
LogLevelStartup [911]	整数	1	0	15	N
LogLevelStatistic [911]	整数	0	0	15	N
LongMessageBuffer [903]	バイ ト	1M	512K	4294967039	N
MaxNoOfAttributes [904]	整数	1000	32	4294967039	N
MaxNoOfConcurrentIndexOperations [902]	整数	512K	0	4294967039	N
MaxNoOfConcurrentOperations [902]	整数	32768	32	4294967039	N
MaxNoOfConcurrentScans [903]	整数	256	2	500	N
MaxNoOfConcurrentTransactions [902]	整数	4096	32	4294967039	N
MaxNoOfFiredTriggers [902]	整数	4000	0	4294967039	N
MaxNoOfIndexes [905] (専用 — 代わりに MaxNoOfOrderedIndexes あるいは MaxNoOfUniqueHashIndexes を使用する)	整数	128	0	4294967039	N
MaxNoOfLocalOperations [902]	整数	UNDEFINED	32	4294967039	N
MaxNoOfLocalScans [903]	整数	UNDEFINED	32	4294967039	N
MaxNoOfOrderedIndexes [905]	整数	128	0	4294967039	N
MaxNoOfSavedMessages [904]	整数	25	0	4294967039	N

クラスタ設定パラメータの概要

MaxNoOfTables [905]	整数	128	8	4294967039	N
MaxNoOfTriggers [905]	整数	768	0	4294967039	N
MaxNoOfUniqueHashIndexes [905]	整数	64	0	4294967039	N
NoOfDiskPagesToDiskAfterRestart [909] (減少 MySQL 5.1.6 の場合)	整数 (20) = 20 * 80KB = (8KB 1.6MB/秒) ページ 数/100 ミリ セカ ンド)	20	1	4294967039	N
NoOfDiskPagesToDiskAfterRestart [909] (減少 MySQL 5.1.6 の場合)	整数 (40) = 40 * 80KB = (8KB 3.2MB/秒) ページ 数/100 ミリ セカ ンド)	40	1	4294967039	N
NoOfDiskPagesToDiskDuringRestart [909] (減少 MySQL 5.1.6 の場合)	整数 (20) = 20 * 80KB = (8KB 1.6MB/秒) ページ 数/100 ミリ セカ ンド)	20	1	4294967039	N
NoOfDiskPagesToDiskDuringRestart [909] (減少 MySQL 5.1.6 の場合)	整数 (40) = 40 * 80KB = (8KB 3.2MB/秒) の ペー ジ 数/100 ミリ セカ ンド)	40	1	4294967039	N
DiskCheckpointSpeed [909] (MySQL 5.1.12 に追加)	整数 (バ イト 数/ 秒)	10M	1M	4294967039	N
DiskCheckpointSpeedInRestart [909] (MySQL 5.1.12 に追加)	整数 (バ イト 数/ 秒)	100M	1M	4294967039	N
DiskSyncSize [909] (MySQL 5.1.12 に追加)	整数 (バ イト 数)	4M	32K	4294967039	N
NoOfFragmentLogFiles [904]	整数	8	1	4294967039	IN

NoOfReplicas [898]	整数	なし	1	4	IS
RedoBuffer [910]	バイト	8M	1M	4294967039	N
RestartOnErrorInsert [906] (デバッグビルドのみ)	true false (1 0)	0	0	1	N
ServerPort [898] (廃版)	整数	1186	0	4294967039	N
StartFailureTimeout [907]	ミリセカンド	0	0	4294967039	N
StartPartialTimeout [907]	ミリセカンド	30000	0	4294967039	N
StartPartitionedTimeout [907]	ミリセカンド	60000	0	4294967039	N
StopOnError [906]	true false (1 0)	1	0	1	N
TimeBetweenGlobalCheckpoints [908]	ミリセカンド	2000	10	32000	N
TimeBetweenInactiveTransactionAbortCheckpoints [908]	ミリセカンド	4000	1000	4294967039	N
TimeBetweenLocalCheckpoints [908]	整数 (ベース2対数としての4バイトの単語数)	20 (= $4 * 2^{20} = 4\text{MB}$ 書き込み)	0	31	N
TimeBetweenWatchDogCheck [906]	ミリセカンド	4000	70	4294967039	N
TransactionBufferMemory [902]	バイト	1M	1K	4294967039	N
TransactionDeadlockDetectionTimeout [902]	ミリセカンド	1200	50	4294967039	N
TransactionInactiveTimeout [908]	ミリセカンド	0	0	4294967039	N

マネジメント ノード設定パラメータ

以下のテーブルは MySQL Cluster マネジメント ノードを設定するために `config.ini` ファイルの `[NDB_MGMD]` あるいは `[MGM]` セクションで使用されるパラメータに関する情報を提供します。これらのパラメータの詳細な説明および補足情報に関しては、[マネジメント サーバーの定義](#) を参照してください。

再起動タイプの カラム値

- **N**:ノードの再起動
- **IN**:最初のノードの再起動
- **S**:システムの再起動
- **IS**:最初のシステムの再起動

これらの略語の説明は [クラスタ設定パラメータの概要](#) を参照してください。

パラメータ名	タイプ/ 単位	デフォルト値	最小値	最大値	再起 起動 タイプ
ArbitrationDelay [897]	ミリ セカ ンド	0	0	4294967039	N
ArbitrationRank [896]	整数	1	0	2	N
DataDir [897]	文字 列	./ (ndb_mgmd ディレ クトリ)	N/A	N/A	IN
ExecuteOnComputer [896]	整数				
HostName [896]	文字 列	localhost	N/A	N/A	IN
Id [896]	整数	なし	1	63	IN
LogDestination [896]	CONSOLE SYSLOG あるいは FILE	CONSOLE (マネジメント コンソールへの定義 参照)	N/A	N/A	N

SQL ノードおよび API ノード設定パラメータ

以下のテーブルは MySQL Cluster SQL ノードおよび API ノードを設定する `config.ini` ファイルの **[SQL]** および **[API]** セクションで使用されるパラメータに関する情報を提供します。これらのパラメータの詳細な説明および補足情報に関しては、[SQL および他の API ノードの定義](#) を参照してください。

再起動タイプの カラム値

- **N**:ノードの再起動
- **IN**:最初のノードの再起動
- **S**:システムの再起動
- **IS**:最初のシステムの再起動

これらの略語の説明は [クラスタ設定パラメータの概要](#) を参照してください。

パラメータ名	タイプ/ 単位	デフォルト値	最小値	最大値	再起 動タ イプ
ArbitrationDelay [913]	ミリ セカ ンド	0	0	4294967039	N

ArbitrationRank [913]	整数	1	0	2	N
BatchByteSize [913]	バイト	32K	1K	1M	N
BatchSize [913]	整数	64	1	992	N
ExecuteOnComputer [913]	integer				
HostName [913]	文字列	localhost	N/A	N/A	IN
Id [913]	整数	なし	1	63	IN
MaxScanBatchSize [914]	バイト	256K	32K	16M	N

ローカル チェックポイントのパラメータの設定

MySQL Cluster のローカル チェックポイントの設定に使用される [Logging and Checkpointing \[903\]](#) および [Data Memory, Index Memory, and String Memory \[898\]](#) は分離しては存在しません、むしろお互い非常に依存しています。このセクションでは— [DataMemory](#)、[IndexMemory](#)、[NoOfDiskPagesToDiskAfterRestartTUP](#)、[NoOfDiskPagesToDiskAfterRestartACC](#)、および [NoOfFragmentLogFiles](#) — を含むこれらのパラメータがどのように実際のクラスタで互いに関連しているかについて説明します。

重要[NoOfDiskPagesToDiskAfterRestartTUP](#) および [NoOfDiskPagesToDiskAfterRestartACC](#) のパラメータは MySQL 5.1.6 では少なくなっています。MySQL 5.1.6 から 5.1.111 では、LCP の際のディスクの書き込みは可能な最高速度で行われます。MySQL 5.1.12 からは、LCP の速度およびスループットはパラメータ [DiskSyncSize](#)、[DiskCheckpointSpeed](#)、および [DiskCheckpointSpeedInRestart](#) を使用して管理しています。 [Defining Data Nodes](#) 参照。

この例では、弊社のアプリケーションは以下のオペレーションの種類を毎時実行すると想定しています。

- 50000 選択
- 15000 挿入
- 15000 更新
- 15000 削除

弊社ではまたアプリケーションで使われるデータについては以下想定しています。

- 40 カラムを持つ 1 つのテーブルを現在開発中です。
- 各カラムは 32 バイトのデータを保持できます。
- アプリケーションによる一般的な [UPDATE](#) の実行では 5 カラムの値に影響します。
- アプリケーションでは [NULL](#) の値は挿入されません。

よい出発点はローカル チェックポイント (LCP) 間の経過時間を決定することです。システムの再起動時には何の価値もありますが、REDO ログを実行するためにこのインターバルの 40-60 パーセントを使います。— 例えば、LCP 間の時間が 5 分 (300 秒) だとすると、REDO の読み込みに 2 ~ 3 分 (120 ~ 180 秒) かかります。

ノード毎の最大データ量は [DataMemory](#) パラメータのサイズによって想定できます。この例では、それは 2 GB と想定しています。 [NoOfDiskPagesToDiskAfterRestartTUP](#) パラメータはユニット時間でのデータがチェックポイントされる量を表します。— しかし、このパラメータは実際は 100 ミリセカンドでチェックポイントされる 8K メモリのページ数で表されます。300 秒で 2 GB がおよそ 1 秒間に 6.8 MB、あるいは 100 ミリセカンドで 700 KB では 100 ミリセカンドでおよそ 85 ページです。

同様に、[NoOfDiskPagesToDiskAfterRestartACC](#) をインデックスに必要なローカルのチェックポイントに時間およびメモリの量で計算できます。— つまり、[IndexMemory](#) です。インデックスに 512 MB を用意すると、このパラメータに対しこれはおよそ 100 ミリセカンドで 20 8-KB ページになります。

次に、必要な REDO ログ ファイル数を決める必要があります。つまり、フラグメント ログ ファイル— 相当するパラメータである [NoOfFragmentLogFiles](#)。少なくとも 3 つのローカル チェックポイントの記録を維持できる十分な REDO ログ ファイルがあることを確認する必要があります。生産の設定では、常に不安定要素— 例えばディスクが常に最高速度あるいは最大のスループットで動作するのかわかりません。このため、最悪の状況を想定して仕様を強化し、6 つのローカル チェックポイントをカバーしたレコードを十分に維持できるフラグメント ログ ファイルを計算します。

ディスクが REDO ログ および UNDO ログへの書き込みもまた処理していることを忘れないことも大切です。ディスクに書き込まれているデータ量が [NoOfDiskPagesToDiskAfterRestartACC](#) および [NoOfDiskPagesToDiskAfterRestartTUP](#) の値で決められた量がディスクの利用できる帯域量に近づいたら、ローカル チェックポイント間の時間を長くしたくなるかもしれません。

ローカル ポイント毎に 5 分 (300 秒) とすると、書き込みログ レコードを最大速度 $6 * 300 = 1800$ 秒でサポートする必要があります。REDO ログ レコードのサイズは 72 バイトに更新されカラム値毎に 4 バイト、それに更新されたカラムの最大サイズを加えると、それにトランザクションで更新された各テーブル レコードに 1 つの REDO ログ レコード、データが常駐する各ノードにあります。このセクションで以前設定したオペレーション数を使用すると、以下が得られます。

- 毎時 50000 選択オペレーションで 0 のログ レコード (よって 0 バイト) ですので、[SELECT](#) ステートメントは REDO ログには記録されません。
- 毎時 15000 [DELETE](#) ステートメントはおおよそ 毎秒 5 削除オペレーションです。(見積もりにおいては控えめにしたいので、ここでまとめると、以下の計算になります。)カラムが削除によって更新されませんので、これらのステートメントはオペレーションごとに 5 オペレーション * 72 バイト = 毎秒 360 バイトになります。
- 毎時 15000 [UPDATE](#) ステートメントはおおよそ 毎秒 5 回の更新に相当します。各更新に 72 バイト、それにカラム毎に 4 バイト * 5 カラムの更新、さらにカラム毎に 32 バイト * 5 カラム— つまりオペレーション毎に $72 + 20 + 160 = 252$ バイトになります。これを毎秒 5 オペレーションを乗算すると毎秒 1260 バイトになります。
- 毎秒 15000 [INSERT](#) ステートメントは毎秒 5 挿入オペレーションに相当します。各挿入は REDO ログ スペース 72 バイトが必要で、それに レコード * 40 カラム毎に 4 バイト、およびカラム * 40 カラム毎に 32 バイトで $72 + 160 + 1280 = 1512$ バイト/オペレーションになります。これに 5 オペレーション/秒で 7560 バイト/秒になります。

ですから 1 秒に書き込まれる REDO ログの合計はおおよそ $0 + 360 + 1260 + 7560 = 9180$ バイトになります。これを 1800 秒で換算すると 16524000 バイトが REDO ログギングに必要になります。おおよそ 15.75 MB です。[NoOfFragmentLogFiles](#) に使用される単位は 4 16-MB ログ ファイル— つまり、64 MB になります。このように、パラメータの最小値 (3) はこの例で示したシナリオに十分で、ですから 64 の 3 倍 = 192 MB、あるいは 12 倍が要求された場合、デフォルトの 8 (あるいは 512 MB) はこのケースでは必要を満たすに十分です。

変更したテーブルのレコードのコピーは UNDO ログにあります。上記で説明したシナリオでは、UNDO ログはデフォルトの設定で提供された以上のスペースは必要ありません。しかしながら、ディスクのサイズの場合には少なくとも 1 GB 割り当てるのが賢明です。

MySQL Cluster のアップグレードおよびダウンロード

MySQL Cluster の章のこの部分では MySQL の 1 つのリリースから他の MySQL Cluster へのアップグレードおよびダウングレードについて説明します。ここでは異なるタイプのクラスタのアップグレードを取り上げ、クラスタのアップグレード/ダウングレードの互換性マトリックス ([クラスタのアップグレードおよびダウングレードの互換性](#) 参照) について説明します。

重要MySQL Cluster のアップグレードおよびダウングレードに先立ち、ここではお客様がすでに MySQL Cluster のインストールおよびその設定についてご理解頂いているものとしてここで説明します。[MySQL Cluster の設定](#) 参照。

このセクションはまだ開発中で、今後アップグレードおよび拡張が行われます。

クラスタのローリング再起動の実行

このセクションでは MySQL Cluster のインストールの ローリング再起動 の実行の仕方について説明します。そのように呼ばれているのはローリング再起動には各ノードを順番に停止および起動 (あるいは再起動) するからで、クラスタそのものは常に稼働しています。これは ローリング アップグレード あるいは ローリング ダウングレードの一部

としてよく行われ、そこではクラスタの高可用性が必須でクラスタ全体のダウンタイムは許されません。ここでアップグレードについて言及した情報は、一般的にはダウングレードにも同様に適用できます。

ローリング再起動が必要な理由はたくさんあります。

- クラスタの設定変更:クラスタの設定を変更するとは、SQL ノードをクラスタに追加したり、設定パラメータを新しい値に変更したりなどあります。
- クラスタのソフトウェアのアップグレード/ダウングレード:クラスタを新しいバージョンの MySQL Cluster のソフトウェアにアップグレード (あるいはダウングレード) します。通常これは「ローリング アップグレード」(あるいは旧バージョンの MySQL Cluster へ戻す場合は「ローリング ダウングレード」) と呼ばれています。
- ノード ホストの変更:1 つ以上のクラスタ ノードが稼働しているハードウェアあるいはオペレーティング システムの変更
- クラスタの再設定:クラスタが望まない状態に近づいたためにクラスタを再設定します。
- リソースの自由化:INSERT および DELETE オペレーションによってテーブルに割り当てられているメモリを他のクラスタ テーブルでできるように自由にします。

ローリング再起動のプロセスは以下のように一般化できます。

1. すべてのクラスタのマネジメント ノード (`ndb_mgmd` プロセス)を停止し、それらを再設定して再起動します。
2. 停止、再設定、次に各クラスタ ノード (`ndbd` プロセス) を順番に再起動します。
3. 停止、再設定、次に各クラスタ SQL ノード (`mysqld` プロセス) を順番に再起動します。

特定のローリング アップグレードを実行する詳細は加える実際の変更によります。ここで詳しいプロセスを説明します。

RESTART TYPE:					
Cluster Configuration Change		Cluster Software Upgrade or Downgrade	Change on Node Host	Cluster Reset	
A. Management node (<i>ndb_mgmd</i>) processes...					
1. Stop all <i>ndb_mgmd</i> processes 2. Make changes in global configuration file(s) 3. Start all <i>ndb_mgmd</i> processes		1. Stop all <i>ndb_mgmd</i> processes 2. Replace each <i>ndb_mgmd</i> binary with new version 3. Start <i>ndb_mgmd</i> processes	1. Stop all <i>ndb_mgmd</i> processes 2. Make desired changes in hardware, operating system, or both 3. Start all <i>ndb_mgmd</i> processes	(OR)	
				1. Stop all <i>ndb_mgmd</i> processes 2. Start all <i>ndb_mgmd</i> processes	Restart all <i>ndb_mgmd</i> processes (optional)
B. For each data node (<i>ndbd</i>) process...					
(OR)		1. Stop <i>ndbd</i> 2. Replace <i>ndbd</i> binary with new version 3. Start <i>ndbd</i>	1. Stop <i>ndbd</i> 2. Make desired changes in hardware, operating system, or both 3. Start <i>ndbd</i>	(OR)	
1. Stop <i>ndbd</i> 2. Start <i>ndbd</i>	Restart <i>ndbd</i>			1. Stop <i>ndbd</i> 2. Start <i>ndbd</i>	Restart <i>ndbd</i>
C. For each SQL node (<i>mysqld</i>) process...					
(OR)		1. Stop <i>mysqld</i> 2. Replace <i>mysqld</i> binary with new version 3. Start <i>mysqld</i>	1. Stop <i>mysqld</i> 2. Make desired changes in hardware, operating system, or both 3. Start <i>mysqld</i>	(OR)	
1. Stop <i>mysqld</i> 2. Start <i>mysqld</i>	Restart <i>mysqld</i>			1. Stop <i>mysqld</i> 2. Start <i>mysqld</i>	Restart <i>mysqld</i>

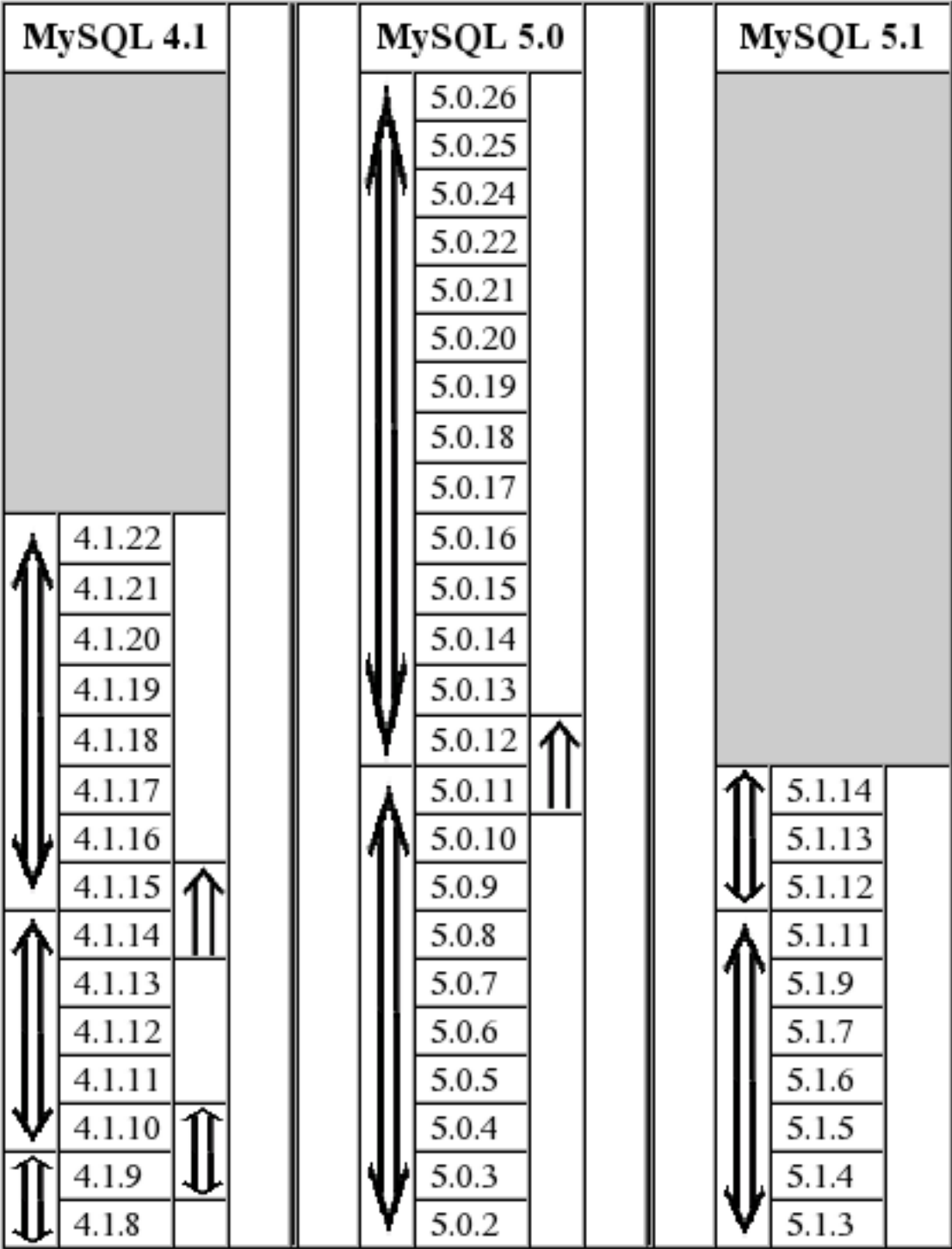
前掲のダイアグラムでは、停止 および 起動 ステップをシェル コマンド (ほとんどの Unix システムの **kill** など) あるいは マネジメント クライアントの **STOP** コマンドを使用して完全に停止して、次に適切な *ndbd* あるいは *ndb_mgmd* 実行ファイルを呼び出してシステムのシェルから再度起動するを説明しました。再起動 はプロセスを *ndb_mgm* マネジメント クライアント **RESTART** コマンドを使用して再起動することを意味します。

クラスタのアップグレードおよびダウングレードの互換性

この項ではアップグレードおよびダウングレードを行うための MySQL Server の異なるバージョン間にクラスタ ソフトウェアおよびテーブル ファイルの互換性に関する情報を提供します。

重要この項では [NDB Cluster](#) に関する MySQL バージョン間の互換性のみ検討し、他の考慮すべき点にまだあります。他の MySQL ソフトウェアのアップグレードあるいはダウングレードと同様に、MySQL Cluster のソフトウェアをアップグレードあるいはダウンロードする前に、移行するあるいは移行される MySQL バージョンの MySQL マニュアルの関連する部分を見直すことを強くお勧めします。[MySQL のアップグレード](#) 参照。

以下のテーブルは異なるバージョンの MySQL Server のクラスタのアップグレードおよびダウングレードの互換性を示しています。



KEY:

Upgrades and downgrades supported

↑↑ Upgrade only

注：

- 4.1 シリーズ:

4.1.8 から 4.1.10 (あるいはそれ以降) に直接アップグレードできません。最初に 4.1.8 を 4.1.9 に、次に 4.1.10 にアップグレードします。同様に 4.1.10 (あるいはそれ以前) から 4.1.8 にダウングレードできません。最初に 4.1.10 から 4.1.9、次に 4.1.8 にダウングレードします。

MySQL Cluster を 4.1.15 にアップグレードするには、最初に 4.1.14、次に 4.1.15 に、それから 4.1.16 あるいはそれ以降にアップグレードします。

クラスタの 4.1.15 から 4.1.14 (あるいはそれ以前) のダウングレードはサポートされていません。

4.1.8 以前のバージョンへの MySQL Server バージョンからのクラスタのアップグレードはサポートされていません。これからアップグレードするには、すべての NDB テーブルを一旦ダンプして、新しいバージョンのソフトウェアをインストールして、次にそのダンプからテーブルを再ロードします。。

- 5.0 シリーズ:

このシリーズでは MySQL 5.0.2 が最初の一般へのリリースです。

MySQL 5.0 から MySQL 4.1 へのクラスタのダウングレードはサポートされていません。

5.0.12 から 5.0.11 (あるいは「それ以前への」) クラスタのダウングレードはサポートされていません。

`ndb_restore` を MySQL 5.1 で動作するクラスタのバックアップを使用して MySQL 5.0 Cluster に復旧することはできません。そのような場合には `mysqldump` を使用する必要があります。

これは MySQL 5.0.23 用には一般にリリースされていません。

- 5.1 シリーズ:

このシリーズでは MySQL 5.1.3 が最初の一般へのリリースです。

MySQL 5.1.6 あるいはそれ以降をディスク データ テーブルを使用して MySQL 5.1.5 あるいはそれ以前のバージョンにダウングレードするには、すべてのそのようなテーブルを in-memory の Cluster テーブルに最初に変換しない限りダウングレードできません。

MySQL 5.1.8 および MySQL 5.1.10 はリリースされていません。

MySQL 5.1.11 (あるいは以前のバージョン) および 5.1.12 (あるいはそれ以降のバージョン) 間でのオンラインでのクラスタのアップグレードおよびダウングレードはクラスタのファイルシステムに大きな変更があるためできません。そのような場合には、バックアップあるいはダンプするか、ソフトウェアをアップグレード (あるいはダウングレード) し、各データ ノードを `--initial` で起動し、次にバックアップあるいはダンプから復旧します。これを行うために NDB バックアップ/復旧あるいは `mysqldump` を使用できます。

MySQL Cluster のプロセス管理

MySQL Cluster の管理を理解するには 4 つの基本的なプロセスに関する知識が必要です。この章の次の数項でクラスタのこれらのプロセスの役割、その使用方法、およびそれらにどのような起動オプションを利用できるかについて説明します。

- [MySQL Cluster のための MySQL Server プロセスの使用法](#)
- [ndbd、ストレージ エンジン ノード プロセス](#)
- [ndb_mgmd、マネジメント サーバ プロセス](#)
- [ndb_mgm、マネジメント クライアント プロセス](#)

MySQL Cluster のための MySQL Server プロセスの使用法

`mysqld` は従来の MySQL サーバーのプロセスです。MySQL Cluster で使用するには、`mysqld` は NDB Cluster ストレージ エンジンのサポート付きでビルドされる必要があります。それは <http://dev.mysql.com/downloads/> からコンパ

イルされてバイナリで利用できます。ソースから MySQL を構築する場合、`configure` を `--with-ndbcluster` オプションと一緒に起動し **NDB Cluster** ストレージ エンジンをサポートします。

`mysqld` バイナリが Cluster サポートするように構築されている場合、**NDB Cluster** ストレージ エンジンはデフォルトはまだ無効です。このエンジンを有効にするために可能なオプションの 2 つのどちらかを使用できます。

- `--ndbcluster` をコマンドラインの起動オプションとして使用 `mysqld` を起動します。
- `my.cnf` ファイルの `[mysqld]` セクションの `ndbcluster` を含む行を挿入します。

お客様のサーバーが **NDB Cluster** ストレージ エンジンが有効で稼働していることを検証するには **SHOW ENGINES** ステートメントを MySQL Monitor (`mysql`) で発行します。**YES** が **Support** 値として **NDBCLUSTER** の行に表示されます。この行で **NO** が表示された場合あるいはそのような行が出力に表示されなかった場合、**NDB** が有効なバージョンの MySQL が動作していないことになります。この行に **DISABLED** が表示された場合、それを今説明した 2 つの方法のいずれかで有効にする必要があります。

クラスタお設定ファイルを読み込むには、MySQL サーバーは少なくとも 3 件の情報が必要です。

- MySQL サーバー自身のクラスタ ノード ID
- マネジメント サーバー (MGM ノード) のホスト名あるいは IP アドレス
- マネジメント サーバーに接続する TCP/IP のポート番号

ノード ID はダイナミックに割り当てられますので、それらを明示的に指定することは必ずしも必要ありません。

`mysqld` パラメータ `ndb-connectstring` はコマンドラインで `mysqld` を起動するときにあるいは `my.cnf` で接続文字列の指定に使用されます。接続文字列はホスト名あるいはマネジメント サーバーがある IP およびそれが使用する TCP/IP ポートを含みます。

以下の例では、`ndb_mgmd.mysql.com` はマネジメント サーバーが常駐するホストで、マネジメント サーバーはポート 1186 のクラスタ メッセージを受信します。

```
shell> mysqld --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

接続文字列に関する情報は **クラスタの 接続文字列** を参照してください。

この情報を与えられると、MySQL サーバーはクラスタの完全な参加者になります。(このように実行されている `mysqld` プロセスを SQL ノードとも言います。) それは完全にクラスタ ノードおよびその状態を認識しており、すべてのデータ ノードに接続を確立します。このように、データ ノードをトランザクション コーオーディネータとして使用してデータを読みデータ ノードを更新します。

`mysql` クライアントで MySQL サーバーがクラスタに **SHOW PROCESSLIST** を使用して接続されているかどうか表示することができます。MySQL サーバーがクラスタに接続されていると、お客様には **PROCESS** 権限がありますので、出力の最初の行はここに表示されているようになります。

```
mysql> SHOW PROCESSLIST \G
***** 1. row *****
  Id: 1
  User: system user
  Host:
  db:
  Command: Daemon
  Time: 1
  State: Waiting for event from ndbcluster
  Info: NULL
```

ndbd、ストレージ エンジン ノード プロセス

ndbd は **NDB Cluster** ストレージ エンジンを使用してテーブルのすべてのデータの処理に使用されるプロセスです。これは配布されたトランザクションの処理、ノードのリカバリ、オンラインバックアップ、および関連のタスクを実行するためのデータ ノードに権限を与えるプロセスです。

MySQL Cluster では、一連の **ndbd** プロセスがデータの処理に協力します。これらのプロセスは同じコンピュータ上あるいは異なるコンピュータ上で実行できます。データ ノードと Cluster ホストとの通信は完全に設定できます。

ndbd generates は config.ini 設定ファイルの DataDir で指定されたディレクトリにある一連のログ ファイルを生成します。これらのログ ファイルを以下に示します。node_id はノードの一意的識別子を表します。例えば、ndb_2_error.log はノード ID が 2 のデータ ノードから生成されたエラーログです。

-
-
- ndb_node_id_trace.log.next は割り当てられる次のトレースファイル番号を追跡したファイルです。
- ndb_node_id_out.log は ndbd プロセスによるデータ出力を含むファイルです。このファイルは ndbd が daemon として実行されたときのみ作成されます。それはデフォルトの振る舞いです。
- ndb_node_id.pid は daemon として実行されたときの ndbd プロセスのプロセス ID を含みます。それはまたノードが同じ識別子で起動されるのを避けるためにロック ファイルとして機能します。
- ndb_node_id_signal.log は ndbd のデバッグ バージョンでのみ使用されるファイルで、すべての受信、送信、およびそのデータを ndbd プロセスに持つ内部メッセージをトレースできます。

NSF で搭載したディレクトリを使用しないようお勧めします。なぜなら、環境によってはこのディレクトリが問題を起こし、.pid ファイルのロックがプロセスが終了しても有効である場合があります。

ndbd を実行するには、マネジメント サーバーのホスト名および接続しているポートを指定する必要があります。オプションで、プロセスが使用するノード ID を指定することもできます。

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

この件に関する詳細は [クラスタの 接続文字列](#) を参照してください。MySQL Cluster プロセスのコマンド オプションは ndbd の他のオプションについて説明します。

ndbd が実行されると、それは実際に 2 つのプロセスを始めます。これらのプロセスの最初は「angel process」と呼ばれています。その唯一の仕事は実行プロセスがいつ完了したかを突き止めることで、次に ndbd プロセスをそれが統合されている場合に実行します。このように、ndbd を Unix kill コマンドで無効にする場合、エンジェル プロセスで初めて両方のプロセスを無効にする必要があります。ndbd プロセスを終了させる好ましい方法としてはマネジメント クライアントを使用してそのプロセスをそこで止めることです。

実行プロセスは他の操作同様、読み込み、書き込み、およびデータのスキャンにスレッドを 1 つ使用しています。このスレッドは数千もの同時発生的な操作を容易に処理するために同期的に導入されます。さらに、監視スレッドはそれがエンドレスのループでフリーズしないよう確認するための実行スレッドを管理します。スレッドのループはファイル I/O を取扱い、各スレッドは 1 つのオープン ファイルをそれぞれ扱います。スレッドは ndbd プロセスのトランスポーターによるトランスポーター接続に使用されます。多数のオペレーション（更新を含む）を実行するマルチ プロセッサ システムの場合、ndbd プロセスはそれが可能な場合には 2 つまでの CPU を使用できます。

多くの CPU を使用しているマシンの場合、異なるノード グループに属すいくつかの ndbd プロセスを使用できます。しかし、そのような設定はそれでも実験的なものであり、生産環境では MySQL 5.1 サポートしていません。MySQL Cluster の既知の制限 参照。

ndb_mgmd、マネジメント サーバードプロセス

マネジメント サーバーはクラスタの設定ファイルを読み込みこの情報をそれを要求したクラスタのすべてのノードに配布するプロセスです。それはまたクラスタの活動に関するログを維持します。マネジメント クライアントはマネジメント サーバーに接続してクラスタの状況をチェックします。

マネジメント サーバーを起動するときには接続文字列を指定することは厳密には必要ありません。しかし、1 つ以上のマネジメント サーバーを使用している場合には、接続文字列を提供しクラスタの各ノードはそのノード ID を明示的に指定する必要があります。

接続の文字列の使用に関する情報は [クラスタの 接続文字列](#) を参照してください。MySQL Cluster プロセスのコマンド オプションは、ndb_mgmd の他のオプションについて説明しています。

以下のファイルはその起動ディレクトリにある ndb_mgmd によって作成あるいは使用され、config.ini で指定されたように DataDir に配置されます。その後のリストでは、node_id は一意的ノード識別子です。

-
- ndb_node_id_cluster.log はクラスタのイベント ログ ファイルです。それらのイベントの例としてチェックポイントの起動および完了、ノードの起動イベント、ノードの失敗、およびメモリ使用のレベルを含みます。完全なクラスタ イベントのリストおよびその説明は [MySQL Cluster の管理](#) にあります。

クラスタのログ サイズが 100 万バイトになると、そのファイルは `ndb_node_id_cluster.log.seq_id` に名前が変わります。そこでは `seq_id` はクラスタのログ ファイルのシーケンス番号です。(例:シーケンス番号の 1、2、3 の番号の付いたファイルが既に存在する場合には、次のログ ファイルは 4 の番号の付いた名前になります。)

- `ndb_node_id_out.log` は `stdout` および `stderr` に使用されるファイルでマネジメント サーバーが daemon として稼働しているときに使用されます。
- `ndb_node_id.pid` はマネジメント サーバーを daemon として使用稼働しているときに使用されるプロセス ID です。

ndb_mgm、マネジメント クライアント プロセス

マネジメント クライアントのプロセスは実際はクラスタを稼働させるには必要ありません。その値はクラスタの状態、バックアップの開始、および他の管理機能をチェックするために一連のコマンドを提供します。マネジメント クライアントは C API を使用してマネジメント サーバーにアクセスします。高度なユーザーは専用のマネジメント プロセスをプログラムするためにこの API を使用し `ndb_mgm` がの実行に類似したタスクを実行できます。

マネジメント クライアントを起動するには、マネジメント サーバーのホスト名とポート番号を提供する必要があります。

```
shell> ndb_mgm [host_name [port_num]]
```

例:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

デフォルトのホスト名とポート番号はそれぞれ `localhost` および 1186 です。

`ndb_mgm` の使用に関する詳細は `ndb_mgm` の [コマンド オプション](#)、および [マネジメント クライアントのコマンド](#) にあります。

MySQL Cluster プロセスのコマンド オプション

MySQL Cluster のすべての実行ファイル (`mysqld` を除く) はこの項で説明したオプションを使用します。MySQL Cluster バージョンのユーザーはこれらのオプションの中には `mysqld` 同様お互いの一貫性を保つために変更されている場合がありますのでご承知ください。MySQL Cluster の実行ファイルと一緒に `--help` オプションを使用しそれがサポートするオプションのリストを表示できます。

以下のオプションはすべての MySQL Cluster 実行ファイルに共通です。

- `--help --usage, -?`

利用できるコマンド オプションのリストをその説明書と一緒に印刷します。

- `--connect-string=connect_string, -c connect_string`

`connect_string` は接続文字列をマネジメント サーバーにコマンドオプションとして設定します。

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

- `--debug[=options]`

このオプションはデバッグを有効にしてコンパイルしたバージョンにのみ使用できます。それは `mysqld` プロセスと同様にデバッグの呼び出しからの出力を可能にするために使用されます。

- `--execute=command, -e command`

はシステム シェルからコマンドをクラスタの実行ファイルに送るために使用されます。例えば、以下のいずれかになります。

```
shell> ndb_mgm -e "SHOW"
```

あるいは

```
shell> ndb_mgm --execute="SHOW"
```

は以下に等価

```
ndb_mgm> SHOW
```

これは `--execute` あるいは `-e` オプションが `mysql` コマンドラインのクライアントとの機能の仕方に類似しています。 [コマンドラインにおけるオプションの使用](#) 参照。

- `--version, -V`

実行ファイルの MySQL Cluster バージョン番号を印刷します。バージョン番号すべてのバージョンが一緒に使用されるとはかぎらないので関連性があります。MySQL Cluster の起動プロセスでは使用されているバイナリのバージョンが同じクラスターの共存できるか検証します。これは MySQL のソフトウェアのアップグレードあるいはダウングレードをオンライン（ローリング）で行う場合にも重要です。（[クラスターのローリング再起動の実行](#) 参照。）

次の数項ではこの **NDB** プログラムに特定のオプションについて説明します。

mysqld の MySQL Cluster 関連コマンド オプション

- `--ndb-connectstring=connect_string`

NDB Cluster ストレージ オプションを使用する際、このオプションはクラスターの設定データを配布するマネジメント サーバーを指定します。

- `--ndbcluster`

MySQL を使用するには **NDB Cluster** ストレージ エンジンが必要です。`mysqld` バイナリが **NDB Cluster** ストレージ エンジンのサポートを含む場合、このエンジンはデフォルトで無効になります。有効にするには `--ndbcluster` オプションを使用します。そのエンジンを明示的に無効にするには `--skip-ndbcluster` を使用します。

ndbdのコマンド オプション

すべての **NDB** プログラムに共通のオプションに関しては、[MySQL Cluster プロセスのコマンド オプション](#) を参照してください。

- `--bind-address`

`ndbd` を特定のネットワーク インターフェースに結合します。このオプションにはデフォルトの値はありません。

このオプションは MySQL 5.1.12 に追加されています。

- `--daemon, -d`

`ndbd` に `daemon` プロセスの実行を指示します。これはデフォルトの振る舞いです。`--nodaemon` はプロセスが `daemon` として実行されないようにします。

- `--initial`

`ndbd` に初期の実行を指示します。初期の実行により以前の `ndbd` のインスタンスでリカバリ目的に作成されたファイルを消去します。またリカバリ ログ ファイルを再度作成します。使用しているオペレーティング システムによってはこのプロセスにはかなり長い時間がかかります。

`--initial` の実行はそれを実行することによって Cluster のファイルシステムのすべてのファイルを削除し、新たにすべての REDO ファイルを作成するため `ndbd` プロセスが実行される一番最初にのみ使用されます。この規則に対する例外は:

- ファイルのコンテンツを変更したソフトウェアのアップグレードを行うとき。
- ノードを `ndbd` の新しいバージョンで再起動するとき。
- 何かの理由でノードあるいはシステムの再起動が繰り返し失敗したときの最後のレポートの方法として。この場合、データ ファイルの破壊によりこのノードはデータの復旧にその後使用できなくなることに注意してください。

このオプションは以下のいずれにも影響は及ぼしません。

- 影響されたノードにより既に作成されたバックアップ ファイル
- Cluster ディスク データ ファイル ([MySQL Cluster ディスク データ ストレージ](#) 参照)。

そのデータ ノード — で作成されたバックアップを維持し `ndbd` を `--initial` オプションを使用せずに実行するには、`DataDir` の `BACKUP` の可能な例外によってデータ ノード `DataDir` のすべてのファイルおよびディレクトリを別の方法 (`rm -r -f` などを使用して) によって削除することで同じ効果を得ることができます。これは Cluster の管理タスクを記述する際に役に立ちます。

- `--initial-start`

このオプションはクラスタを部分的に初期起動する際に使用します。各ノードおよび `--no-wait-nodes` はこのオプションで起動します。

例えば、データ ノードの ID が 2、3、4、および 5 を持つ 4-ノード クラスタを持っているとします。それをノードの 2、4、および 5 のみを使用して部分的な初期の起動をする場合。— つまりノード 3 を除く：

```
ndbd --ndbd-nodeid=2 --no-wait-nodes=3 --initial-start
ndbd --ndbd-nodeid=4 --no-wait-nodes=3 --initial-start
ndbd --ndbd-nodeid=5 --no-wait-nodes=3 --initial-start
```

このオプションは MySQL 5.1.9 に追加されています。

- `--nowait-nodes=node_id_1[, node_id_2[, ...]]`

このオプションはクラスタが起動前に待たないデータ ノードのリストを扱っています。

これをクラスタが分割した状態での起動に使用できます。例えば、クラスタを 4-ノード クラスタを稼働しているデータ ノードの半分だけ (ノード 2、3、4、および 5) で起動するには、かく `ndbd` プロセスを `--nowait-nodes=3,5` で実行します。この場合、クラスタはノード 2 および 4 が接続すると直ぐ起動し、ノード 3 および 5 が接続するまで `StartPartitionedTimeout` ミリ秒を待ちません。

前の例の同じクラスタを 1 つの `ndbd` — 無しで起動したい場合、例えば、ノード 3 のホスト マシンのハードウェアに問題があり — ノード 2、4、および 5 を `--no-wait-nodes=3` で起動する場合。次にクラスタが直ぐにノード 2、4、および 5 を接続しノード 3 の起動を待たない場合。

このオプションは MySQL 5.1.9 に追加されています。

- `--nodaemon`

`ndbd` に daemon プロセスとして実行しないよう指示を出します。これは `ndbd` がデバッグされて出力を画面に向ける場合に有効です。

- `--nostart, -n`

`ndbd` に自動的に実行しないよう指示をだします。このオプションを使用すると、`ndbd` がマネジメント サーバーに接続して、マネジメント サーバーから設定データを取得し、通信オブジェクトを初期化します。しかし、それはマネジメント サーバーによって特にそうするように要求されるまで実行エンジンを起動しません。これはマネジメント クライアントで適切な `START` コマンドを出すと実行されます ([マネジメント クライアントのコマンド](#) 参照)。

ndb_mgmd のコマンド オプション

NDB プログラムに共通のオプションについては、[MySQL Cluster プロセスのコマンド オプション](#) を参照してください。

- `--config-file=filename, -f filename`

マネジメント サーバーにその設定ファイルにどのファイルを使用するかを指示します。このオプションは指定する必要があります。そのファイル名は `config.ini` に既定です。

注:このオプションには `-c file_name`を付けることもできますが、このショートカットは既に陳腐なため新しいインストールでは使用していません。

- `--daemon, -d`

`ndb_mgmd` に `daemon` プロセスを実行する指示を出します。これはデフォルトの操作です。

- `--nodaemon`

`ndb_mgmd` に `daemon` `daemon` として開始しないように指示します。

ndb_mgm のコマンド オプション

NDB プログラムに共通のオプションについては、[MySQL Cluster プロセスのコマンド オプション](#) を参照してください。

- `--try-reconnect=number`

マネジメント サーバーへの接続が切断された場合、ノードは接続できるまで 5 秒ごとに接続を試みます。このオプションを使用して、再接続の回数を `number` に制限して再接続を止めエラーとして報告することもできます。

MySQL Cluster の管理

MySQL Cluster の管理には様々なタスクを含み、その最初は設定および MySQL Cluster の起動です。これは [MySQL Cluster の設定](#)、および [MySQL Cluster のプロセス管理](#) で説明しています。

以下の項では MySQL Cluster の運用管理について説明します。

MySQL Cluster の運用管理には基本的に 2 つの手法があります。これらの最初はマネジメント クライアントの入力したコマンドの使用でそこではクラスタの状態をチェックし、ログ レベルを変更し、バックアップを開始して停止し、ノードを停止して起動します。2 番目の手法はクラスタ ログ `ndb_node_id_cluster.log` のコンテンツを調べることを含みます。これは通常マネジメント サーバーの `DataDir` ディレクトリにあります。このロケーションは `LogDestination` オプションで無効になります。— 詳細は [マネジメント サーバーの定義](#) を参照してください。(`node_id` がその活動がログされるノードの一意の識別子を表すことを思い出してください。)クラスタ ログは `ndbd` で生成されたイベント レポートを含みます。クラスタ ログのエントリを Unix システム ログに送ることもできます。

MySQL Cluster 起動フェーズ

この項ではクラスタが起動したときのステップを説明します。

以下に示すようにいくつかの異なる起動タイプおよびモードがあります。

- 最初の起動: クラスタはすべてのデータ ノードの未使用のファイルシステムで起動します。これはクラスタがまさに一番最初の起動時に発生する、あるいは `--initial` オプションで再起動したときに発生します。

注: ディスク データ ファイルはノードを `--initial` で再起動したときには削除されません。

- システムの再起動: クラスタが起動しデータ ノードに保存されたデータを読み込みます。これはクラスタが使用された後にシャットダウンされ場合、クラスタが離れたところからオペレーションを再開する際に発生します。
- ノードの再起動: これはクラスタそのものが稼働しているときのクラスタ ノードのオンラインの再起動です。
- 最初のノードの再起動: これはノード再初期化され未使用のファイルシステム起動されたときを除いて、ノードの再起動と同じです。

起動に先立ち、すべてのデータ ノード (`ndbd` プロセス) は初期化する必要があります。初期化には以下のステップが含まれます。

1. ノード ID の取得。
2. 設定データの取り出し。
3. インターノード通信のポートの割り当て。
4. 設定ファイルから取得した設定に基づいたメモリの割り当て。

データ ノードあるいは SQL ノードが最初にマネジメント ノードに接続すると、クラスタ ノード ID を予約します。他のノードが同じノード ID を割り当てていないか確認するために、この ID はノードがクラスタおよびノードが接続

された少なくとも 1 つの `ndbd` レポートに接続されるまで維持されます。このノード ID の維持は問題のノードおよび `ndb_mgmd` 間の接続によって保護されます。

通常、ノードに問題が発生した場合、ノードの接続がマネジメント サーバーから切断され、その接続に使用されているソケットが閉じて、予約されたノード ID の予約が開放されます。しかし、ノードの接続が突然切断された場合—例えば、クラスタのホストの 1 つのハードウェアの問題、あるいはネットワークの問題で切断されると、—オペレーティング システムによるソケットの通常の閉鎖が行われなくなります。この場合、そのノード ID は予約されたまま、TCP のタイムアウトが 10 分あるいはそれ以降には起こるまで開放されません。

この問題に対処するために `PURGE STALE SESSIONS` を使用できます。このステートメントを実行するとすべての予約したノード ID をチェックします。そして、実際に接続されていてノードによって使用されていないノード ID の予約が外されます。

MySQL 5.1.11 以降では、ノード ID の割り当てのタイムアウトの処理が導入されています。これによりおよそ 20 秒後に ID 利用を自動的にチェックするため、`PURGE STALE SESSIONS` は通常のクラスタの起動においては必要なくなります。

各ノードが初期化された後、クラスタの起動プロセスが開始されます。クラスタがこのプロセスで行うステージを以下に示します。

- ステージ 0

クラスタのファイルシステムをクリアします。このステージはクラスタが `--initial` オプションで起動されたときのみ発生します。

- ステージ 1

このステージはクラスタの接続を設定し、インターノード通信を確立し、クラスタのハートビートを始動します。

注: 残りのノードがフェーズ 2 でハングしている時にフェーズ 1 で 1 つ以上のノードがハングした場合は、ネットワークの問題が考えられます。そのような問題で考えられる原因の 1 つは 1 つ以上のホストが複数のネットワークのインターフェースを持っている場合です。この条件の別の共通の問題点はクラスタ ノードの通信に必要な TCP/IP ポートのブロックです。後者のケースでは、ファイアウォールの設定の仕方に問題がある場合がよくあります。

- ステージ 2

アービトラータ ノードが選択されます。これがシステムの再起動の場合、クラスタは最新の保存できるグローバルチェックポイントを決定します。

- ステージ 3

このステージでは多くのクラスタの変数を初期化します。

- ステージ 4

最初の起動あるいは最初のノードに起動に対し、やり直し (redo) ログファイルが作成されます。これらのファイル数は `NoOfFragmentLogFiles` と同じです。

システムの再起動:

- スキーマを読んでください。
- ローカルのチェックポイントからデータ読み込みログを元に戻し (undo) ます。
- 最新の保存できるグローバル チェックポイントになるまでやり直し (redo) 情報を適用します。

ノードの再起動には、やり直し (redo) ログの最後を探します。

- ステージ 5

これが最初の起動の場合、`SYSTAB_0` および `NDB$EVENTS` 内部システム テーブルを作成します。

ノードの再起動あるいは最初のノードの再起動:

1. このノードはトランザクションの取扱い操作に含まれています。

2. ノード スキーマはそのマスタに比較されそれと同期します。
 3. [INSERT](#) フォームでこのノード グループの他のデータ ノードから受信した同期データ
 4. すべてのケースにおいて、アービトラーターによって決められたローカル チェックポイントの完了を待ちます。
- ステージ 6
内部変数を更新します。
 - ステージ 7
内部変数を更新します。
 - ステージ 8
システムの再起動で、すべてのインデックスを再構築します。
 - ステージ 9
内部変数を更新します。
 - ステージ 10
この段階でノードの再起動あるいは最初のノードに再起動で、API がノードに接続してイベントを受け取ります。
 - ステージ 11
この段階でノードの再起動あるいは最初の再起動で、イベントの配布はクラスタを結合するノードに渡されます。新たに結合されたノードはそのプライマリのデータをサブスクラバーに届ける責任を負います。
- 最初の起動およびシステムの再起動のこのプロセスが完了すると、トランザクションの取扱いが有効になります。ノードの再起動あるいは最初のノードの再起動で、起動プロセスの完了はノードがトランザクションのコーディネーターとしての役割を始めたことを意味します。

マネジメント クライアントのコマンド

クライアントの設定に加えて、クラスタがマネジメント クライアント [ndb_mgm](#) で利用できるコマンドライン インターフェースを通じて操作されます。これは稼働中のクラスタのプライマリの管理インターフェースです。

イベント ログのコマンドは [MySQL Cluster で生成されたイベント レポート](#) にあります。バックアップの作成およびバックアップからの保存は [MySQL Cluster のオンライン バックアップ](#) で提供しています。

マネジメント クライアントには以下の基本的なコマンドがあります。その後のリストでは、[node_id](#) はデータベースのノード ID あるいはキーワード [ALL](#) を表し、それによってコマンドはすべてのクラスタのデータ ノードに適用されます。

-
-
-
-
-
-
-
-
-
-

MySQL Cluster は 2 種類のイベント ログを提供します。

- クラスタ ログ、はすべてのクラスタ ノードが生成したイベントを含みます。クラスタ ログは単一のロケーションでのクラスタ全体のログ情報を提供するにほとんどのケースに使用に推薦しているログです。

デフォルトでは、クラスタ ログは `ndb_bgm` バイナリが常駐する同じディレクトリでファイル名 `ndb_node_id_cluster.log` で保存されます。(ここでは `node_id` はマネジメント サーバーのノード ID です)。

クラスタのログ情報をファイルに追加あるいはファイルに保存される代わりに、`DataDir` および `LogDestination` 設定パラメータに設定された値によって決められたように `stdout` あるいは `syslog` に送られます。これらのパラメータに関する情報は、[マネジメント サーバーの定義](#) を参照してください。

- ノード ログ は各ノードに対してローカルです。

ノードのイベント ログにより生成された出力はノードの `DataDir` のファイル `ndb_node_id_out.log` (ここでは `node_id` はノードのノード ID です) に書き込まれます。ノードのイベント ログはマネジメント ノードおよびデータ ノードの両方に生成されます。

ノード ログはアプリケーションの開発時、あるいはアプリケーション コードのデバッグ時のみ使用されるように意図されたものです。

両方のログ タイプはイベントの異なるサブセットのログに設定できます。

各レポートされるイベントは 3 つの異なる基準で分類されます。

- Category:これは以下の値のいずれかになります。[STARTUP](#)、[SHUTDOWN](#)、[STATISTICS](#)、[CHECKPOINT](#)、[NODERESTART](#)、[CONNECTION](#)、[ERROR](#)、あるいは [INFO](#)。
- Priority:これは 1 ~ 15 の番号で表示され、1 は「最も重要」で 15 は「それほど重要でない」ことを意味します。
- 深刻度:これは以下の値のいずれかになります。[ALERT](#)、[CRITICAL](#)、[ERROR](#)、[WARNING](#)、[INFO](#)、あるいは [DEBUG](#)。

クラスタ ログおよびノード ログはどちらもこれらのプロパティでフィルタできます。

クラスタ ログで使用されているフォーマットを以下示します。

```
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 1: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 2: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 3: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 4: Node 9: API version 5.1.15
2007-01-26 19:59:22 [MgmSrvr] ALERT -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT -- Node 2: Node 7 Disconnected
```

クラスタ ログの各行には以下の情報が含まれます。

- タイムスタンプ `YYYY-MM-DD HH:MM:SS` フォーマット。
- ログを実行しているノード タイプクラスタ ログでは、これは常に `[MgmSrvr]` です。
- イベントの深刻度です。

- イベントをレポートするノード ID です。
- イベントの説明です。ログの表示され最も一般的なイベントのタイプ f はチェックポイントが発生したときのクラスタの異なるノード間の接続と切断です。ケースによっては、その記述の中に状況の情報が含まれます。

ログの管理コマンド

以下のマネジメント コマンドはクラスタ のログに関するものです。

- **CLUSTERLOG ON**

クラスタのログをオンにします。

- **CLUSTERLOG OFF**

クラスタのログをオフにします。

- **CLUSTERLOG INFO**

クラスタのログ設定に関する情報を提供します。

- **node_id CLUSTERLOG category=threshold**

ログの **category** イベントでクラスタ ログの **threshold** 以下あるいは同等のイベントです。

- **CLUSTERLOG FILTER severity_level**

severity_level で指定されたイベントのクラスタ ログを切り換えます。

以下のテーブルはクラスタ ログのカテゴリの閾値のデフォルトの設定 (すべてのデータ ノードの) を示しています。イベントが優先度の閾値より低いがあるいは同じ場合値を持つ場合、それはクラスタ ログにレポートされます。

イベントはデータ ノード毎にレポートされ、閾値は異なるノードの異なる値で設定できます。

カテゴリ	デフォルトの閾値 (すべてのデータ ノード)
STARTUP	7
SHUTDOWN	7
STATISTICS	7
CHECKPOINT	7
NODERESTART	7
CONNECTION	7
ERROR	15
INFO	7

STATISTICS カテゴリには非常に多くの役に立つデータがあります。詳細については、**CLUSTERLOG STATISTICSの活用** をご参照してください。

閾値は各カテゴリ内のイベントをフィルタできます。例えば、重要度 3 の **STARTUP** イベントは **STARTUP** の閾値が 3 あるいはそれ以下の変更されないとログされません。重要度が 3 あるいはそれ以下のイベントが閾値が 3 の場合送られます。

以下のテーブルではイベントの深刻度を表示します。(注:これらは **LOG_EMERG** および**LOG_NOTICE**、それは使用されたりマップされない) を除く Unix の **syslog** レベルに一致します。

1	ALERT	破損したシステムのデータベースなどのように至急修正が必要な条件
2	CRITICAL	デバイスのエラーあるいは不十分なりソースなどのクリティカルが条件
3	ERROR	設定エラーなど修正を必要な条件

4	WARNING	エラーではないが、特別な扱いが必要な条件
5	INFO	情報に関するメッセージ
6	DEBUG	NDB Cluster の開発に使用されるデバッグ メッセージ

イベントの深刻度レベルはオン/オフにできます。(上記 [CLUSTERLOG FILTER](#) — 参照)。深刻度レベルをオンにすると、深刻度がカテゴリの閾値より低いあるいは同じすべてのイベントがログされます。深刻度レベルをオフにするとその深刻度レベルに属すイベントはログされません。

ログ イベント

イベント ログでレポートされたイベント レポートのフォーマットは以下のようになります。

```
datetime [string] severity -- message
```

例:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

この項では各カテゴリのカテゴリおよび深刻度レベルで規定されたレポートが必要なイベントのすべてについて説明します。

イベントの説明での、GCP および LCP はそれぞれ「グローバル チェックポイント」および「ローカル チェックポイント」を意味します。

CONNECTION イベント

これらのイベントはクラスタ間の接続に関連しています。

イベント	優先度	深刻度レベル	説明
データ ノード接続済み	8	INFO	データ ノード接続済み
データ ノード接続済み	8	INFO	データ ノード接続済み
通信閉鎖	8	INFO	SQL ノードあるいはデータ ノード接続切断
通信再開	8	INFO	SQL ノードあるいはデータ ノード接続の再開

CHECKPOINT イベント

ここに示したログ メッセージはチェックポイントに関するものです。

イベント	優先度	深刻度レベル	説明
計算が GCI を維持中の LCP の停止	0	ALERT	LCP 停止
ローカル チェックポイントの断片化完了	11	INFO	断片の LCP 完了
グローバル チェックポイント完了	10	INFO	GCP 終了
グローバル チェックポイントの開始	9	INFO	GCP の開始:REDO ログのディスクへの書き込み
ローカル チェックポイントの完了	8	INFO	LCP 通常に完了
ローカル チェックポイントの開始	7	INFO	L C P の開始:データのディスクへの書き込み
undo ログのレポートのブロック	7	INFO	UNDO ログのブロック; バッファのオーバーフロー

STARTUP イベント

以下のイベントがノードあるいはクラスタの起動、またはその成功不成功により生成されます。それらによってログの活動を含む起動プロセスの進展に関連する情報も提供されます。

イベント	優先度	深刻度レベル	説明
内部の開始信号受信 STTORY	15	INFO	再起動完了後にブロックを受信
Undo レコード実行	15	INFO	
新しい REDO ログ開始	10	INFO	GCI 維持 X 、最新の保存可能な GCI Y
新しいログの開始	10	INFO	ログ パート X 、MB の開始 Y 、MB の停止 Z
ノードのクラスタへ包含が拒否されました。	8	INFO	設定の間違いによりノードをクラスタに含めることができません。通信の確立の失敗、または他の問題
データ ノード ネーバー	8	INFO	近くのデータ ノードの表示
データ ノードの起動フェーズ X 完了	4	INFO	データ ノードの起動フェーズの完了
ノードが無事クラスタに含まれました	3	INFO	ノード、管理ノード、およびダイナミック ID の表示
データ ノードの起動フェーズの開始	1	INFO	NDB クラスタ ノードの開始
データ ノードのすべての起動フェーズの完了	1	INFO	NDB クラスタノードの開始
データ ノードのシャットダウン開始	1	INFO	データ ノードのスアットダウンが開始されました
データ ノードのシャットダウンの失敗	1	INFO	データ ノードの通常のシャットダウン不可

NODERESTART Events

以下のイベントがノードを再起動すると生成されノード再起動の成功あるいは失敗に関連します。

イベント	優先度	深刻度レベル	説明
ノードの失敗フェーズの完了	8	ALERT	ノード失敗フェーズの完了レポート
ノードが失敗しました。ノードの状態は X	8	ALERT	ノード失敗のレポート
アービトラーターの結果のレポート	2	ALERT	アービトレーション試行で 8 つの頃なる結果があります - アービトレーションのチェックに失敗— 残りは 1/2 以下のノード - アービトレーションのチェックの完了— ノードグループの過半数 - アービトレーションのチェック失敗— 不明なノードグループ - ネットワークのパーティショニング— アービトレーションが必要 - アービトレーション無事終了— ノードX から確認の応答 - Arbitration 失敗 - ノード X から否定的な応答 - ネットワーク パーティショニング - アービトラーターは利用不可 - ネットワーク パーティショニング - アービトラーターの設定なし
フラグメントのコピー完了	10	INFO	
ディレクトリ情報にコピー完了	8	INFO	

配布情報のコピー完了	8	INFO	
フラグメントのコピー開始	8	INFO	
すべてのフラグメントのコピー完了	8	INFO	
GCP 引継ぎ開始	7	INFO	
GCP 引継ぎ開始	7	INFO	
LCP 引継ぎ開始	7	INFO	
LCP 引継ぎ完了 (ステート = X)	7	INFO	
アービトラーターの有無のレポート	6	INFO	アービトラーターには 7 種類の結果が可能 • マネジメント サーバーのアービトレーション スレッドの開始 [ステート=X] • アービトラーターノードの準備 X [チケット=Y] • アービトラーターノードの受信 X [チケット=Y] • アービトラーターノードの開始 X [チケット=Y] • 紛失したアービトラーターノード X - プロセス不良 [ステート=Y] • f 不明なアービトラーターノード X - プロセス不良 [ステート=Y] • 不明なアービトラーターノード X <error msg> [ステート=Y]

STATISTICS イベント

以下のイベントは統計的なものです。トランザクション数、および他のオペレーション、転送したデータ量あるいは受信した個々のノード、およびメモリの使用などに関する情報を提供します。

イベント	優先度	深刻度	説明
ジョブ スケジューリング統計レポート	9	INFO	平均内部ジョブ スケジューリング統計
送信バイト数	9	INFO	平均ノード送信バイト数X
受信バイト数	9	INFO	ノードからの平均受信バイト数X
トランザクション統計のレポート	8	INFO	回数:トランザクション、実行、読み込み、単純読み込み、書き込み、現在のオペレーション、属性情報、および失敗
オペレーションのレポート	8	INFO	オペレーション数
テーブル作成のレポート	7	INFO	
メモリの使用	5	INFO	データおよびメモリの使用状況 (80%, 90%, and 100%)

ERROR イベント

これらのイベントはクラスタのエラーおよび警告に関するものです。1 つ以上の記録がここにある場合、それは重大な誤動作および不具合が発生したことを示しています。

イベント	優先度	深刻度	Description
不明なハートビートによるシステム停止	8	ALERT	ノードX による「不明ハートビートによる」システム停止の宣言
トランスポーターエラー	2	エラー	

トランスポーター警告	8	WARNING	
不明なハートビート	8	WARNING	ノードX 不明なハートビート数Y
一般的な警告イベント	2	WARNING	

INFO イベント

これらのイベントはクラスタおよびログおよびハートビート伝送などのクラスタのメンテナンス活動の状態に関する一般的な情報を提供します。

イベント	優先度	深刻度	説明
伝送ハートビート	12	INFO	ノードへのハートビート伝送X
ログ バイトの作成	11	INFO	ログの部分、ログ ファイル、MB
一般的な情報イベント	2	INFO	

CLUSTERLOG STATISTICSの活用

NDB マネジメント クライアントの **CLUSTERLOG STATISTICS** コマンドではその出力に対する様々な統計を提供します。以下の統計がトランザクション コーオーディネーターからレポートされます。

統計	説明 (回数...)
Trans.Count	コーオーディネーターとしてこのノードで試されたトランザクション回数
Commit Count	コーオーディネーターとしてこのノードで実行されたトランザクション
Read Count	プライマリ キーの読み込み数 (すべて)
Simple Read Count	プライマリ キーの最新の実行値の読み込み
Write Count	プライマリ キーの書き込み (INSERT, UPDATE、および DELETE オペレーションのすべてを含む)
AttrInfoCount	受信したすべての読み込みおよびかみこみの説明に使用したデータの単語数
Concurrent Operations	レポートが取得された時の実行中のすべての同時進行オペレーション数
Abort Count	このノードがコーオーディネーターとして実行したトランザクションの失敗した数
Scans	スキャン (すべて)
Range Scans	インデックスのスキャン数

ndbd プロセスには無限ループで実行されるスケジューラがあります。各ループ スケジューラは以下のタスクを実行します。

1. ソケットからジョブバッファに受信メッセージを読み込みます。
2. 時間を区切ったメッセージが実行されているかチェックします。実行されている場合、同様にジョブ バッファに導入します。
3. ジョブ バッファのメッセージを実行 (ループで) します。
4. ジョブ バッファでメッセージを実行することによって生成された配布メッセージを送信します。
5. メッセージの受信を待ちます。

3 番目のステップで実行されたループ数が **Mean Loop Counter** としてレポートされます。この統計のサイズが TCP/IP のバッファの利用が改善することによって大きくなります。新しいプロセスをクラスタに追加する際にこのモニタ パフォーマンスを使用できます。

Mean send size および **Mean receive size** の統計でノード間の書き込みおよび読み込みの効率を測ることができます。これらの値はバイトで表されます。大きい値はバイト毎の送受信のコストが低いことを意味します。最大は 64k です。

すべてのクラスタ ログの統計をログするには、**NDB** マネジメント クライアントの以下のコマンドを使用できます。

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

注:**STATISTICS** の閾値を 15 に設定するとクラスタ ログが非常に冗長になり、クラスタの ノード数およびクラスタの活動量に直接比例して直ぐにサイズが大きくなります。

単一ユーザーモード

単一ユーザーモード ではデータベースの管理者が MySQL サーバー (SQL ノード) あるいは**ndb_restore** のインスタンスのようにデータベースへのアクセスを単一の API ノードに制限できます。単一ユーザーモードに鳴ると、すべての他の API ノードが優雅に閉じられすべての実行中のトランザクションが中断されます。新しいトランザクションは実行されません。

クラスタが単一ユーザーモードになると、指定された API ノードのみがデータベースにアクセスできます。

クラスタが単一ユーザーモードに何時なった表示するには**ALL STATUS** コマンドを使用できます。

例:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

コマンドが実行されクラスタが単一ユーザーモードになると、ノード ID が 5 の API ノードがクラスタの唯一許可されたユーザーになります。

前のコマンドで指定されたノードは API ノードのなりますので、他のノード タイプを指定しようとしても拒否されます。

注:前のコマンドが呼び出されると、指定されたノードで実行中のすべてのトランザクションは中断されて接続の閉じますので、サーバーを再起動する必要があります。

EXIT SINGLE USER MODE のコマンドはクラスタのデータ ノードの状態を単一ユーザーモードから通常モードに変更します。接続を待っている（つまり、クラスタが準備が整い利用できる状態）MySQL Server—などのAPI ノード—は、再度接続を許可されます。単一ユーザーノードとしてAPI ノードはその状態が変更中あるいは変更後も（接続されている場合）実行し続けます。

例:

```
ndb_mgm> EXIT SINGLE USER MODE
```

単一ユーザーモードで稼働中のノード不良を処理する 2 つの方法をお勧めしています。

- メソッド 1:
 1. すべての単一ユーザーモードのトランザクションを終了する
 2. **EXIT SINGLE USER MODE** コマンドを発行する
 3. クラスタのデータ ノードを再起動する

- メソッド 2:

単一ユーザーモードに入る前にデータベース ノードを再起動する

MySQL Cluster のオンライン バックアップ

この項ではバックアップの作成の仕方およびその後のバックアップのデータベースの保存に仕方について説明します。

クラスタ バックアップの概念

バックアップは所定の時間でのデータベースのスナップショットです。バックアップは 3 つの主要要素で構成されます。

- メタデータ:すべてのデータベース テーブルの名前と定義
- テーブル レコード:バックアップしたときのデータベース テーブルに保存された実際のデータ
- トランザクション ログ:データがデータベースにどのように何時保存されたを記録したシーケンシャルなレコード

これらの各要素はバックアップに使用されたすべてのノードに保存されます。バックアップ中に、各ノードはこれらの 3 つの要素をディスクの 3 つのファイルに保存します。

- [BACKUP-backup_id.node_id.ctl](#)

コントロール情報およびメタデータを含むコントロール ファイルです。各ノードは同じテーブル定義 (クラスタのすべてのテーブルの) をこのファイルのそれ自身のバージョンに保存します。

- [BACKUP-backup_id-0.node_id.data](#)

テーブル レコードを含むデータ ファイルで、フラグメント毎に保存されます。つまり、バックアップ中に異なるノードが異なるフラグメントを保存します。各ノードに保存されたファイルがテーブルにどのデータが入っているかを示すヘッダーから始まります。レコードのリストに続いてすべてのレコードのチェックサムを含むフッターが表示されます。

- [BACKUP-backup_id.node_id.log](#)

実行されたトランザクションのレコードを含むログ ファイルバックアップで保存されたテーブルのトランザクションのみがログの保存されます。バックアップに関わったノードは異なるノードは異なるデータベースのフラグメントをホストするために異なるレコードを保存します。

上記おリストで、[backup_id](#) はバックアップ識別子を表しており、[node_id](#) はファイルを作成する一意の識別子を表します。

バックアップを作成するためのマネジメント クライアントの使用

バックアップを開始する前に、クラスタが実行するものに対して適切に設定されているか確認します。([クラスタバックアップの設定](#) 参照。)

マネジメント クライアントを使用したバックアップの作成には以下のステップが含まれます。

1. マネジメント クライアントを起動 ([ndb_mgm](#)) します。
- 2.
3. マネジメント クライアントが次のように応答します。

```
Waiting for completed, this may take several minutes
Node 1: Backup backup\_id started from node management\_node\_id
```

ここでは、[backup_id](#) は特定のバックアップのための一意の識別子です。(設定が保存できるようになっている場合、この識別子もクラスタ ログの保存できます。) [management_node_id](#) はマネジメント クライアントが接続されるマネジメントのノード ID です。

これはクラスタがバックアップの要求を受信して処理することを意味しています。それはバックアップが完了したことを意味するものではありません。

注:バックアップのメッセージは MySQL 5.1.12 or 5.1.13 のクラスタには記録されませんでした。バックアップ オペレーションのログは MySQL 5.1.14 には保存されました (Bug #24544 参照)。

4. バックアップが完了すると、マネジメント .クライアントはバックアップ完了を以下のように示します。

```
Node 1: Backup backup\_id started from node management\_node\_id completed
StartGCP: 417599 StopGCP: 417602
#Records: 105957 #LogRecords: 0
```

Data: 99719356 bytes Log: 0 bytes

`StartGCP`、`StopGCP`、`#Records`、`#LogRecords`、`Data`、および `Log` に表示された値はクラスタの特性により変化します。

クラスタのバックアップは各データ ノードの `DataDir` の `BACKUP` サブディレクトリのデフォルトによって作成されます。これは 1 つ以上のデータ ノードに個別に、または `config.ini` ファイルのすべてのクラスタ データ ノードに `BackupDataDir` 設定パラメータを使用して [Identifying Data Nodes \[897\]](#) で説明したようにオーバーライドされます。所定の `backup_id` でバックアップに作成されたバックアップファイルはディレクトリの `BACKUP-backup_id` の名前のサブディレクトリに保存されます。

既に実行中のバックアップを中断します。

1. マネジメント クライアントを終了します。
2. このコマンドを実行します。

```
ndb_mgm> ABORT BACKUP backup_id
```

`backup_id` の番号はバックアップが開始されたとき (`Backup backup_id started from node management_node_id` のメッセージで) のマネジメント クライアントの応答に含まれるバックアップの識別子です。

3. マネジメント クライアントは中断要求を `Abort of backup backup_id ordered` で認識します。注:この段階で、マネジメント クライアントはクラスタ データ ノードからこの要求の応答を受け取っておらず、バックアップは実際のところまだ中断されていません。
4. バックアップが中断されると、マネジメント クライアントは中断されたことを以下に類似した方法でレポートします。

```
Node 1: Backup 3 started from 5 has been aborted. Error: 1321 - Backup aborted by user request: Permanent error: User defined error
Node 3: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
```

この例では、4 つのデータ ノードを持つクラスタのサンプル出力を例示しました。そこでの中断されるバックアップのシーケンス番号は 3 で、クラスタ マネジメント クライアントが接続されるマネジメント ノードのノード ID は 5 です。その中断の理由がユーザーの要求によるバックアップ レポートの中断を完了する最初のノード(残りのノードはバックアップの中断は未指定の内部エラーによるものだとレポートします。)注:クラスタ ノードが `ABORT BACKUP` コマンドに対し特定の順序で応答するかは保証はありません

`Backup backup_id started from node management_node_id has been aborted` メッセージはバックアップが終了し、このバックアップに関連したすべてのファイルがクラスタのファイルシステムから削除されたことを意味します。

このコマンドを使用してシステム シェルから実行中のバックアップを中断することもできます。

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

注:`ABORT BACKUP` が発行された時に ID `backup_id` を持つバックアップが無い場合、マネジメント クライアントは応答せず、無効な中断コマンドが送信されましたの表示もされません。

クラスタのバックアップの復旧方法

クラスタの復旧プログラムは個別のコマンドライン ユーティリティ `ndb_restore` として実装されており、通常は MySQL `bin` ディレクトリにあります。このプログラムはバックアップの結果生成されたファイルを読みデータベースに保存された情報を挿入します。

`ndb_restore` をバックアップを作成するために使用した `START BACKUP` コマンドにより作成された各バックアップファイルに対し一度実行する必要があります。(バックアップを作成するためのマネジメント クライアントの使用 参照)これはバックアップが作成された時点のクラスタのデータ ノード数の相当します。

注:`ndb_restore` を使用する前に、並列で複数のデータ ノードを復旧していない限り、クラスタが単一ユーザーモードで動作させることをお勧めします。単一ユーザーモードに関する情報は、[単一ユーザーモード](#) を参照してください。

このユーティリティの一般的なオプションを以下に示します。

```
ndb_restore [-c connectstring] [-n node_id [-m] -b backup_id -r /path/to/backup/files
```

-c オプションは `ndb_restore` にクラスタ マネジメント サーバーの位置を知らせる接続文字列を指定するために使用します。(接続文字列に関する情報は、[クラスタの 接続文字列](#) を参照してください。)このオプションを使用していない場合、`ndb_restore` が `localhost:1186` のマネジメント サーバーに接続を試みます。このユーティリティは API ノードとして機能し、クラスタ マネジメント サーバーに接続するにはフリー接続の「slot」が必要です。このことは少なくとも 1 つの [\[API\]](#) あるいは [\[MYSQLD\]](#) セクションがあり、クラスタの `config.ini` ファイルで使用できるということです。最低でも 1 つの空の [\[API\]](#) あるいは [\[MYSQLD\]](#) セクションを この目的のために MySQL サーバーあるいは他のアプリケーションで使用されていない `config.ini` の持つことはいい考えです ([SQL および他の API ノードの定義](#) 参照)。

`ndb_restore` がクラスタに `ndb_mgm` マネジメント クライアントの `SHOW` コマンドを使用して接続されていることを検証できます。この検証を以下のシステム シェルで行うこともできます。

```
shell> ndb_mgm -e "SHOW"
```

-n はバックアップが行われたデータ ノードのノード ID を指定するために使用します。

最初に `ndb_restore` 復旧プログラムを実行する際には、メタデータも復旧する必要があります。換言すれば、データベースのテーブルを再度作成する必要があります—これはそれを **-m** オプションで行うことで実行できます。バックアップを復旧する際にはクラスタに空のデータベースが無ければならないことを忘れないでください。(言い換えれば、復旧を行う前に `ndbd` を `--initial` で起動する必要があります。またデータ ノード `DataDir` にあるディスク データ ファイルを手動で削除する必要があります。)

-b オプションはバックアップの ID あるいはシーケンス番号を指定するために使用し、バックアップの完了時に `Backup backup_id completed` メッセージのマネジメント クライアントに表示される番号と同じです。([バックアップを作成するためのマネジメント クライアントの使用](#) 参照。)

-r オプションは必要で、`ndb_restore` にバックアップ ファイルのあるディレクトリの所在を知らせるために使用されます。重要クラスタのバックアップを復旧する際、同じバックアップ ID を持つバックアップのすべてのデータ ノードの復旧を確認する必要があります。

作成されたものとは異なる設定のデータベースにバックアップを復旧することができます。例えば、[2](#) および [3](#) のノード ID を持つ 2 つのデータベース ノードの持つクラスタで作成されたバックアップ ID [12](#) を持つバックアップは、4 つのノードを持つクラスタに保存する必要があります。次に `ndb_restore` 2 回実行する必要があります — バックアップが行われたクラスタの各データベース ノードに 1 回ずつです。しかし、`ndb_restore` は MySQL の 1 つのバージョンで動作しているクラスタで作成されたバックアップを異なる MySQL バージョンで動作中のクラスタの常復旧できるとは限りません。詳細については、[クラスタのアップグレードおよびダウングレードの互換性](#) をご参照してください。

注:復旧を素早く行いたい場合には、十分な数のクラスタの接続が可能な場合データは並列で復旧できます。つまり、並列で複数のノードに復旧する際、各同時進行の `ndb_restore` プロセスに利用できるクラスタ `config.ini` ファイルに [\[API\]](#) あるいは [\[MYSQLD\]](#) セクションを持つ必要があります。しかし、データ ファイルは常にログの前に適用する必要があります。

このプログラムのオプションの完全なリストを以下のテーブルに示します。

長いフォーム	短いフォーム	Description	デフォルト値
<code>--backup-id</code>	<code>-b</code>	バックアップ シーケンス ID	0
<code>--backup_path</code>	該当なし	バックアップ ファイルへのパス	./
<code>--character-sets-dir</code>	該当なし	文字セット情報が得られるディレクトリの指定	該当なし
<code>--connect</code> , <code>--connectstring</code> , あるいは <code>--ndb-connectstring</code>	<code>-c</code> あるいは <code>-C</code>	接続文字列の設定 <code>[nodeid=node_id;] [host=]host[:port]</code> フォーマット	該当なし
<code>--core-file</code>	該当なし	エラーの場合コア ファイルを記述	TRUE
<code>--debug</code>	<code>-#</code>	デバッグ ログの出力	d:t:O,/tmp/ ndb_restore.trace

<code>--dont_ignore_systab_0</code>	<code>-f</code>	復旧の際システム テーブルを無視しない— 試験用で生産用ではありません	<code>FALSE</code>
<code>--help</code> あるいは <code>--usage</code>	<code>-?</code>	利用可能なオプションと現在の値を含むヘルプ メッセージの表示、そして終了	[該当なし]
<code>--ndb-mgmd-host</code>	<code>None</code>	マネジメント サーバーの接続のためのホストとポートを <code>host[:port]</code> フォーマットに設定。これは <code>--connect</code> , <code>--connectstring</code> 、あるいは <code>--ndb-connectstring</code> と同じだが、 <code>nodeid</code> の指定はない	該当無し
<code>--ndb-nodegroup-map</code>	<code>-z</code>	ノード グループのマッピングの指定 — 構文: リスト (<code>source_nodegroup</code> , <code>destination_nodegroup</code>)	該当なし
<code>--ndb-nodeid</code>	該当なし	次にノード ID の指定 <code>ndb_restore</code> プロセス	<code>0</code>
<code>--ndb-optimized-node-selection</code>	該当なし	トランザクションのノード選択の最適化	<code>TRUE</code>
<code>--ndb-shm</code>	該当なし	利用可能な場合共有メモリの使用	<code>FALSE</code>
<code>--no-restore-disk-objects</code>	<code>-d</code>	テーブル スペースやログ ファイル グループなどのディスク データ オブジェクトは復旧しない	<code>FALSE</code> (換言すると、このオプションが使用されない場合ディスク データ オブジェクトを復旧する)
<code>--nodeid</code>	<code>-n</code>	ノードのバックアップ ファイルを指定した ID で使用する	<code>0</code>
<code>--parallelism</code>	<code>-p</code>	復旧プロセスで使用される並列トランザクションを 1 ~ 1024 設定する	<code>128</code>
<code>--print</code>	該当なし	データとログと次にプリントする <code>stdout</code>	<code>FALSE</code>
<code>--print_data</code>	該当なし	データと次のプリントする <code>stdout</code>	<code>FALSE</code>
<code>--print_log</code>	該当なし	ログと次にプリントする <code>stdout</code>	<code>FALSE</code>
<code>--print_meta</code>	該当なし	メタデータを次にプリントする <code>stdout</code>	<code>FALSE</code>
<code>--restore_data</code>	<code>-r</code>	データとログの復旧	<code>FALSE</code>
<code>--restore_epoch</code>	<code>-e</code>	エポック データをステータス テーブルに復旧する。適切な場合 <code>ID0</code> を持つ <code>cluster.apply_status</code> の行に挿入あるいは更新されます。—これはレプリケーションを MySQL Cluster レプリケーション スレーブで開始する場合に便利です。	<code>FALSE</code>
<code>--restore_meta</code>	<code>-m</code>	テーブル メタデータの復旧	<code>FALSE</code>
<code>--version</code>	<code>-V</code>	バージョン情報を更新して終了	[該当なし]

注:`ndb_restore` は一時的および永久的なエラーをレポートします。一時的なエラーの場合、そのエラーから回復することができる場合があります。MySQL 5.1.12 以降では、そのような場合、`Restore successful, but encountered temporary error, please look at configuration` のようにレポートされます。

クラスタ バックアップの設定

バックアップには 4 つの設定パラメータが必要です。

- `BackupDataBufferSize`
ディスクに書き込まれる前のデータのバッファに必要なメモリ容量
-

BackupLogBufferSize

これがディスクに書き込まれる前のログ レコードのバッファに必要なメモリ容量

- ### BackupMemory

データベース ノードで割り当てられたバックアップ用のメモリの合計これはデータ バックアップ用のバッファおよびログのバックアップ用バッファに割り当てられたメモリの合計になります。

- ### BackupWriteSize

ディスクに書き込まれたブロックのデフォルトのサイズこれはデータのバックアップ用バッファおよびログのバックアップ用バッファの両方に適用されます。

- ### BackupMaxWriteSize

ディスクに書き込まれたブロックの最大サイズこれはデータのバックアップ用バッファおよびログのバックアップ用バッファの両方に適用されます。

これらのパラメータに関する詳細は [Backup Parameters \[911\]](#) にあります。

バックアップのトラブルシューティング

バックアップ要求を出した際にエラーコードが返された場合は、その原因はメモリあるいはディスクのスペース不足の場合がよくあります。バックアップに十分なメモリが割り当てられているか確認する必要があります。重要[BackupDataBufferSize](#) および [BackupLogBufferSize](#) を設定してその合計が 4MB 以上の場合、[BackupMemory](#) も同様に設定する必要があります。[BackupMemory \[912\]](#) 参照。

バックアップに必要な容量に対してハードドライブのパーティションに十分なスペースがあることも確認する必要があります。

NDB は繰り返しの読み込みをサポートしていないので、これが復旧プロセスで問題を引き起こします。バックアップのプロセスは「hot」ですが、バックアップから MySQL クラスタを復旧するのは 100% 「hot」 なプロセスではありません。これは復旧プロセスの期間に対してトランザクションの実行によって復旧データから繰り返し出来ない読み込みがあるからです。これは復旧を実行中のデータの状態が一貫していないからです。

クラスタ ユーティリティ プログラム

この項では [mysql/bin](#) ディレクトリにある MySQL Cluster のユーティリティ プログラムについて説明します。これらのすべて—[ndb_size.pl](#) および [ndb_error_reporter](#) を除く—はスタンドアロンのバイナリでシステム シェルから使用でき、MySQL サーバー (MySQL サーバーをクラスタに接続する必要もなし) に接続する必要はありません。

これらのユーティリティは NDB API を使用したお客様のアプリケーションの記述する際の例として使用できます。これらのプログラムのソースコードは MySQL5.1 ツリーの [storage/ndb/tools](#) ディレクトリにあります ([ソースのディストリビューションを使用した MySQL のインストール](#) 参照)。NDB API はこのマニュアルでは説明していません。この API に関する情報は [NDB API Guide](#) を参照してください。

すべてのNDB ユーティリティが簡単な説明とともにここに掲載しています。

- [ndb_config](#): クラスタ設定オプション値を取り出します。
- [ndb_delete_all](#): 所定のテーブルからすべての行を削除します。
- [ndb_desc](#): NDB テーブルのすべてのプロパティのリストです。
- [ndb_drop_index](#): NDB テーブルから指定したインデックスを削除します。
- [ndb_drop_table](#): NDB テーブルを削除します。
- [ndb_error_reporter](#): クラスタの問題を分析する情報を集めるために使用します。

- `ndb_mgm`:これは [マネジメント クライアントのコマンド](#) で説明したMySQL Cluster のマネジメント クライアントです。
- `ndb_print_backup_file`:クラスタのバックアップ ファイルから取得した分析情報をプリントします。
- `ndb_print_schema_file`:クラスタ スキーマ ファイルから取得したの分析情報をプリントします。
- `ndb_print_sys_file`:クラスタ システム ファイルから取得した分析情報をプリントします。
- `ndb_restore`:バックアップからクラスタを復旧するために使用されるユーティリティです。詳細については、[クラスタのバックアップの復旧方法](#) をご参照してください。
- `ndb_select_all`:NDB テーブルのすべての行をプリントします。
- `ndb_select_count`:1 つ以上の NDB テーブルの行番号を取得します。
- `ndb_show_tables`:クラスタのすべての NDB テーブルを表示します。
- `ndb_size.pl`:所定の非クラスタ データベースのすべてのテーブルを調べ、NDB ストレージ エンジンを使用するために変換された時に必要なそれぞれのストレージの容量を計算します。
- `ndb_waiter`:マネジメント クライアントのコマンド `ALL STATUS` を同じような方法でクラスタのデータ ノードの状態をレポートします。

これらのユーティリティの多くは機能するためにはクラスタのマネジメント サーバーの接続する必要があります。その例外は `ndb_size.pl` (以下参照)、およびクラスタのデータ ノード ファイルシステムにアクセスしよってデータノードのホストで動作する必要のある以下のユーティリティです。

- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

`ndb_size.pl` は Perl スクリプトでこれもまたシェルで使用されます。しかし、それは MySQL アプリケーションですので MySQL サーバーに接続できることが必要です。このスクリプトを使用する際の要件については、[ndb_size.pl — NDBCluster サイズ仕様エステイメーター](#) を参照してください。

`ndb_error_reporter` もまた Perl スクリプトです。それはクラスタのデータノードおよびマネジメント ノード ログをバグ レポートと一緒に tarball に集めるために使用します。ノードのファイルシステムに遠隔でアクセスするために `ssh` あるいは `scp` を使用できます。

これらのユーティリティ (`ndb_mgm` および `ndb_restore` を除く) のそれぞれに関する情報は次項以降で説明します。

注:これらのすべてのユーティリティ (`ndb_size.pl` および `ndb_config` を除く) は [MySQL Cluster プロセスのコマンド オプション](#) で説明したオプションを使用できます。各ユーティリティプログラムに特定のその他のオプションはそれぞれのプログラム一覧表で説明しています。

これらのオプションを使用する順番は一般的にはそれほど重要ではありません。例えば、これらのすべてのコマンドはすべての同じ出力をもたらします。

- `ndb_desc -c localhost fish -d test`
- `ndb_desc fish -c localhost -d test`
- `ndb_desc -d test fish -c localhost`

ndb_config — NDB 設定情報の抽出

このツールはデータ ノード、SQL ノード、および API ノードの設定情報をクラスタのマネジメント ノード (およびその `config.ini` ファイル) から抽出します。

使用法:

ndb_config options

このユーティリティーに利用できるoptionsはどことなく他のユーティリティーに使用されるものとは異なっておりますので、次項でいくつかの例と共にその全体を説明します。

オプション:

- `--usage`, `--help`, あるいは `-?`

は `ndb_config` に利用可能なオプションのリストをプリントさせ、次に終了します。

- `--version`, `-V`

は `ndb_config` にバージョン情報の文字列をプリントさせ、次に終了します。

- `--ndb-connectstring=connect_string`

マネジメント サーバーの接続に使用される `connectstring` を指定します。接続文字列のフォーマットは [クラスタの接続文字列](#) での説明と同じで、`localhost:1186` のデフォルトです。

このオプションへの `-c` の短いバージョンとして使用は `ndb_config` に対しては MySQL 5.1.12 以降でポートしています。

- `--config-file=path-to-file`

マネジメント サーバーの設定ファイル (`config.ini`) にパスを提供します。これは相対あるいは絶対パスです。マネジメント ノードが `ndb_config` が呼び出されるものと異なるホストにある場合、絶対パスを使用する必要があります。

- `--query=query-options`, `-q query-options`

これはコンマ区切りの クエリ オプションのリスト — つまり返される 1 つ以上のノード属性です。これらには `id` (ノード ID)、タイプ (ノード タイプ — つまり、`ndbd`、`mysqld`、あるいは `ndb_mgmd`)、およびそれらの値が取得される設定パラメータが含まれています。

例、`--query=id,type,indexmemory,datamemory` はノード ID、ノード タイプ、`DataMemory`、および `IndexMemory` を各ノードに返します。

注:所定のパラメータがあるタイプのノードに適用できない場合、空の文字列が相当する値に返されます。詳細はこの項の後の部分にあるサンプルを参照してください。

- `--host=hostname`

設定情報が返されるノードのホスト名を指定します。

- `--id=node_id`, `--nodeid=node_id`

設定情報が返されるノードのノード ID の指定に使用されます。

- `--nodes`

(`ndb_config` に `[ndbd]` セクションでのみ定義されたパラメータの情報のプリントを指示します。現在、このオプションは値のチェックのみで機能していませんが将来的には `[tcp]` および他のクラスタ 設定ファイルのセクションで設定されたパラメータのクエリに使用する予定です。

- `--type=node_type`

指定された `node_type` (`ndbd`、`mysqld`、あるいは `ndb_mgmd`) ノードに適用した設定値のみが返されるように結果にフィルタをかけます。

- `--fields=delimiter`, `-f delimiter`

結果のフィールドを分けるために使用する `delimiter` 文字列を指定します。デフォルトは `「、」` (コンマです)。

注:`delimiter` がスペースあるいはエスケープ (ラインフィード文字の `\n` など) を含む場合、引用する必要があります。

- `--rows=separator, -r separator`

結果の行を分けるために使用する `separator` 文字列を指定します。デフォルトはスペース文字です。

注:`separator` がスペースあるいはエスケープ (ラインフィード文字の `\n` など) を含む場合、それを引用する必要があります。

例:

1. クラスタのノード ID および各ノードのタイプを取得する。

```
shell> ./ndb_config --query=id,type --fields=';' --rows='\n'
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqlld
7:mysqlld
8:mysqlld
9:mysqlld
```

この例では、`--fields` オプションを使用して ID および各ノードのタイプをコロン (:) で分けました。`--rows` オプションは出力の新しい行の各ノードに値をつけました。

2. データ、SQL、および API ノードがマネジメント サーバーの接続に使用する接続文字列を作成します。

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini --query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
192.168.0.179:1186
```

3. `ndb_config` を実行してデータ ノードのみ (`--type` オプションを使用して) をチェックし、各ノード ID およびホスト名、およびその `DataMemory`、`IndexMemory`、並びに `DataDir` パラメータを表示します。

```
shell> ./ndb_config --type=ndbd --query=id,host,datamemory,indexmemory,datadir -f ':' -r '\n'
1 : 192.168.0.193 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
2 : 192.168.0.112 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
3 : 192.168.0.176 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
4 : 192.168.0.119 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

この例では、短いオプション `-f` および `-r` をフィールド区切り文字および行の分離符号の設定に使用しました。

4. 1 つの特別なものを除いてホストを結果から除外するには `--host` オプションを使用します。

```
shell> ./ndb_config --host=192.168.0.176 -f ':' -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

この例でも、クエリされる属性を決定するために短いフォーム `-q` を使用します。

同様に、`--id` あるいは `--nodeid` オプションを使用して特定の ID を持つノードに結果を制限できます。

ndb_delete_all — NDB テーブルからのすべての行を削除する

`ndb_delete_all` は所定の NDB テーブルからすべての行を削除します。場合によっては、これは `DELETE` あるいはむしろ `TRUNCATE` よりもかなり速くできます。

使用法:

```
ndb_delete_all -c connect_string tbl_name -d db_name
```

これにより データベース `db_name` のテーブル `tbl_name` の行をすべて削除します。それは MySQL の `TRUNCATE db_name.tbl_name` で実行するのと全く同じです。

その他のオプション:

- `--transactional, -t`

このオプションを使用すると単一のトランザクションとして実行されるオペレーションを削除します。

警告:非常に大きなテーブルでは、このオプションを使用したこれを使用することで超過している多くのオペレーションが利用できるようになります。ter to be exceeded.

ndb_desc — NDB テーブルの説明

ndb_desc は 1 つ以上の NDB テーブルの詳細な説明を提供します。

使用法:

```
ndb_desc -c connect_string tbl_name -d db_name
```

サンプル出力:

MySQL テーブルの作成およびポピュレーション ステートメント :

```
USE test;

CREATE TABLE fish (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(20),

  PRIMARY KEY pk (id),
  UNIQUE KEY uk (name)
) ENGINE=NDBCLUSTER;

INSERT INTO fish VALUES
('guppy'), ('tuna'), ('shark'),
('manta ray'), ('grouper'), ('puffer');
```

ndb_desc の出力:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 16777221
Fragment type: 5
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 268
Row Checksum: 1
Row GCI: 1
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
name Varchar(20;latin1_swedish_ci) NULL AT=SHORT_VAR ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
uk(name) - OrderedIndex
PRIMARY(id) - OrderedIndex
uk$unique(name) - UniqueHashIndex

-- Per partition info --
Partition Row count Commit count Frag fixed memory Frag varsized memory
2      2      2      65536      327680
1      2      2      65536      327680
3      2      2      65536      327680

NDBT_ProgramExit: 0 - OK
```

その他のオプション:

- `--extra-partition-info, -p`

テーブルのパーティションに関するその他の情報のプリント

- 複数のテーブルに関する情報はスペースで分離されたそれらの名前を使用して `ndb_desc` の 1 回の実行で取得できます。すべてのテーブルは同じデータベースにあることが必要です。

ndb_drop_index

説明:NDB テーブルから指定したインデックスを削除します。このユーティリティは NDB API アプリケーションを記述する際の例として使用することをお勧めします。 — 詳細はこの項の後にある警告を参照してください。

使用法:

```
ndb_drop_index -c connect_string table_name index -d db_name
```

上記のステートメントは次の名前のインデックス `index` を `database` の `table` から削除します

その他のオプション:このアプリケーションに特定のものはありません。

警告:NDB API を使用してクラスタ テーブルのインデックスで実行したオペレーションは MySQL には表示されず MySQL サーバーではテーブルを使用できません。インデックスを削除するためにこのプログラムを使用して SQL ノードにアクセスしようとすると、以下のようにエラーになります。

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *****
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.1.12-beta-20060817

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+-----+
| Tables_in_ctest1 |
+-----+
| a                |
| bt1              |
| bt2              |
| dogs             |
| employees        |
| fish             |
+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

そのような場合、テーブルを MySQL で再度使用できるようにする 唯一 のオプションはテーブルを削除して再度作成することです。テーブルを削除するには SQL ステートメント `DROP TABLE` あるいは `ndb_drop_table` ユーティティ (`ndb_drop_table` 参照) のいずれかを使用します。

ndb_drop_table

説明:指定された NDB テーブルを削除します。(NDB ではなくストレージ エンジンで作成されたテーブルでこれを使用すると、エラー 723 で失敗します。そのようなテーブルは存在しません。)ケースによってはこのオペレーションは非常に速く — MySQL の NDB テーブルの `DROP TABLE` を使用した時よりも非常に速い順序になります。

使用法:

```
ndb_drop_table -c connect_string tbl_name -d db_name
```

その他のオプション:無し

ndb_error_reporter

説明:アーカイブをバグあるいは他のクラスタの問題の分析を支援するデータ ノードあるいはマネジメント ノードからアーカイブを作成します。MySQL Cluster でバグのレポートをファイルするときはこのユーティリティを使用することを強くお勧めします。

使用法:

```
ndb_error_reporter path/to/config-file [username] [--fs]
```

このユーティリティはマネジメント ノードのホストで使用されるようになっており、マネジメント ホストの設定ファイル ([config.ini](#)) が必要です。オプションでは、SSH でクラスタのデータ ノードにアクセスできるユーザーの名前を提供できます。それによりデータ ノードのログ ファイルをコピーします。次に、それが実行されている同じディレクトリで作成されたアーカイブのこれらのファイルを含みます。アーカイブの名前は [ndb_error_report_YYYYMMDDHHMMSS.tar.bz2](#)、ここでは [YYYYMMDDHHMMSS](#) は日時の文字列です。

`--fs` を使用すると、データ ノードのファイルシステムはマネジメント ホストにもコピーされ、このスクリプトで作成されたアーカイブに含まれます。データ ノードのファイルシステムは圧縮後も非常に大きくなりますので、このオプションを使用して作成したアーカイブを特別な要求が無い限り MySQL ABに送らないようお願いします。

ndb_print_backup_file

説明:クラスタのバックアップ ファイルから分析情報を取得します。

使用法:

```
ndb_print_backup_file file_name
```

[file_name](#) はクラスタのバックアップ ファイル名です。これはクラスタのバックアップ ディレクトリにあるどのファイル ([.Data](#)、[.ctl](#)、あるいは [.log](#) ファイル) でもできます。これらのファイルはサブディレクトリ [BACKUP-#](#) の下のデータ ノードのバックアップ ディレクトリにあります。そこでは <#> はバックアップのシーケンス番号です。クラスタのバックアップ ファイルおよびそれらのコンテンツに関する詳細は、[クラスタ バックアップの概念](#) を参照してください。

[ndb_print_schema_file](#) および [ndb_print_sys_file](#) (そして、マネジメント サーバーホストで動作するあるいはマネジメント サーバーに接続する他のほとんどの [NDB](#) とは異なり) [ndb_print_backup_file](#) は、それが直接データ ノードのファイルシステムにアクセスするため、クラスタのデータ ノードで動作する必要があります。それはマネジメント サーバーを使用しないため、このユーティリティはマネジメント サーバーが稼動していない時、およびクラスタが完全にシャットダウンされた時に使用されます。

その他のオプション:無し

ndb_print_schema_file

説明:クラスタのスキーマ ファイルから分析情報を取得します。

使用法:

```
ndb_print_schema_file file_name
```

[file_name](#) はクラスタのスキーマのファイル名です。

[ndb_print_backup_file](#) および [ndb_print_sys_file](#) (そして、マネジメント サーバーホストで動作するあるいはマネジメント サーバーに接続する他のほとんどの [NDB](#) ユーティリティとは異なり) [ndb_schema_backup_file](#) は、それが直接データ ノードのファイルシステムにアクセスするため、クラスタのデータ ノードで動作する必要があります。それはマネジメント サーバーを使用しないため、このユーティリティはマネジメント サーバーが稼動していない時、およびクラスタが完全にシャットダウンされた時に使用されます。

その他のオプション:なし

ndb_print_sys_file

説明:クラスタのシステム ファイルから分析情報を取得します。

使用法:

```
ndb_print_sys_file file_name
```

`file_name` はクラスタのシステム ファイル (sysfile) 名です。クラスタ システム ファイルはノードのデータ ディレクトリ (DataDir) にあります。このディレクトリのシステム ファイルへのパスは `ndb # fs/D#/DBD/H/P#.sysfile` に一致します。それぞれのケースで、`#` は番号 (必ずしも同じ番号である必要はありません) を表します。.

`ndb_print_backup_file` および `ndb_print_schema_file` (そして、マネジメント サーバーで動作するあるいはマネジメント サーバーに接続するほとんどの他の NDB ユーティリティとは異なり) `ndb_print_backup_file` は、それがデータ ノードのファイルシステムと直接アクセスするので、クラスタのデータ ノードで動作する必要があります。それはマネジメント サーバーを使用しないため、このユーティリティはマネジメント サーバーが稼動していない時、およびクラスタが完全にシャットダウンされた時に使用されます。

その他のオプション:なし

ndb_select_all

説明:すべての行を NDB テーブルから `stdout` にプリントします。

使用法:

```
ndb_select_all -c connect_string tbl_name -d db_name [> file_name]
```

その他のオプション:

- `-l lock_type`, `--lock=lock_type`

テーブルを読み込んでいるときにロックを使用します。`lock_type` に可能な値は:

- `0`: ロックを読む
- `1`: ホールドしてロックを読む
- `2`: 排他的読み込みロック

このオプションにはデフォルトの値はありません。

- `-o index_name`, `--order=index_name`

`index_name` の名前のインデックスに基づいて出力を調整します。これはオンデックスの名前で、カラムの名前でなく、インデックスは作成された時に、明示的に名前を付ける必要があります。

- `-z`, `--descending`

降順で出力を分類します。このオプションは `-o` (`--order`) オプションに関連してのみ使用されます。

- `--header=FALSE`

出力からカラムのヘッダーを除外します。

- `-x`, `--useHexFormat`

すべての数値を16進数のフォーマットで表示します。これは文字列あるいは日時値に含まれる数値の出力に影響を与えません。

`-D character`, `--delimiter=character`

`character` をカラムの区切り文字で使えるようにします。この区切り文字ではターベルのデータ カラムのみが区切られます。

デフォルトの区切り文字はタブの文字です。

- `--disk`

出力にディスク リファレンス カラムを追加します。このカラムは非インデックス カラムを持つディスク データ テーブルにのみノン エンプティです。

- `--rowid`

行が保存されるフラグメントに関する情報を提供する `ROWID` カラムを追加します。

- `--gci`

各行が最後に更新された時にグローバル チェックポイントを表示する出力にカラムを追加します。チェックポイントに関する詳細は、[MySQL Cluster の用語](#) および [ログ イベント](#) を参照してください。

- `-t, --tupscan`

tubpe の順序でテーブルをスキャンします。

- `--nodata`

テーブルのデータを削除します。

サンプル出力:

MySQL `SELECT` ステートメントからの出力

```
mysql> SELECT * FROM ctest1.fish;
+-----+
| id | name |
+-----+
| 3 | shark |
| 6 | puffer |
| 2 | tuna |
| 4 | manta ray |
| 5 | grouper |
| 1 | guppy |
+-----+
6 rows in set (0.04 sec)
```

`ndb_select_all` に相当する実行からの出力:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id  name
3   [shark]
6   [puffer]
2   [tuna]
4   [manta ray]
5   [grouper]
1   [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

`ndb_select_all` の出力ではすべての文字列の値は角括弧 (「[...]」) で括ります。他の例については、作成されたテーブルを考慮し以下のように配布します。

```
CREATE TABLE dogs (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(25) NOT NULL,
  breed VARCHAR(50) NOT NULL,
  PRIMARY KEY pk (id),
  KEY ix (name)
```

```
)
TABLESPACE ts STORAGE DISK
ENGINE=NDB;
```

```
INSERT INTO dogs VALUES
('Lassie', 'collie'),
('Scooby-Doo', 'Great Dane'),
('Rin-Tin-Tin', 'Alsatian'),
('Rosscoe', 'Mutt');
```

これはいくつかの追加の `ndb_select_all` オプションの使用に関して例示します。

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI id name breed DISK_REF
834461 2 [Scooby-Doo] [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878 4 [Rosscoe] [Mutt] [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463 3 [Rin-Tin-Tin] [Alsatian] [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657 1 [Lassie] [Collie] [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned
NDBT_ProgramExit: 0 - OK
```

ndb_select_count

説明: 1 つ以上の **NDB** テーブルの行数をプリントします。1 つのテーブルでは、その結果は MySQL ステートメント `SELECT COUNT(*) FROM tbl_name` を使用して得た結果と同じです。.

使用法:

```
ndb_select_count [-c connect_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

その他のオプション: このアプリケーションに特定のものはありません。しかし、このコマンドを実行したときに、以下の Sample Output に示す様にスペースで分離されたテーブル名を記載することで同じデータベースの複数のテーブルの行計算を取得できます。

サンプル出力:

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs
NDBT_ProgramExit: 0 - OK
```

ndb_show_tables

説明: クラスタのすべての **NDB** データベース オブジェクトのリストを表示します。デフォルトでは、これにはユーザー作成のテーブルおよび **NDB** システム テーブルの両方が含まむだけでなく、**NDB**-特定のインデックス、内部トリガ、およびクラスタ ディスク データ オブジェクトも同様に含まれます。

使用法:

```
ndb_show_tables [-c connect_string]
```

その他のオプション:

- `-l, --loops`

ユーティリティが実行すべき回数を指定します。このオプションが指定されない時はこれは 1 ですが、このオプションを使用する場合には、その整数の引数を提供する必要があります。

- `-p, --parsable`

このオプションを使用するとその出力を `LOAD DATA INFILE` の使用に適したフォーマットにします。

- `-t, --type`

以下に示す整数タイプのコードで指定された 1 種類のオブジェクトに出力を制限するために使用します。

- 1: System table
- 2: User-created table
- 3: Unique hash index

他の値はすべての NDB データベース オブジェクトをリスト (デフォルト) にします。

- `-u, --unqualified`

これを指定すると、許可されていないオブジェクト名を表示します。

注: ユーザー作成のクラスタ テーブルのみが MySQL からアクセスできます。SYSTAB_0 は `mysqld` には表示されません。しかし、`ndb.select.all` などの NDB API アプリケーションを使用してシステム テーブルのコンテンツを調べることができます (`ndb.select.all` 参照)。

ndb_size.pl — NDBCluster サイズ仕様エスティメーター

これはそれが NDBCluster ストレージ エンジンを使用するために変換される場合の MySQL のデータベースで必要なスペースの量を見積もるために使用される Perl のスクリプトです。この項で説明した他のユーティリティとは異なり、MySQL Cluster (実際のところ、接続する理由はないのだが) とのアクセスは必要ありません。しかしながら、データベースの常駐をテストする MySQL サーバーへのアクセスは必要です。

仕様:

- 稼働中の MySQL サーバーのインターフェースは MySQL Cluster にサポートを提供する必要はありません。
- Perl の稼働中のインストール。
- DBI および HTML::Template モジュール、この両方はそれらが Perl のインストールの一部で無い場合 CPAN から取得できます。(多くの Linux および他のオペレーティング システムの配布はこれらの 1 つあるいは両方のバイナリにそれ自身のパッケージを提供します。
- `ndb_size.tmpl` テンプレート ファイル、は MySQL のインストールの `share/mysql` ディレクトリに見つかるはずで、このファイルは `ndb_size.pl` と同じディレクトリにコピーあるいは移動する必要があります。もしそれがそこに既に無い場合— スクリプトを実行する前に
- 必要な権限を持つ MySQL ユーザーアカウント既存のアカウントの使用の希望されない場合、`GRANT USAGE ON db_name` を使用して新たに作成します。* — ここでは `db_name` は検査されるデータベース名で— この目的には十分です。

`ndb_size.pl` および `ndb_size.tmpl` は `storage/ndb/tools` の MySQL ソースにあります。これらのファイルが MySQL のインストールに見つからない場合、[MySQLForge project page](#) で入手できます。

使用法:

```
perl ndb_size.pl db_name hostname username password > file_name.html
```

表示されたコマンドは `password` を持つユーザー `username` で MySQL サーバーに `hostname` で接続し、データベース `db_name` のすべてのテーブルを分析し、ファイル `file_name.html` に送られるレポートを HTML フォーマットで生成します。(送り先を変えない場合、その出力は `stdout` に送られます。)数字はウェブ ブラウザで表示される部分的なサンプル出力です。

MySQL Cluster analysis for world

This is an automated analysis of the DBI:mysql:database=world;host=192.168.0.176 database for migration into [MySQL](#) Cluster. No warranty is made to the accuracy of the information.

This information should be valid for MySQL 4.1 and 5.0. Since 5.1 is not a final release yet, the numbers should be used as a guide only.

Parameter Settings

NOTE the configuration parameters below do not take into account system tables and other requirements.

Parameter	4.1	5.0	5.1
DataMemory (kb)	544	544	544
IndexMemory (kb)	192	136	136
MaxNoOfTables	3	3	3
MaxNoOfAttributes	24	24	24
MaxNoOfOrderedIndexes	3	3	3
MaxNoOfUniqueHashIndexes	3	3	3
MaxNoOfTriggers	12	12	12

Memory usage because of parameters

Usage is in kilobytes. Actual usage will vary as you should set the parameters larger than those listed in the table above.

Parameter	4.1	5.0	5.1
Attributes	5	5	5
Tables	60	60	60
OrderedIndexes	30	30	30
UniqueHashIndexes	45	45	45

Table List

- [City](#)
- [Country](#)
- [CountryLanguage](#)

City

Column	Type	Size	Key	4.1 NDB Size	5.0 NDB Size	5.1 NDB Size
ID	int	11	PRI	4	4	4
District	char	20		20	20	20
Name	char	35		36	36	36
Population	int	11		4	4	4
CountryCode	char	3		4	4	4

Indexes

We assume that indexes are ORDERED (not created USING HASH). If order is not required, 10 bytes of data memory can be saved per row if the index is created USING HASH

Index	Type	Columns	4.1 IdxMem	5.0 IdxMem	5.1 IdxMem	4.1 DatMem	5.0 DatMem	5.1 DatMem
PRIMARY	BTREE	ID	29	25	25	10	10	10

DataMemory Usage

	4.1	5.0	5.1
Row Overhead	16	16	16
Column DataMemory/Row	68	68	68
Index DataMemory/Row	10	10	10
Total DataMemory/Row	94	94	94
Rows per 32 kb page	347	347	347
Current number of rows	4079	4079	4079
Total DataMemory (kb)	384	384	384

IndexMemory Usage

このスクリプトの出力は以下を含みます。

- テーブルの分析に必要な `DataMemory`、`IndexMemory`、`MaxNoOfTables`、`MaxNoOfAttributes`、`MaxNoOfOrderedIndexes`、`MaxNoOfUniqueHashIndexes` および `MaxNoOfTriggers` の設定パラメータ。
- データベースで定義されたすべてのテーブル、属性、順序付けインデックス、および一意のハッシュ インデックスのメモリ要件。
- テーブルおよびテーブル行毎に必要な `IndexMemory` および `DataMemory`。

ndb_waiter

説明:すべてのクラスタ データ ノードのプリント アウトをクラスタが所定のステータスになるまであるいは `--timeout` 制限になりまで繰り返し (各 100 ミリ秒毎)、その後終了します。デフォルトでは、すべてのノードが起動しクラスタに接続する状態を示すクラスタが `STARTED` ステータスになるまで待ちます。これは `--no-contact` および `--not-started` オプション ([Additional Options \[961\]](#) 参照) を使用することによって書き換えられます。

このノードはこのユーティリティで以下レポートされます。

- `NO_CONTACT`:このノードは接続できません。
- `UNKNOWN`:このノードは接続できません。そのステータスはまだ未詳です。通常、この表示はノードが `START` あるいは `RESTART` コマンドをマネジメント サーバーから受信しても、まだ実行されない状態を示します。
- `NOT_STARTED`:このノードは停止したが、まだクラスタと接続している状態を示します。この表示はノードをマネジメント クライアントの `RESTART` コマンドを使用して再起動したときに表示されます。
- `STARTING`:ノードの `ndbd` プロセスが実行されたが、ノードがまだクラスタに接続していない状態を示します。
- `STARTED`:ノードは使用できるが、まだクラスタに接続していない状態です。
- `SHUTTING_DOWN`:ノードがシャットダウンされています。
- `SINGLE USER MODE`:これはクラスタが単一ユーザーモードの場合のすべてのクラスタ データ ノードの状態です。

使用法:

```
ndb_waiter [-c connect_string]
```

その他のオプション:

- `-n, --no-contact`
`STARTED` ステートを待つ代わりに、`ndb_waiter` はクラスタが `NO_CONTACT` ステータスになるまで継続し、終了すること示しています。
- `--not-started`
`STARTED` ステートを待つ代わりに、`ndb_waiter` がクラスタが `NOT_STARTED` ステータスになるまで継続して、終了すること示しています。
- `-t, --timeout=#`
待ち時間プログラムは指定した待ち時間 (秒) 内に実行されない場合終了します。デフォルトは 120 秒 (1200 レポート回数) です。

サンプル出力:

2 つのノードがシャットダウンし手動で起動した 4 つのノード クラスタを起動した際の出力 `ndb_waiter` を示したものです。レポートにコピー (「...」による) は削除されています。

```
shell> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
```

```

State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED
...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED
...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED
...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED
NDBT_ProgramExit: 0 - OK

```

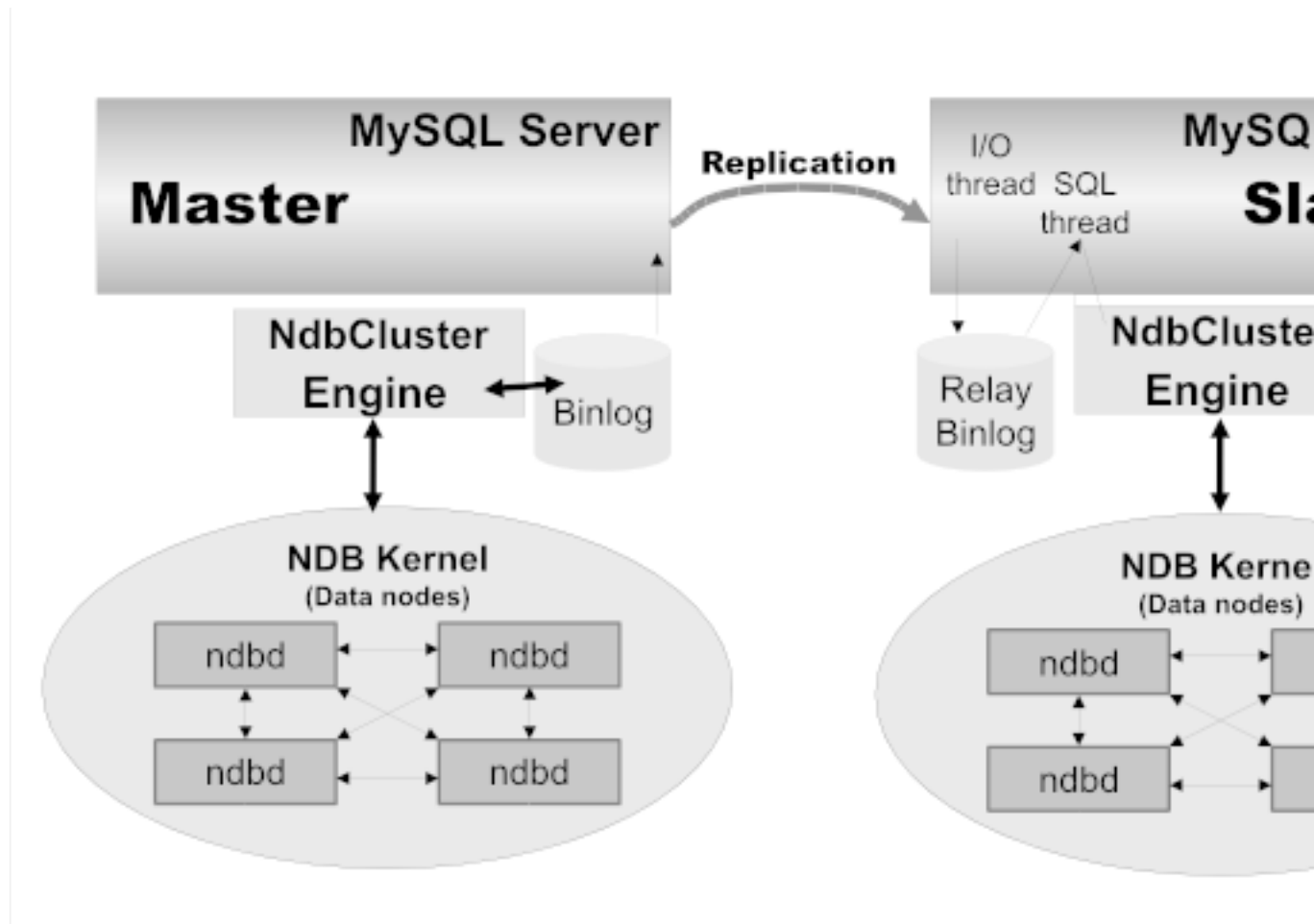
Note: 接続文字列が指定されていない場合、`ndb_waiter` がマネジメントに `localhost` で接続を試み、`Connecting to mgmsrv at (null)` をレポートします。

MySQL Cluster レプリケーション

MySQL 5.1.6 以前のバージョンは、asynchronous replication、単に「replication」とよく呼ばれているものは、MySQL Cluster を使用した場合は利用できませんでした。MySQL 5.1.6 では MySQL Cluster のデータベースにマスタスレーブのレプリケーションを導入しています。この項では設定方法および設定の管理について説明します。そこでは MySQL Cluster として運用されている 1 つのグループのコンピュータを 2 番目コンピュータあるいはグループのコンピュータにレプリケートします。標準の MySQL レプリケーションに関しては読者の中にはこのマニュアルのどこかで説明した内容をご存知の方もおられると思います。([5章レプリケーション](#) 参照。)

通常 (非クラスタ) のレプリケーションは「master」サーバーおよび「slave」を、マスタは運用のソースとして、データはレプリケートされるものとして、スレーブはこれらの受皿として含まれています。MySQL Cluster では、レプリケーションは概念的には非常に類似していますが、2 つの完全なクラスタ間のレプリケーションを含む多くの異なる設定をカバーするために拡張できるなど実際の運用においてはさらに複雑にできます。MySQL Cluster そのものは [NDB Cluster](#) ストレージ エンジンにそのクラスタ機能を依存していますが、スレーブのクラスタ ストレージ エン

ジンを使用する必要はありません。しかしながら、最大の可用性を引き出すために、1つのMySQL Clusterから別へレプリケートでき、これが以下の図に示すここで説明する設定タイプです。



このシナリオでは、レプリケーション プロセスはマスタ クラスタの連続的なステートがログされスレーブのクラスタに保存されます。このプロセスは NDB binlog インジェクタースレッドとして知られる特別なスレッドによって実現され、それは書く MySQL サーバー上で動作しバイナリのログ (**binlog**) を生成します。このスレッドはクラスタのすべての変更がバイナリのログを生成し、MySQL サーバーによって影響を受けたそれらの変更だけではなく正確なシリアル番号にログに挿入されることを確認します。ここでは MySQL レプリケーション マスタおよびレプリケーション サーバーとしてのレプリケーション スレーブあるいはレプリケーション ノード、およびそのデータ フローあるいは replication channel としてのそれらの間の通信ラインについて言及します。

略語と記号

この項を通じて、マスタおよびスレーブのクラスタ、クラスタあるいはクラスタ ノードで実行されるプロセス並びにコマンドに関する以下の略語および記号を使用します。

記号あるいは略語	の説明 (参照...)
M	(プライマリ) レプリケーション マスタとしてのクラスタ
S	(プライマリ) レプリケーション スレーブとして機能するクラスタ
shellM>	マスタ クラスタで発行されるシェル コマンド
mysqlM>	マスタ クラスタの SQL ノードとして動作する単一の MySQL サーバーで発行された MySQL クライアント コマンド
mysqlM*>	レプリケーション マスタに使用されているすべての SQL ノードで発行される MySQL クライアント コマンド

shellS>	スレーブ クラスタで発行される シェル コマンド
mysqlS>	スレーブ クラスタで SQL ノードとして実行される単一の MySQL サーバーで発行される MySQL コマンド
mysqlS*>	MySQL client command to be issued on all SQL nodes participating in the replication slave cluster
C	Primary replication channel
C'	Secondary replication channel
M'	Secondary replication master
S'	Secondary replication slave

仮定条件と一般要件

レプリケーション チャンネルには 2 台の MySQL サーバーをレプリケーション サーバー (マスタおよびスレーブに各 1 台ずつ) が必要です。例えば、2 つのレプリケーション チャンネル (冗長性を確保するために予備のチャンネルを提供する) でレプリケーションの設定をする場合、それぞれのクラスタ各 2 つのレプリケーション ノードがあるため合計で 4 つのレプリケーション ノードになることを意味します。

この項で説明する MySQL Cluster のレプリケーションおよびそれに続くものは行ベースのレプリケーションに依存しています。このことはレプリケーションのマスタ MySQL サーバーは [レプリケーションの開始 \(シングル レプリケーション チャンネル\)](#) の説明のように `--binlog-format=row` を一緒に起動する必要があります。行ベースのレプリケーションの一般的な情報は、[レプリケーション フォーマット](#) を参照してください。

(MySQL Cluster をステートメントをベースとしたレプリケーションでレプリケートすることができます。しかし、この場合、以下の制限が適用されます。マスタとの役割を担うクラスタのデータ行のすべての更新は単一の MySQL サーバーに割り当てる必要があります。いくつかの MySQL レプリケーション プロセスを使用して同時にクラスタをレプリケートすることはできません。SQL レベルでの変更のみレプリケートされます。)

クラスタのいずれかのレプリケーションに使用される MySQL サーバーは、いずれかのクラスタ に使用しているすべての MySQL レプリケーションサーバー間で一意で認識される必要があります (同じ ID を共有するマスタおよびスレーブのクラスタの両方にレプリケーション サーバーを持つことはできません)。これは各 SQL ノードを `--server-id=id` オプションを使用して起動することで可能です。そこでは `id` は固有の整数です。必ずしも必要ではありませんが、ここここでは説明のために、すべての MySQL のインストールは同じバージョンであることを想定しています。

いずれの場合でも、レプリケーションに使用する両方の MySQL サーバーは使用するレプリケーション プロトコルおよびそれらがサポートする SQL 機能セットに於いてお互い互換性がある必要があります。この場合にこれを確認する最も簡素で容易な方法は同じ MySQL バージョンを使用するすべてのサーバーに使用することです。多くの場合マスタのバージョンより古いバージョンを使用した MySQL バージョンで動作するスレーブでレプリケートすることはできません—詳細は [MySQL バージョン間のレプリケーション互換性](#) を参照してください。

スレーブ サーバーあるいはクラスタはマスタのレプリケーション専用であり、他のデータはそれに保存されないものと想定しています。

既知の問題

以下はレプリケーションを MySQL 5.1 の MySQL Cluster で行う場合の既知の問題あるいは懸案事項です。

- MySQL Cluster のレプリケーション スレーブ `mysqld` はマスタの接続が切断され、ログがバッファされないことを検知する方法がありません。このため、マスタが使用できなくなったりあるいはネットワークの問題が発生した場合、スレーブがマスタに対して一貫性が無くなる場合があります。

この問題を避けるために、複数のレプリケーション ラインを設け、プライマリのレプリケーション ラインでマスタ `mysqld` を監視し、必要に応じてフェールオーバーを 2 次側のラインに設定します。

この種の設定を行うための情報に関しては、[2 つのレプリケーション チャンネルを使用する](#)、および [MySQL Cluster にフェールオーバーを導入する](#) を参照してください。

弊社では現在この問題の解決策を今後の MySQL Cluster のリリースも含めて検討しています。(この問題に関する説明は Bug #21494 にあります。)

- 循環的なレプリケーションはクラスタのレプリケーションではサポートしていません。これは特定の MySQL Cluster で作成されたすべてのログ イベントの マスタとして使用されている MySQL サーバーサーバーID のタグが間違っており、元のサーバーのサーバーID ではないからです。

その ID の間違いにより、MySQL サーバー A→B→A、そこでは B はクラスタ A に接続された MySQL サーバーで、クラスタテーブル A からの変更 (ログ エントリ) により元のサーバーの識別名を B から A で失う 「lose」 からです。これによりその変更はサーバー A に再び適用されます。

- [CREATE TABLE](#)、[DROP TABLE](#)、および [ALTER TABLE](#) などのデータ定義ステートメントの使用はバイナリのログにそれが発行された MySQL サーバーにのみ記録されます。
- MySQL 5.1.6 では、明示のプライマリ キーを持つ [NDB](#) のみがレプリケートされます。この制限は MySQL 5.1.7 が解除されています。
- クラスタを `--initial` オプションで再起動すると GCI およびエポック番号が 0 から始まります。(これは一般的には MySQL Cluster では当たり前でクラスタを使用したレプリケーションのシナリオに限ったことではありません。レプリケーションに使用された MySQL サーバーはの場合レプリケートされます。この後、[RESET MASTER](#) および [RESET SLAVE](#) ステートメントを使用して [ndb_binlog_index](#) および [ndb_apply_status](#) テーブルをそれぞれクリアします。
- [auto_increment_offset](#) および [auto_increment_increment](#) サーバースystem に変数を設定しようすると予測できない結果がでます。これらの変数の使用は MySQL Cluster のレプリケーションではサポートしていません。

レプリケーション スキーマおよびテーブル

MySQL Cluster でのレプリケーションではレプリケートされるクラスタおよびレプリケーション スレーブ (単一のサーバーあるいはクラスタのいずれか) としての役割を果たす各 MySQL Server インスタンスの `mysql` データベースの多くの専用のテーブルを使用しています。これらのテーブルは `mysql_install_db` スクリプトによって MySQL のインストール中に作成され、バイナリ ログのインデックス データを保存するテーブルを含んでいます。[ndb_binlog_index](#) テーブルは各 MySQL サーバーに対してはローカルで、クラスタ化には使用されず、[MyISAM](#) ストレージ エンジンを使用します。このことはそれはマスタのクラスタに使用される各 `mysqld` で個別に作成されることを意味しています。(しかし、binlog そのものはレプリケートされるクラスタの MySQL サーバーの更新を含んでいます。) このテーブルは以下のように定義されます。

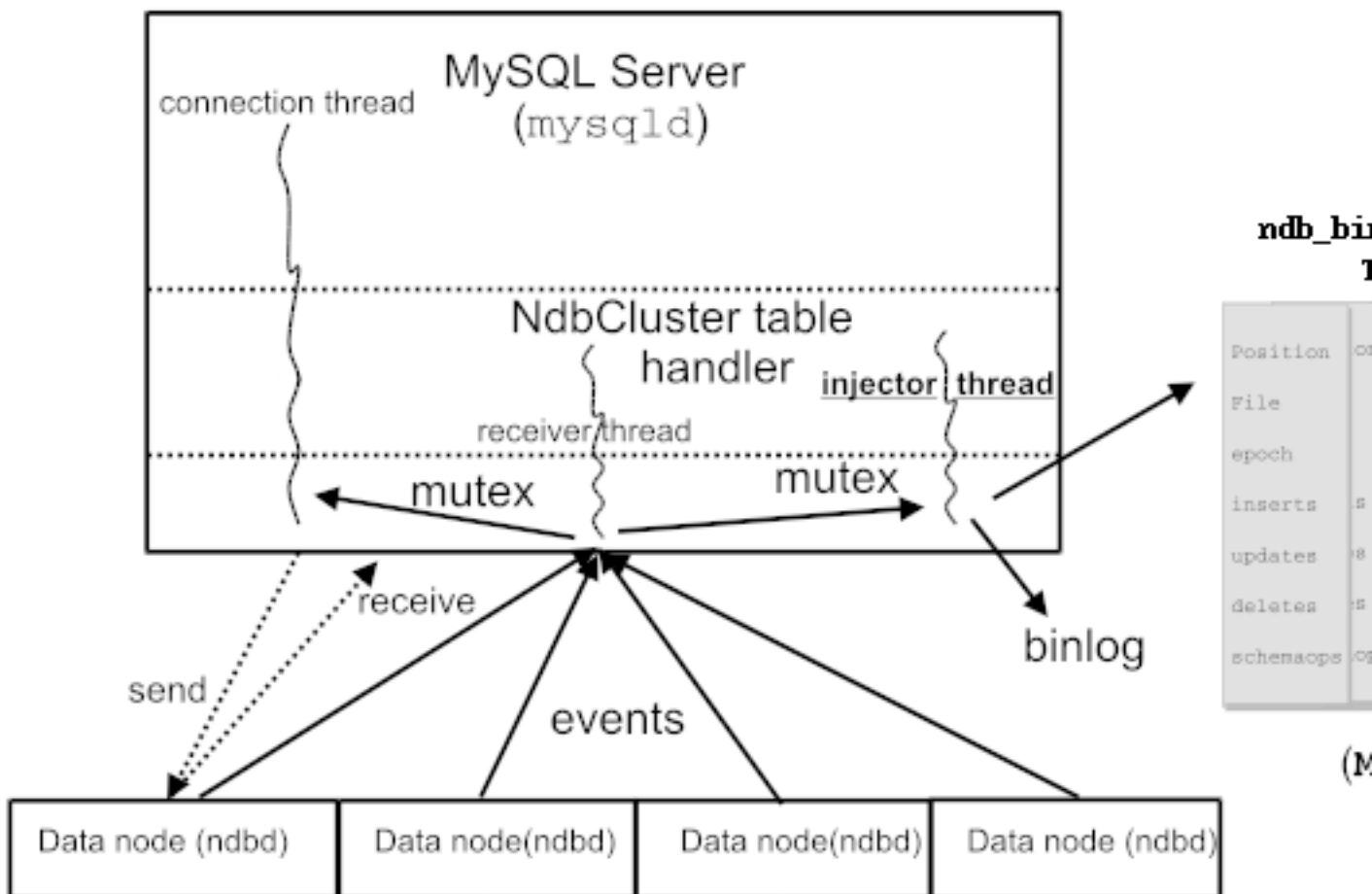
```
CREATE TABLE `ndb_binlog_index` (
  `Position` BIGINT(20) UNSIGNED NOT NULL,
  `File` VARCHAR(255) NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `inserts` BIGINT(20) UNSIGNED NOT NULL,
  `updates` BIGINT(20) UNSIGNED NOT NULL,
  `deletes` BIGINT(20) UNSIGNED NOT NULL,
  `schemaops` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY (`epoch`)
) ENGINE=MYISAM DEFAULT CHARSET=latin1;
```

重要MySQL 5.1.14 以前のバージョンでは、[ndb_binlog_index](#) テーブルは `binlog_index` として知られ個別 `cluster` データベースで維持され、それは MySQL 5.1.7 あるいはそれ以前のバージョンでは `cluster_replication` データベースとして知られていました。同様に、[ndb_apply_status](#) および [ndb_schema](#) テーブルは `apply_status` および `schema` として知られており、`cluster` (以前は `cluster_replication`) データベースにありました。しかし、MySQL 5.1.14 以降では、すべての MySQL Cluster のレプリケーション テーブルは `mysql` システム データベースにあります。

この変更が MySQL Cluster 5.1.13 および 5.1.14 およびそれ以降のバージョンのアップグレードに影響を及ぼすかは [Changes in release 5.1.14 \(05 December 2006\)](#) を参照してください。

以下の図は MySQL Cluster のレプリケーション マスタ サーバー、その binlog インジェクタースレッド、および `mysql.ndb_binlog_index` テーブルのレプリケーションを表しています。

MySQL Replication Between Clusters, Injecting into



`ndb_apply_status` の名前の新たなテーブルがマスタからスレーブにレプリケートされたオペレーションの記録を取るために使用されます。`ndb_binlog_index` の場合と違って、このテーブルのデータはクラスタのどの SQL ノード独自のものではないため、`ndb_apply_status` は以下に示すように **NDB Cluster** ストレージ エンジンを使用できます。

```
CREATE TABLE `ndb_apply_status` (
  `server_id` INT(10) UNSIGNED NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY USING HASH (`server_id`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```

`ndb_binlog_index` および `ndb_apply_status` テーブルはそれらがレプリケートされる必要が無いため **mysql** データベースで作成されます。それらのいずれの作成および維持に関してユーザーが関与することは通常ありません。`ndb_binlog_index` および `ndb_apply_status` テーブルの双方は **NDB** インジェクタースレッドで維持されます。これが **NDB** ストレージ エンジンで実行される変更に対する更新されたマスタ **mysqld** プロセスを維持します。**NDB** binlog インジェクター スレッドは **NDB** ストレージ エンジンから直接イベントを受け取ります。**NDB** インジェクターはクラスタのすべてのデータ イベントの取得を行い、データの変更、挿入、あるいは削除するすべてのイベントが

`ndb_binlog_index` テーブルに記録されたことを確認します。スレーブ I/O スレッドはイベントをマスタのバイナリ ログからスレーブのリレーログに転送します。

しかし、これらのテーブルの存在と品質をレプリケーションに MySQL Cluster を準備する初期のステップで確認することをお勧めします。マスタで直接 `mysql.binlog_index` テーブルをクエリするとバイナリ ログに記録されたイベント データを表示できます。これをレプリケーション マスタあるいはスレーブ MySQL サーバーのいずれかの `SHOW BINLOG EVENTS` ステートメントを使用して表示することもできます。

注:MySQL 5.1.14 以降では、`apply_status` テーブルがスレーブに無い場合、`ndb_restore` で作成できます。(Bug #14612)

レプリケーションにクラスタを準備する

MySQL Cluster のレプリケーションに準備をするには以下のステップを踏みます。

1. すべての MySQL サーバーのバージョンの互換性をチェックします (仮定条件と一般要件 参照)。
2. マスタ クラスタに権限付きのスレーブのアカウントを作成します。

```
mysqlM> GRANT REPLICATION SLAVE
-> ON *.* TO 'slave_user'@'slave_host'
-> IDENTIFIED BY 'slave_password';
```

`slave_user` がスレーブ アカウントのユーザー名の場合、`slave_host` がレプリケーション スレーブのホスト名あるいは IP アドレスで、`slave_password` はこのアカウントを割り当てるパスワードです。

例えば、スレーブ ユーザーアカウントを「`myslave` の名前で作成する場合、」ホスト名「`rep-slave` でログインし、」パスワード「`53cr37` を使用して、」以下の `GRANT` ステートメントを使用します。

```
mysqlM> GRANT REPLICATION SLAVE
-> ON *.* TO 'myslave'@'rep-slave'
-> IDENTIFIED BY '53cr37';
```

セキュリティのためには、一意のユーザーアカウント—他の目的に使用されていない—をレプリケーション スレーブ アカウントに使用するようお勧めします。

3. マスタを使用出来るようにスレーブを設定します。MySQL Monitor を使用すると、これを `CHANGE MASTER TO` ステートメントで実行できます。

```
mysqlS> CHANGE MASTER TO
-> MASTER_HOST='master_host',
-> MASTER_PORT=master_port,
-> MASTER_USER='slave_user',
-> MASTER_PASSWORD='slave_password';
```

`master_host` がレプリケーション マスタのホスト名あるいは IP アドレスの場合、`master_port` はマスタの接続に使用されるスレーブのポートで、`slave_user` はマスタのスレーブの設定したユーザー名で、`slave_password` は前のステップのユーザーアカウントに設定されたパスワードです。

例えば、スレーブにホスト名が「`rep-master` の MySQL サーバーからレプリケートするよう指示した場合、」前のステップで作成したレプリケーションのスレーブ アカウントを使用して、以下のステートメントを使用します。

```
mysqlS> CHANGE MASTER TO
-> MASTER_HOST='rep-master'
-> MASTER_PORT=3306,
-> MASTER_USER='myslave'
-> MASTER_PASSWORD='53cr37';
```

(このステートメントで使用する節の完全なリストは、[CHANGE MASTER TO 構文](#) を参照してください。)

スレーブのサーバー `my.cnf` ファイルの相当する設定オプションを設定してマスタを使用するスレーブを設定することもできます。前述の例 `CHANGE MASTER TO` ステートメントと同じようにスレーブを設定するには、スレーブの `my.cnf` ファイルに以下の情報を含める必要があります。

```
[mysqld]
```

```
master-host=rep-master
master-port=3306
master-user=myslave
master-password=53cr37
```

レプリケーション スレーブに `my.cnf` で設定できるその他のオプションについては、[レプリケーションのオプションと変数](#) を参照してください。

注:レプリケーションのバックアップ機能を検証するには、レプリケーションのプロセスを始める前に `ndb-connectstring` オプションのスレーブの `my.cnf` ファイルに追加する必要があります。詳細は [MySQL Cluster のレプリケーションによるバックアップ](#) を参照してください。

4. マスタ クラスタが既に使用されている場合、マスタのバックアップを作成しそれをスレーブにアップロードしてスレーブがマスタに同期する時間を節約することができます。スレーブが MySQL Cluster で稼働している場合には、バックアップを使用して [MySQL Cluster のレプリケーションによるバックアップ](#) で説明したようにその手順を保存することでこれを実現できます。

```
ndb-connectstring=management_host[:port]
```

レプリケーション スレーブで MySQL Cluster を使用していない 場合、レプリケーション マスタのこのコマンドを使用してバックアップを作成できます。

```
shellM> mysqldump --master-data=1
```

ダンプ ファイルをスレーブにコピーしてその結果のデータをスレーブにインポートします。この後で、ここに示すように `mysql` クライアントを使用してデータをダンプファイルからスレーブのデータベースにインポートし、そこでは `dump_file` マスタの `mysqldump` を使用して生成したファイルの名前で、`db_name` はレプリケートされるデータベースの名前です。

```
shellS> mysql -u root -p db_name < dump_file
```

`mysqldump` で使用する完全なオプションのリストは、[mysqldump — データベースバックアッププログラム](#) を参照してください。

このようにデータをスレーブにコピーするには、スレーブがコマンドラインで `--skip-slave-start` オプションで起動していることを確認するかあるいは `skip-slave-start` をスレーブの `my.cnf` ファイルに入れてマスタへの接続させないようにしてすべてのデータがロードされる前にレプリケーションを始めます。データのローディングが完了したら、次の 2 項で説明するステップに従います。

5. レプリケーション マスタとしての各 MySQL サーバーが一意的サーバー ID で設定されているか、バイナリのロギングが有効になっているか、行フォーマットを使用しているか確認します。([レプリケーション フォーマット](#) 参照。)これらのオプションはマスタの `mysql` プロセスを実行する際にマスタのサーバー `my.cnf` ファイル、あるいはコマンドラインのいずれかに設定できます。後者のオプションのに関する情報については、[レプリケーションの開始 \(シングル レプリケーション チャンネル\)](#) を参照してください。

レプリケーションの開始 (シングル レプリケーション チャンネル)

この項ではシングルのレプリケーション チャンネルを使用した MySQL Cluster のレプリケーションに手順に付いて説明します。

1. このコマンドを発行して MySQL レプリケーション マスタ サーバーを起動します。

```
shellM> mysqld --ndbcluster --server-id=id \
--log-bin --binlog-format=row &
```

ここでは `id` はこのサーバーの一意的 ID です ([仮定条件と一般要件](#) 参照)。これによりサーバーの `mysqld` プロセスが適切なロギング フォーマットを使用してバイナリのロギングを有効にして実行されます。

2. 以下の示すように MySQL レプリケーション スレーブ サーバーを起動します。

```
shellS> mysqld --ndbcluster --server-id=id &
```

ここでは `id` はスレーブ サーバーの一意的 ID です。レプリケーション スレーブでロギングを有効にする必要はありません。

このコマンドで `--skip-slave-start` オプションを使用するか、あるいはレプリケーションを直ぐ始めることを望まないかぎり `skip-slave-start` をサーバーの `my.cnf` ファイルに入れる必要があります。このオプションを使用すると、レプリケーションの開始が以下のステップ 4 で説明したように適切な `START SLAVE` ステートメントが発行されるまで遅延します。

- スレーブ サーバーをマスタ サーバーのレプリケーション binlog に同期する必要があります。バイナリのロギングがマスタで事前に実行されない場合、以下のステートメントをスレーブで実行します。

```
mysqlS> CHANGE MASTER TO
-> MASTER_LOG_FILE=",
-> MASTER_LOG_POS=4;
```

これによりスレーブがマスタのバイナリ ログをログ 開始ポイントから読み込みを開始します。さもなければ — つまり、バックアップを使用してマスタからデータをローディングするには — そのような場合の `MASTER_LOG_FILE` および `MASTER_LOG_POS` に使用する正しい値の取得に関する情報は [MySQL Cluster にフェールオーバーを導入する](#) を参照してください。

- 最後に、レプリケーション スレーブの `mysql` クライアントからコマンドを発行してレプリケーションを適用するようにスレーブに指示する必要があります。

```
mysqlS> START SLAVE;
```

これによってレプリケーションのデータのスレーブへの転送を始めます。

また次項で説明するような手順で 2 つのレプリケーション チャンネルを使用することもできます。これとシングルのレプリケーション チャンネルを使用する場合の違いは [2 つのレプリケーション チャンネルを使用する](#) で説明しています。

2 つのレプリケーション チャンネルを使用する

さらに完全な例のシナリオでは、2 つのレプリケーション チャンネルを使用することで冗長性を提供することによって、1 つのレプリケーション チャンネルで考えられる問題を回避できるものと考えています。これには合計 4 台のレプリケーション サーバーが必要で、2 台のマスタ サーバーをマスタ クラスタに、後の 2 台のスレーブ サーバーをスレーブ クラスタに使用します。これからの説明のために、一意の識別子を以下のように割り当てたものとします。

サーバー ID	説明
1	マスタ - プライマリ レプリケーション チャンネル (M)
2	マスタ - 二次レプリケーション チャンネル (M')
3	スレーブ - プライマリ レプリケーション チャンネル (S)
4	スレーブ - 二次レプリケーション チャンネル (S')

2 つのチャンネルでの設定は 1 つのチャンネルのレプリケーション チャンネルの設定とそれ程大きく異なりません。最初にプライマリと二次レプリケーション マスタの `mysqld` プロセスは、プライマリおよび二次スレーブの実行に続いて実行します。次に、各スレーブで `START SLAVE` ステートメントを発行してレプリケーションを始めます。コマンドおよびその発行順序を以下に示します。

- プライマリのレプリケーション マスタを起動します。

```
shellM> mysqld --ndbcluster --server-id=1 \
--log-bin --binlog-format=row &
```

- 二次レプリケーション マスタを起動します。

```
shellM> mysqld --ndbcluster --server-id=2 \
--log-bin --binlog-format=row &
```

- プライマリのレプリケーション スレーブ サーバーを起動します。

```
shellS> mysqld --ndbcluster --server-id=3 \
--skip-slave-start &
```

- 二次レプリケーション スレーブを起動します。

```
shellS'> mysqld --ndbcluster --server-id=4 \
--skip-slave-start &
```

- 最後に、以下に示すようにプライマリ スレーブの **START SLAVE** ステートメントを実行してプライマリ チャネルでレプリケーションを開始します。

```
mysqlS'> START SLAVE;
```

前述同様、バイナリのロギングをレプリケーション スレーブで有効にする必要はありません。

MySQL Cluster にフェールオーバーを導入する

プライマリのクラスタ レプリケーション プロセスが失敗した場合、二次レプリケーション チャネルに切り替えることができます。この切り替えを行うために必要なステップの手順を以下説明します。

- 最も最新のグローバル チェックポイント (GCP) の時間の取得。つまり、最も最新のエポックをスレーブ クラスタの **ndb_apply_status** テーブルで決める必要があります。それ以下のクエリで検索できます。

```
mysqlS'> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status;
```

- ステップ 1 で説明したクエリから取得した情報を使用して、マスタ クラスタの **ndb_binlog_index** テーブルから以下のように相当するレコードを取得します。

```
mysqlM'> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch > @latest
-> ORDER BY epoch ASC LIMIT 1;
```

これらはプライマリ レプリケーション チャネルの失敗以来のマスタに保存されたレコードです。ここではステップ 1 で取得した値を表すユーザー変数 **@latest** を使用しています。勿論、1 つの **mysqld** インスタンスで別のサーバーのインスタンスに設定されたユーザー変数に直接アクセスすることはできません。これらの値は手動で 2 番目のクエリあるいはアプリケーション コードに「plugged in」する必要があります。

- ここで二次のスレーブ サーバーに以下のクエリを実行して二次のチャネルを同期できます。

```
mysqlS'> CHANGE MASTER TO
-> MASTER_LOG_FILE=@file,
-> MASTER_LOG_POS=@pos;
```

再度ユーザー変数 (この場合は **@file** および **@pos**) を使用してステップ 2 で取得しステップ 3 で適用した値を表します。実際はこれらの値は手動で挿入するか、あるいは使用している両方のサーバーにアクセスするアプリケーション コードを使用します。

@file は **'/var/log/mysql/replication-master-bin.00001'** などの文字列の値で、SQL あるいはアプリケーション コードで使用されるときに引用される必要があります。しかし、**@pos** で表される値は引用する必要はありません。MySQL は通常文字列を番号を変換しようとしますが、この場合は例外です。

- 今二次スレーブ **mysqld** の適切なコマンドを発行して二次チャネルでレプリケーションを開始できます。

```
mysqlS'> START SLAVE;
```

二次のレプリケーション チャネルが利用できるようになったら、プライマリの不具合を調べ問題を解決します。問題の解決にはプライマリ チャネルの問題を正しく見極めた正確な対応が必要です。

その問題が 1 台のサーバーに限定されるのであれば、その不具合は (論理的には) **M** から **S'**、あるいは **M'** から **S** にレプリケートできます。しかし、この件はまだ検証していません。

MySQL Cluster のレプリケーションによるバックアップ

この項ではバックアップの作成とそのバックアップの MySQL Cluster レプリケーションを使用した保存について説明します。レプリケーション サーバーが以前に説明 ([レプリケーションにクラスタを準備する](#) およびその直後の項の説

明を参照してください)した設定になっていることが前提です。その設定になっている場合、バックアップの作成およびそのバックアップの保存手順は以下のようになります。

1. バックアップを開始する 2 つの異なる方法があります。

- 方法 A:

この方法ではレプリケーションのプロセスを開始する前にクラスタのバックアップのプロセスがマスタ サーバーで有効になっている必要があります。これは以下の行を

```
ndb-connectstring=management_host[:port]
```

を `my.cnf` file の `[MYSQL_CLUSTER]` の項に含めること可能で、そこでは `management_host` はマスタ クラスタの NDB サーバーの IP アドレスあるいはホスト名で、`port` はマネジメント サーバーのポート番号です。ポート番号はデフォルトのポート (1186) が使用されていない場合にのみ指定する必要があります。(MySQL Cluster のポートおよびポートの割り当てに関する情報は、[マルチコンピュータの設定](#) を参照してください。)

この場合、そのバックアップはこのステートメントをレプリケーション マスタで実行することで開始されます。

```
shellM> ndb_mgm -e "START BACKUP"
```

- 方法 B:

`my.cnf` ファイルがマネジメント ホストの場所を指定していない場合、バックアップ プロセスはこの情報を NDB マネジメント クライアントに `START BACKUP` コマンドの一部として渡すことで開始できます。以下のようになります。

```
shellM> ndb_mgm management_host:port -e "START BACKUP"
```

ここでは `management_host` および `port` はマネジメント サーバーのホスト名およびポート番号です。前に述べたようなシナリオ ([レプリケーションにクラスタを準備する](#) 参照) で、以下のように実行できます。

```
shellM> ndb_mgm rep-master:1186 -e "START BACKUP"
```

どの方法の場合でも、未処理のトランザクションがある場合にはそれをバックアップを開始する前に完了し、バックアップの実施中に新たにトランザクションを実施しないようお願いします。

2. クラスタおバックアップファイルを行に入れるスレーブにコピーします。マスタ クラスタの `ndbd` プロセスで稼働している各システムはそのシステム上にクラスタのバックアップファイルを持ち、これらの all のファイルは保存の確認をするためにスレーブにコピーされます。バックアップ ファイルは、MySQL および NDB バイナリがそのディレクトリの許可を読む限りスレーブ マネジメント ホストが常駐するコンピュータのどのディレクトリにコピーできます。このように、これらのファイルはディレクトリ `/var/BACKUPS/BACKUP-1` にコピーできるものと想定されます。

スレーブ クラスタが `ndbd` プロセス (データ ノード) とマスタとして同じ番号を持つ必要ありませんが、この番号が同じにするように強くお勧めします。レプリケーション プロセスが準備不足で起動しないように、スレーブを `--skip-slave-start` オプションで起動することが必要です。

3. データベースをスレーブのレプリケートされるマスタ クラスタのスレーブ クラスタで作成します。重要レプリケートされる各データベースに相当する `CREATE SCHEMA` ステートメントをスレーブ クラスタの各データ ノードで実行します。
4. MySQL Monitor のこのステートメントを使用してスレーブのクラスタをリセットします。

```
mysqlS> RESET SLAVE;
```

スレーブの `apply_status` テーブルが保存プロセスを実行する前にレコードを含んでいないことが重要です。スレーブの SQL ステートメントを実行することでこれを実現できます。

```
mysqlS> DELETE FROM mysql.ndb_apply_status;
```

5. ここで各バックアップ ファイルに対して順番に `ndb_restore` コマンドを使用してレプリケーション プロセスを開始できます。これらを実行する前に、クラスタのメタデータを保存するには `-m` オプションを含めることが必要です。

```
shellS> ndb_restore -c slave_host:port -n node-id \
-b backup-id -m -r dir
```

`dir` はバックアップ ファイルがレプリケーション スレーブの置かれるディレクトリへのパスです。残りのバックアップ ファイルに相当する `ndb_restore` コマンドに対しては、`-m` オプションは使用しないでください。

マスタ クラスタからバックアップ ファイルがディレクトリ `/var/BACKUPS/BACKUP-1` にコピーされる 4 つのデータ ノード (MySQL Cluster レプリケーション の図を参照) に保存するには、スレーブで実行されるコマンドのシーケンスは以下のようになります。

```
shellS> ndb_restore -c rep-slave:1186 -n 2 -b 1 -m \
-r ./VAR/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 3 -b 1 \
-r ./VAR/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 4 -b 1 \
-r ./VAR/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 5 -b 1 -e \
-r ./VAR/BACKUPS/BACKUP-1
```

このコマンドのシーケンスにより最も最新のエポック レコードをスレーブの `ndb_apply_status` テーブルに書き込みます。

- ここで最も最新のエポックをスレーブの `ndb_binlog_index` テーブルから取得する必要があります (MySQL Cluster にフェールオーバーを導入する の説明参照):

```
mysqlS> SELECT @latest:=MAX(epoch)
FROM mysql.ndb_binlog_index;
```

- `@latest` を前のステップで取得したエポック値として使用して、以下のクエリを使用してマスタの `mysql.ndb_binlog_index` テーブルから正しいバイナリ ログ ファイル `@file` の正しい起動位置 `@pos` を取得できます。

```
mysqlM> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch > @latest
-> ORDER BY epoch ASC LIMIT 1;
```

- 前のステップで取得した値を使用して、スレーブの `mysql` クライアントの適切な `CHANGE MASTER TO` ステートメントを発行できます。

```
mysqlS> CHANGE MASTER TO
-> MASTER_LOG_FILE='@file',
-> MASTER_LOG_POS=@pos;
```

- スレーブが `binlog` ファイルがマスタからデータを読み込むどのポイントかを「知って」 いるので、この標準の MySQL ステートメントでスレーブにレプリケーションを開始させることができます。

```
mysqlS> START SLAVE;
```

バックアップを実行して二次のレプリケーション チャンネルで保存するには、これらのステップを繰り返し、適切と思われる場合二次マスタおよびスレーブのホスト名および ID をプライマリのマスタおよびスレーブレプリケーションに置き換えて、前のステートメントをそれらの上で実行するのみで可能です。

クラスタのバックアップおよびバックアップからのクラスタの保存に関する詳細は、MySQL Cluster のオンラインバックアップを参照してください。

スレーブのマスタ binlog への自動同期

前項で説明した多くのプロセスを自動化できます (MySQL Cluster のレプリケーションによるバックアップ 参照)。以下の Perl スクリプト `reset-slave.pl` はこの方法に関する例を提供します。

```
#!/user/bin/perl -w

# file: reset-slave.pl
```

```

# Copyright ©2005 MySQL AB

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to:
# Free Software Foundation, Inc.
# 59 Temple Place, Suite 330
# Boston, MA 02111-1307 USA
#
# Version 1.1

##### Includes #####

use DBI;

##### Globals #####

my $m_host="";
my $m_port="";
my $m_user="";
my $m_pass="";
my $s_host="";
my $s_port="";
my $s_user="";
my $s_pass="";
my $dbhM="";
my $dbhS="";

##### Sub Prototypes #####

sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetSlaveEpoch;
sub GetMasterInfo;
sub UpdateSlave;

##### Program Main #####

CollectCommandPromptInfo;
ConnectToDatabases;
GetSlaveEpoch;
GetMasterInfo;
UpdateSlave;
DisconnectFromDatabases;

##### Collect Command Prompt Info #####

sub CollectCommandPromptInfo
{
    ### Check that user has supplied correct number of command line args
    die "Usage:\n
        reset-slave >master MySQL host< >master MySQL port< \n
            >master user< >master pass< >slave MySQL host< \n
            >slave MySQL port< >slave user< >slave pass< \n
        All 8 arguments must be passed. Use BLANK for NULL passwords\n"
        unless @ARGV == 8;

    $m_host = $ARGV[0];
    $m_port = $ARGV[1];
    $m_user = $ARGV[2];
    $m_pass = $ARGV[3];
    $s_host = $ARGV[4];
    $s_port = $ARGV[5];
    $s_user = $ARGV[6];

```

```

$s_pass = $ARGV[7];

if ($m_pass eq "BLANK") { $m_pass = "";}
if ($s_pass eq "BLANK") { $s_pass = "";}
}

##### Make connections to both databases #####

sub ConnectToDatabases
{
    ### Connect to both master and slave cluster databases

    ### Connect to master
    $dbhM
    = DBI->connect(
        "dbi:mysql:database=mysql:host=$m_host;port=$m_port",
        "$m_user", "$m_pass")
    or die "Can't connect to Master Cluster MySQL process!
        Error: $DBI::errstr\n";

    ### Connect to slave
    $dbhS
    = DBI->connect(
        "dbi:mysql:database=mysql:host=$s_host",
        "$s_user", "$s_pass")
    or die "Can't connect to Slave Cluster MySQL process!
        Error: $DBI::errstr\n";
}

##### Disconnect from both databases #####

sub DisconnectFromDatabases
{
    ### Disconnect from master

    $dbhM->disconnect
    or warn " Disconnection failed: $DBI::errstr\n";

    ### Disconnect from slave

    $dbhS->disconnect
    or warn " Disconnection failed: $DBI::errstr\n";
}

##### Find the last good GCI #####

sub GetSlaveEpoch
{
    $sth = $dbhS->prepare("SELECT MAX(epoch)
        FROM mysql.ndb_apply_status;")
    or die "Error while preparing to select epoch from slave: ",
        $dbhS->errstr;

    $sth->execute
    or die "Selecting epoch from slave error: ", $sth->errstr;

    $sth->bind_col (1, \$epoch);
    $sth->fetch;
    print "\tSlave Epoch = $epoch\n";
    $sth->finish;
}

##### Find the position of the last GCI in the binlog #####

sub GetMasterInfo
{
    $sth = $dbhM->prepare("SELECT
        SUBSTRING_INDEX(File, '/', -1), Position
        FROM mysql.ndb_binlog_index
        WHERE epoch > $epoch
        ORDER BY epoch ASC LIMIT 1;")
    or die "Prepare to select from master error: ", $dbhM->errstr;

    $sth->execute
    or die "Selecting from master error: ", $sth->errstr;
}

```

```

$sth->bind_col (1, \${binlog});
$sth->bind_col (2, \${binpos});
$sth->fetch;
print "\tMaster bin log = ${binlog}\n";
print "\tMaster Bin Log position = ${binpos}\n";
$sth->finish;
}

##### Set the slave to process from that location #####

sub UpdateSlave
{
    $sth = $dbhS->prepare("CHANGE MASTER TO
        MASTER_LOG_FILE='${binlog}',
        MASTER_LOG_POS='${binpos}';")
    or die "Prepare to CHANGE MASTER error: ", $dbhS->errstr;

    $sth->execute
    or die "CHANGE MASTER on slave error: ", $sth->errstr;
    $sth->finish;
    print "\tSlave has been updated. You may now start the slave.\n";
}

# end reset-slave.pl

```

MySQL Cluster ディスク データ ストレージ

MySQL 5.1.6 では、ディスクの **NDB** テーブルの非インデックスのカラムを、以前のバージョンの MySQL Cluster の RAM 以外に保存できるようになりました。

クラスタ ディスク データの作業の一貫で、ノードのリカバリおよび再起動時に多量 (テラ バイト) のデータの取扱い効率を上げるための多くの改善を加えています。これらの改善点の中には非常に大きなデータセットを持つノードの起動の同期する「no-steal」アルゴリズムが含まれています。詳細については MySQL Cluster の開発者 Mikael Ronström および Jonas Orelund による説明書 [Recovery Principles of MySQL Cluster 5.1](#) を参照してください。

MySQL 5.1.6 あるいはそれ以降で移動する MySQL Cluster をすべてのノード (マネジメント および SQL ノードを含む) を設定したと仮定すると、ディスクでクラスタ テーブル作成の基本的なステップは以下のようになります。

1. log file group を作成し、1つ以上の **UNDO** ログ ファイルをそれに割り当て (**UNDO** ログ ファイルは **UNDOFILE** とも言われます) ます。
2. tablespace を作成し、1 つ以上のデータ ファイルおよびログ ファイル グループをそれに割り当てます。
3. データのストレージにテーブルスペースを使用するあディスク データ テーブルを作成します。

これらの各タスクは以下の例に示すように SQL ステートメントで実行できます。

1. **lg_1** の名前のログ ファイル グループを **CREATE LOGFILE GROUP** を使用して作成します。このログ ファイルのグループは 2 つの **UNDO** ログ ファイルで構成され、それぞれの名前を **undo_1.dat** および **undo_2.dat** とし、それらの初期サイズはそれぞれ 16 MB および 12 MB です。(ログ ファイルをログ ファイル グループに追加する際はそれらの初期サイズを指定する必要があります。オプションで、ログ ファイル グループの **UNDO** バッファのサイズを指定するか、デフォルト値の 8 MB のまま使用することもできます。この例では、UNDO バッファのサイズを 2 MB にしています。**UNDO** ログ ファイルと一緒にログ ファイルのグループを作成する必要があります。ここでは **undo_1.dat** を **lg_1** にこの **CREATE LOGFILE GROUP** ステートメントで追加します。

```

CREATE LOGFILE GROUP lg_1
ADD UNDOFILE 'undo_1.dat'
INITIAL_SIZE 16M
UNDO_BUFFER_SIZE 2M
ENGINE NDB;

```

undo_2.dat をログ ファイルのグループに追加するには、以下の **ALTER LOGFILE GROUP** ステートメントを使用します。

```

ALTER LOGFILE GROUP lg_1

```

```
ADD UNDOFILE 'undo_2.dat'
INITIAL_SIZE 12M
ENGINE NDB;
```

いくつかの項目に関する備考

- ここで使用されている `.dat` のファイル拡張は必要ありません。ここで使用しているのはログおよびデータ ファイルが分かり易いように使用しているだけです。
- すべての `CREATE LOGFILE GROUP` および `ALTER LOGFILE GROUP` ステートメントには `ENGINE` 節を含める必要があります。MySQL 5.1 では、この節に許可された値は `NDB` および `NDBCLUSTER` です。

重要MySQL 5.1.8 およびそれ以降では、所定の時間ではログ ファイル グループは 1 つだけです。

- `UNDO` ログ ファイルをログ ファイル グループに `ADD UNDOFILE 'filename'` を使用して追加するとき、`filename` の名前のファイルがクラスタの各データ ノードの `Data Directory` の `ndb_nodeid_fs` ディレクトリで作成されます。そこでは `nodeid` はデータ ノードのノード ID です。
- `UNDO_BUFFER_SIZE` は利用できるシステム メモリの容量によって制限されます。
- `CREATE LOGFILE GROUP` ステートメントの詳細に関しては [CREATE LOGFILE GROUP 構文](#) を参照してください。`ALTER LOGFILE GROUP` の詳細は、[ALTER LOGFILE GROUP 構文](#) を参照してください。

2.

ここでテーブルスペースを作成します。テーブルスペースは MySQL Cluster ディスク データ テーブルで使用されるそれらのデータを保存するファイルを含みます。テーブルスペースは特定のログ ファイルのグループに関連付けられています。新たにテーブルスペースを作成する際は、`UNDO` ロギングで使用するログ ファイルのグループを指定する必要があります。データ ファイルも指定する必要があります。テーブルスペースを作成した後にさらにテーブルスペースを追加することもできます。データスペースからデータ ファイルを削除することもできます (データ ファイルの削除の例はこの項で後ほど提供します)。

`ts_1` の名前で `lg_1` でログ ファイルのグループとして使用されるテーブルスペースを作成するものとします。このテーブルスペースは 2 つのデータ ファイル `data_1.dat` および `data_2.dat` を含むものとし、それぞれの初期のサイズをそれぞれ 32 MB および 48 MB とします。これを 2 つの SQL ステートメントを使用して行います。`CREATE TABLESPACE`、`ts_1` をデータ ファイル `data_1.dat` で作成し、`ts_1` をログ ファイル グループ `lg_1` に関連付けし、`ALTER TABLESPACE` は 2 番目のデータ ファイルを追加します。以下のこれらのステートメントを示します。

```
CREATE TABLESPACE ts_1
ADD DATAFILE 'data_1.dat'
USE LOGFILE GROUP lg_1
INITIAL_SIZE 32M
ENGINE NDB;

ALTER TABLESPACE ts_1
ADD DATAFILE 'data_2.dat'
INITIAL_SIZE 48M
ENGINE NDB;
```

いくつかの項目に関する備考

- ここで `UNDO` ログ ファイルに使用されたファイル名の場合と同様、`.dat` ファイル拡張の特別な重要性はありません。分かり易くするためにのみ使用しています。
- すべての `CREATE TABLESPACE` および `ALTER TABLESPACE` ステートメントは `ENGINE` 節を含む必要があり、テーブルスペースと同じストレージ エンジンを使用しているテーブルのみがテーブルスペースで作成されます。MySQL 5.1 では、この節に許可された値は `NDB` および `NDBCLUSTER` だけです。

`CREATE TABLESPACE` および `ALTER TABLESPACE` ステートメントに関する詳細は、[CREATE TABLESPACE 構文](#)、および [ALTER TABLESPACE 構文](#) を参照してください。

3.

ここで非インデックスのカラムがテーブルスペース `ts_1` のディスクの保存されるテーブルを作成できます。

```
CREATE TABLE dt_1 (
member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
```



```
last_name VARCHAR(50) NOT NULL,
first_name VARCHAR(50) NOT NULL,
dob DATE NOT NULL,
joined DATE NOT NULL,
INDEX(last_name, first_name)
)
TABLESPACE ts_1 STORAGE DISK
ENGINE NDB;
```

TABLESPACE ...STORAGE DISK 節が **NDB Cluster** ストレージ エンジンにテーブルスペース **ts_1** をディスク データ ストレージに使用するよう指示します。

以下のようにテーブル **ts_1** が以下のように作成されたら、他の MySQL テーブルできるように **INSERT**、**SELECT**、**UPDATE**、および **DELETE** ステートメントを実行できます。

テーブル **dt_1** にはここで定義されたように、**dob** および **joined** カラムのみがディスクに保存できます。これは **id** にインデックスがあるからで、**last_name**、および **first_name** カラム、並びにこれらのカラムに所属するが RAM に保存されます。MySQL 5.1 では、非インデックスカラムのみがディスクの保存されます。インデックスおよびインデックスの付いたカラムはメモリに保存されます。このインデックスと RAM の保存間の兼ね合いはディスク データ テーブルを設計する際に忘れてはならないものです。

重要MySQL 5.1 のディスク データ テーブルには、変数長カラムがある一定のスペースを使用します。各行では、そのカラムの一番大きな値を保存するのに必要なスペースに相当します。(これらの計算に関するヘルプに関しては、[データタイプが必要とする記憶容量](#) を参照してください。)

注:クラスタを **--initial** オプションで起動するとディスク データ ファイルは削除されません。クラスタを最初の再起動を実行するときに手動でそうする必要があります。

これらを使用しているログ ファイル グループ、テーブルスペース、およびディスク データ テーブルは特別な順序で作成する必要があります。これらのオブジェクトを削除する際も同様です。

- ログ ファイル グループはテーブルスペースがそれを使用している場合は削除できません。
- テーブルスペースはそれがデータ ファイルを含んでいる場合には削除できません。
- テーブルスペースを使用しているテーブルが残っている限りテーブルスペースからデータ ファイルを削除することはできません。
- MySQL 5.1.12 以降では、ファイルが作成されたもの以外の異なるテーブルスペースに関連して作成されたファイルは削除できなくなりました。(Bug #20053)

例えば、この項でこれまで作成してきたすべてのオブジェクトを削除するには、以下のステートメントを使用します。

```
mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_2.dat'
-> ENGINE NDB;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_1.dat'
-> ENGINE NDB;

mysql> DROP TABLESPACE ts_1
-> ENGINE NDB;

mysql> DROP LOGFILE GROUP lg_1
-> ENGINE NDB;
```

これらのステートメントは表示された順序で実行する必要があります。**ALTER TABLESPACE ...DROP DATAFILE** のこの 2 つのステートメントはどちらの順序でも実行できる場合があります。

INFORMATION_SCHEMA データベースで **FILES** テーブルにクエリしてディスク データ テーブルに使用されるデータ ファイルに関する情報を取得できます。undo ログ ファイルに関する情報を提供するために特別な「**NULL 行**」が MySQL 5.1.14 に追加されています。使用例に関する詳細は、[INFORMATION_SCHEMA FILES テーブル](#) を参照してください。 .

パラメータの設定により以下を含むディスク データ の振る舞いに影響を及ぼします。

- [DiskPageBufferMemory](#)
これによりディスクのキャッシュ ページに使用されるスペースを決め、[config.ini](#) ファイルの [\[NDBD\]](#) あるいは [\[NDBD DEFAULT\]](#) セクションに設定されます。そのスペースはバイトで測定されます。各ページは 32k です。このことは N が負の整数以外の場合、クラスタ デスク データ ストレージは常に $N * 32k$ メモリを使用することを意味します。
- [SharedGlobalMemory](#)
これはログ バッファに使用されるメモリ容量、ディスク オペレーション (ページ リクエストおよび待ちキューなど)、およびテーブルスペースのメタデータ、ログ ファイル グループ、[UNDO](#) ファイル、ならびにデータ ファイルを決めます。それは [config.ini](#) 設定ファイルの [\[NDBD\]](#) あるいは [\[NDBD DEFAULT\]](#) セクションで設定され、バイトで測定されます。

注:[OPTIMIZE TABLE](#) ステートメントはディスク データ テーブルには影響を及ぼしません。

MySQL Cluster での高速インターコネクトを使用する

1996 年に [NDB Cluster](#) の設計を始める以前から既に、ネットワークのノード間の通信が並列のデータベースの構築で問題になるだろうということは明らかになっていました。このため、[NDB Cluster](#) が多くの異なるデータ転送メカニズムに使用するためにその初期の段階から設計されました。本マニュアルでは、これらの用語に transporter を使用しています。

MySQL Cluster のコードベースでは 4 つの異なるトランスポートをサポートしています。

- TCP/IP は 100 Mbps あるいはギガ バイト Ethernet を [Cluster TCP/IP Connections](#) の説明にあるように使用しています。
- 直接 (マシン対マシン) TCP/IP、このトランスポートは同じ TCP/IP プロトコルを以前説明したアイテムと使用していますが、ハードウェアの設定を別にし設定も同様に変える必要があります。このため、それは MySQL Cluster の個別の転送メカニズムと考えられています。詳細は [直接接続を使用した TCP/IP の接続](#) を参照してください。
- 共有メモリ (SHM)。SHM に関する情報は、[共有メモリ接続](#) を参照してください。
- スケーラブル コヒーレント インターフェース (SCI)、本章の次項、[SCI トランスポート接続](#) で説明します。

多くのユーザーはそれがユビキタスであるため現在 TCP/IP を Ethernet で使用しています。TCP/IP はまたこれまで MySQL Cluster での使用に於いては最も検証されたトランスポートです。

[ndbd](#) プロセスによる通信を「chunks」でできるように現在準備しています。これができるようになるとすべてのデータ 転送に恩恵を齎します。

ユーザーが希望されるのであれば、クラスタのインターコネクトを使用してパフォーマンスをより向上させることも可能です。これを実現する 2 つの方法があります。カスタムのトランスポートをこれに使用できるように設計するか、あるいは TCP/IP スタックを拡張できるようにバイパスするソケットを使用することによって実現できます。弊社では [Dolphin](#) 社が開発した SCI (スケーラブル コヒーレント インターフェース) テクノロジを使用してこの 2 つのテクノロジを実験しました。

SCI ソケットを使用するための MySQL Cluster の設定

この項では、クラスタを通常の TCP/IP 通信に SCI ソケットを使用できるようにするために設定について説明します。この説明は 2004 年 10 月 1 日現在の SCI Socket バージョン 2.3.0 に基づいています。

必要条件

SCI ソケットを使用するマシンは SCI カード実装が必要です。

SCI ソケットはどのバージョンの MySQL Cluster でも使用できます。それは既に MySQL Cluster で利用できる通常のソケット呼び出しを使用していますので特別な構築は必要ありません。しかし、SCI Scket は現在 Linux 2.4 および

2.6 kernels 上でしかサポートされていません。SCI トランスポーターはその他のオペレーティング システムでも検証されていますが今までのところ弊社では Linux 2.4 でしかこれらの検証を行っておりません。

これらは SCI Socket の使用する際の基本的な要件です。

- SCI Socket ライブラリの構築。
- SCI Socket kernel ライブラリのインストール。
- 1 つ以上の設定ファイルのインストール。
- SCI Socket kernel ライブラリはマシン全体あるいは MySQL Cluster プロセスが実行されるシェルのいずれかに有効でなければなりません。

このプロセスは SCI Socket をインターノード通信に使用する際のクラスタの各マシンに繰返されます。

SCI Socket を動作させるには 2 つのパッケージを取り出す必要があります。

- SCI Socket ライブラリの DIS サポート ライブラリを含むソースコード パッケージ。
- SCI Socket ライブラリそのもののソースコード パッケージ

現在は、これらはソースコードのフォーマットでしか利用できません。本マニュアルを書いている段階でのこれらのパッケージの最新のバージョンは (それぞれ) [DIS_GPL_2_5_0_SEP_10_2004.tar.gz](#) および [SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz](#) が利用できました。これらは (または最新バージョン) は <http://www.dolphinics.no/support/downloads.html> で入手できます。

パッケージのインストール

これらのライブラリのパッケージを入手したら、以下のステップでそれらを適切なディレクトリに解凍し、そこで SCI Socket ライブラリを DIS コードの下ディレクトリに入れます。次に、ライブラリを構築する必要があります。この例ではこのタスクを実行するための Linux/x86 でのコマンドを示します。

```
shell> tar xzf DIS_GPL_2_5_0_SEP_10_2004.tar.gz
shell> cd DIS_GPL_2_5_0_SEP_10_2004/src/
shell> tar xzf ../SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz
shell> cd ../adm/bin/Linux_pkgs
shell> ./make_PSB_66_release
```

これらのライブラリを 64 ビットのプロセッサに構築できます。64 ビット拡張を使用してライブラリを Opteron CPU に構築するには、[make_PSB_66_X86_64_release](#) を実行します。[make_PSB_66_release](#) ではありません。それを Itanium マシンに構築した場合には、[make_PSB_66_IA64_release](#) を使用する必要があります。X86-64 バリエーションは Intel EM64T アーキテクチャで動作しますが、これはまだ (弊社の知っている限り) まだ検証されていません。

構築が完了したら、コンパイルしたライブラリは zip の tar ファイルで [DIS-<operating-system>-time-date](#) の名前で検索できます。ここでパッケージを適切なスペースにインストールします。この例ではインストールを [/opt/DIS](#) に入れます。(注:多くの場合以下をシステム [root](#) ユーザーとして実行する必要があります。)

```
shell> cp DIS_Linux_2.4.20-8_181004.tar.gz /opt/
shell> cd /opt
shell> tar xzf DIS_Linux_2.4.20-8_181004.tar.gz
shell> mv DIS_Linux_2.4.20-8_181004 DIS
```

ネットワークの設定

すべてのライブラリおよびバイナリが準備できたら、SCI カードが SCI アドレス スペースで適切なノード ID を持っているか確認する必要があります。

次に進む前にネットワーク構成で決める必要があります。これ例で利用できるネットワーク構成は 3 種類あります。

- 簡単な一次元リング
- スイッチ ポートと毎に 1 つのリングを持つ 1 つ以上の SCI スイッチ
- 2 あるいは 3 次元トラス

これらの各トポロジにはノード ID を提供するそれぞれの手法があります。簡単にそれぞれについて説明します。

簡単なリングは非ゼロの 4 の倍数を使用します。4, 8, 12,...

次の例は SCI スイッチを使用しています。SCI スイッチには 8 ポートあり、それぞれのポートはリングをサポートします。異なるリングは異なるノード ID スペースを使用することを確認する必要があります。一般的な設定では、最初のポートは 64 (4-60) 以下のノード ID を使用し、次の 64 ノード ID (68-124) は次のポートに割り当てられ、そのように続いてノード ID 452-508 は 8 番目のポートに割り当てられます。

2、3 次元のトーラス ネットワーク構成は各ノードは各次元のどこに配置され、最初の次元で各ノードを 4 で増分し、2 次元では 64、および (適用できる場合) 3 次元では 1024 で増分します。さらに詳しい説明については [Dolphin 社のウェブサイト](#) を参照してください。

弊社の試験ではスイッチを使用しました。大きなクラスターのインストールでは 2 あるいは 3 次元のローラス構成を使用します。スイッチを使用する利点は、2 つの SCI および 2 つのスイッチでは、比較的容易に冗長ネットワークを構築でき、そこでは SCI ネットワークの標準フェールオーバー時間は一般的に 100 ミリセカンドです。これは MySQL Cluster の SCI トランスポーターでサポートされており、現在 SCI Socket に導入するために開発中です。

2D/3D トーラスでのフェールオーバーは可能ですがすべてのノードに対して新しいルート インデックスを送る必要があります。しかし、これには それを完了するには 100 ミリセカンドあるいはそれくらいが必要で、多くの高可用性を要求されるケースで使用せきなればなりません。

クラスターのデータ ノードをスイッチのアーキテクチャに適切に配置することで、2 つのスイッチを使用して 16 台のコンピュータをインターコネクトし 1 つの不具合が他の 1 つ以上に影響を及ぼさない構成を構築できます。32 台のコンピュータと 2 つのスイッチで、1 つの不具合が 2 つ以上のノードの損失につながらないようにクラスターを設定することができます。この場合、その組のノードが影響されたかを知ることも可能です。このように、2 つのノードを個別のノード グループに配置することで、「安全な」MySQL Cluster のインストールを構築できます。

ノード ID を SCI カードに設定するには `/opt/DIS/sbin` ディレクトリの以下のコマンドを使用します。この例では、`-c 1` は SCI カード (これはマシンにカードが 1 つだけの場合には常に 1 です) の番号を表し、`-a 0` はアダプタ 0、および `68` はノード ID を意味します。

```
shell> ./sciconfig -c 1 -a 0 -n 68
```

同じマシンに複数の SCI カードがある場合、以下のコマンド (ここでは使用するディレクトリを `/opt/DIS/sbin` とします) を発行することでどのカードにどのスロットがあるか決めることができます。

```
shell> ./sciconfig -c 1 -gsn
```

これにより SCI カードのシリアル番号が決まります。次にマシンの各カードにこのプロシージャを `-c 2` などで繰り返します。各カードをスロットに合わせたら、ノード ID をすべてのカードに設定できます。

必要なライブラリおよびバイナリをインストールすると、SCI ノードと ID が設定され、次のステップでホスト名 (あるいは IP アドレス) から SCI ノード ID のマッピングを設定します。これは SCI Socket の設定ファイルで行われ、`/etc/sci/scisock.conf` として保存します。このファイルで、各 SCI ノード ID は適切な SCI カードから通信するホスト名あるいは IP アドレスにマップされます。ここにその様な設定ファイルの極めて簡単な例を示します。

```
#host      #nodeid
alpha      8
beta       12
192.168.10.20 16
```

設定をこれらのホストの利用できるポートのサブセットにのみ適用できるように制限することも可能です。これを行うために別の設定ファイル `/etc/sci/scisock_opt.conf` を以下のように使用できます。

```
#key      -type      -values
EnablePortsByDefault  yes
EnablePort      tcp      2200
DisablePort      tcp      2201
EnablePortRange  tcp      2202 2219
DisablePortRange  tcp      2220 2231
```

ドライバのインストール

設定ファイルの用意ができたら、ドライバをインストールできます。

最初に、低レベルのドライバ、次に SCI ソケット ドライバをインストールする必要があります。

```
shell> cd DIS/sbin/
shell> ./drv-install add PSB66
shell> ./scisocket-install add
```

任意で、SCI ソケットの設定ファイルのすべてのノードがアクセスできることを検証するスクリプトを実行してインストールをチェックできます。

```
shell> cd /opt/DIS/sbin/
shell> ./status.sh
```

エラーが発生し SCI ソケットの設定の変更が必要な場合、このタスクを実行するためには [ksocketconfig](#) を使用する必要があります。

```
shell> cd /opt/DIS/util
shell> ./ksocketconfig -f
```

設定テスト

SCI ソケットが実際に使用されているか確認するには、[latency_bench](#) テスト プログラムを使用します。このユーティリティのサーバーコンポーネントを使用して、接続のレイテンシーをテストするためにサーバーに接続できます。SCI が有効であるかを確認するにはこのレイテンシーを確認することで用意に分かります。(注:[latency_bench](#) を使用する前に、[LD_PRELOAD](#) 環境変数をこの項の後で述べるように設定する必要があります。

サーバーを設定するには、以下を使用します。

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -server
```

クライアントを起動するには、[latency_bench](#) を、この場合は [-client](#) オプションを除いて再度使用します。

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -client server_hostname
```

SCI ソケットの設定はこれで完了し、MySQL Cluster の SCI ソケット および SCI トランスポート ([SCI トランスポート接続](#) 参照) を使用する用意ができました。

クラスタの起動

プロセスの次のステップで MySQL Cluster が起動します。SCI ソケットの使用を有効にするには、環境変数 [LD_PRELOAD](#) を [ndbd](#)、[mysqld](#)、および [ndb_mgmd](#) を実行する前に設定します。この変数は SCI ソケットの kernel ライブラリに向ける必要があります。

[ndbd](#) をバッシュ シェルで起動するには、以下に従います。

```
bash-shell> export LD_PRELOAD=/opt/DIS/lib/libkscisock.so
bash-shell> ndbd
```

tcsh 環境では、以下で同様のことが出来ます。

```
tcsh-shell> setenv LD_PRELOAD=/opt/DIS/lib/libkscisock.so
tcsh-shell> ndbd
```

注:MySQL Cluster は SCI ソケットの kernel 派生品のみ使用できます。

Cluster インターコネクトの理解する

[ndbd](#) プロセスには多くの簡単な構成があり、MySQL Cluster のアクセスに使用します。それらのパフォーマンスおよび様々なインターコネクトがそのパフォーマンスに及ぼす影響をチェックする簡単なベンチマークを作成しました。

4 つのアクセス方法があります。

- プライマリ キーアクセス

これはレコードのそのプライマリ キーによるアクセスです。最も簡単なケースでは、一度に 1 つのレコードのみによるアクセスさせます。それによって 多くの TCP/IP のメッセージの設定の全コストおよびスイッチングのコンテキストの多くのコストはこの 1 つのリクエストから生じます。この場合複数のプライマリ キーのアクセスは 1 回のバッチで送られ、それらのアクセスは TCP/IP メッセージの必要な設定およびスイッチのコンテキストに要する費用を分担します。TCP/IP メッセージのディスティネーションが異なる場合、新たに TCP/IP メッセージを設定する必要があります。

- 一意のキーアクセス

一意のキーアクセスは一意のキーアクセスはテーブルのプライマリ キーアクセスに続くインデックス テーブルの読み込みとして実行され以外はプライマリ キーアクセスに類似しています。しかし、MySQL サーバーからはリクエストが 1 つだけ送られ、インデックス テーブルの読み込みは [ndbd](#) によって処理されます。それらのリクエストはバッチにより恩恵を受けます。

- テーブルの完全スキャン

テーブルのルックアップにインデックスが存在しない場合、テーブルの完全スキャンを実施します。これは 1 つのリクエストとして [ndbd](#) プロセスに送られ、そこでテーブルスキャンをすべてのクラスタ [ndbd](#) プロセスの一連の並列スキャンに分割します。将来的には MySQL Cluster のバージョン、SQL ノードはこれらのスキャンのいくつかをフィルタできるようにします。

- 順序付けされたインデックスを仕様したレンジ スキャン

順序付けされたインデックスを使用すると、SQL サーバー (SQL ノード) により転送されたクエリの範囲にあるそれらのレコードのみをスキャンする以外は、全なテーブルのスキャンと同様にスキャンを実施します。すべての結合した属性がパーティション キーのすべての属性を含む場合パーティションは並列でスキャンされます。

これらのアクセス方法の基本的なパフォーマンスをチェックするために、一連のベンチマークを開発しました。そのようなベンチマークの一つは、[testReadPerf](#)、で n 簡単なバッチのプライマリおよび一意のキーアクセスをテストします。このベンチマークはまた 1 つのレコードを返すスキャンを発行することで範囲スキャンの設定コストを測定します。範囲スキャンを使用してレコードのバッチを取り出すこのベンチマークの派生版もあります。

この様に、基本となるアクセス方法で、使用している通信メディアの影響を測定するばかりでなく、1 つのキーアクセスおよび 1 つのレコードスキャン アクセスの両方のコストを決定できます。

弊社のテストでは、これらのベンチマークを TCP/IP ソケットを使用した通常のトランスポートおよび SCI ソケットを使用した類似の設定の両方に使用しています。以下テーブルのレポートした図はアクセス毎に 20 レコードの小さなアクセスに使用します。シリアルおよびバッチのアクセスは 2KB のレコードを使用した場合 3 と 4 の因数が減少します。SCI ソケットは 2KB レコードでテストしていません。テストは AMD 社の MP1900+ プロセッサを搭載した 2 つのデュアル CPU マシン上で動作する 1 つのクラスタに 2 つのデータ ノードで実行しました。

アクセス タイプ	TCP/IP ソケット	SCI ソケット
シリアル pk アクセス	400 マイクロセカンド	160 マイクロセカンド
バッチ pk アクセス	28 マイクロセカンド	22 マイクロセカンド
シリアル uk アクセス	500 マイクロセカンド	250 マイクロセカンド
バッチ uk アクセス	70 マイクロセカンド	36 マイクロセカンド
インデックス eq-バウンド	1250 マイクロセカンド	750 マイクロセカンド
インデックス レンジ	24 マイクロセカンド	12 マイクロセカンド

SCI ソケット vis-à-vis と SCI トランスポートのパフォーマンス、およびこれら両方を TCP/IP トランスポートと比較したパフォーマンスをチェックしました。これらすべてのテストにはプライマリ キーアクセスをシリアルおよび複数のスレッド、あるいは複数のスレッドおよびバッチのいずれかに使用しました。

このテストでは SCI ソケットが TCP/IP よりもおおよそ 100% 早いことが分かりました。SCI トランスポートは SCI ソケットに比べると殆どのケースで速くなっています。テスト プログラムの多くのスレッドで注目すべきことが発生した。それは SCI トランスポートは [mysqld](#) プロセスに使用した場合あまりよい結果を示しませんでした。

全体的な結論としては、通信のパフォーマンスが懸案でないときの珍しいインスタンスを除いて、ほとんどのベンチマークで、SCI ソケットは TCP/IP に対しておよそ 100% パフォーマンスを改善することです。これはスキンのフィルタが殆どのプロセス時間を使った場合あるいはプライマリ キーの非常に大きなバッチ アクセスが達成されたときに起こります。その様な場合、**ndbd** プロセスの CPU の処理がオーバーヘッドのかなり大きな部分になります。

SCI ソケットの代わりに SCI トランスポートを使用するには、**ndbd** プロセス間の通信の端に興味によるものです。SCI トランスポートを使用するのも単なる興味からで、SCI トランスポートがこのプロセスが決して停止しないことを確認するため、CPU を **ndbd** プロセスに専用にできるかとの単なる興味からのものです。**ndbd** プロセスの優先権がプロセスが、Linux 2.6 でプロセスをロックできるように、時間を延長して実行されても優先権がなくならないように設定されているか確認する必要があります。そのような設定が可能な場合、**ndbd** プロセスは SCI ソケットを使用した場合に比べて 10-70% 恩恵があることになります。(更新および並列スキンのオペレーションでは大きな数字になります。)

コンピュータのクラスタに Myrinet、ギガビット Ethernet、Infiniband および VIA インターフェースなど他にもいくつかの最適化ソケットの実装があります。これまでに MySQL Cluster を SCI ソケットだけでしかテストしていません。通常の TCP/IP を MySQL Cluster 使用した SCI ソケットの設定方法については、[SCI ソケットを使用するための MySQL Cluster の設定](#) を参照してください。

MySQL Cluster の既知の制限

この項では、MySQL Cluster の 5.1.x シリーズのリリースの **MyISAM** および **InnoDB** ストレージ エンジンを使用した際に利用できる機能との比較における既知の制限のリストを提供します。現在、これからリリースされる MySQL 5.1 でこれらを試す計画はありません。しかし、今後のリリースではこれらの問題で解決されたことを提供するつもりです。「Cluster」カテゴリを <http://bugs.mysql.com> の MySQL バグ データベースでチェックすると、MySQL 5.1 の今後のリリースで修正する既知のバグ (「5.1」の印の) を検索できます。

ここに掲げるリストは以下の条件で完成する予定です。何か齟齬があった場合には [質問またはバグの報告](#) の指示に従って MySQL バグ データベースにレポートできます。その問題を MySQL 5.1 で修正しない場合には、それをリストに追加します。

(注:現在のバージョンで解決された MySQL 5.0 Cluster の問題のリストについてはこの項の末尾を参照してください。)

MySQL Cluster のレプリケーションに特定の制限とその他の問題に関しては、[既知の問題](#) の説明を参照してください。

- 構文の不承諾 (既存のアプリケーションを実行中のエラーによる):
 - テンポラリ テーブルはサポートしていません。
 - テキスト インデックスはサポートしていません。つまり、**TEXT** データベースのカラムにはインデックスを作成できません。また **NDB** ストレージ エンジンは **FULLTEXT** インデックス (これらは **MyISAM** のみサポートしています。) をサポートしていません。しかし、**NDB** テーブルの **VARCHAR** カラムはインデックスできます。
 - A **BIT** カラムはプライマリ キー、インデックスにはできません。またコンポジットのプライマリ キー、一意のキー、あるいはインデックスの一部を構成することもできません。
- ジオメトリデータ タイプは (**WKT** および **WKB**) はMySQL の **NDB** テーブルでサポートされています。5.1.しかし、スペーシャル インデックスはサポートされていません。
- **CREATE TABLE** ステートメントは 4096 文字以内です。この制限は MySQL 5.1.6、5.1.7、および 5.1.8 のみに影響を及ぼします。(See Bug #17813)
- MySQL 5.1.7 およびそれ以前では、**INSERT IGNORE**、**UPDATE IGNORE**、および **REPLACE** はプライマリ キーのみサポートしており、一意のキーはサポートしていません。この制約を外す一つの解決策は一意のインデックスを削除し、挿入を実行し、次に一意のインデックスを再度追加することです。
この制限は MySQL 5.1.8 の**INSERT IGNORE** および **REPLACE** には削除されています (Bug #17431)。
- 行ベースのレプリケーションを MySQL Cluster で使用する場合、バイナリのロギングは無効にできません。つまり、**NDB** ストレージ エンジンは **SQL_LOG_BIN** の値を無視します。(Bug #16680)

- `auto_increment_increment` および `auto_increment_offset` サーバースステムの変数はクラスタのレプリケーションをサポートしていません。
- 制限あるいは振る舞いの不承認 (既存のアプリケーションを実行した場合エラーになる場合があります。):
 - エラーの報告:
 - キーエラー 2 回でエラーメッセージ `ERROR 23000` が返されます。:書き込みできません。キーをtbl_name' で、
 - 他の MySQL ストレージ エンジン同様、`NDB` ストレージ エンジンは最大の `AUTO_INCREMENT` カラムをテーブル毎に扱います。しかし、明示のプライマリ キーのないクラスタ テーブルの場合、`AUTO_INCREMENT` カラムは自動的に定義され「非表示」のプライマリ キーとして使用されます。このため、`AUTO_INCREMENT` カラムを持つテーブルはカラムもまた `PRIMARY KEY` オプションを使用して宣言されない限りテーブルを定義することはできません。

テーブルのプライマリ キーではない `AUTO_INCREMENT` カラムのテーブルを作成しようとして `NDB` ストレージ エンジンを使用すると、エラーになり失敗します。
 - トランザクションの取扱い:
 - `NDB Cluster` は `READ COMMITTED` トランザクションの分離レベルのみサポートします。
 - トランザクションの部分的なロールバックはありません。複製キーおよび同様のエラーはトランザクション全体のロールバックにつながります。
 - 重要クラスタ テーブルの `SELECT` が `BLOB` あるいは `TEXT` カラムを含む場合、`READ COMMITTED` トランザクションの分離は読み込みロックで読み込みに変換されます。これはこれらのタイプのカラムに保存された値の一部は実際は個別のテーブルから読まれるので一貫性を保証するために行われます。
 - 本章で気付かれたことと思いますが、MySQL Cluster は大きなトランザクションの取扱いは得意ではありません。非常にたくさんのオペレーションを含む 1 つの大きなトランザクションを扱うよりはオペレーションの数が少ない多くの小さなトランザクションの扱いに向いています。

とりわけ、大きなトランザクションは非常に大きなメモリを必要とします。このため、多くの MySQL ステートメントのトランザクションの振る舞いが以下のリストで説明するように影響を受けます。

 - `TRUNCATE` は `NDB` テーブルで使用するするとトランザクションを行いません。`TRUNCATE` がテーブルを空に出来ない場合、成功するまで続ける必要があります。
 - `DELETE FROM (WHERE 節がない場合でも)` はトランザクションできます。非常に多くの行を含むテーブルの場合、いくつかの `DELETE FROM ...LIMIT ...` ステートメントを使用して削除操作を「チャンク」することでパフォーマンスが改善する場合があります。テーブルを空にしたい場合には、代わりに `TRUNCATE` を使用します。
 - `TRUNCATE` は `NDB` テーブルで使用するするとトランザクションを行いません。そのようなオペレーション中には、`NDB` エンジンは指示に従って実行されます。

`LOAD DATA FROM MASTER` は MySQL Cluster ではサポートされていません。

 - テーブルを `ALTER TABLE` の一部としてコピーする場合、コピーの作成は非トランザクションです。(いずれの場合も、このオペレーションはコピーが削除されるとロールバックします。)
 - ノードの起動、停止、あるいは再起動:ノードの起動、停止、あるいは再起動によってトランザクションの失敗につながるテンポラリエラーが増える場合があります。これらには以下のケースが含まれます。
 - 最初にノードを起動する場合、エラー 1204 のテンポラリな不具合、配布の変更、 および類似のテンポラリエラーが表示される場合があります。
 - データ ノードの停止あるいは不具合によって多くの異なるノード不具合エラーにつながる場合があります。(しかし、クラスタの予定したシャットダウンを実行中にはトランザクションの中断はありません。)

これらのいずれのケースの場合にも、生成されたエラーはアプリケーション内で処理する必要があります。トランザクションを再試行することによって行われます。

- 設定可能な多くのハードの制限がありますが、クラスタの設定制限のメインメモリで利用できます。[設定ファイル](#)の設定パラメータの完全なリストを参照してください。多くの設定パラメータはオンラインで更新できます。これらのハードの制限には以下が含まれます。

- データベースのメモリ サイズとインデックスのメモリ サイズ ([DataMemory](#) および [IndexMemory](#)、それぞれ)。

[DataMemory](#) には 32KB ページが割り当てられています。各 [DataMemory](#) ページが使用されると、それは特定のテーブルに割り当てられます。一度割り当てられるとこのメモリはテーブルを削除しない限り自由にはできません。

[DataMemory](#) および [IndexMemory](#) の詳細は、[Defining Data Nodes](#) を参照してください。

- トランザクション毎に実行できる最大のオペレーション数は設定パラメータ [MaxNoOfConcurrentOperations](#) および [MaxNoOfLocalOperations](#) で設定できます。バルクでローディングする場合、[TRUNCATE TABLE](#)、および [ALTER TABLE](#) が複数のトランザクションを実行することで特別なケースとして扱われ、よってこの制限は適用されません。
- テーブルおよびインデックスに関連した異なる制限例えば、テーブル毎の最大数の順序付けされたインデックスが [MaxNoOfOrderedIndexes](#) で決められた場合。

- データベース名、テーブル名および属性名は [NDB](#) テーブルでは他のテーブル ハンドラーより長くすることはできません。属性名は 31 文字に切り捨てられ、切り捨ての後一意でない場合にはえらーになります。データベース名およびテーブル名の合計は 122 文字です。(つまり、[NDB Cluster](#) のテーブル名の最大長は 122 文字からテーブルが属しているデータベースの数を引いたものになります。

- クラスタのデータベースの最大テーブル数は 20320 に制限されています。
- MySQL 5.1.10 および以前のバージョンでは、非表示のプライマリ キーを含む [AUTO_INCREMENT](#) カラム — を持つ最大のテーブル数 — は 2048 です。

この制限は MySQL 5.1.11 では解除されています。

- テーブル毎の最大属性数は 128 に制限されています。
- 1 行の許可された最大サイズは 8KB です。それぞれの [BLOB](#) あるいは [TEXT](#) カラムはこの合計に対して $256 + 8 = 264$ バイトです。
- キー毎の属性の最大数は 32 です。

- サポートされていない機能 (エラーにはならないが、サポートされていないもしくは強化できない) :

- 外部キーの構成は、[MyISAM](#) テーブルの場合と同様無視されます。
- セーブポイントおよびサブポイントへのロールバックは [MyISAM](#) で無視されます。
- [OPTIMIZE](#) オペレーションはサポートされていません。
- [LOAD TABLE ...FROM MASTER](#) はサポートされていません。

- パフォーマンスおよび制限に関連した問題:

- [NDB](#) ストレージ エンジンへのシーケンシャルなアクセスによるクエリのパフォーマンスの問題があります。多くの範囲のスキャンをするのは [MyISAM](#) あるいは [InnoDB](#) で行うより比較的高価になります。

- **Records in range** 統計は利用できますが検証が済んでいないあるいは公式にはサポートしていません。これにより場合によっては non-optimal query plan になります。必要に応じてこの目的のために **USE INDEX** あるいは **FORCE INDEX** を使用できます。
- **USING HASH** で作成された一意のハッシュ インデックスは **NULL** がキーの一部として与えられた場合はテーブルのアクセスに使用できません。
- **SQL_LOG_BIN** はデータのオペレーションには影響を及ぼしません。それはスキーマのオペレーションにサポートされています。

MySQL Cluster は **BLOB** カラムを持ちしかもプライマリ キーではないテーブルの binlog は生成しません。

以下のスキーマのみがステートメントを実行する **mysqld** にはない クラスタの binlog にログインされます。

- **CREATE TABLE**
- **ALTER TABLE**
- **DROP TABLE**
- **CREATE DATABASE / CREATE SCHEMA**
- **DROP DATABASE / DROP SCHEMA**
- **CREATE TABLESPACE**
- **ALTER TABLESPACE**
- **DROP TABLESPACE**
- **CREATE LOGFILE GROUP**
- **ALTER LOGFILE GROUP**
- **DROP LOGFILE GROUP**
- 不明な機能:
 - 唯一サポートされている分離レベルは **READ COMMITTED** です。(InnoDB は **READ COMMITTED**、**READ UNCOMMITTED**、**REPEATABLE READ**、および **SERIALIZABLE** をサポートしています。)これがバックアップおよびクラスタ データベースの保存に与える影響に関する情報は、[バックアップのトラブルシューティング](#) を参照してください。
 - ディスクで複製できないコミットコミットは複製できますがログのコミットのフラッシュは保証していません。
- 複数の MySQL サーバー (**MyISAM** あるいは **InnoDB** には無関係):
 - **ALTER TABLE** は複数の MySQL サーバー (配布テーブル ロックがない場合) を稼働中は完全にロックできません。
 - DDL オペレーションはノード不具合が起こる場合があります。これらの 1 つ (**CREATE TABLE** あるいは **ALTER TABLE** など) を実行しようとする、データディクショナリがロックされクラスタを再起動しない限り DDL ステートメントはそれ以上実行されません。
- MySQL Cluster 専用に発行 (**MyISAM** あるいは **InnoDB** には関連せず):
 - クラスタで使用されるすべてのマシンは同じアーキテクチャである必要があります。つまり、ノードをホストするすべてのマシンは big-endian あるいは little-endian のいずれかである必要があり、その両方を一緒に使用することはできません。例えば、x86 マシンで動作しているデータ ノードに指示する PowerPC で動作しているマネジメント ノードを持つことはできません。この制限は単に **mysql** あるいはクラスタの SQL ノードにアクセスする他のクライアントで稼働しているマシンには適用されません。

- ノードをオンラインで追加あるいは削除することはできません (そのような場合クラスタを再起動する必要があります)。
- 1 台のホストで同時に複数のクラスタのプロセスを実行できますが、他の要因はともかくパフォーマンスおよび高可用性の理由によってそのように使用することは推奨できません。とくに、MySQL 5.1 は 1 つ以上の `ndbd` プロセスが 1 台の物理マシンで実行されている MySQL Cluster の開発を使用している生産をサポートしていません。

今後の MySQL リリースではホスト毎の複数のデータ ノードを以下のテストを行うことでサポートします。しかし、MySQL 5.1 では、そのような設定は実験レベルだけでの検討になります。

- クラスタをディスク無しで稼働している場合ディスク データ テーブルはサポートされていません。MySQL 5.1.12 以降のバージョンではそれはどちらも許可されていません。(Bug #20008)

- 複数のマネジメント サーバーを使用する場合：

- ノード ID の自動割り当ては複数のマネジメント サーバー間では機能しないので接続文字列でノードに明示の ID を与える必要があります。
- すべてのマネジメント サーバーに同じ設定をする際は特別な注意が必要です。この件に関してクラスタでは特別なチェックありません。
- データ ノード毎の複数のネットワークはサポートされていません。これらを使用すると問題が発生します。データ ノードの不具合が発生した場合には、そのデータ ノードに別のルートが開放されていますので SQL はデータ ノードが停止ししかもそれを受信しなくなるまでその確認を待ちます。これにより効果的にクラスタの稼働を止めます。

1 つのデータ ノードに複数のネットワーク ハードウェア インターフェース (Ethernet カードなど) を使用できますが、これらは同じアドレスを使用する必要があります。これは `config.ini` ファイルで接続毎に 1 つ以上の `[TCP]` セクションを使用できないことを意味しています。詳細については、[Cluster TCP/IP Connections](#) をご参照してください。

- データ ノードの最大数は 48 です。
- MySQL Cluster のノードの最大数は 63 です。この数にはすべての SQL ノード (MySQL サーバー)、API ノード (MySQL サーバー以外にクラスタにアクセスするアプリケーション)、データ ノード、およびマネジメント サーバーが含まれています。
- MySQL 5.1 Cluster のメタデータ オブジェクトの最大数は 20320 です。この制限はハードで制限されています。

MySQL 5.1 で解決された MySQL Cluster 以前のバージョンの問題:

- [NDB Cluster](#) ストレージ エンジンは今では in-memory テーブルにサポートしています。

以前は、これは— 例えば — 比較的に小さい値をもつ 1 つ以上の `VARCHAR` フィールドを持ち、`NDBCluster` ストレージ エンジンを使用するときと同じ テーブルおよびデータを持つ `MyISAM` エンジンを使用した場合に比べてより多くのメモリやディスク スペースを必要としたクラスタ テーブルのことを意味します。言い換えれば、`VARCHAR` カラムの場合で、同じサイズの `CHAR` カラムと同じストレージ容量が要求されるカラムのです。MySQL 5.1 では、これはもはや in-memory テーブルには当てはまらず、そこでは `VARCHAR` および `BINARY` などの変数カラムのストレージ要件が `MyISAM` テーブルで使用された場合これらのカラム タイプに対してそれらと互換性があります([データタイプが必要とする記憶容量](#) 参照)。

重要MySQL Cluster のディスク データ テーブルでは、固定幅の制限はそのまま適用されます。[MySQL Cluster ディスク データ ストレージ](#) 参照。

- MySQL のレプリケーションを Cluster データベースに使用できるようになりました。詳細に関しては [MySQL Cluster レプリケーション](#) を参照して下さい。

しかしながら、循環型レプリケーションは現在 MySQL では現在サポートされていません。[既知の問題](#) 参照。

- 所定の `mysqld` が既に動作していてデータベースが異なる `mysqld` で作成されると同時にクラスタに接続される場合、データベースの自動検索は現在同じ MySQL Cluster にアクセスする複数の MySQL サーバーをサポートしています。

このことは `mysqld` プロセスが `db_name` の名前のデータベースが作成された後に最初にクラスタの接続すると、それが最初に MySQL Cluster にアクセスしたときに「新しい」MySQL サーバーで `CREATE SCHEMA db_name` ステートメントを発行する必要があります。これが完了すると、その「新しい」`mysqld` はそのデータベースのテーブルをエラーなしで削除できます。

このことはまたオンラインで `NDB` テーブルのスキーマの変更が可能であることを意味しています。つまり、クラスタの SQL ノードで実行された `ALTER TABLE` および `CREATE INDEX` などのオペレーションの結果がなんら他の操作をしなくてもクラスタの他の SQL ノードに見えるということです。

- MySQL 5.1.10 以降では、クラスタのバックアップを行って異なるアーキテクチャ間で保存できます。以前は—例えば—big-endian プラットフォームで動作しているクラスタをバックアップできず、またそのバックアップから little-endian システムで動作しているクラスタに保存できませんでした。(Bug #19255)
- MySQL 5.1.10 以降は MySQL をクラスタのサポート付きで非デフォルトのロケーションにインストールして `--basedir` あるいは `--character-sets-dir` オプションのいずれかを使用してフロント ディスクリプション ファイルの検索パス `k` を変更できます。(以前は MySQL 5.1 の `ndbd` はデフォルトのパスのみを検索していました—一般的には `/usr/local/mysql/share/mysql/charsets`—文字セットに対して)
- MySQL 5.1 では、それはもはや必要なくなり、複数のマネジメント サーバーを稼働するとき、マネジメント ノードがお互いを見るようにするにはすべてのクラスタのデータ ノードを再起動します。

MySQL Cluster 開発ロードマップ

この項では、MySQL 5.1 を MySQL 5.0 に比較した場合の MySQL Cluster の実装における変更点について説明します。

MySQL 5.0 に比較した場合、MySQL 5.1 への `NDB Cluster` ストレージ エンジンの実装では多くの大きな変更点があります。これらの変更点の概要については、[MySQL 5.1 における MySQL Cluster の変更点](#) を参照してください。

MySQL 5.1 における MySQL Cluster の変更点

MySQL Cluster の多くの新機能が MySQL 5.1 に導入されています。:

- MySQL Cluster の MySQL レプリケーションへの統合:この統合によってクラスタのどの MySQL サーバーからの更新が可能になりクラスタの MySQL サーバーの 1 台による MySQL レプリケーションを持ち、スレーブ スライドの状態がマスタとして機能するクラスタとの一貫性を維持しています。

詳しくは [MySQL Cluster レプリケーション](#) を参照してください。

- ディスク ベースのレコードのサポート:ディスクのレコードは今サポートされています。プライマリ キーのハッシュ インデックスを含むインデックス領域はまだ RAM に保存される必要がありますが、他のすべての領域はディスクに保存できます。

詳しくは [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

- 変数サイズのレコード:`VARCHAR(255)` として定義されたカラムは現在特定のレコードに保存されたものとは別個に 260 バイトのストレージを使用しています。MySQL 5.1 のクラスタ テーブルでは、実際はレコードで使われるカラムの一部のみが保存されます。これによりそのようなカラムのスペースを多くのケースで 5 倍ほど削減します。
- ユーザー定義のパーティショニング:ユーザーはプライマリ キーの一部であるカラムのパーティションを定義できます。`NDB` テーブルを `KEY` および `LINEAR KEY` スキーマに基づいてパーティションできます。この機能は多くの他の MySQL ストレージ エンジンにも利用できます。それによって `NDB Cluster` テーブルでは利用できない特別なパーティショニングをサポートします。

MySQL 5.1 のユーザー定義のパーティショニングの一般情報の詳細に関しては、[15章パーティショニング](#) を参照してください。パーティショニングの詳細については [パーティショニングのタイプ](#) で説明します。

MySQL サーバーはいくつかのパーティションを [WHERE](#) 節から「剪定」することもできます。 [パーティションの刈り込み](#) 参照。

- テーブル スキーマの変更点の自動検索 MySQL 5.1 では、[FLUSH TABLES](#) あるいは「ダミー」 [SELECT](#) を [NDB](#) テーブルあるいは 1 つの SQL ノードの既存の [NDB](#) テーブルのスキーマに加えた変更に対して発行して クラスタの他の SQL ノードで見えるようにする必要がなくなりました。

注:新たにデータベースを作成するとき、それでも [CREATE DATABASE](#) あるいは [CREATE SCHEMA](#) ステートメントをクラスタの各 SQL ノードに発行する必要があります。

詳細に関しては [MySQL Cluster issues from previous versions that have been resolved in MySQL 5.1 \[987\]](#) を参照してください。

MySQL Cluster の用語

以下の用語は MySQL Cluster の理解あるいは MySQL Cluster に関する特別な意味を持ちます。

- クラスタ:

一般的な意味は、クラスタはユニットで動作する一連のコンピュータで 1 つのタスクを実行するために一緒に作業します。

[NDB Cluster](#):

これは MySQL のストレージ エンジンでいくつかのコンピュータを束ねてデータ ストレージ、検索、および管理を行うためのものです。

MySQL Cluster:

これは [NDB](#) ストレージ エンジンを使用した一群のコンピュータが一緒に作業して、in-memory storage を使用してアーキテクチャを共有しない構成で分散型 MySQL データベースをサポートすることを意味します。

- 設定ファイル:

テキスト ファイルで命令およびクラスタ、そのホスト、およびそのノードに関する情報を含んでいます。これらはクラスタが起動したときにクラスタのマネジメント ノードで読み込まれます。詳細は [設定ファイル](#) を参照してください。

- Backup:

クラスタ、トランザクションおよびログの完全なコピーで、ディスクあるいは他の長期使用のストレージに保存されます。

- 復旧:

クラスタをバックアップで保存した前の状態に戻します。

- チェックポイント:

一般的には、データがディスクの保存された場合、チェックポイントに達したと言われています。クラスタに特化した場合には、すべての実行されたトランザクションがディスクに時宜をえて保存された時点の意味します。[NDB](#) ストレージ エンジンに関しては、2 種類のチェックポイントがありそれが共同して安定したクラスタの表示を維持します。

- ローカル チェックポイント (LCP):

これは 1 つのノードに特化したチェックポイントですが、LCP はクラスタではほぼ同時にすべてのノードで行われます。LCP はノードのすべてのデータのディスクへの保存に関わり、通常は数分間隔でそれを実行します。この正確な間隔はノードで保存されるデータ量、クラスタの作業量、および他の要因によってばらつきがあります。

- グローバル チェックポイント (GCP):

GCP はすべてのノードのトランザクションの同期および redo-log のディスクへのフラッシュ時に数秒毎に実行されます。

- クラスタ ホスト:

MySQL Cluster の一部を構成するコンピュータ。クラスタには物理構成と論理構成の両方があります。物理的には、クラスタ ホスト (あるいはもっと単に ホスト) と呼ばれる多くのコンピュータで構成されます。以下のノードおよびノード グループを参照してください。

- ノード:

これは MySQL Cluster の論理および関数単位に関するもので、クラスタ ノード とも呼ばれます。MySQL Cluster の説明では、「ノート」用語をクラスタの物理コンポーネントよりはむしろプロセスとして使用しています。MySQL Cluster の構築するには 3 種類のノードが必要です。

- マネジメント (MGM) ノード:

MySQL Cluster の他のノードを管理します。それは他のノードに設定データ、つまりノードの起動および停止、ネットワークのパーティショニングの取扱い、バックアップの作成およびその保存などの設定データを提供します。

- SQL (MySQL サーバー) ノード:

MySQL サーバーのインスタンスで、クラスタの データ ノード に保持されたデータへのフロントエンドの役割を果たします。データを保存、抽出、あるいは更新を希望するクライアントは他の MySQL サーバーにアクセスするように SQL ノードにアクセスし、通常の認証手法および API を使用し、ノード グループ間のデータの配布をユーザーやアプリケーションに透明にします。SQL ノードは異なるデータ ノードあるいはホスト間のデータの配布には関係なくクラスタのデータベース全体にアクセスします。

- データ ノード:

これらのノードが実際のデータを保存します。テーブル データのフラグメントは一連のノード グループに保存されます。各ノード グループはテーブル データの異なるサブセットを保存します。ノード グループを構成する各ノードはノード グループが担当するフラグメントのコピーを保存します。現在は 1 つのクラスタは合計で 48 データ ノードまでサポートしています。

1 つ以上のノードが 1 台のマシンに共存できます。(実際は、1 台のマシンに完全なクラスタを構築できますが、これを実際の生産環境で使用しようとは思わないでしょう。MySQL Cluster で作業するとき、用語のホストはクラスタの物理コンポーネントで、一方ノード は論理あるいは機能のコンポーネント (つまり、プロセス) を覚えておくくと便利です。

用語に関する備考:旧バージョンの MySQL Cluster の説明書では、データ ノードを「データベース ノード」を書いてきたこともあります。「ストレージ ノード」の用語を使っていたときもあります。さらに、SQL ノードを「クライアント ノード」と呼んだときもあります。古い用語は混乱を避けるために使われなくなっており、その意味においては使わないほうがいいでしょう。それらはまた「API ノード」とも言われます — SQL ノードは実際はクラスタへの API ノードのインターフェースを提供します。

- ノード グループ:

一連のデータ ノードノード グループのすべてのデータ ノードは同じデータ (フラグメント) を含みます。そして 1 つのグループのすべてのノードは異なるホストに常駐する必要があります。どのノードがどのノード グループに属するか管理できます。

詳細は、[MySQL Cluster ノード](#)、[ノード グループ](#)、[レプリカ](#)、および[パーティション](#) を参照してください。

- ノード不具合:

MySQL Cluster はクラスタを構成する 1 つのノードのだけに依存している訳ではないので、クラスタは 1 つ以上のノードが失敗しても動作を続けます。所定のクラスタが耐えられるノードの不具合の正確な数はノード数およびクラスタの構成によります。

- ノードの再起動:

失敗したクラスタ ノードの再起動プロセス

• 内部ノードの再起動:

ファイルシステムを削除したクラスタ ノードの起動プロセス。これはソフトウェアのアップグレードおよび他の特殊な環境でたまに使用されます。

• システム クラッシュ (あるいは システム不具合):

これは多くのクラスタ ノードに不具合が発生してクラスタがそれ以上耐えられなくなると起こります。

• システムの再起動:

クラスタの再起動およびディスクのログおよびチェックポイントからの初期化のプロセスです。これはクラスタの予定されたあるいは予定外のシャットダウンの後に必要になります。

• フラグメント:

データベース テーブルの一部で、[NDB ストレージ エンジン](#)では、テーブルは分割されたたくさんのフラグメントで保存されます。フラグメントはたまに「パーティション」とも呼ばれます。しかし、「フラグメント」のほうが好ましい用語です。テーブルは マシンとノード間の負荷分散のために MySQL Cluster クラスタでフラグメント化 (分割) されます。

• レプリカ:

[NDB ストレージ エンジン](#)では、各テーブルのフラグメントは冗長性を提供するために他のデータで保存された多くのレプリカ (コピー) 保持します。現在は、フラグメント毎に 4 つのレプリカがあります。

• トランスポーター:

ノード間のデータ伝送を提供するプロトコルです。MySQL Cluster は現在 4 種類のトランスポーター接続をサポートしています。

• TCP/IP

これは、勿論、馴染みのネットワーク プロトコルでインターネット上の HTTP、FTP (など) を基礎をなすものです。TCP/IP はローカルおよびリモート接続の両方に使用できます。

• SCI

スケーラブル コヒーレント インターフェースはマルチプロセッサ システムおよび並列処理アプリケーションを構築する高速のプロトコルです。SCI を MySQL Cluster で使用するには [SCI ソケットを使用するための MySQL Cluster の設定](#) で説明した特別なハードウェアが必要です。SCI に関する基本的な説明は [this essay at dolphinics.com](#) を参照してください。

• SHM

Unix-style のshared memory セグメントです。サポートされている場合、SHM は同じホストで動作しているノードを自動的に接続するために使用します。この件に関する詳細な情報を得るには[Unix man page for shmop\(2\)](#) が最適です。

注:クラスタのトランスポーターはクラスタに対しては内部的です。MySQL Cluster を使用したアプリケーションは SQL ノードとそれらが他のバージョンの MySQL サーバー (TCP/IP 経由、あるいは Unix のソケット ファイルあるいは Windows パイプ) と同じように通信します。クエリが送られ標準の MySQL クライアント API を使用して結果を取り出します。

• [NDB](#):

これは Network Database のことで、MySQL Cluster を使用するためのストレージ エンジンのことです。[NDB ストレージ エンジン](#)はすべての通常の MySQL データ タイプおよび SQL ステートメントをサポートし、ACID に準拠しています。このエンジンはまたトランザクション (実行およびロールバック) をフル サポートしています。

• 非共有アーキテクチャ:

MySQL Cluster の理想絶的なアーキテクチャです。それはまさに何も共有しない設定で、各ノードは個別のホストで動作します。そのような配列の利点はどのホストやノードもシングル ポイントの不具合あるいは全体としてシステムのパフォーマンスのボトルネックになり得ないということです。

- In-memory ストレージ:

各データ ノードに格納されたすべてのデータはノードのホスト コンピュータのメモリに保持されます。クラスタの各データ ノードには、データベースにレプリカを乗算し、それをデータ ノードで除算した数に相当する RAM 容量を持つ必要があります。このように、データベースのメモリが 1GB で、クラスタを 4 つのレプリカと 8 つのデータ ノードで構成する場合、メモリはノード毎に最低 500MB 必要になります。これはオペレーティング システムおよびホストで動作する他のアプリケーションが必要とするメモリに追加されます。

MySQL 5.1 では、非印インデックス カラムがディスクで保存される ディスク データ も作成でき、このようにクラスタが必要とするメモリ容量を減らすことができます。インデックスおよびインデックスを付したカラムはそのまま RAM に保持されます。[MySQL Cluster ディスク データ ストレージ](#) 参照。

- テーブル:

リレーショナル データベースの場合と同様、用語の「テーブル」は全く同一に構成されたレコードを意味します。MySQL Cluster では、データベースのテーブルは一連のフラグメントとしてデータ ノードに保持され、それらはそれぞれ追加されたデータ ノードでレプリケートされます。データ ノードのセットは同じフラグメントをレプリケートしあるいはフラグメントのセットは ノード グループ として言及されます。

- Cluster のプログラム:

これらは MySQL の動作、設定、管理に使用されるコマンドラインのプログラムです。それらにはサーバーのデーモン (daemons) が含まれます。:

- [ndbd](#):

データ ノード デーモン (データ ノードのプロセスを実行します)

- [ndb_mgmd](#):

マネジメント サーバーデーモン (マネジメント サーバーのプロセスを実行します)

およびクライアント プログラム:

- [ndb_mgm](#):

マネジメント クライアント (マネジメント コマンドを実行するためのインターフェースを提供します)

- [ndb_waiter](#):

クラスタのすべてのノードの状態を検証するために使用します。

- [ndb_restore](#):

バックアップからのクラスタのデータの復旧

これらのプログラムおよびそれらの使用法については [MySQL Cluster のプロセス管理](#) を参照してください。.

- イベント ログ:

MySQL Cluster はイベントをカテゴリ (起動、シャットダウン、エラー、チェックポイントなど)、優先順位、および重要度別のログに記録します。レポートする (記録に残す) すべてのイベントの完全なリストは [MySQL Cluster で生成されたイベント レポート](#) にあります。イベント ログは 2種類あります。

- クラスタ ログ:

クラスタの全体に関するレポートを希望するイベントのレコードを維持します。

- ノード ログ:

各個別のノードを記録した個別のログ

通常環境では、クラスタ ログのみの維持管理が必要でまたそれで十分です。ノード ログはアプリケーションの開発およびデバッグ処理にのみ使用されます。

第15章 パーティショニング

目次

MySQL パーティショニングの概要	996
パーティショニングのタイプ	998
RANGE パーティショニング	999
LIST パーティショニング	1001
HASH パーティショニング	1003
KEY パーティショニング	1006
サブパーティショニング	1007
MySQLパーティショニングの NULL 値の取り扱い	1010
パーティショニング管理	1013
RANGE と LIST パーティションの管理	1014
HASH や KEY パーティションの管理	1019
パーティションのメンテナンス	1019
パーティション情報の取得	1020
パーティションの刈り込み	1022
パーティショニングの制約と制限	1025

コンパイル時の不手際のため、MySQL 5.1.12のバイナリ配布にはNDBクラスタやパーティショニングは含まれていませんでした。ご不便をお掛けし恐縮です。バージョン5.1.14へ更新してください。ソースからコンパイルする場合には、[--with-ndbcluster](#)、[--with-partition](#)オプションとともに[configure](#)を実行して下さい。

この章ではMySQL 5.1 において実装される ユーザ定義パーティショニング について述べています。

パーティショニングの概要およびコンセプトについては [MySQL パーティショニングの概要](#) を参照してください。

MySQLは [パーティショニングのタイプ](#) で述べられている数種類のパーティショニングのほか、[サブパーティショニング](#) で説明されているサブパーティショニングもサポートしています。

既存のパーティショニングされたテーブルへの、パーティション追加、削除、変更に関しては [パーティショニング管理](#) を参照してください。

パーティショニングされたテーブルと使用するテーブルメンテナンスコマンドについては、[パーティションのメンテナンス](#) を参照してください。

重要バージョン 5.1.6 以前の MySQL で作成されたパーティショニングされたテーブルは 5.1.6 版以降の MySQL Server では読み取れません。加えて、[INFORMATION_SCHEMA.TABLES](#)テーブルが5.1.6サーバで存在している場合、使用不可能です。MySQL 5.1.7 以降、サーバによって適合しないパーティショニングされたテーブルの存在を知らせる警告が発せられます。

重要MySQL 5.1.5 以前で作成されたパーティショニングされたテーブルを使用している場合、MySQL 5.1.6 以降にアップグレードする前に 必ず [Changes in release 5.1.6 \(01 February 2006\)](#) を参照して推奨されている追加情報を入力してください。

MySQL 5.1でのパーティショニングの実装はまだ開発途中です。MySQL パーティショニングに関して判明している問題などに関しては、[パーティショニングの制約と制限](#) を参照してください。

また、パーティショニングされたテーブルを使用して作業をこなす際に以下の情報を活用してください。

追加情報

- [MySQL パーティショニングフォーラム](#)

以下は MySQL のパーティショニング技術で研究・実験したいユーザのための公式ディスカッションフォーラムです。MySQL開発者等からの発表や更新を記載しています。このフォーラムはパーティショニング開発・レポートチームによってモニターされています。

- [Mikael Ronström のブログ](#)

MySQL パーティショニングデザイナー兼リード開発者 Mikael Ronström が MySQL のパーティショニングや MySQL クラスタに関する情報を頻繁に掲載・更新しています。

- [PlanetMySQL](#)

MySQL 関連のブログ。MySQL を使用しているユーザにとって有用な情報が記載されています。MySQL のパーティショニング作業を行っているユーザ等が更新するブログへのリンクが記載されていますので、頻繁にチェックするか、自身のブログを追加する場合にリンクにアクセスすることをお勧めします。

MySQL 5.1 バイナリは <http://dev.mysql.com/downloads/mysql/5.1.html> で提供されています。ただし、最新のバグフィックスと追加情報に関しては、BitKeeper 庫からソースを取得できます。パーティショニングを有効化するには、`--with-partition` オプションを使用してサーバをコンパイルしてください。MySQL の構築に関する追加情報には、[ソースのディストリビューションを使用した MySQL のインストール](#) を参照してください。パーティショニングが有効化されている MySQL 5.1 構築をコンパイルする上で問題が発生する場合は、[MySQLパーティショニングフォーラム](#)を参照し、すでに解決策が投稿されていない場合そこでヘルプを要求してください。

MySQL パーティショニングの概要

このセクションでは MySQL 5.1 パーティショニングの概要のコンセプト説明を提供しています。

パーティショニングの制限またはフィーチャーされている限界については [パーティショニングの制約と制限](#) を参照してください。

SQL 基準は実際のデータ保存に関するガイダンスをあまり提供していません。SQL 言語自体、スキーマ、テーブル、行、そしてカラム等の基盤となるデータ構造やメディアとは独立して作動することを目的としています。それにもかかわらず、進んでいるデータベースマネジメントシステムのほとんどが何らかの方法を使ってファイルシステム、ハードウェア、もしくはその両方において特定のデータを保存するための実際のロケーションを決定づける手段を発展させてきました。MySQL では、[InnoDB](#) ストレージエンジンはテーブルスペースの概念をサポートしてきました。MySQL サーバも、パーティショニングが可能になる前から異なる実際のディレクトリを指定して異なるデータベースの保存を行うようコンフィギュアすることが可能でした。(方法に関しては、[シンボリックリンクの使用](#) を参照してください)

Partitioningはこの概念を進化させたもので、必要に応じて大まかに設定したルールに従い、各々のテーブルの一部分をファイルシステム上で分布することを可能にしています。結果的に、テーブルの異なる一部分は異なるテーブルとして別々のロケーションに保存されます。データのパーティショニングをとりおこなうユーザによって選択されたルールは関数と呼ばれ、MySQL では係数、レンジや値のリストに対する照合、内部ハッシュファンクション、もしくはリニアハッシュファンクションになります。関数はユーザによって指定されたパーティショニングの種類によって選択され、ユーザによって提供された表現の値をパラメータとして取り入れます。この表現は整数カラム値、もしくは一つまたは複数のカラム値に対して働く関数が整数を返していることが考えられます。この表現の値はパーティショニング関数へ渡され、その特定のレコードが記憶されるべきパーティションを示す整数値を返します。この関数はコンスタントでもランダムであっていいかもしれません。クエリを含んでいないかもしれませんが、実質MySQL上有効などのSQL表現も使用可能です。ただし、[MAXVALUE](#) (可能な最大のポジティブ整数)未満のポジティブ整数を返す表現である必要があります。パーティショニング関数の例はこの章の後部に登場するパーティショニング種類のディスカッション ([パーティショニングのタイプ](#) を参照してください) のほかに、パーティショニング構文の記述に含まれています ([CREATE TABLE 構文](#) を参照してください) 。

これは 水平パーティショニング ー と称されていて、テーブル内の異なる行が異なる実際のパーティショニングに割り当てられることが有ります。MySQL 5.1 は、テーブル内の異なるカラムが実際に異なるパーティショニングに割り当てられる、垂直パーティショニングをサポートしていません。現在、垂直パーティショニングをMySQL 5.1に組み込む予定はありません。

パーティショニングのサポートはMySQL 5.1の-max版に含まれています(つまり、5.1 -maxバイナリは--with-partitionで構築されます。MySQL バイナリがパーティショニングサポートで構築されている場合、有効化するための追加作業は必要ありません (例えば、[my.cnf](#)ファイルへの特殊エントリは要求されません。))ユーザのMySQLサーバがパーティショニングをサポートしているか否かは、以下のような[SHOW VARIABLES](#)コマンドを使用して確認することができます。

```
mysql> SHOW VARIABLES LIKE '%partition%';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
```

```
| have_partitioning | YES |
+-----+-----+
1 row in set (0.00 sec)
```

妥当な `SHOW VARIABLES` の出力上に上記のように `YES` 値が `have_partitioning` 変数と現れない場合、ユーザの MySQL はパーティショニングをサポートしていません。

MySQL 5.1.6以前では、この変数は `have_partition_engine` と名づけられていました。(Bug #16718)

パーティショニングされたテーブルを作成するには、MySQLサーバにサポートされるほとんどの記憶エンジンを使用することができます。MySQLパーティショニングエンジンは別の層で作動しており、これらのどのエンジンとも対話できます。MySQL 5.1では、同じパーティショニングされたテーブルのパーティショニングは同じ記憶エンジンを使用していなければいけません。例えば、`MyISAM`を1つのパーティションに使い、別のパーティションに`InnoDB`を使用することはできません。ただし、同じMySQLサーバもしくは同じデータベース上で異なるパーティショニングされたテーブルに異なる記憶エンジンを使用することを阻むものではありません。

注:MySQL パーティショニングは`MERGE`もしくは`CSV`記憶エンジンと使うことはできません。MySQL 5.1.15に始まり、`FEDERATED`テーブルもパーティショニングできません (Bug #22451)。MySQL 5.1.6より前では、`BLACKHOLE`記憶エンジンを使用してパーティショニングされたテーブルを作ることは不可能でした (Bug #14524)。`KEY` を使用してのパーティショニングは `NDBCluster` 記憶エンジンを使用することでサポートされていますが、MySQL 5.1 内のクラスターテーブルでは他の種類のユーザによって定義されるパーティショニングはサポートされません。

パーティショニングテーブルに特定の記憶エンジンを使用するには、非パーティショニングされたテーブルでそうするように`[STORAGE] ENGINE`のみ使用する必要があります。ただし、`[STORAGE] ENGINE` (と他のテーブルオプション) はパーティショニングオプションが`CREATE TABLE`ステートメントで使用される前に記述される必要があります。この例はハッシュを使って6のパーティショニングに分けられ、`InnoDB`記憶エンジンを使用しているテーブルの作成方法を示しています。

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
ENGINE=INNODB
PARTITION BY HASH( MONTH(tr_date) )
PARTITIONS 6;
```

(各 `PARTITION` 節は `[STORAGE] ENGINE` オプションを含むことはできますが、MySQL 5.1 では効果がありません。)

注:パーティショニングはテーブルの全データとインデックスに対して適用されます。インデックスのみ、またはvice versaテーブルの1部分のみをパーティショニングすることはできません。

各パーティションのデータやインデックスは、パーティショニングされたテーブル作成のために使用される`CREATE TABLE`ステートメントの `partitioning` 節の`DATA DIRECTORY`と`INDEX DIRECTORY`オプションを使用して、特定のディレクトリに割り当てることができます。それに加え、`MAX_ROWS`と`MIN_ROWS`は各パーティショニングに記憶可能な最大と最低の数値を割り出すことができます。これらのオプションについては、[パーティショニング管理](#) を参照してください。

パーティショニングの利点にはこういったものが含まれます。

- 1つのディスクもしくはファイルシステムのパーティションで記憶できる量よりも多くのデータをテーブルで記憶することができます。
- 有用性を失うデータはそのデータのみを含んでいるパーティションを消去することでテーブルから取り除くことができます。また逆に、新しいデータを追加するプロセスは、そのデータ特有のパーティションを用意することで大きく援助することができる場合があります。
- ある特定の`WHERE`節を満たすデータが1つまたは1つ以上のパーティションのみで記憶されることによって、いくつかのクエリは最適化されます。これにより、検索から残りのパーティションを除外します。パーティショニングされたテーブル作成後にパーティショニングを変更できるため、パーティショニングスキームが最初に設定された当初は現れなかった頻繁なクエリを促進させるためにデータを再編成することができます。この機能は `パーティションの刈り込み` と呼ばれており、MySQL 5.1.6で実装されました。追加情報に関しては [パーティションの刈り込み](#) を参照してください。

他のパーティショニングによるベネフィットが次のリストの記述されています。これらの特徴は現MySQLパーティショニングには実装されていませんが、優先事項のトップにあります。

- `SUM()` や `COUNT()` といった集約関数を含むクエリは、簡単に並列化させることができます。そのようなクエリの単純な例：`SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id;`。「並列化」とは、各パーティションで同時にクエリを作動させることが可能で、且つ最終結果は全パーティションの結果の合計として取得することができるという意味を含んでいます。
- 複数のディスク上でデータシークを広めるために、さらに優れたクエリ処理能力を取得できます。

パーティショニングの開発は続いているので、引き続きこのセクションと章をチェックしてください。

パーティショニングのタイプ

このセクションではMySQL 5.1で提供されているパーティショニングのタイプについて記述されています。これらは：

- **RANGE** パーティショニング:あるレンジに当てはまるカラム値に対するパーティションに行を割り当てます。[RANGE パーティショニング](#) を参照してください。
- **LIST** パーティショニング:レンジによるパーティショニングと似ていますが、離散的値と合致するカラムに対してパーティションが選択されます。[LIST パーティショニング](#) を参照してください。
- **HASH** パーティショニング:テーブルに挿入される行のカラム値に作用する、ユーザによって定義された表現が返す値に対してパーティションが選択されます。関数は、MySQL内で非ネガティブ整数値を生み出す有効な表現で構成されます。[HASH パーティショニング](#) を参照してください。
- **KEY** パーティショニング:ハッシュによるパーティショニングと似ていますが、評価されるひとつか1つ以上のカラムが提供され、MySQL サーバは自身のハッシュ関数を提供している。MySQL に提供されるハッシュ関数はカラムデータタイプに左右されない整数の値を約束するため、これらのカラムは整数値以外の値を含むことができません。[KEY パーティショニング](#) を参照してください。

非常に一般的なデータベースパーティショニングは、日付によってデータを分けることで行われます。いくつかのデータベースシステムは明確なデータパーティショニングをサポートしています。これは、MySQL 5.1では実装されません。ただし、MySQLで**DATE**、**TIME**、または**DATETIME**カラムを使用して、もしくはそれらのカラムを使用してできた表現をもとにパーティショニングされたスキーマを作成することは難しくありません。

KEY または**LINEAR KEY** を使用してパーティショニングする場合、**DATE**、**TIME**、または**DATETIME** カラムを、カラム値の改良をすることなくパーティショニングカラムとして使用することができます。例えば、このテーブル作成ステートメントはMySQLにおいて完全に有効です。

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

しかし、MySQL の他のタイプのパーティショニングは整数値もしくは **NULL** を生み出すパーティショニング表現が要求されます。**RANGE**、**LIST**、**HASH** または **LINEAR HASH** を使用して日付によるパーティショニングを使用する場合は、**DATE**、**TIME**、または **DATETIME** カラムで作動し、且つ以下に示されるようにそのような値を返す関数を使用できます。

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
  PARTITION p0 VALUES LESS THAN (1960),
  PARTITION p1 VALUES LESS THAN (1970),
  PARTITION p2 VALUES LESS THAN (1980),
  PARTITION p3 VALUES LESS THAN (1990),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

日付を使用してのパーティショニングの例はここでも紹介されています。

- [RANGE パーティショニング](#)
- [HASH パーティショニング](#)
- [LINEAR HASH パーティショニング](#)

さらに複雑な日付を使用してのデータベースパーティショニングに関しては、以下を参照してください。

- [パーティションの刈り込み](#)
- [サブパーティショニング](#)

MySQL パーティショニングは `TO_DAYS()` と `YEAR()` 関数での使用に対して最適化されています。ただし、整数や `NULL` を返す日付・時間関数を使用できます。たとえば、`WEEKDAY()`、`DAYOFYEAR()`、または `MONTH()` を使用することができます。詳細については、[日付時刻関数](#) を参照してください。

— は、使用されているパーティショニングの種類によらず、— は重要で、パーティショニングは常に 0 に始まり自動的且つシーケンスにしたがって作成されます。新しい行がパーティショニングされたテーブルに挿入された時、正しいパーティションを識別するのに使用されるのはこれらのパーティション番号です。例えば、ユーザのテーブルが4つのパーティションを使用している場合、これらのパーティションには 0、1、2、そして 3 と番号付けされます。`RANGE` と `LIST` パーティショニング型に関しては、各パーティション番号ごとにパーティションが定義されることが必要です。`HASH` パーティショニングに関しては、使用されているユーザ関数は 0 より大きい整数の値を返さなければいけません。`KEY` パーティショニングでは、MySQL サーバが内部で使用しているハッシュ関数によってこの問題は自動的に対処されることになります。

パーティションの名前は、一般的に他のMySQL 識別子を支配するルールに沿っています。例えば、テーブルやデータベースのそれと同じように扱われます。ただし、パーティションの名前は大文字・小文字によって区別されないの、注意してください。例えば、以下の `CREATE TABLE` ステートメントは失敗します。

```
mysql> CREATE TABLE t2 (val INT)
-> PARTITION BY LIST(val)(
-> PARTITION mypart VALUES IN (1,3,5),
-> PARTITION MyPart VALUES IN (2,4,6)
-> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

これは `mypart` と `MyPart` の違いをMySQLが察知できないために失敗します。

このテーブルのパーティションの数を指定する時、それは正の値であって、ゼロで始まらない、ゼロではない整数文字であり、`0.8E+01` や `6-2` は整数として成立しても使用することはできません。(MySQL 5.1.12からは、小数点の分数は切り捨てられるのではなく、完全に使用不可能となりました。)

続くセクションでは、各パーティショニング型を作成するためのあらゆる構文を提供しているわけではありません。追加情報は、[CREATE TABLE 構文](#) を参照してください。

RANGE パーティショニング

レンジによってパーティショニングされたテーブルは、特定のレンジにおいて行のパーティショニング表現値が置かれるように、パーティショニングされます。レンジは連続していますがかぶることは無く、`VALUES LESS THAN` 演算子を使用して定義されます。次のいくつかの例では、ユーザが20のビデオレンタル店の個人情報を含む1から20のテーブルを作成しているとします。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);
```

このテーブルは、ユーザのニーズによってさまざまな方法でレンジによるパーティショニングを行うことができます。1つには、`store_id` カラムを使用した方法があります。たとえば、`PARTITION BY RANGE` 節を使用して、4方法でテーブルをパーティショニングすると決めたとします。


```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
  PARTITION p0 VALUES LESS THAN (6),
  PARTITION p1 VALUES LESS THAN (11),
  PARTITION p2 VALUES LESS THAN (16),
  PARTITION p3 VALUES LESS THAN (21)
);
```

このパーティショニングスキーマでは、**p0** パーティションに記憶されるのは店舗1-5で働く店員を含む行で、**p1** パーティションに記憶されるのは店舗6-10の店員になります。各パーティションは小から大まで順番どおりにパーティションが分けられていることに注目してください。これは **PARTITION BY RANGE** 構文の要求です。**switch ... case**、**C**、**Java**などと同義であると考えていいでしょう。

(72, 'Michael', 'Widenius', '1998-06-25', NULL, 13) データを含む新しい行が **p2** パーティションに挿入されていることは断定できますが、チェーンに 21 番目の店舗が追加された時はどうでしょう。このスキーマでは、**store_id** が20よりも大きい行をカバーしているというルールが無いため、サーバが何処に情報を置くかを判断しかねるため、エラーが発生します。「キャッチオール」**VALUES LESS THAN** 節を **CREATE TABLE** ステートメントで使用するによって、明確に挙げられる最高値よりも高い値全てに対応することができます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
  PARTITION p0 VALUES LESS THAN (6),
  PARTITION p1 VALUES LESS THAN (11),
  PARTITION p2 VALUES LESS THAN (16),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

MAXVALUE はありうる最高の整数値を表しています。**store_id** カラム値が16かそれ以上のものは、(定義される最高値) **p3** パーティションに記憶されます。いずれ将来、— 25、30、もしくはさらに店舗が増えた時、— **ALTER TABLE** ステートメントを使用して21-25、26-30の店舗のために新しいパーティションを作成することができます。(方法に関しては、[パーティショニング管理](#) を参照してください。)

同様に、雇用者コードに合わせてテーブルをパーティションすることができます。—それは、**job_code**カラム値によるレンジで可能になります。例えば、— 二桁雇用コードがストア店員、3桁コードが事務・サポート員に、そして4桁コードが基幹職を示している場合、—以下を使用してパーティショニングされたテーブルを作成することができます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
  PARTITION p0 VALUES LESS THAN (100),
  PARTITION p1 VALUES LESS THAN (1000),
  PARTITION p2 VALUES LESS THAN (10000)
);
```

この場合、全てのストア店員に関する行は **p0** パーティションで記憶され、事務・サポート要因に関する行は **p1**、そして基幹職は **p2** となります。

他にも **VALUES LESS THAN** 節で表現を使用することができます。ただし、MySQLは表現の返される値を **LESS THAN (<)** 比較の一部として評価できることが前提となります。

店舗番号ごとにテーブルデータをパーティショニングするよりも、2つのうち1つの **DATE** カラムの表現を使用することができます。例えば、社員が会社を辞めた年度ごとにパーティショニングするとします。それは、**YEAR(separated)** の値となります。そのようなパーティショニングされたスキーマを実装する **CREATE TABLE** ステートメントの例がここに記されています。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1996),
  PARTITION p2 VALUES LESS THAN (2001),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

このスキーマでは、1991より前に辞めた社員に限って、行は **p0** パーティションで記憶されています。1991から1995の間に辞めた社員は **p1**、1996 から2000の間に辞めた社員は **p2** そして2000以降に辞めた社員は **p3** で記憶されています。

レンジパーティショニングは以下のような時に特に有用です。

- 「old」 データを呼び出したい、もしくは削除したい。上記で記されているパーティショニングされたスキーマを使用している場合、**ALTER TABLE employees DROP PARTITION p0;** 1991より前に辞めた全ての従業員に関する行を削除することができます。(詳細については、**ALTER TABLE 構文** と **パーティショニング管理** を参照して下さい。)行の非常に多いテーブルに関しては、**DELETE FROM employees WHERE YEAR(separated) <= 1990;** といったような **DELETE** クエリを使用して効率よく作業を行うことができます。
- 日付や時間の値、もしくは他のシリーズからなる値を含むカラムを使用したい場合。
- テーブルパーティショニングのために使用されるカラムに直接従属するクエリを頻繁に使用する。例えば、**SELECT COUNT(*) FROM employees WHERE YEAR(separated) = 2000 GROUP BY store_id;** といったクエリを実行する場合、MySQLは **p2** パーティションのみスキャンする必要があると断定します。これは残りのパーティションが **WHERE** 節を充たすレコードを含むことができないからです。これがどのように達成されるかについては、**パーティションの刈り込み** を参照してください。

LIST パーティショニング

MySQLのLISTパーティショニングは多くの点でRANGEパーティショニングに似ています。**RANGE** によるパーティショニング同様、各パーティションは明確に定義されていなければいけません。決定的な違いは、リストパーティショニングでは、隣接する値のレンジ内のセットの1つとしてではなく、各パーティションは1セットの値のリストの中のカラム値メンバーシップによって定義・選択されます。これは **PARTITION BYLIST (expr)** によって実行されます。その時 **expr** はカラム値もしくは返される整数値に基づいたカラム値や表現であり、**VALUES IN (value_list)** の **value_list** はカンマによって分けられた整数のリストになります。

注:MySQL 5.1では、**LIST** によるパーティショニングを行う時、整数のリストのみに対して照合(そして **NULL** —MySQLパーティショニングの **NULL 値の取り扱い** を参照してください)をすることが可能です。

パーティショニングがRANGEによって定義されたケースとは異なり、リストパーティショニングは特定の順番で宣言される必要はありません。さらに詳しい構文に関する情報については、**CREATE TABLE 構文** を参照してください。

以下の例では、パーティショニングされるテーブルの基本的な定義は **CREATE TABLE** ステートメントによって提供されているものとします。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
```

```
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
);
```

(これは **RANGE パーティショニング** の例として使用されるテーブルと同様のものです。)

例えば、以下のテーブルの様に20のビデオレンタル店が4つのフランチャイズで分布されているとします。

地域	店舗ID
北	3, 5, 6, 9, 17
東	1, 2, 10, 11, 19, 20
西	4, 12, 13, 14, 18
中央	7, 8, 15, 16

同じ地域の店舗を示す行が同パーティションに含まれるようテーブルをパーティショニングする場合は、以下の様に **CREATE TABLE** ステートメントを使用することができます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY LIST(store_id) (
  PARTITION pNorth VALUES IN (3,5,6,9,17),
  PARTITION pEast VALUES IN (1,2,10,11,19,20),
  PARTITION pWest VALUES IN (4,12,13,14,18),
  PARTITION pCentral VALUES IN (7,8,15,16)
);
```

これによって、テーブルから特定の地域の店舗の従業員情報を簡単に追加・削除することが可能になります。例えば、西地域の全店舗が別の会社に売られたとします。その地域で雇用されている従業員を示す全ての行は **ALTER TABLE employees DROP PARTITION pWest**; クエリを使用して削除することができ、これは同等の **DELETE** ステートメントの **DELETE FROM employees WHERE store_id IN (4,12,13,14,18)**; よりはるかに効率よく実行することができます。

RANGE や **HASH** パーティショニングのように、**NULL** が整数ではない値を持つカラムでテーブルをパーティショニングする場合、そのような値を返すカラムに基づいてパーティショニング表現を使用する必要があります。例えば、以下のように従業員データを含むテーブルが記されていたとします。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code CHAR(1),
  store_id INT
);
```

この **employees** テーブルのバージョンでは、雇用コードは数値ではなく文字になります。各文字が特定の職種に対応しており、同様の職種についている、もしくは同じ職場の従業員は同じパーティションに含まれるよう、以下のスキーマでテーブルをパーティショニングしたいとします。

職種・ 職場	雇用コード
管理	D, M, O, P
営業	B, L, S
技術	A, E, G, I, T
事務	K, N, Y
サポート	C, F, J, R, V

割り振りなし

「空」

値のリストでキャラクタ値がしようできないため、これらを整数もしくは **NULL** 等に変換する必要があります。このため、**ASCII()** 関数をカラム値に使用することができます。加えて、— 異なる時間帯、ロケーションでの異なるアプリケーションの使用により、— これらのコードは大文字、もしくは小文字になりえ、「割り振られていない」を示す「空」値は、**NULL**、空の文字列、もしくはスペースを表しているかもしれません。このスキーマを実装しているパーティショニングされたテーブルが以下に示されています。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code CHAR(1),
  store_id INT
)
PARTITION BY LIST(ASCII( UCASE(job_code) )) (
  PARTITION management VALUES IN(68, 77, 79, 80),
  PARTITION sales VALUES IN(66, 76, 83),
  PARTITION technical VALUES IN(65, 69, 71, 73, 84),
  PARTITION clerical VALUES IN(75, 78, 89),
  PARTITION support VALUES IN(67, 70, 74, 82, 86),
  PARTITION unassigned VALUES IN(NULL, 0, 32)
);
```

評価式がパーティションのリストに与える値として許可されていないため、照合される文字に関しては文字列ではなく ASCII コードをリストしなければいけません。**ASCII(NULL)** が **NULL** を返すことに注意してください。

重要もしリストに含まれないカラム値 (もしくはパーティショニング表現が返す値) に行を挿入しようとした場合、**INSERT** クエリはエラーを表示し、失敗に終わります。例えば、先ほど概要の説明がされた **LIST** リストパーティショニングスキーマでは、このクエリは失敗します。

```
INSERT INTO employees VALUES
(224, 'Linus', 'Torvalds', '2002-05-01', '2004-10-12', 'Q', 21);
```

失敗は、81 (大文字 'Q' ASCII のアスキーコード) がパーティションを定義する値のリストに含まれていないためおこります。値のリストに含まれない値を承認する、**VALUES LESS THAN(MAXVALUE)** と同義のパーティションリストの「キャッチオール」定義は存在しません。つまり、照合する全ての値は、値のリスト中に存在していなければいけません。

RANGE パーティショニングと同様に、**LIST** パーティショニングとキー、ハッシュパーティショニングを合わせることで合成パーティショニングを生成できます(サブパーティショニング)。**サブパーティショニング** を参照してください。

HASH パーティショニング

HASH によるパーティショニングは前もって決められた数のパーティショニングの中でデータを均等に割り振るために使用されます。レンジやリストパーティショニングでは、どのパーティションにカラム値やカラム値のセットが記憶されるか特定しなければいけません。ハッシュパーティショニングでは、MySQL がこれを自動的に実行してくれるため、ハッシュされるカラム値に対してカラム値や表現と、パーティショニングされたテーブルのパーティションの数だけ特定すれば事足ります。

HASH パーティショニングを使用してテーブルをパーティショニングする場合、**CREATE TABLE** ステートメントに **PARTITION BY HASH (expr)** 節を付加する必要があります。この時、**expr** は整数を返す表現です。これは単純に、MySQL の整数タイプと同様のタイプのカラムの名前でけっこうです。加えて、**PARTITIONS num** 節で続かせるのが定石です。この時、**num** はパーティショニングされたテーブルのパーティションの数を表現するネガティブ値ではない整数になります。

例えば、以下のステートメントは **store_id** カラムに対してハッシュを行い、4つのパーティションに分かれるテーブルを作成します。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
```

```
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

PARTITIONS 節を含めない場合、パーティションの数はデフォルトで 1 となります。

PARTITIONS キーワードの続きに数字が使用されていない場合、構文エラーとなります。

expr に対して整数を返す SQL 表現を使用することもできます。例えば、雇用年度に対してパーティショニングを行いたいとします。以下のようにできます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

非定数、非ランダム整数値を返すという条件をクリアしていれば、MySQL で有効な **expr** に対してどの関数でも表現でも使用することができます。(言い換えれば、多様性があっても、断言的でなければいけません。)ただし、行が挿入もしくは更新 (削除) されるたび、この表現が評価されます。これは、特に多数の行に影響を及ぼすオペレーション (例えばバッチ挿入) を実行する際、きわめて複雑な表現はパフォーマンス問題を浮上させる可能性があります。

最も効率的なハッシュ関数では、単一のテーブルカラムに対して実行され、その値がカラム値に比例して増大、減少します。これにより、パーティションのレンジを「pruning」することができます。つまり、表現が基となるカラムの値に対して比例すればするほど、MySQL はその表現をハッシュパーティショニングにその分だけ効率よく用いることができます。

たとえば、**date_col** が **DATE** のカラム型である場合、表現 **TO_DAYS(date_col)** は **date_col** の値に対して直接比例します。これは **date_col** の値が変更されるたびに、表現の値が一定の割合で変更されるからです。**YEAR(date_col)** 表現の **date_col** に対する変化は、**TO_DAYS(date_col)** に対する変化と比べて直接的ではありません。これは、**date_col** 内の変化の全てが **YEAR(date_col)** に対して同等の変更を促すとは限らないからです。それでも、**YEAR(date_col)** はハッシュ関数の優れた候補となります。**date_col** の一部と直接比例する上、**date_col** 内には **YEAR(date_col)** に対して不均衡な変化を促すものが無いからです。

逆に、**INT** 型を持つ **int_col** というカラムがあるとします。この表現 **POW(5-int_col,3) + 6** を検討してください。これはハッシュ関数に使用するには優れた候補とはいえません。なぜなら、**int_col** の値に変化がおきた時、値の表現に対して比例する変化が起きる保証がないからです。**int_col** の値を変更すれば、表現の値にもさまざまな変化を促します。例えば **int_col** を 5 から 6 に変えると、表現の値に -1 という変化をもたらしますが、**int_col** の値を 6 から 7 に変えると、表現の値に -7 という変化をもたらします。

言い換えると、グラフのカラム値に対して versus 表現の値が $y=nx$ この時 n は 0 以外の定数という条件のもと直線をなぞるほど、その表現はハッシュにふさわしいものになります。これは、表現が非直線状的であればあるほど、パーティション内で割り振られるデータが不均衡になりがちであることと関係しています。

セオリーでは、「刈り込み」は 1 つ以上のカラム値を含む表現にも使用できますが、ふさわしい表現の特定は難しい上に、多くの時間を要します。このため、複数のカラムに対してハッシュ表現を使用することは推奨できません。

PARTITION BY HASH が使用される時 MySQL は **num** どのパーティションが使用されるかを、ユーザ関数の結果係数に基づいて断定します。言い換えれば、**expr** の表現に対して、レコードが記憶されたパーティションの番号は **N** であり、 $N = \text{MOD}(\text{expr}, \text{num})$ となります。例えば、テーブル **t1** が以下の様に定義され、4 のパーティションがあるとします。

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY HASH( YEAR(col3) )
PARTITIONS 4;
```

t1 にレコードを挿入し、**col3** 値が '2005-09-15' の場合、それが記憶されるパーティションは以下の様に断定されます。

```
MOD(YEAR('2005-09-01'),4)
= MOD(2005,4)
= 1
```

MySQL 5.1 は [HASH](#) パーティショニングの変形である、linear hashing もサポートしており、これはパーティショニングされたテーブルに新しい行を配置する際にさらに複雑なアルゴリズムを使用します。詳細については、[LINEAR HASH パーティショニング](#) を参照してください。

レコードが挿入もしくは更新されるたびに、ユーザ関数は評価されます。それは — 状況によって、— レコードが消去される際に評価されることもあります。

注:パーティショニングされるテーブルに [UNIQUE](#) キーがある場合、[HASH](#) ユーザ関数や [KEY](#) の [column_list](#) に対してアークギュメントとして提供されたカラムは、そのキーの一部出なければいけません。例外:この制限は [NDBCluster](#) 記憶エンジンを使用しているテーブルには当てはまりません。

LINEAR HASH パーティショニング

MySQL はリニアハッシュもサポートします。リニアハッシュが通常のハッシュと異なるところは、ハッシュがハッシュ関数値の係数を使用するところ、リニアハッシュはリニア二乗アルゴリズムを使用します。

構文的に、リニアハッシュパーティショニングと通常のハッシュパーティショニングの唯一の違いは、以下に示されるよう、[PARTITION BY](#) 節内の [LINEAR](#) キーワードの追加です。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;
```

[expr](#) の表現に対して、リニアハッシュが使用される際にレコードが記憶されるパーティションは [num](#) パーティション内の [N](#) となります。この時、[N](#) は以下のアルゴリズムにより派生します。

1. [num](#) よりも大きい二乗を探してください。この値を [V](#) と称します。以下の様に計算することができます。

```
V = POWER(2, CEILING(LOG(2, num)))
```

(たとえば、[num](#) が 13 とします。そうすると [LOG\(2,13\)](#) は 3.7004397181411 になります。 [CEILING\(3.7004397181411\)](#) は 4 となり、[V = POWER\(2,4\)](#)、は 16 となります。)

2. セット [N](#) = [F\(column_list\)](#) & ([V](#) - 1)。

3. [N](#) >= [num](#) の場合

- [V](#) = [CEIL\(V / 2\)](#) とセットしてください
- [N](#) = [N](#) & ([V](#) - 1) とセットしてください

たとえば、リニアハッシュパーティショニングをして6つのパーティション分かれていたテーブル [t1](#) が、以下のステートメント使用して作成されたとします。

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR HASH( YEAR(col3) )
PARTITIONS 6;
```

[col3](#) カラム値 '2003-04-14' と '1998-10-19' を持つ [t1](#) に 2 つのレコードを挿入したいとします。最初のパーティション番号は以下のように決定されます。

```
V = POWER(2, CEILING( LOG(2,7) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
  = 2003 & 7
  = 3
```

```
(3 >= 6 is FALSE: record stored in partition #3)
```

2番目のレコードが記憶されているパーティションの番号は以下の様に計算されます。

```
V = 8
N = YEAR('1998-10-19') & (8-1)
  = 1998 & 7
  = 6

(6 >= 6 is TRUE: additional step required)

N = 6 & CEILING(8 / 2 - 1)
  = 6 & 3
  = 2

(2 >= 6 is FALSE: record stored in partition #2)
```

リニアハッシュによるパーティショニングの利点は、パーティションの追加、削除、結合、そして分裂のスピードアップが図れることです。これは、大量のデータ（テラバイト級）を含むテーブルを取り扱う際に、効果的です。欠点は、通常のハッシュパーティショニングを使用した時に比べデータがパーティションの間で不均等に割り振られていることがあります。

KEY パーティショニング

キーによるパーティショニングはハッシュによるパーティショニングと似ていますが、ハッシュパーティショニングがユーザによって定義された表現を使用するところ、キーパーティショニングのハッシュ関数は MySQL サーバによって提供されます。MySQL クラスタはこのために [MD5\(\)](#) を使用します。他のストレージエンジンを使用しているテーブルには、サーバは自身の [PASSWORD\(\)](#) アルゴリズムに基づいた内部ハッシュ関数を使用します。

[CREATE TABLE ... PARTITION BY KEY](#) の構文ルールは、ハッシュによってパーティショニングされたテーブルを作成するものと類似しています。最大の相違点は

- **HASH** よりも **KEY** が使用されることにあります。
- **KEY** は 1 以上のカラム名のみとります。MySQL 5.1.5に始まり、パーティショニングキーとして使用されるカラムはテーブルのプライマリキーの一部もしくは前部を構成しなければいけません。これは、テーブルにプライマリキーがある場合のみです。

MySQL 5.1.6に始まり、**KEY** は 0 以上のカラム名をとります。パーティショニングキーとしてカラム名が特定されていない場合、存在する場合に限ってテーブルのプライマリキーが使用されます。たとえば、以下の [CREATE TABLE](#) ステートメントはMySQL 5.1.6以降有効です。

```
CREATE TABLE k1 (
  id INT NOT NULL PRIMARY KEY,
  name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 2;
```

プライマリキーが無くユニークキーがある場合、パーティショニングキーにユニークキーが使用されます。

```
CREATE TABLE k1 (
  id INT NOT NULL,
  name VARCHAR(20),
  UNIQUE KEY (id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

ただし、**NOT NULL** としてユニークキーカラムが定義されていない場合、先のステートメントは失敗に終わります。

両方の場合、パーティショニングキーは **id** カラムになります。これは [SHOW CREATE TABLE](#) や [INFORMATION_SCHEMA.PARTITIONS](#) テーブルの [PARTITION_EXPRESSION](#) カラムで記されていなくても同じです。

他のパーティショニングタイプと違い、**KEY** によるパーティショニングに使用されたカラムは整数や **NULL** 値に制限されません。例えば、以下の [CREATE TABLE](#) ステートメントは有効です。

```
CREATE TABLE tm1 (
```



```
s1 CHAR(32) PRIMARY KEY
)
PARTITION BY KEY(s1)
PARTITIONS 10;
```

他のパーティショニングの種類が指定された場合、先のステートメントは有効ではありません。(注:この場合、単純に `PARTITION BY KEY()` を使用すれば、有効である上 `PARTITION BY KEY(s1)` と同等の効果となります。これは、`s1` がテーブルのプライマリキーとなるからです。

この件の詳細については [パーティショニングの制約と制限](#) を参照してください。

注:MySQL 5.1.6に始まり、`NDB Cluster` ストレージエンジンを使用しているテーブルは、テーブルのプライマリキーをパーティショニングキーとして、`KEY` によって暗黙にパーティショニングされています。クラスターテーブルに明確にプライマリキーがない場合、各クラスターテーブルの `NDB` ストレージエンジンによって生成された「hidden」プライマリキーがパーティショニングキーとして使用されます。

重要`NDB Cluster` 以外のMySQLストレージ エンジンを使用しているキーパーティショニングされたテーブルは、`ALTER TABLE DROP PRIMARY KEY` を実行することができません。なぜなら、実行した場合は以下のエラーテキストが現れるからです: `ERROR 1466 (HY000): Field in list of fields for partition function not found in table`。これは `KEY` によってパーティショニングされたMySQLクラスターテーブルにとっては問題になりません。その場合、「hidden」プライマリキーをテーブルの新しいパーティショニングキーとしてテーブルが再構築されます。[14章MySQL Cluster](#) を参照してください。

リニアキーを使用してテーブルをパーティショニングすることもできます。ここに単純な例を記します。

```
CREATE TABLE tk (
  col1 INT NOT NULL,
  col2 CHAR(5),
  col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

`LINEAR` を使用するのには `KEY` パーティショニングに対しても `HASH` パーティショニングに対しても同様の効果をもたらします。この時パーティショニングはモジュール算術よりも二乗アルゴリズムを使用してパーティショニング番号が派生されます。アルゴリズムとその意味合いについては、[LINEAR HASH パーティショニング](#) を参照してください。

サブパーティショニング

サブパーティショニング、— もしくは 複合パーティショニング — はパーティショニングされたテーブルのパーティションをさらに分けることを指します。例えば、以下の `CREATE TABLE` ステートメントを検討してください。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) )
SUBPARTITIONS 2 (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

テーブル `ts` は3つの `RANGE` パーティショニングを含んでいます。これらの各パーティション — `p0`、`p1`、そして `p2` — はさらに2つのサブパーティションに分けられます。結果的に、テーブル全体が $3 * 2 = 6$ パーティションに分けられます。ただし、`PARTITION BY RANGE` 節の作動によって、最初の2つは `purchased` カラムで 1990 の値より少ないレコードのみが記憶されます。

MySQL 5.1 では、`RANGE` や `LIST` によってパーティショニングされたテーブルをさらにサブパーティショニングすることが可能です。サブパーティショニングは `HASH` または `KEY` パーティショニングを用いることがあります。これは、複合パーティショニング と呼ばれます。

`SUBPARTITION` 節を使用して各々のサブパーティショニングのオプションを特定することで、サブパーティションを定義することができます。例えば、以前の例通りの `ts` を回りくどく作成する場合。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
```

```
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2,
    SUBPARTITION s3
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4,
    SUBPARTITION s5
  )
);
```

構文的な注意点：

- 各パーティションは同じ数のサブパーティションを擁していなければいけません。
- もしパーティショニングされたテーブルにおいて **SUBPARTITION** を使用して明示的にサブパーティションを定義した場合、残る全てのパーティションにおいてもサブパーティションを定義しなければいけません。言い換えれば、以下のステートメントは失敗します。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s2,
    SUBPARTITION s3
  )
);
```

このステートメントは、**SUBPARTITIONS 2** 節を含んでいたとしても失敗します。

- 各 **SUBPARTITION** 節は(最低でも) サブパーティションの名称を含んでいなければいけません。でなければ、サブパーティションに要求どおりのオプションを設定するか、そのオプションのデフォルト設定にもどします。
- MySQL 5.1.7 以前では、サブパーティションの名称は各パーティション内ではユニークである必要がありました。が、テーブル全体の中でユニークである必要はありませんでした。MySQL 5.1.8 に始まり、サブパーティション名称はテーブル全体においてユニークであることが必要になりました。たとえば、以下の **CREATE TABLE** ステートメントはMySQL 5.1.8以降有効です。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2,
    SUBPARTITION s3
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4,
    SUBPARTITION s5
  )
);
```

(以前のステートメントは MySQL 5.1.8. 以前でも有効です。)

サブパーティショニングはデータやインデックスを複数のディスク上分布するために特に大きなテーブルと使用することができます。例えば、**/disk0**、**/disk1**、**/disk2** とつづく6つのディスクを重ねていたとします。以下の例を検討してください。

```
CREATE TABLE ts (id INT, purchased DATE)
```

```

PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0
      DATA DIRECTORY = '/disk0/data'
      INDEX DIRECTORY = '/disk0/idx',
    SUBPARTITION s1
      DATA DIRECTORY = '/disk1/data'
      INDEX DIRECTORY = '/disk1/idx'
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2
      DATA DIRECTORY = '/disk2/data'
      INDEX DIRECTORY = '/disk2/idx',
    SUBPARTITION s3
      DATA DIRECTORY = '/disk3/data'
      INDEX DIRECTORY = '/disk3/idx'
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4
      DATA DIRECTORY = '/disk4/data'
      INDEX DIRECTORY = '/disk4/idx',
    SUBPARTITION s5
      DATA DIRECTORY = '/disk5/data'
      INDEX DIRECTORY = '/disk5/idx'
  )
);

```

この場合、データと各 **RANGE** インデックスごとに別のディスクが使用されています。他にもバリエーションが考えられます。例えば：

```

CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE(YEAR(purchased))
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0a
      DATA DIRECTORY = '/disk0'
      INDEX DIRECTORY = '/disk1',
    SUBPARTITION s0b
      DATA DIRECTORY = '/disk2'
      INDEX DIRECTORY = '/disk3'
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s1a
      DATA DIRECTORY = '/disk4/data'
      INDEX DIRECTORY = '/disk4/idx',
    SUBPARTITION s1b
      DATA DIRECTORY = '/disk5/data'
      INDEX DIRECTORY = '/disk5/idx'
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s2a,
    SUBPARTITION s2b
  )
);

```

ここでは、ストレージは以下のようになります。

- **purchased** 日付が1990 以前の行は大容量を必要とするため、4つに分けられています。これは別のディスクをデータと各サブパーティション専用に割り当て、(s0a と s0b) p0 パーティションを作成する。言い換えると：
 - サブパーティション **s0a** のデータは **/disk0** に記憶されています。
 - サブパーティション **s0a** のインデックスは **/disk0 1** に記憶されています。
 - サブパーティション **s0b** のデータは **/disk2** に記憶されています。
 - サブパーティション **s0b** のインデックスは **/disk3** に記憶されています。
- 1990 から1999を含む行は(パーティション **p1**) 1990 以前のものと比べ、容量を多く要求しません。p0 で記憶されている4つのディスクのレガシイレコードと比べ、これらは2つのディスク上振り分けられています。(disk4 と disk5)

- **p1** の最初のサブパーティションに含まれるデータやインデックスは(**s1a**) は **/disk4** に記憶され、 — **/disk4/data** のデータ、**/disk4/idx** 内のインデックス
- **p1** の2番目のサブパーティションに含まれるデータやインデックスは(**s1b**) は **/disk5** に記憶され、 — **/disk5/data** のデータ、**/disk5/idx** 内のインデックス
- 2000年から現在を示す(パーティション **p2**) 行は、以前の2レンジで必要とされたほどのスペースは要求されません。現在では、デフォルト位置にこれら全てを記憶することで事足ります。

将来的に、2000年から始まった購入のデータ量がデフォルト位置内で支えきれなくなった時、**ALTER TABLE ... REORGANIZE PARTITION** ステートメントを使用してそれらの行は移動させることができます。詳細については、**パーティショニング管理** を参照してください。

MySQLパーティショニングの **NULL** 値の取り扱い

MySQLでのパーティションは、カラム値であろうと、ユーザによって提供された表現であろうと、**NULL** をパーティショニング表現の値として禁じるようなことは一切しません。**NULL** 値を、整数を生み出す表現の値として使用することが許可されていますが、**NULL** は数値でないことを覚えておいてください。MySQL5.1.8より、パーティションは**NULL** を全ての非 **NULL** 値より少ないものと認めます。これは、**ORDER BY** でも同じです。

これにより、**NULL** の取り扱いは異なるパーティショニングの種類によって、予期せぬ事態を招くことがあります。これにより、この章では各MySQLのパーティショニングのタイプが、行が記憶されるパーティションを選択するさい、どのように **NULL** 値を取り扱うかを紹介し、例を取り上げます。

パーティションを判定するカラム値が **NULL** となるよう **RANGE** によりパーティショニングされたテーブルに行を挿入した場合、行は最も低いパーティションに挿入されます。例えば、以下の2つの実装、作成されたテーブルを記します。

```
mysql> CREATE TABLE t1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY RANGE(c1) (
->   PARTITION p0 VALUES LESS THAN (0),
->   PARTITION p1 VALUES LESS THAN (10),
->   PARTITION p2 VALUES LESS THAN MAXVALUE
-> );
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> CREATE TABLE t1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY RANGE(c1) (
->   PARTITION p0 VALUES LESS THAN (-5),
->   PARTITION p1 VALUES LESS THAN (0),
->   PARTITION p1 VALUES LESS THAN (10),
->   PARTITION p2 VALUES LESS THAN MAXVALUE
-> );
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
```

Query OK, 1 row affected (0.00 sec)

```
mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
```

Query OK, 1 row affected (0.00 sec)

```
mysql> SELECT * FROM t1;
```

```
+-----+
| id | name |
+-----+
| NULL | mothra |
+-----+
```

1 row in set (0.00 sec)

```
mysql> SELECT * FROM t2;
```

```
+-----+
| id | name |
+-----+
```

```
| NULL | mothra |
+-----+
1 row in set (0.00 sec)
```

どのパーティションに行が記憶されているかは、ファイルシステムを検査し、パーティションと対応している **.MYD** ファイルのサイズを比較することで割り出すことができます。

```
/var/lib/mysql/test> ls -l *.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 03:27 t1#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t1#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t1#P#p2.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 03:27 t2#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t2#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t2#P#p2.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 03:17 t2#P#p3.MYD
```

(パーティションのファイルは **table_name#P#partition_name.extension** フォーマットにより名づけられます。**t1#P#p0.MYD** はテーブル **t1** のパーティション **p0** データが記憶されているところです。注: MySQL 5.1.5以前には、これらのファイルはそれぞれ **t1_p0.MYD** や **t2_p0.MYD** と名づけられていました。この変化が更新に対してどういう影響を及ぼすかは、[Changes in release 5.1.6 \(01 February 2006\)](#) とバグ#13437を参照してください。)

これらの行が各テーブルの最も低いパーティションに記憶されていたことを証明するには、これらのパーティションを削除し、**SELECT** ステートメントを起動します。

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

(**ALTER TABLE ... DROP PARTITION** に関する情報については、**ALTER TABLE 構文** を参照してください。)

これはSQL関数を使用するパーティショニング表現に対しても同様です。例えば、以下のようなテーブルがあるとします。

```
CREATE TABLE tndate (
  id INT,
  dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

他の MySQL 関数同様、**YEAR(NULL)** は **NULL** を返します。**NULL** の **dt** カラム値を持つ行は、パーティショニング表現がそれ以外の値よりも少ない値に評価されたかのように扱われるため、**p0** パーティションに挿入される。

LIST によってパーティショニングされたテーブルが、**NULL** 値を認めるのは、**NULL** を含む値のリストを使用して一部のパーティションが定義されている場合のみです。これの逆は、**LIST** によってパーティショニングされたテーブルで、値のリスト拒否で **NULL** を明確にしようしないため、以下のような**NULL** 値のパーティショニング表現に至ります。

```
mysql> CREATE TABLE ts1 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (0, 3, 6),
->   PARTITION p1 VALUES IN (1, 4, 7),
->   PARTITION p2 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts1 VALUES (9, 'mothra');
```

```
ERROR 1504 (HY000): Table has no partition for value 9
```

```
mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL
```

唯一 0 から 8 の間に `c1` の値が含まれる行が `ts1` に挿入可能です。NULL は 9 の様に、このレンジ外に位置します。NULL を含む値リストを持つテーブル `ts2` や `ts3` を、以下のとおり作成することができます。

```
mysql> CREATE TABLE ts2 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (0, 3, 6),
->   PARTITION p1 VALUES IN (1, 4, 7),
->   PARTITION p2 VALUES IN (2, 5, 8),
->   PARTITION p3 VALUES IN (NULL)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
->   c1 INT,
->   c2 VARCHAR(20)
-> )
-> PARTITION BY LIST(c1) (
->   PARTITION p0 VALUES IN (0, 3, 6),
->   PARTITION p1 VALUES IN (1, 4, 7, NULL),
->   PARTITION p2 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.01 sec)
```

パーティションのために値のリストを定義している時、NULL は他の値と同様に扱えます。よって、VALUES IN (NULL) と VALUES IN (1, 4, 7, NULL) は両方有効です (VALUES IN (1, NULL, 4, 7)、VALUES IN (NULL, 1, 4, 7)、以下同様)。テーブル `ts2` と `ts3` 両方に NULL を持つ行をカラム `c1` に挿入することができます。

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

ファイルシステムを検査することで、テーブル `ts2` のパーティション `p3` にはこれらステートメントの最初のものが挿入され、テーブル `ts3` のパーティション `p1` には2番目のステートメントが挿入されたことを確認することができます。

```
/var/lib/mysql/test> ls -l ts2*.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:35 ts2#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:35 ts2#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:35 ts2#P#p2.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 10:35 ts2#P#p3.MYD

/var/lib/mysql/test> ls -l ts3*.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:36 ts3#P#p0.MYD
-rw-rw---- 1 mysql mysql 20 2006-03-10 10:36 ts3#P#p1.MYD
-rw-rw---- 1 mysql mysql 0 2006-03-10 10:36 ts3#P#p2.MYD
```

新しい例のとおり、これらファイルをリストするため、ユニックスオペレーティングシステム上で `bash` シェルを使用します。この件に関しては、ユーザのプラットフォームが提供するものを使用してください。たとえば、Windows のOS上でDOSシェルを使用している場合、最後のリストと等価のものは `C:\Program Files\MySQL\MySQL Server 5.1\data\test` ディレクトリ内の `dir ts3*.MYD` コマンドを起動することで取得できる可能性があります。

このセクションの前部で紹介したとおり、削除し、SELECT を実行することで値の記憶に使用されたパーティションを確認することができます。

NULL は HASH や KEY によってパーティショニングされたテーブルとは同様に扱われます。こういったケースでは、NULL 値を生み出すパーティショニング表現は返される値が 0 であるかのように扱われます。この動作を確認するには、HASH によってパーティショニングされたテーブルを作成し、適当な値を含むレコードで実装させることでファイルシステムへの影響を調べることができます。たとえば、以下のステートメントで、test データベース内のテーブル `th` が作成されたとします。

```
mysql> CREATE TABLE th (
```



```
-> c1 INT,
-> c2 VARCHAR(20)
-> )
-> PARTITION BY HASH(c1)
-> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
```

Linux上でのMySQLのRPMインストールを想定すると、このステートメントは2つの `.MYD` ファイルを `/var/lib/mysql/test` につくり、それは `bash` シェルで以下の様に現れます。

```
/var/lib/mysql/test> ls th*.MYD -l
-rw-rw---- 1 mysql mysql 0 2005-11-04 18:41 th#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2005-11-04 18:41 th#P#p1.MYD
```

各ファイルのサイズが0バイトであることに注目してください。では、`c1` カラム値が `NULL` の行を `th` に挿入して、その行が挿入されたことを認証してください。

```
mysql> INSERT INTO th VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+-----+-----+
| c1    | c2    |
+-----+-----+
| NULL  | mothra |
+-----+-----+
1 row in set (0.01 sec)
```

どの整数 `N` にとっても、`NULL MOD N` の値は常に `NULL` になります。`HASH` や `KEY` によってパーティショニングされたテーブルに関しては、この結果は正しいパーティションを `0` として確定するために扱われます。システムシェルに戻ると、(このため `bash` を使用します)、再度データファイルをリストすることで、値が最初のパーティションに挿入されたことを確認できます。(デフォルトで `p0` と名づけられる)

```
/var/lib/mysql/test> ls *.MYD -l
-rw-rw---- 1 mysql mysql 20 2005-11-04 18:44 th#P#p0.MYD
-rw-rw---- 1 mysql mysql 0 2005-11-04 18:41 th#P#p1.MYD
```

他のデータファイルに影響することなく (ディスク上でサイズを増大)、`th#P#p0.MYD` ファイルのみ、`INSERT` ステートメントが改良したことが確認できます。

重要MySQL 5.1.8以前では、`RANGE` パーティショニングは、配置の断定に関して、パーティショニング表現値 `NULL` をゼロとして扱いました。(これを回避する方法は、ヌルを許容しないテーブルをデザインすることで、通常はカラムを `NOT NULL` と宣言することで実行しました)。この前期の動作による `RANGE` パーティショニングスキーマがある場合、MySQL 5.1.8以降にアップグレードする時再実装しなければいけません。

パーティショニング管理

MySQL 5.1 はパーティショニングされたテーブルの改良方法をいくつか提供しています。存在するパーティションを追加、削除、再定義、結合、そして分離させることができます。これらのアクションの全てが `ALTER TABLE` コマンドのパーティショニング拡張によって行うことができます。(構文の定義については[ALTER TABLE 構文](#)を参照してください)パーティショニングされたテーブルやパーティションそれ自体の情報を取得する方法もあります。続くセクションでこれらのトピックを紹介します。

- `RANGE` や `LIST` によってパーティショニングされたテーブルのパーティション管理に関する情報については、[RANGE と LIST パーティションの管理](#) を参照してください。
- `HASH` や `KEY` パーティショニングの管理に関しては、[HASH や KEY パーティションの管理](#) を参照してください。
- MySQL 5.1 で提供されるパーティショニングやパーティショニングされたテーブルの収集メカニズムについては、[パーティション情報の取得](#) を参照してください。
- パーティションのメンテナンスを行う場合、[パーティションのメンテナンス](#) を参照してください。

注:MySQL 5.1 では、パーティショニングされたテーブルのパーティションは全て同じ数のサブパーティションを持たなければいけません。また、一度テーブルが作成されてからサブパーティションを変更することは不可能です。

ステートメント `ALTER TABLE ... PARTITION BY ...` は MySQL 5.1.6 より稼動しています。MySQL 5.1 では、構文的には認められていてもステートメント事態は効果がありませんでした。

テーブルのパーティショニングスキーマを変更するには、`ALTER TABLE` コマンドを `partition_options` 節と共に使用することのみが要求されます。この節は `CREATE TABLE` のパーティショニングされたテーブルを作成するのに使用される同じ構文をもち、必ず `PARTITION BY` のキーワードで始まる。たとえば、以下の `CREATE TABLE` ステートメントを使用してレンジによりパーティショニングされたテーブルがあるとします。

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (2000),
  PARTITION p3 VALUES LESS THAN (2005)
);
```

このテーブルを再度パーティショニングし、その際キーにより二つのパーティションに分けられるように、キーのベースに `id` カラム値を用い、以下のステートメントを使用してください。

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

これはテーブルを削除し `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;` を使用して再作成するのと同様の効果をテーブルの構成にもたらしめます。

重要MySQL 5.1.7 と MySQL 5.1 のリリースでは、`ALTER TABLE ... ENGINE = ...` は影響されたテーブルの全てのパーティションを取り除く動作をしました。MySQL 5.1.8 に始まり、このステートメントはテーブルに使用されるストレージエンジンだけを変更し、テーブルのパーティショニングスキーマは保たれます。MySQL 5.1.8以降はテーブルのパーティションを取り除くには、`ALTER TABLE ... REMOVE PARTITIONING` を使用してください。詳細については `ALTER TABLE` 構文 を参照してください。

RANGE と LIST パーティションの管理

パーティションの追加や削除が行われる点に関しては、レンジパーティショニングもリストパーティショニングもよく似ています。このため、これらのパーティションの管理をこのセクションで紹介しします。ハッシュやキーでパーティショニングされたテーブルの使用については、`HASH` や `KEY パーティションの管理` を参照してください。`RANGE` や `LIST` パーティションの削除は追加よりも単純なので、先にこちらを紹介します。

`RANGE` か `LIST` によってパーティショニングされたテーブルからパーティションを削除するには、`ALTER TABLE` ステートメントを `DROP PARTITION` 節と使用することで達成できます。ここに単純な例を記します。すでにレンジで作成され、`CREATE TABLE` と `INSERT` ステートメントを使用して10のレコードで実装されたテーブルがあるとします。

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
-> PARTITION BY RANGE( YEAR(purchased) ) (
-> PARTITION p0 VALUES LESS THAN (1990),
-> PARTITION p1 VALUES LESS THAN (1995),
-> PARTITION p2 VALUES LESS THAN (2000),
-> PARTITION p3 VALUES LESS THAN (2005)
-> );
```

Query OK, 0 rows affected (0.01 sec)

```
mysql> INSERT INTO tr VALUES
-> (1, 'desk organiser', '2003-10-15'),
-> (2, 'CD player', '1993-11-05'),
-> (3, 'TV set', '1996-03-10'),
-> (4, 'bookcase', '1982-01-10'),
-> (5, 'exercise bike', '2004-05-09'),
-> (6, 'sofa', '1987-06-05'),
-> (7, 'popcorn maker', '2001-11-22'),
-> (8, 'aquarium', '1992-08-04'),
-> (9, 'study desk', '1984-09-16'),
-> (10, 'lava lamp', '1998-12-25');
```

Query OK, 10 rows affected (0.01 sec)

どのアイテムが `p2` パーティションに挿入されるべきかは、以下で確認できます。

```
mysql> SELECT * FROM tr
-> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+-----+-----+
| id | name | purchased |
+-----+-----+
| 3 | TV set | 1996-03-10 |
```

```
| 10 | lava lamp | 1998-12-25 |
+-----+-----+
2 rows in set (0.00 sec)
```

p2 と名づけられたパーティションを削除するには、以下のコマンドを実行してください。

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

注:MySQL 5.1 では、**NDBCLUSTER** ストレージ エンジンでは **ALTER TABLE ... DROP PARTITION** をサポートしません。ただし、この章で紹介される他の **ALTER TABLE** パーティショニングに関連する拡張はサポートされます。

パーティションを削除する時、そのパーティション内で記憶されていたデータも全て削除される ことに注意してください。以前の **SELECT** クエリを再起動させることでこれを確認できます。

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

これにより、MySQL 5.1.10 では **ALTER TABLE ...DROP PARTITION** を実行する前にテーブルの **DROP** 権限があります。

テーブルの定義とパーティショニングスキーマを保持したままパーティションからデータを削除したい場合、**TRUNCATE TABLE** コマンドを実行してください。(詳しくは **TRUNCATE** 構文 をご確認ください。)

データを失わずに、テーブルのパーティションを変更する場合、**ALTER TABLE ...REORGANIZE PARTITION** を代わりに使用してください。**REORGANIZE PARTITION** に関する情報については、以下が **ALTER TABLE** 構文 を参照してください。

これで **SHOW CREATE TABLE** コマンドを実行すれば、テーブルのパーティショニングの構造がどう変更したか確認できます。

```
mysql> SHOW CREATE TABLE tr\G
***** 1. row *****
Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.01 sec)
```

'1995-01-01' と '2004-12-31' 間の **purchased** カラム値を変更されたテーブルに行挿入する時、**p3** パーティションに記憶されます。以下の様に証明できます。

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
+-----+-----+
| id | name       | purchased |
+-----+-----+
| 11 | pencil holder | 1995-07-12 |
| 1 | desk organiser | 2003-10-15 |
| 5 | exercise bike | 2004-05-09 |
| 7 | popcorn maker | 2001-11-22 |
+-----+-----+
4 rows in set (0.00 sec)

mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

ALTER TABLE ... DROP PARTITION の結果テーブルから削除された行の数は、同等の **DELETE** クエリによって処理された場合と違い、サーバによって報告されません。

LIST パーティショニングの削除は、**RANGE** パーティショニングの削除と同様の **ALTER TABLE ... DROP PARTITION** 構文を使用します。ただし、テーブルの使用に対して、1つ重大な違いがあります。テーブルに、削除されたパーティションを定義する値リストに含まれていた値を、行挿入することができません。(LIST パーティショニングで例を参照してください。)

以前にパーティショニングされたテーブルに新しいレンジ、もしくはリストパーティションを追加する場合は、**ALTER TABLE ... ADD PARTITION** ステートメントを使用してください。**RANGE** によってパーティショニングされているテーブルには、これを行うことによって存在するパーティションのリストに新しいレンジを追加できます。例えば、ユーザの所属する機関のメンバーデータを含むパーティショニングされたテーブルが、以下のようにあるとします。

```
CREATE TABLE members (
  id INT,
  fname VARCHAR(25),
  lname VARCHAR(25),
  dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970),
  PARTITION p1 VALUES LESS THAN (1980),
  PARTITION p2 VALUES LESS THAN (1990)
);
```

例えば、さらに、メンバーの最低年齢が16とします。2005年に近づく、1990年に生まれたメンバー (そしてその年以降) を受け付けていることに気づき始めるでしょう。**members** テーブルを改良して、1990-1999に生まれた新メンバーを表すことができます。

```
ALTER TABLE ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

重要レンジによりパーティショニングされたテーブルでは、パーティションのリストに新しいパーティションを追加するため、**ADD PARTITION** を使用することができます。このようにして、存在するパーティションのまえ、もしくは間に新しいパーティションを追加すると、以下のようなエラー表示となります。

```
mysql> ALTER TABLE members
> ADD PARTITION (
> PARTITION p3 VALUES LESS THAN (1960));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
increasing for each partition
```

同じように、**LIST** によってパーティショニングされたテーブルに新しいパーティションを追加することができます。例えば、以下の様に定義されたテーブルでは：

```
CREATE TABLE tt (
  id INT,
  data INT
)
PARTITION BY LIST(data) (
  PARTITION p0 VALUES IN (5, 10, 15),
  PARTITION p1 VALUES IN (6, 12, 18)
);
```

data カラム値 7、14、そして 21 含む行を記憶する新しいパーティションを追加することができます。

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

存在するパーティションの値リストに含まれる値を包含する新しい **LIST** パーティションを追加することはできません。試みると、以下のエラーが発生します。

```
mysql> ALTER TABLE tt ADD PARTITION
> (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
in list partitioning
```

data カラム値 12 を含む行はすでにパーティション **p1** に割り振られているため、テーブル **tt** に 12 を値リストに含む新しいパーティションを作成することはできません。これを達成するには、**p1** を削除し、**np** を追加してから、改良された定義の新しい **p1** を作成する必要があります。ただし、以前述べたとおり、**p1** に記憶れた全てのデータの損失

につながります。— 大抵、ユーザの意思とはかけ離れた結果となります。また、別の解決法で、新しいパーティションを含んだテーブルのコピーを作成し、かつデータを `CREATE TABLE ...SELECT ...` を使用して書き込み、古いテーブルを削除し新しいテーブルの名前をつけなおすことができますが、量の多いデータを取り扱っている時など、非常に多くの時間を要することがあります。加えて、これは高い有効性が求められている状況では、あまり推奨できる手段ではありません。

MySQL 5.1.6 二始まり、以下の様に1つの `ALTER TABLE ... ADD PARTITION` ステートメントに複数のパーティションを追加することができます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
  PARTITION p5 VALUES LESS THAN (2010),
  PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

幸い、MySQLのパーティショニング実装はデータを損失することなくパーティショニングを再定義する方法を提供しています。では、`RANGE` パーティショニングの例をいくつか見てみましょう。以下のように定義されている、`members` テーブルを思い出してください。

```
mysql> SHOW CREATE TABLE members\G
***** 1. row *****
Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) default NULL,
  `fname` varchar(25) default NULL,
  `lname` varchar(25) default NULL,
  `dob` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1980) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2000) ENGINE = MyISAM
)
```

例えば、1960年以前に生まれたメンバーを示す行を別のパーティションに移動させたいとします。すでに見たように、`ALTER TABLE ... ADD PARTITION` を使用して行うのは不可能です。ただし、別の `ALTER TABLE` のパーティショニング関連の拡張をすることでこれを達成できます。

```
ALTER TABLE members REORGANIZE PARTITION p0 INTO (
  PARTITION s0 VALUES LESS THAN (1960),
  PARTITION s1 VALUES LESS THAN (1970)
);
```

実質的に、このコマンドはパーティション `p0` を2つの新しいパーティション `s0` と `s1` に分けます。`p0` に記憶されていたデータを2つの `PARTITION ... VALUES ...` 節で記されるルールによってデータが移動させられます。これにより、`s0` は `YEAR(dob)` が1960未満の、そして `s1` は `YEAR(dob)` 1960 以上、1970 未満のレコードを含む行が存在します。

`REORGANIZE PARTITION` 節を使用して隣接するパーティションを結合することができます。以下の様に `members` テーブルを以前のパーティションに戻すことができます。

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
  PARTITION p0 VALUES LESS THAN (1970)
);
```

`REORGANIZE PARTITION` を使用してパーティションを分離、結合することでデータが失われることはありません。上記のステートメントを実行する際、MySQLは `s0` と `s1` に記憶されていたパーティションを `p0` に移動させます。

REORGANIZE PARTITION の一般的な構文は

```
ALTER TABLE tbl_name
  REORGANIZE PARTITION partition_list
  INTO (partition_definitions);
```

ここでは、`tbl_name` はパーティショニングされたテーブルの名前で、`partition_list` はカンマによって分けられた、変更すべき存在するパーティションの名前です。`partition_definitions` はカンマによって分けられた、新しいパーティションの定義のリストで、`CREATE TABLE` で使用される `partition_definitions` リストと同様のルールに従います。(CREATE TABLE 構文を参照してください)。REORGANIZE PARTITION を使用する時、複数のパーティションを1つに結合する、もしくは1つのパーティションを複数のパーティションに分離することだけに制限されているわけではありません。例えば、以下の様に、`members` テーブル内の4つのパーティションを2つに再編成することができます。

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
  PARTITION m0 VALUES LESS THAN (1980),
  PARTITION m1 VALUES LESS THAN (2000)
);
```

REORGANIZE PARTITION を LIST によってパーティショニングされたテーブルと使用することもできます。それでは、リストによってパーティショニングされたテーブル `tt` に新しいパーティションを追加する問題に戻ってみましょう。この問題は、新しいパーティションに、すでに存在するパーティションの値リストに含まれる値が新しいパーティションにも含まれていたため、失敗に終わりました。これは、衝突しない値を含むパーティションを追加することで対処し、新しいパーティションを存在するパーティションを再編成する際に、存在するパーティション内に含まれていた値が新しいパーティションに移動することで、解決できます。

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
  PARTITION p1 VALUES IN (6, 18),
  PARTITION np VALUES in (4, 8, 12)
);
```

RANGE や LIST を使用してパーティショニングされたテーブルを再度パーティショニングする際に使用される ALTER TABLE ... REORGANIZE PARTITION に関する重要なポイントは、以下のとおりです。

- 新しいパーティショニングスキーマを決定するのに使用される PARTITION 節は CREATE TABLE ステートメントで使用されているものに対して同じルールが適用されます。

さらに重要なのは、新しいパーティショニングスキーマには重複するレンジや値のセットがあってはならないことです(RANGE や LIST によってパーティショニングされたテーブルを再編成する際に適用する)。

注:MySQL 5.1.4以前では、INTO 節内で存在するパーティションの名前を、それらのパーティションが再定義、もしくは削除されていても再利用することはできませんでした。詳細については、[Changes in release 5.1.4 \(21 December 2005\)](#) を参照してください。

- `partition_definitions` リストに含まれるパーティションのコンビネーションは、`partition_list` で名づけられている結合されたパーティションと同じレンジ、値のセットになります。

たとえば、このセクションで例として使用されている `members` テーブル では、パーティション `p1` と `p2` は合わせて1980から1999の期間をカバーしているということになります。よって、これらパーティションのどの再編成も、最終的には同じ期間をカバーすることになります。

- RANGE によりパーティショニングされたテーブルに関しては、隣接するパーティションのみ再編成することができます。レンジパーティションを飛び越すことはできません。

たとえば、このセクションで使用されている `members` テーブルを ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ... で始まるステートメントに再編成することはできません。なぜなら、`p0` 1970 以前の年度をカバーしており、`p2` は1990から1999をカバーするため、この二つは隣接するパーティションにはなりえません。

- REORGANIZE PARTITION を使用してテーブルのパーティショニングのタイプを変更することはできません。それは、RANGE パーティショニングを HASH パーティショニングや、その逆 vice versa もまた不可能ということになります。また、このコマンドを使用してパーティショニング表現やカラムを変更することができます。両方のタスクを、テーブルを削除もしくは再作成せずに行う場合、ALTER TABLE ...PARTITION BY ... を使用することができます。例:

```
ALTER TABLE members
```



```
PARTITION BY HASH( YEAR(dob) )
PARTITIONS 8;
```

HASH や KEY パーティションの管理

ハッシュやキーによりパーティショニングされたテーブルの変更はパーティショニングの設定を行う際と手順が似ており、レンジやリストによりパーティショニングされたテーブルとは違いがあります。そのため、このセクションはハッシュやキーのみによりパーティショニングされたテーブルの変更に関するトピックに絞って説明をおこないます。レンジやリストによりパーティショニングされたテーブルの追加や削除については、[RANGE と LIST パーティションの管理](#) を参照してください。

HASH や KEY によりパーティショニングされたテーブルを RANGE や LIST によりパーティショニングされたテーブルと同じように削除することはできません。ただし、HASH や KEY パーティショニングは ALTER TABLE ...COALESCE PARTITION コマンドを使って結合することはできます。例えば、12のパーティションに分かれた、クライアントのデータを含むテーブルがあるとします。clients テーブルは以下の様に定義されています。

```
CREATE TABLE clients (
  id INT,
  fname VARCHAR(30),
  lname VARCHAR(30),
  signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

パーティションの数を12から8に減らす場合、以下の ALTER TABLE コマンドを実行することができます。

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

COALESCE は HASH、KEY、LINEAR HASH、または LINEAR KEY によりパーティショニングされたテーブルとは同じ様に使用できます。ここに、以前の例と類似している例を記します。LINEAR KEY によりパーティショニングされているという点のみ、異なります。

```
mysql> CREATE TABLE clients_lk (
-> id INT,
-> fname VARCHAR(30),
-> lname VARCHAR(30),
-> signed DATE
-> )
-> PARTITION BY LINEAR KEY(signed)
-> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> ALTER TABLE clients_lk COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

COALESCE PARTITION は残りのパーティションに結合されるパーティションの数を示しています。— 言い換えると、テーブルから取り除くパーティションの数を指します。

テーブルに含まれている以上のパーティションを取り除こうとすると、以下のようなエラーが表示されます。

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

clients テーブル内のパーティションの数を12から18に増やす場合、ALTER TABLE ...ADD PARTITIONを、以下に示されるとおり使用してください。

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

パーティションのメンテナンス

MySQL では複数のパーティショニングメンテナンスタスクを行うことができます。5.1.MySQLでは CHECK TABLE、OPTIMIZE TABLE、ANALYZE TABLE、そして REPAIR TABLEのようなコマンドをパーティショニングされたテーブル用にサポートされてはいません。代わりに、MySQL5.1.5で実装された ALTER TABLE の拡張のいくつかを使用することができます。これらはこのような種類のオペレーションを直接1つ、もしくは複数のパーティションに実行するのに使用できます。以下のリストを参照してください。

- REBUILD PARTITIONパーティションを再構築します。パーティションに記憶されているレコードを削除し、再度挿入するのと同じ効果があります。これはデフラグメンテーションのために有効に使えます。

例:

```
ALTER TABLE t1 REBUILD PARTITION p0, p1;
```

- OPTIMIZE PARTITIONもしパーティションから多くの行を削除もしくは異なる長さの行を含むパーティショニングされたテーブルに変更を加えた場合、(つまり、[VARCHAR](#)、[BLOB](#)、または[TEXT](#)カラムを含む)[ALTER TABLE ...OPTIMIZE PARTITION](#)を使用してパーティションのデータファイルをデフラグすることによって使用されていないスペースを再利用できます。

例:

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

[OPTIMIZE PARTITION](#) をあるパーティションで使用することは [CHECK PARTITION](#)、[ANALYZE PARTITION](#)、そして [REPAIR PARTITION](#) をそのパーティションで作動させることと同義です。

- ANALYZE PARTITION:これはパーティションのキーの分散を読み込み・記憶します。

例:

```
ALTER TABLE t1 ANALYZE PARTITION p3;
```

- REPAIR PARTITIONこれは破壊されたパーティションを修復します。

例:

```
ALTER TABLE t1 REPAIR PARTITION p0,p1;
```

- CHECK PARTITION[CHECK TABLE](#) をパーティショニングされていないテーブルでチェックのため使用できるように、パーティションのエラーをチェックすることができます。

例:

```
ALTER TABLE trb3 CHECK PARTITION p1;
```

このコマンドはテーブル [t1](#) のデータやインデックスが [p1](#) 破壊されているかを知らせます。この場合、[ALTER TABLE ...REPAIR PARTITION](#)を使用してパーティションを修復することができます。

[mysqlcheck](#) や [myisamchk](#) ユーティリティを使用してこれらのタスクを達成することができます。これらは、テーブルをパーティショニングすることによって別々の [.MYI](#) ファイルに生成されています。[mysqlcheck — テーブル メンテナンスと修復プログラム](#) を参照してください。

パーティション情報の取得

このセクションでは、存在するパーティションの情報を取得するいくつかの方法を紹介します。これらは：

- [SHOW CREATE TABLE](#) ステートメントを使用して、パーティショニングされたテーブルの作成に使用された [PARTITION](#)句を一覧する。
- [SHOW TABLE STATUS](#) ステートメントを使用して、テーブルがパーティショニングされているかを判定する。
- [INFORMATION_SCHEMA.PARTITIONS](#) をクエリする。
- [EXPLAIN PARTITIONS SELECT](#) ステートメントを使用して、どのパーティションが [SELECT](#) で使用されているか判別する。

この章で別途紹介されているように、[SHOW CREATE TABLE](#) はその出力にパーティショニングされたテーブルの作成に用いる [PARTITION BY](#) 節を含んでいる。例:

```
mysql> SHOW CREATE TABLE trb3\G
***** 1. row *****
Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) default NULL,
```

```
`name` varchar(50) default NULL,
`purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.00 sec)
```

注:MySQL 5.1の最初のリリースでは、[PARTITIONS](#) 節は、[HASH](#) や [KEY](#) によってパーティショニングされたテーブルでは表示されませんでした。この問題は MySQL 5.1.6. で修正されました。

[SHOW TABLE STATUS](#) はパーティショニングされたテーブルにおいても動作します。MySQL 5.1.9に始まり、パーティショニングされていないテーブルと出力は同じですが、[Create_options](#) カラムに [partitioned](#) 文字列を含んでいます。MySQL 5.1.8 以前では、[Engine](#) カラムには必ず [PARTITION](#) の値が含まれていました。MySQL 5.1.9 に始まり、このカラムはテーブルに使用される全てのパーティションのストレージエンジンの名前を含んでいます。(このコマンドの詳細については、[SHOW TABLE STATUS 構文](#) を参照してください。)

パーティションの情報は [PARTITIONS](#) テーブルが含まれる、[INFORMATION_SCHEMA](#) からも取得できます。[INFORMATION_SCHEMA PARTITIONS テーブル](#) を参照してください。

MySQL 5.1.5 に始まり、[EXPLAIN PARTITIONS](#) を使用して、パーティショニングされたテーブルのどのパーティションが [SELECT](#) と関係しているかを判定することができます。[PARTITIONS](#) キーワードはクエリと照合されるレコードを記したパーティションを表示する、[EXPLAIN](#) の出力結果に [partitions](#) カラムを追加します。

例えば、以下の様に定義・実装されたテーブル [trb1](#) があるとします。

```
CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE(id)
(
  PARTITION p0 VALUES LESS THAN (3),
  PARTITION p1 VALUES LESS THAN (7),
  PARTITION p2 VALUES LESS THAN (9),
  PARTITION p3 VALUES LESS THAN (11)
);

INSERT INTO trb1 VALUES
(1, 'desk organiser', '2003-10-15'),
(2, 'CD player', '1993-11-05'),
(3, 'TV set', '1996-03-10'),
(4, 'bookcase', '1982-01-10'),
(5, 'exercise bike', '2004-05-09'),
(6, 'sofa', '1987-06-05'),
(7, 'popcorn maker', '2001-11-22'),
(8, 'aquarium', '1992-08-04'),
(9, 'study desk', '1984-09-16'),
(10, 'lava lamp', '1998-12-25');
```

以下の様に、どのパーティションが [SELECT * FROM trb1](#); といったクエリで使用されているかを確認することができます。

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: trb1
  partitions: p0,p1,p2,p3
         type: ALL
possible_keys: NULL
          key: NULL
        key_len: NULL
         ref: NULL
         rows: 10
      Extra: Using filesort
```

この場合、全4つのパーティションが検索されます。ただし、パーティショニングキーを使用して作成された検索条件がクエリに追加された時、以下のように、合致する値を含むパーティションのみが検索されます。

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5G
```

```
***** 1. row *****
id: 1
select_type: SIMPLE
table: trb1
partitions: p0,p1
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 10
Extra: Using where
```

EXPLAIN PARTITIONS は標準的な **EXPLAIN SELECT** ステートメントのように、キーや可能性のあるキーに関する情報を提供します。

```
mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10 Duplicates: 0 Warnings: 0

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5;G
***** 1. row *****
id: 1
select_type: SIMPLE
table: trb1
partitions: p0,p1
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: NULL
rows: 7
Extra: Using where
```

以下の、**EXPLAIN PARTITIONS** の制限や規制について注意してください。

- **EXTENDED** と **PARTITIONS** キーワードを、同じ **EXPLAIN ... SELECT** ステートメントで使用することはできません。これを試みると、構文エラーが発生します。
- もし **EXPLAIN PARTITIONS** がパーティショニングされていないテーブルに対するクエリを診断する場合、エラーは発生しませんが、**partitions** カラムの値は常に **NULL** となります。

EXPLAINを使用して、**クエリを最適化する** も参照してください。

パーティションの刈り込み

このセクションではパーティションの刈り込みを紹介します。この最適化は、MySQL 5.1.6.でパーティショニングされたテーブル用に実装されています。

パーティション刈り込みのコンセプトは単純です。「合致する値が存在し得ないパーティションはスキャンしない」というものです。例えば、以下のステートメントに定義されたパーティショニングされたテーブル **t1** があるとします。

```
CREATE TABLE t1 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
  PARTITION p0 VALUES LESS THAN (64),
  PARTITION p1 VALUES LESS THAN (128),
  PARTITION p2 VALUES LESS THAN (192)
  PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

以下のようなクエリから結果を取得しようとしているケースを検討してください。

```
SELECT fname, lname, postcode, dob
FROM t1
WHERE region_code > 125 AND region_code < 130;
```

これを見れば、返されるはずの行が **p0** か **p3** のどちらかのパーティションに含まれていることが容易に理解できます。つまり、**p1** と **p2** パーティションのみを検索する必要があるということがわかります。そうすることによって、テーブル内のパーティションをスキャンするよりも、一致する行を探すことに時間を費やすことができます。この不必要なパーティションを「省く」ことを、刈り込みといいます。オプティマイザがパーティションの刈り込みをクエリの実行に使用できると、パーティショニングされていないテーブルに含まれる同じカラム定義やデータに対して行うことに比べると一段速く、クエリの実行ができます。

WHERE 状態が以下の2つになる場合、クエリオプティマイザは刈り込みを実行することができます。

- **partition_column = constant**
- **partition_column IN (constant1, constant2, ..., constantN)**

ケース1の場合、オプティマイザは単純に与えられた値のパーティショニング表現を評価し、どのパーティションにその値があるかを判定、そしてそのパーティションのみを検索します。ケース2の場合、オプティマイザはリストに含まれる各値に対してパーティショニング表現を評価し一致するパーティションのリストを作成、そしてそのリストにあるパーティションだけを検索します。

刈り込みはショートレンジにも適用できます。この際、オプティマイザは値が等価のリストに変換できます。例えば、前の例では、**WHERE** 節は **WHERE region_code IN (125, 126, 127, 128, 129, 130)** に変換できます。この後、オプティマイザはリストに含まれる最初の3つの値が **p1** パーティションに含まれると断定、残りの3つの値が **p2** そして、他のパーティションには一致する値が含まれないと判断できるため、残りの検索を省くことができます。

この類の最適化はパーティショニング表現が等価もしくはレンジにより構成され、等価のセットに縮小できる場合、あるいはパーティショニング表現が増減する関係を表している場合に適用できます。刈り込みは **DATE** や **DATETIME** カラムでパーティショニングされたテーブルに対しても適用することができます。この時、パーティショニング表現は **YEAR()** または **TO_DAYS()** 関数を使用しています。(注:将来的にリリースされるMySQLには刈り込みサポートを追加する予定があります。これは、**DATE**、**DATETIME** 値、整数を返し、増減に対して追加関数を行う形になります。)例えば、以下の様に定義されたテーブル **t2** が **DATE** カラムでパーティショニングされていたとします。

```
CREATE TABLE t2 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( YEAR(dob) ) (
  PARTITION d0 VALUES LESS THAN (1970),
  PARTITION d1 VALUES LESS THAN (1975),
  PARTITION d2 VALUES LESS THAN (1980),
  PARTITION d3 VALUES LESS THAN (1985),
  PARTITION d4 VALUES LESS THAN (1990),
  PARTITION d5 VALUES LESS THAN (2000),
  PARTITION d6 VALUES LESS THAN (2005),
  PARTITION d7 VALUES LESS THAN MAXVALUE
);
```

t2 の以下のクエリが刈り込みを利用することができます。

```
SELECT * FROM t2 WHERE dob = '1982-06-23';

SELECT * FROM t2 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';

SELECT * FROM t2 WHERE YEAR(dob)
  IN (1979, 1980, 1983, 1985, 1986, 1988);

SELECT * FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'
```

最後のクエリに関しては、オプティマイザは以下の様に作動します。

1. レンジの下辺を含むパーティションを検索することができます。
YEAR('1984-06-21') は **1984** の値を生み出し、**d3** パーティションで発見されます。
2. レンジの上辺を含むパーティションを検索することができます。
YEAR('21.06.99') は **1999** の値を生み出し、**d5** パーティションで発見されます。

3. これら2つのパーティションと、それらの間にあるパーティションのみを検索します。.

この場合、パーティション **d3**、**d4**、**d5** のみが検索されます。残りのパーティションは安全に無視することができます。(無視されます。)

これまで、**RANGE** パーティショニングを含む例のみを挙げましたが、刈り込みは他のパーティショニングのタイプでも利用することができます。

LIST によってパーティショニングされたテーブルを検討します。パーティショニング表現が増減を繰り返している、以下のようなテーブル **t3** を検証してみましょう。(この例では、詳細を省くため **region_code** カラムが1から10の値の間に制限されているとします。)

```
CREATE TABLE t3 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY LIST(region_code) (
  PARTITION r0 VALUES IN (1, 3),
  PARTITION r1 VALUES IN (2, 5, 8),
  PARTITION r2 VALUES IN (4, 9),
  PARTITION r3 VALUES IN (6, 7, 10)
);
```

SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3 のようなクエリに関して、1、2、そして3の値がどのパーティションに含まれているかを判定します。(r0 と r1) 残りの(r2 と r3 はスキップします)。

HASH や **KEY** を使用してパーティショニングされているテーブルに関しては、パーティションの刈り込みは **WHERE** 節が単純な等価比較(=)をカラムのパーティショニング表現に対して行うことにより実現可能です。このように作成されたテーブルを検討してください。

```
CREATE TABLE t4 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

このようなクエリは全て刈り込みの対象となります。

```
SELECT * FROM t4 WHERE region_code = 7;
```

刈り込みはショートレンジに使用できます。これは、オプティマイザがそのような状態を **IN** 関係に変換することが可能なためです。例えば、以前定義された同テーブル **t4** を使用して、以下のようなクエリを刈り込みすることができます。

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;
```

```
SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

両ケースの場合、**WHERE** 節はオプティマイザによって **WHERE region_code IN (3, 4, 5)** に変更されます。重要この最適化はレンジの規模がパーティションの数よりも小さい場合のみ使用されます。このクエリを検討してください。

```
SELECT * FROM t4 WHERE region_code BETWEEN 4 AND 8;
```

WHERE 節のレンジは5つの値をカバーします(4, 5, 6, 7, 8)。しかし、**t4** には4つのパーティションしか存在しません。これは、以前のクエリを刈り込みできないことを意味しています。

刈り込みは **HASH** や **KEY** によってパーティショニングされたテーブルの整数カラムにのみ使用できます。例えば、テーブル **t4** のクエリは、**dob** が **DATE** カラムであるため、刈り込みを使用することができません。

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

ただし、テーブルが **INT** カラム内で年度を示す値を記憶している場合、**WHERE year_col >= 2001 AND year_col <= 2005** を含むクエリは刈り込みされます。

パーティショニングの制約と制限

このセクションでは現在MySQLパーティショニングサポートに課せられている制約と制限を紹介します。

- MySQL 5.1.12 より、以下の生成子はパーティショニング表現で許可されていません。
 - 入れ子関数コール(例えば、`func1( func2(col_name) )`) といった生成子を指します。
 - 記憶された関数、ストアド プロシージャ、UDF、プラグイン。
 - 宣言された変数やユーザ変数。
- MySQL 5.1.12 より、以下特定の MySQL 関数はパーティショニング表現で許容されていません。
 - `GREATEST()`
 - `ISNULL()`
 - `LEAST()`
 - `CASE()`
 - `IFNULL()`
 - `NULLIF()`
 - `BIT_LENGTH()`
 - `CHAR_LENGTH()`
 - `CHARACTER_LENGTH()`
 - `FIND_IN_SET()`
 - `INSTR()`
 - `LENGTH()`
 - `LOCATE()`
 - `OCTET_LENGTH()`
 - `POSITION()`
 - `STRCMP()`
 - `CRC32()`
 - `ROUND()`
 - `SIGN()`
 - `DATEDIFF()`
 - `PERIOD_ADD()`
 - `PERIOD_DIFF()`
 - `TIMESTAMPDIFF()`
 - `UNIX_TIMESTAMP()`
 - `WEEK()`

- [CAST\(\)](#)
- [CONVERT\(\)](#)
- [BIT_COUNT\(\)](#)
- [INET_ATON\(\)](#)
- [+](#)、[-](#)、[×](#)、そして [/](#) といった数的演算子はパーティショニング表現で許容されています。ただし、結果は整数値もしくは [NULL](#) でなければいけません。([\[LINEAR\] KEY](#) パーティショニングは例外となります。— 詳細については [パーティショニングのタイプ](#) を参照してください)。

MySQL 5.1.12 にはじまり、[|](#)、[&](#)、[^](#)、[<<](#)、[>>](#)、そして [~](#) といったビット演算子はパーティショニング表現では許容されていません。

- MySQL 5.1.12 より、以下特定の MySQL 関数のみがパーティショニング表現で許容されています。

- [ABS\(\)](#)
- [ASCII\(\)](#)
- [CEILING\(\)](#)
- [DAY\(\)](#)
- [DAYOFMONTH\(\)](#)
- [DAYOFWEEK\(\)](#)
- [DAYOFYEAR\(\)](#)
- [EXTRACT\(\)](#)
- [FLOOR\(\)](#)
- [HOUR\(\)](#)
- [MICROSECOND\(\)](#)
- [MINUTE\(\)](#)
- [MOD\(\)](#)
- [MONTH\(\)](#)
- [ORD\(\)](#)
- [QUARTER\(\)](#)
- [SECOND\(\)](#)
- [TIME_TO_SEC\(\)](#)
- [TO_DAYS\(\)](#)
- [WEEKDAY\(\)](#)
- [WEEKOFYEAR\(\)](#)
- [YEAR\(\)](#)
- [YEARWEEK\(\)](#)

- 重要サーバ SQL モード次第で、いくつもの MySQL 関数や演算子の結果が変更される可能性に注意してください。このため、パーティショニングされたテーブルを作成したあとモードを変更することは推奨できません。[SQL モード](#)を参照してください。
- [ASCII\(\)](#) や [ORD\(\)](#) といった関数を使用して文字列を(たとえば [CHAR](#) あるいは [VARCHAR](#) カラム)整数に変換するのは、文字列が8ビットキャラクタセットを使用している時のみ可能です。文字列に使用される照合順序は関連キャラクタセットのどの照合順序でも可能です。ただし、[latin1_german2_ci](#)、[latin2_czech_cs](#)、そして [cp1250_czech_cs](#) などの照合順序は1対複数の変換を擁するため、使用は不可能です。
- パーティションの最大数は 1024 になります。これはサブパーティションを含めた値です。

もし、多くのパーティションを使用してテーブルを作成する場合(しかし上記の最大数よりも少ない場合)、以下のエラーメッセージが発生します。[Got error 24 from storage engine](#) これは、[open_files_limit](#) システム変数の値を増加させなければいけないという意味です。['File' Not Found and Similar Errors](#)を参照してください。
- パーティショニングされたテーブルは外部キーをサポートしません。これは、[InnoDB](#)ストレージエンジンを使用しているテーブルも含まれます。
- パーティショニングされたテーブルは [FULLTEXT](#) をサポートしません。これは、[MyISAM](#) ストレージエンジンを使用しているテーブルも含まれます。
- パーティショニングされたテーブルは [GEOMETRY](#) カラムをサポートしません。
- MySQL 5.1.8 以降、テンポラリテーブルはパーティショニングできません。(Bug #17497)
- [MERGE](#) ストレージ エンジンを使用しているテーブルはパーティショニングできません。

[FEDERATED](#) テーブルのパーティショニングはサポートされていません。MySQL 5.1.15 からは、パーティショニングされた [FEDERATED](#) テーブルを作成する事は不可能になりました。将来的に、この制限をMySQLから取り除く方向で開発を進めています。

[CSV](#) ストレージ エンジンを使ったパーティショニングはサポートされません。MySQL 5.1.12 からは、パーティショニングされた [CSV](#) テーブルを作成する事は不可能になりました。

MySQL 5.1.6 以前では、[BLACKHOLE](#) ストレージエンジンを使用しているテーブルはパーティショニング不可能でした。

[KEY](#) (あるいは [LINEAR KEY](#))によるパーティショニングが、[NDB](#) ストレージエンジンでサポートされる唯一のパーティショニングです。MySQL 5.1.12 で始まり、[\[LINEAR\] KEY](#)以外のパーティショニングのタイプを使用してクラスタテーブルを作成するのが不可能になりました。実行するとエラーが発生します。
- アップグレードを実行中 [KEY](#) によりパーティショニングされ、[NDBCLUSTER](#) 以外のストレージエンジンを使用しているテーブルは、いったんダンプしてからリストアする必要があります。
- テーブルのパーティションとサブパーティションすべてが(後者の場合、存在していれば)同じストレージ エンジンを使用していなければいけません。将来的に、この制限をMySQLから取り除く方向で開発を進めています。
- パーティショニングキーは整数カラム、もしくは整数に帰結する表現でなければいけません。カラム、もしくは表現値は [NULL](#) となりえます。(詳しくは [MySQLパーティショニングの NULL 値の取り扱い](#) をご確認ください。)

この規制にたいする唯一の例外は[\[LINEAR\] KEY](#) — を使用してパーティショニングする際に発生します。この時、パーティショニングキーとして他タイプのカラムを使用することができるのは、— MySQL の内部キーハッシュ関数はこれらの型から正しいデータ型を生成するからです。例えば、以下の [CREATE TABLE](#) ステートメントは有効です。

```
CREATE TABLE tkc (c1 CHAR)
PARTITION BY KEY(c1)
PARTITIONS 4;
```

この例外は **BLOB** や **TEXT** カラム型に 適用されません。

- サブクエリが整数値もしくは **NULL** に帰結するとしても、サブクエリをパーティショニングキーとして利用することは出来ません。
-
- サブパーティショニングは **HASH** や **KEY** パーティショニングに限定されます。**HASH** や **KEY** パーティショニングに対してサブパーティショニングを行うことはできません。

第16章 Spatial Extensions

目次

Introduction to MySQL Spatial Support	1030
The OpenGIS Geometry Model	1030
The Geometry Class Hierarchy	1030
Class Geometry	1031
Class Point	1032
Class Curve	1033
Class LineString	1033
Class Surface	1033
Class Polygon	1033
Class GeometryCollection	1034
Class MultiPoint	1034
Class MultiCurve	1034
Class MultiLineString	1035
Class MultiSurface	1035
Class MultiPolygon	1035
Supported Spatial Data Formats	1035
Well-Known Text (WKT) Format	1035
Well-Known Binary (WKB) Format	1036
Creating a Spatially Enabled MySQL Database	1037
MySQL Spatial Data Types	1037
Creating Spatial Values	1037
Creating Spatial Columns	1040
Populating Spatial Columns	1040
Fetching Spatial Data	1041
Analyzing Spatial Information	1041
Geometry Format Conversion Functions	1041
Geometry Functions	1042
Functions That Create New Geometries from Existing Ones	1047
Functions for Testing Spatial Relations Between Geometric Objects	1047
Relations on Geometry Minimal Bounding Rectangles (MBRs)	1048
Functions That Test Spatial Relationships Between Geometries	1048
Optimizing Spatial Analysis	1049
Creating Spatial Indexes	1050
Using a Spatial Index	1051
MySQL Conformance and Compatibility	1052

MySQL supports spatial extensions to allow the generation, storage, and analysis of geographic features. These features are available for [MyISAM](#), [InnoDB](#), [NDB](#), and [ARCHIVE](#) tables. (However, the [ARCHIVE](#) engine does not support indexing, so spatial columns in [ARCHIVE](#) columns cannot be indexed. MySQL Cluster also does not support indexing of spatial columns.)

Although spatial extensions are supported in [InnoDB](#) tables, use of spatial indexes may cause a crash. (Bug #15860)

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

Additional resources

- The Open Geospatial Consortium publishes the OpenGIS® Simple Features Specifications For SQL, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>.
- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: <http://forums.mysql.com/list.php?23>.

Introduction to MySQL Spatial Support

MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at <http://www.opengis.org/>.

In 1997, the Open Geospatial Consortium published the OpenGIS® Simple Features Specifications For SQL, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>. It contains additional information relevant to this chapter.

MySQL implements a subset of the SQL with Geometry Types environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describes a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A geographic feature is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, a postcode area, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term geospatial feature to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word geometry meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: geographic feature, geospatial feature, feature, or geometry. Here, the term most commonly used is geometry, defined as a point or an aggregate of points representing anything in the world that has a location.

The OpenGIS Geometry Model

The set of geometry types proposed by OGC's SQL with Geometry Types environment is based on the OpenGIS Geometry Model. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- **Geometry** (non-instantiable)
 - **Point** (instantiable)

- `Curve` (non-instantiable)
 - `LineString` (instantiable)
 - `Line`
 - `LinearRing`
- `Surface` (non-instantiable)
 - `Polygon` (instantiable)
- `GeometryCollection` (instantiable)
 - `MultiPoint` (instantiable)
 - `MultiCurve` (non-instantiable)
 - `MultiLineString` (instantiable)
 - `MultiSurface` (non-instantiable)
 - `MultiPolygon` (instantiable)

It is not possible to create objects in non-instantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.
- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as non-instantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

Class `Geometry`

`Geometry` is the root class of the hierarchy. It is a non-instantiable class but has a number of properties that are common to all geometry values created from any of the `Geometry` subclasses. These properties are described in the following list. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its type. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its SRID, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

- Its coordinates in its Spatial Reference System, represented as double-precision (eight-byte) numbers. All non-empty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the planar coordinate system and the distance on the geocentric system (coordinates on the Earth's surface) are different things.

- Its interior, boundary, and exterior.

Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its MBR (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- Whether the value is simple or non-simple. Geometry values of types ([LineString](#), [MultiPoint](#), [MultiLineString](#)) are either simple or non-simple. Each type determines its own assertions for being simple or non-simple.
- Whether the value is closed or not closed. Geometry values of types ([LineString](#), [MultiString](#)) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is empty or non-empty. A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a [NULL](#) value). An empty geometry is defined to be always simple and has an area of 0.
- Its dimension. A geometry can have a dimension of -1, 0, 1, or 2:
 - -1 for an empty geometry.
 - 0 for a geometry with no length and no area.
 - 1 for a geometry with non-zero length and zero area.
 - 2 for a geometry with non-zero area.

[Point](#) objects have a dimension of zero. [LineString](#) objects have a dimension of 1. [Polygon](#) objects have a dimension of 2. The dimensions of [MultiPoint](#), [MultiLineString](#), and [MultiPolygon](#) objects are the same as the dimensions of the elements they consist of.

Class [Point](#)

A [Point](#) is a geometry that represents a single location in coordinate space.

[Point](#) Examples

- Imagine a large-scale map of the world with many cities. A [Point](#) object could represent each city.
- On a city map, a [Point](#) object could represent a bus stop.

[Point](#) Properties

- X-coordinate value.
- Y-coordinate value.
- [Point](#) is defined as a zero-dimensional geometry.
- The boundary of a [Point](#) is the empty set.

Class [Curve](#)

A [Curve](#) is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of [Curve](#) define the type of interpolation between points. [Curve](#) is a non-instantiable class.

[Curve](#) Properties

- A [Curve](#) has the coordinates of its points.
- A [Curve](#) is defined as a one-dimensional geometry.
- A [Curve](#) is simple if it does not pass through the same point twice.
- A [Curve](#) is closed if its start point is equal to its endpoint.
- The boundary of a closed [Curve](#) is empty.
- The boundary of a non-closed [Curve](#) consists of its two endpoints.
- A [Curve](#) that is simple and closed is a [LinearRing](#).

Class [LineString](#)

A [LineString](#) is a [Curve](#) with linear interpolation between points.

[LineString](#) Examples

- On a world map, [LineString](#) objects could represent rivers.
- In a city map, [LineString](#) objects could represent streets.

[LineString](#) Properties

- A [LineString](#) has coordinates of segments, defined by each consecutive pair of points.
- A [LineString](#) is a [Line](#) if it consists of exactly two points.
- A [LineString](#) is a [LinearRing](#) if it is both closed and simple.

Class [Surface](#)

A [Surface](#) is a two-dimensional geometry. It is a non-instantiable class. Its only instantiable subclass is [Polygon](#).

[Surface](#) Properties

- A [Surface](#) is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple [Surface](#) as a geometry that consists of a single 「patch」 that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple [Surface](#) is the set of closed curves corresponding to its exterior and interior boundaries.

Class [Polygon](#)

A [Polygon](#) is a planar [Surface](#) representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the [Polygon](#).

[Polygon](#) Examples

- On a region map, [Polygon](#) objects could represent forests, districts, and so on.

[Polygon](#) Assertions

- The boundary of a `Polygon` consists of a set of `LinearRing` objects (that is, `LineString` objects that are both simple and closed) that make up its exterior and interior boundaries.
- A `Polygon` has no rings that cross. The rings in the boundary of a `Polygon` may intersect at a `Point`, but only as a tangent.
- A `Polygon` has no lines, spikes, or punctures.
- A `Polygon` has an interior that is a connected point set.
- A `Polygon` may have holes. The exterior of a `Polygon` with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a `Polygon` a simple geometry.

Class `GeometryCollection`

A `GeometryCollection` is a geometry that is a collection of one or more geometries of any class.

All the elements in a `GeometryCollection` must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a `GeometryCollection`, although the subclasses of `GeometryCollection` described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a `MultiPoint` may contain only `Point` elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

Class `MultiPoint`

A `MultiPoint` is a geometry collection composed of `Point` elements. The points are not connected or ordered in any way.

`MultiPoint` Examples

- On a world map, a `MultiPoint` could represent a chain of small islands.
- On a city map, a `MultiPoint` could represent the outlets for a ticket office.

`MultiPoint` Properties

- A `MultiPoint` is a zero-dimensional geometry.
- A `MultiPoint` is simple if no two of its `Point` values are equal (have identical coordinate values).
- The boundary of a `MultiPoint` is the empty set.

Class `MultiCurve`

A `MultiCurve` is a geometry collection composed of `Curve` elements. `MultiCurve` is a non-instantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the 「mod 2 union rule」 (also known as the 「odd-even rule」): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `MultiCurve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

Class `MultiLineString`

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

Class `MultiSurface`

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a non-instantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Two `MultiSurface` surfaces have no interiors that intersect.
- Two `MultiSurface` elements have boundaries that intersect at most at a finite number of points.

Class `MultiPolygon`

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.
- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.
- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each `Curve` in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every `Curve` in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form.

Examples of WKT representations of geometry objects:

- A [Point](#):

```
POINT(15 20)
```

Note that point coordinates are specified with no separating comma.

- A [LineString](#) with four points:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

Note that point coordinate pairs are separated by commas.

- A [Polygon](#) with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A [MultiPoint](#) with three [Point](#) values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A [MultiLineString](#) with two [LineString](#) values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A [MultiPolygon](#) with two [Polygon](#) values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A [GeometryCollection](#) consisting of two [Point](#) values and one [LineString](#):

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

A Backus-Naur grammar that specifies the formal production rules for writing WKT values can be found in the OpenGIS specification document referenced near the beginning of this chapter.

Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specification. It is also defined in the ISO SQL/MM Part 3: Spatial standard.

WKB is used to exchange geometry data as binary streams represented by [BLOB](#) values containing geometric WKB information.

WKB uses one-byte unsigned integers, four-byte unsigned integers, and eight-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to [POINT\(1 1\)](#) consists of this sequence of 21 bytes (each represented here by two hex digits):

```
0101000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F
```

Component representation is as follows:

- The byte order may be either 0 or 1 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.

- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate [Point](#), [LineString](#), [Polygon](#), [MultiPoint](#), [MultiLineString](#), [MultiPolygon](#), and [GeometryCollection](#).
- A [Point](#) value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- [GEOMETRY](#)
- [POINT](#)
- [LINESTRING](#)
- [POLYGON](#)

[GEOMETRY](#) can store geometry values of any type. The other single-value types ([POINT](#), [LINESTRING](#), and [POLYGON](#)) restrict their values to a particular geometry type.

The other data types hold collections of values:

- [MULTIPOINT](#)
- [MULTILINESTRING](#)
- [MULTIPOLYGON](#)
- [GEOMETRYCOLLECTION](#)

[GEOMETRYCOLLECTION](#) can store a collection of objects of any type. The other collection types ([MULTIPOINT](#), [MULTILINESTRING](#), [MULTIPOLYGON](#), and [GEOMETRYCOLLECTION](#)) restrict collection members to those having a particular geometry type.

Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as input parameters a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

[GeomFromText\(\)](#) accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- [GeomCollFromText\(wkt\[,srid\]\)](#), [GeometryCollectionFromText\(wkt\[,srid\]\)](#)

Constructs a [GEOMETRYCOLLECTION](#) value using its WKT representation and SRID.

- [GeomFromText\(wkt\[,srid\]\)](#), [GeometryFromText\(wkt\[,srid\]\)](#)

Constructs a geometry value of any type using its WKT representation and SRID.

- [LineFromText\(wkt\[,srid\]\)](#), [LineStringFromText\(wkt\[,srid\]\)](#)

Constructs a [LINESTRING](#) value using its WKT representation and SRID.

- [MLineFromText\(wkt\[,srid\]\)](#), [MultiLineStringFromText\(wkt\[,srid\]\)](#)

Constructs a [MULTILINESTRING](#) value using its WKT representation and SRID.

- [MPointFromText\(wkt\[,srid\]\)](#), [MultiPointFromText\(wkt\[,srid\]\)](#)

Constructs a [MULTIPOINT](#) value using its WKT representation and SRID.

- [MPolyFromText\(wkt\[,srid\]\)](#), [MultiPolygonFromText\(wkt\[,srid\]\)](#)

Constructs a [MULTIPOLYGON](#) value using its WKT representation and SRID.

- [PointFromText\(wkt\[,srid\]\)](#)

Constructs a [POINT](#) value using its WKT representation and SRID.

- [PolyFromText\(wkt\[,srid\]\)](#), [PolygonFromText\(wkt\[,srid\]\)](#)

Constructs a [POLYGON](#) value using its WKT representation and SRID.

The OpenGIS specification also defines the following optional functions, which MySQL does not implement. These functions construct [Polygon](#) or [MultiPolygon](#) values based on the WKT representation of a collection of rings or closed [LineString](#) values. These values may intersect.

- [BdMPolyFromText\(wkt,srid\)](#)

Constructs a [MultiPolygon](#) value from a [MultiLineString](#) value in WKT format containing an arbitrary collection of closed [LineString](#) values.

- [BdPolyFromText\(wkt,srid\)](#)

Constructs a [Polygon](#) value from a [MultiLineString](#) value in WKT format containing an arbitrary collection of closed [LineString](#) values.

Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as input parameters a [BLOB](#) containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

[GeomFromWKB\(\)](#) accepts a WKB of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- [GeomCollFromWKB\(wkb\[,srid\]\)](#), [GeometryCollectionFromWKB\(wkb\[,srid\]\)](#)

Constructs a [GEOMETRYCOLLECTION](#) value using its WKB representation and SRID.

- [GeomFromWKB\(wkb\[,srid\]\)](#), [GeometryFromWKB\(wkb\[,srid\]\)](#)

Constructs a geometry value of any type using its WKB representation and SRID.

- [LineFromWKB\(wkb\[,srid\]\)](#), [LineStringFromWKB\(wkb\[,srid\]\)](#)

Constructs a [LINESTRING](#) value using its WKB representation and SRID.

- [MLineFromWKB\(wkb\[,srid\]\)](#), [MultiLineStringFromWKB\(wkb\[,srid\]\)](#)

Constructs a [MULTILINESTRING](#) value using its WKB representation and SRID.

- [MPointFromWKB\(wkb\[,srid\]\)](#), [MultiPointFromWKB\(wkb\[,srid\]\)](#)

Constructs a [MULTIPOINT](#) value using its WKB representation and SRID.

- [MPolyFromWKB\(wkb\[,srid\]\)](#), [MultiPolygonFromWKB\(wkb\[,srid\]\)](#)

Constructs a [MULTIPOLYGON](#) value using its WKB representation and SRID.

- [PointFromWKB\(wkb\[,srid\]\)](#)

Constructs a [POINT](#) value using its WKB representation and SRID.

- [PolyFromWKB\(wkb\[,srid\]\)](#), [PolygonFromWKB\(wkb\[,srid\]\)](#)

Constructs a [POLYGON](#) value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing [Polygon](#) or [MultiPolygon](#) values based on the WKB representation of a collection of rings or closed [LineString](#) values. These values may intersect. MySQL does not implement these functions:

- [BdMPolyFromWKB\(wkb,srid\)](#)

Constructs a [MultiPolygon](#) value from a [MultiLineString](#) value in WKB format containing an arbitrary collection of closed [LineString](#) values.

- [BdPolyFromWKB\(wkb,srid\)](#)

Constructs a [Polygon](#) value from a [MultiLineString](#) value in WKB format containing an arbitrary collection of closed [LineString](#) values.

Creating Geometry Values Using MySQL-Specific Functions

MySQL provides a set of useful non-standard functions for creating geometry WKB representations. The functions described in this section are MySQL extensions to the OpenGIS specification. The results of these functions are [BLOB](#) values containing WKB representations of geometry values with no SRID. The results of these functions can be substituted as the first argument for any function in the [GeomFromWKB\(\)](#) function family.

- [GeometryCollection\(g1,g2,...\)](#)

Constructs a WKB [GeometryCollection](#). If any argument is not a well-formed WKB representation of a geometry, the return value is [NULL](#).

- [LineString\(pt1,pt2,...\)](#)

Constructs a WKB [LineString](#) value from a number of WKB [Point](#) arguments. If any argument is not a WKB [Point](#), the return value is [NULL](#). If the number of [Point](#) arguments is less than two, the return value is [NULL](#).

- [MultiLineString\(ls1,ls2,...\)](#)

Constructs a WKB [MultiLineString](#) value using WKB [LineString](#) arguments. If any argument is not a WKB [LineString](#), the return value is [NULL](#).

- [MultiPoint\(pt1,pt2,...\)](#)

Constructs a WKB [MultiPoint](#) value using WKB [Point](#) arguments. If any argument is not a WKB [Point](#), the return value is [NULL](#).

- [MultiPolygon\(poly1,poly2,...\)](#)

Constructs a WKB [MultiPolygon](#) value from a set of WKB [Polygon](#) arguments. If any argument is not a WKB [Polygon](#), the return value is [NULL](#).

- [Point\(x,y\)](#)

Constructs a WKB [Point](#) using its coordinates.

- [Polygon\(ls1,ls2,...\)](#)

Constructs a WKB [Polygon](#) value from a number of WKB [LineString](#) arguments. If any argument does not represent the WKB of a [LinearRing](#) (that is, not a closed and simple [LineString](#)) the return value is [NULL](#).

Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with [CREATE TABLE](#) or [ALTER TABLE](#). Currently, spatial columns are supported for [MyISAM](#), [InnoDB](#), [NDB](#), and [ARCHIVE](#) tables. See also the annotations about spatial indexes under [Creating Spatial Indexes](#).

- Use the [CREATE TABLE](#) statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the [ALTER TABLE](#) statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;  
ALTER TABLE geom DROP pt;
```

Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format:

- Perform the conversion directly in the [INSERT](#) statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));  
  
SET @g = 'POINT(1 1)';  
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the [INSERT](#):

```
SET @g = GeomFromText('POINT(1 1)');  
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';  
INSERT INTO geom VALUES (GeomFromText(@g));  
  
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 5))';  
INSERT INTO geom VALUES (GeomFromText(@g));  
  
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';  
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use [GeomFromText\(\)](#) to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';  
INSERT INTO geom VALUES (PointFromText(@g));  
  
SET @g = 'LINESTRING(0 0,1 1,2 2)';  
INSERT INTO geom VALUES (LineStringFromText(@g));  
  
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 5))';  
INSERT INTO geom VALUES (PolygonFromText(@g));  
  
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';  
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a [POINT\(1 1\)](#) value with hex literal syntax:

```
mysql> INSERT INTO geom VALUES
-> (GeomFromWKB(0x010100000000000000000000F03F000000000000F03F));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of **BLOB** type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See `mysql_real_escape_string()`.

Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

- Fetching spatial data in internal format:

Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- Fetching spatial data in WKT format:

The `AsText()` function converts a geometry from internal format into a WKT string.

```
SELECT AsText(g) FROM geom;
```

- Fetching spatial data in WKB format:

The `AsBinary()` function converts a geometry from internal format into a **BLOB** containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

Analyzing Spatial Information

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats
- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as `mysql` or MySQL Query Browser
- Application programs written in any language that supports a MySQL client API

Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- `AsBinary(g)`

Converts a value in internal geometry format to its WKB representation and returns the binary result.

```
SELECT AsBinary(g) FROM geom;
```

- [AsText\(g\)](#)

Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+-----+
| AsText(GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+
```

- [GeomFromText\(wkt\[,srid\]\)](#)

Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as [PointFromText\(\)](#) and [LineFromText\(\)](#). See [Creating Geometry Values Using WKT Functions](#).

- [GeomFromWKB\(wkb\[,srid\]\)](#)

Converts a binary value from its WKB representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as [PointFromWKB\(\)](#) and [LineFromWKB\(\)](#). See [Creating Geometry Values Using WKB Functions](#).

Geometry Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return **NULL** if the argument is of an incorrect geometry type. For example, [Area\(\)](#) returns **NULL** if the object type is neither [Polygon](#) nor [MultiPolygon](#).

General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- [Dimension\(g\)](#)

Returns the inherent dimension of the geometry value [g](#). The result can be -1, 0, 1, or 2. The meaning of these values is given in [Class Geometry](#).

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+
```

- [Envelope\(g\)](#)

Returns the Minimum Bounding Rectangle (MBR) for the geometry value [g](#). The result is returned as a [Polygon](#) value.

The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| POLYGON((1 1,2 2,2 1,1 1)) |
+-----+
```

- [GeometryType\(g\)](#)

Returns as a string the name of the geometry type of which the geometry instance `g` is a member. The name corresponds to one of the instantiable [Geometry](#) subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT |
+-----+
```

- [SRID\(g\)](#)

Returns an integer indicating the Spatial Reference System ID for the geometry value `g`.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
| 101 |
+-----+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- [Boundary\(g\)](#)

Returns a geometry that is the closure of the combinatorial boundary of the geometry value `g`.

- [IsEmpty\(g\)](#)

Returns 1 if the geometry value `g` is the empty geometry, 0 if it is not empty, and -1 if the argument is `NULL`. If the geometry is empty, it represents the empty point set.

- [IsSimple\(g\)](#)

Currently, this function is a placeholder and should not be used. If implemented, its behavior will be as described in the next paragraph.

Returns 1 if the geometry value `g` has no anomalous geometric points, such as self-intersection or self-tangency. [IsSimple\(\)](#) returns 0 if the argument is not simple, and -1 if it is `NULL`.

The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See [The Geometry Class Hierarchy](#).)

Point Functions

A [Point](#) consists of X and Y coordinates, which may be obtained using the following functions:

- [X\(p\)](#)

Returns the X-coordinate value for the point `p` as a double-precision number.

```
mysql> SET @pt = 'Point(56.7 53.34)';
mysql> SELECT X(GeomFromText(@pt));
+-----+
| X(GeomFromText(@pt)) |
+-----+
| 56.7 |
+-----+
```

- [Y\(p\)](#)

Returns the Y-coordinate value for the point `p` as a double-precision number.

```
mysql> SET @pt = 'Point(56.7 53.34)';
```

```
mysql> SELECT Y(GeomFromText(@pt));
+-----+
| Y(GeomFromText(@pt)) |
+-----+
|          53.34 |
+-----+
```

LineString Functions

A **LineString** consists of **Point** values. You can extract particular points of a **LineString**, count the number of points that it contains, or obtain its length.

- **EndPoint(Is)**

Returns the **Point** that is the endpoint of the **LineString** value **Is**.

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@Is)));
+-----+
| AsText(EndPoint(GeomFromText(@Is))) |
+-----+
| POINT(3 3) |
+-----+
```

- **GLength(Is)**

Returns as a double-precision number the length of the **LineString** value **Is** in its associated spatial reference.

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@Is));
+-----+
| GLength(GeomFromText(@Is)) |
+-----+
|          2.8284271247462 |
+-----+
```

GLength() is a non-standard name. It corresponds to the OpenGIS **Length()** function.

- **NumPoints(Is)**

Returns the number of **Point** objects in the **LineString** value **Is**.

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@Is));
+-----+
| NumPoints(GeomFromText(@Is)) |
+-----+
|          3 |
+-----+
```

- **PointN(Is,N)**

Returns the **N**-th **Point** in the **LineString** value **Is**. Points are numbered beginning with 1.

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@Is),2));
+-----+
| AsText(PointN(GeomFromText(@Is),2)) |
+-----+
| POINT(2 2) |
+-----+
```

- **StartPoint(Is)**

Returns the **Point** that is the start point of the **LineString** value **Is**.

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@Is)));
+-----+
| AsText(StartPoint(GeomFromText(@Is))) |
+-----+
```

```
| POINT(1 1) |
+-----+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- [IsRing\(ls\)](#)

Returns 1 if the [LineString](#) value *ls* is closed (that is, its [StartPoint\(\)](#) and [EndPoint\(\)](#) values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *ls* is not a ring, and -1 if it is [NULL](#).

MultiLineString Functions

- [GLength\(mls\)](#)

Returns as a double-precision number the length of the [MultiLineString](#) value *mls*. The length of *mls* is equal to the sum of the lengths of its elements.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5));'
mysql> SELECT GLength(GeomFromText(@mls));
+-----+
| GLength(GeomFromText(@mls)) |
+-----+
| 4.2426406871193 |
+-----+
```

[GLength\(\)](#) is a non-standard name. It corresponds to the OpenGIS [Length\(\)](#) function.

- [IsClosed\(mls\)](#)

Returns 1 if the [MultiLineString](#) value *mls* is closed (that is, the [StartPoint\(\)](#) and [EndPoint\(\)](#) values are the same for each [LineString](#) in *mls*). Returns 0 if *mls* is not closed, and -1 if it is [NULL](#).

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5));'
mysql> SELECT IsClosed(GeomFromText(@mls));
+-----+
| IsClosed(GeomFromText(@mls)) |
+-----+
| 0 |
+-----+
```

Polygon Functions

- [Area\(poly\)](#)

Returns as a double-precision number the area of the [Polygon](#) value *poly*, as measured in its spatial reference system.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1));'
mysql> SELECT Area(GeomFromText(@poly));
+-----+
| Area(GeomFromText(@poly)) |
+-----+
| 4 |
+-----+
```

- [ExteriorRing\(poly\)](#)

Returns the exterior ring of the [Polygon](#) value *poly* as a [LineString](#).

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1));'
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----+
| LINESTRING(0 0,0 3,3 0,0 0) |
+-----+
```

- [InteriorRingN\(poly,N\)](#)

Returns the **N**-th interior ring for the **Polygon** value **poly** as a **LineString**. Rings are numbered beginning with 1.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));'
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1) |
+-----+
```

- **NumInteriorRings(poly)**

Returns the number of interior rings in the **Polygon** value **poly**.

```
mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));'
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+-----+
| NumInteriorRings(GeomFromText(@poly)) |
+-----+
| 1 |
+-----+
```

MultiPolygon Functions

- **Area(mpoly)**

Returns as a double-precision number the area of the **MultiPolygon** value **mpoly**, as measured in its spatial reference system.

```
mysql> SET @mpoly =
-> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+-----+
| Area(GeomFromText(@mpoly)) |
+-----+
| 8 |
+-----+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- **Centroid(mpoly)**

Returns the mathematical centroid for the **MultiPolygon** value **mpoly** as a **Point**. The result is not guaranteed to be on the **MultiPolygon**.

- **PointOnSurface(mpoly)**

Returns a **Point** value that is guaranteed to be on the **MultiPolygon** value **mpoly**.

GeometryCollection Functions

- **GeometryN(gc,N)**

Returns the **N**-th geometry in the **GeometryCollection** value **gc**. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3));'
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+-----+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+-----+
| POINT(1 1) |
+-----+
```

- **NumGeometries(gc)**

Returns the number of geometries in the **GeometryCollection** value **gc**.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3));'
mysql> SELECT NumGeometries(GeomFromText(@gc));
+-----+
| NumGeometries(GeomFromText(@gc)) |
+-----+
| 2 |
+-----+
```

Functions That Create New Geometries from Existing Ones

Geometry Functions That Produce New Geometries

[Geometry Functions](#), discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- [Envelope\(g\)](#)
- [StartPoint\(ls\)](#)
- [EndPoint\(ls\)](#)
- [PointN\(ls,N\)](#)
- [ExteriorRing\(poly\)](#)
- [InteriorRingN\(poly,N\)](#)
- [GeometryN\(gc,N\)](#)

Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

These functions are not implemented in MySQL. They may appear in future releases.

- [Buffer\(g,d\)](#)
Returns a geometry that represents all points whose distance from the geometry value [g](#) is less than or equal to a distance of [d](#).
- [ConvexHull\(g\)](#)
Returns a geometry that represents the convex hull of the geometry value [g](#).
- [Difference\(g1,g2\)](#)
Returns a geometry that represents the point set difference of the geometry value [g1](#) with [g2](#).
- [Intersection\(g1,g2\)](#)
Returns a geometry that represents the point set intersection of the geometry values [g1](#) with [g2](#).
- [SymDifference\(g1,g2\)](#)
Returns a geometry that represents the point set symmetric difference of the geometry value [g1](#) with [g2](#).
- [Union\(g1,g2\)](#)
Returns a geometry that represents the point set union of the geometry values [g1](#) and [g2](#).

Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides several functions that test relations between minimal bounding rectangles of two geometries *g1* and *g2*. The return values 1 and 0 indicate true and false, respectively.

- **MBRContains(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
```

```
+-----+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+-----+
| 1 | 0 |
+-----+
```

- **MBRDisjoint(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- **MBREqual(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- **MBRIntersects(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- **MBROverlaps(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap.

- **MBRTouches(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch.

- **MBRWithin(*g1*,*g2*)**

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
```

```
+-----+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+
| 1 | 0 |
+-----+
```

Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. They test the relationship between two geometry values *g1* and *g2*.

Currently, MySQL does not implement these functions according to the specification. Those that are implemented return the same result as the corresponding MBR-based functions. This includes functions in the following list other than **Distance()** and **Related()**.

These functions may be implemented in future releases with full support for spatial analysis, not just MBR-based support.

The return values 1 and 0 indicate true and false, respectively.

- [Contains\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) completely contains [g2](#).

- [Crosses\(g1,g2\)](#)

Returns 1 if [g1](#) spatially crosses [g2](#). Returns [NULL](#) if [g1](#) is a [Polygon](#) or a [MultiPolygon](#), or if [g2](#) is a [Point](#) or a [MultiPoint](#). Otherwise, returns 0.

The term spatially crosses denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

- [Disjoint\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) is spatially disjoint from (does not intersect) [g2](#).

- [Distance\(g1,g2\)](#)

Returns as a double-precision number the shortest distance between any two points in the two geometries.

- [Equals\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) is spatially equal to [g2](#).

- [Intersects\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) spatially intersects [g2](#).

- [Overlaps\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) spatially overlaps [g2](#). The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- [Related\(g1,g2,pattern_matrix\)](#)

Returns 1 or 0 to indicate whether the spatial relationship specified by [pattern_matrix](#) exists between [g1](#) and [g2](#). Returns -1 if the arguments are [NULL](#). The pattern matrix is a string. Its specification will be noted here if this function is implemented.

- [Touches\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) spatially touches [g2](#). Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- [Within\(g1,g2\)](#)

Returns 1 or 0 to indicate whether [g1](#) is spatially within [g2](#).

Optimizing Spatial Analysis

Search operations in non-spatial databases can be optimized using indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point

- Region queries that search for all objects that overlap a given region

MySQL uses R-Trees with quadratic splitting to index spatial columns. A spatial index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. You are required to declare a prefix for any (non-spatial) index on a spatial column except for **POINT** columns.

Creating Spatial Indexes

MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the **SPATIAL** keyword. Currently, spatial columns that are indexed must be declared **NOT NULL**. The following examples demonstrate how to create spatial indexes:

- With **CREATE TABLE**:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- With **ALTER TABLE**:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With **CREATE INDEX**:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

For **MyISAM** tables, **SPATIAL INDEX** creates an R-tree index. For other storage engines that support spatial indexing, **SPATIAL INDEX** creates a B-tree index. A B-tree index on spatial values will be useful for exact-value lookups, but not for range scans.

To drop spatial indexes, use **ALTER TABLE** or **DROP INDEX**:

- With **ALTER TABLE**:

```
ALTER TABLE geom DROP INDEX g;
```

- With **DROP INDEX**:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table **geom** contains more than 32,000 geometries, which are stored in the column **g** of type **GEOMETRY**. The table also has an **AUTO_INCREMENT** column **fid** for storing object ID values.

```
mysql> DESCRIBE geom;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| fid   | int(11) |      | PRI | NULL    | auto_increment |
| g     | geometry |      |      |          |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+-----+
| count(*) |
+-----+
| 32376    |
+-----+
1 row in set (0.00 sec)
```

To add a spatial index on the column **g**, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0
```

Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` or `MBRWithin()` in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
            31000 15000,
            31000 16000,
            30000 16000,
            30000 15000));'
mysql> SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g);
+----+-----+
| fid | AsText(g) |
+----+-----+
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+----+-----+
20 rows in set (0.00 sec)
```

Use `EXPLAIN` to check the way this query is executed:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
            31000 15000,
            31000 16000,
            30000 16000,
            30000 15000));'
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
         key: g
        key_len: 32
         ref: NULL
         rows: 50
   Extra: Using where
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
            31000 15000,
            31000 16000,
            30000 16000,
            30000 15000));'
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
-> MBRContains(GeomFromText(@poly),g)\G
```

```
***** 1. row *****
id: 1
select_type: SIMPLE
table: geom
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 32376
Extra: Using where
1 row in set (0.00 sec)
```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
           31000 15000,
           31000 16000,
           30000 16000,
           30000 15000));'
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
-> MBRContains(GeomFromText(@poly),g);
+----+-----+
| fid | AsText(g) |
+----+-----+
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136. ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+----+-----+
20 rows in set (0.46 sec)
```

In future releases, spatial indexes may also be used for optimizing other functions. See [Functions for Testing Spatial Relations Between Geometric Objects](#).

MySQL Conformance and Compatibility

MySQL does not yet implement the following GIS features:

- Additional Metadata Views

OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()`

The problem is that there is an existing SQL function `Length()` that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

第17章 ストアドプロシージャとファンクション

目次

ストアドルーチンとgrantテーブル	1054
ストアドルーチン構文	1054
CREATE PROCEDUREおよびCREATE FUNCTION 構文	1054
ALTER PROCEDURE および ALTER FUNCTION 構文	1058
DROP PROCEDURE および DROP FUNCTION 構文	1058
CALLステートメント構文	1058
BEGIN ... END 複合ステートメント構文	1059
DECLAREステートメント用構文	1060
ストアドルーチン内の変数	1060
条件とハンドラ	1061
カーソル	1063
フローコントロール・コンストラクト	1064
ストアドプロシージャ、ファンクション、トリガ並びにLAST_INSERT_ID()	1066
ストアドルーチンとトリガのバイナリログ	1066

ストアドルーチン（プロシージャとファンクション）が MySQL 5.1ではサポートされています。ストアドプロシージャはサーバが保存することができるSQLステートメントの組です。これが実行されると、クライアントは各ステートメントを発行し続ける必要がなくなり、代わりにストアドプロシージャを参照します。

MySQL中に保存されているルーチンとその利用に関する一般質問に対する答えは、[MySQL 5.1 FAQ — Stored Procedures](#)で見ることができます。更に、ストアドルーチンに関する一般的な質問に対する答えについては、[MySQL 5.1 FAQ — Stored Routines, Triggers, and Replication](#)をご参照ください。

ストアドルーチンが特に有用な幾つかの状況：

- 複数のクライアントアプリケーションが、複数の言語で書かれている場合、または異なるプラットフォームで動作するが、同じデータベースオペレーションを行う必要がある場合。
- セキュリティを最優先する場合。例えば、銀行はすべての共通オペレーションにストアドプロシージャとファンクションを使います。これは一貫して安全な環境を提供するので、ルーチンは各オペレーションが適切にログされていることを保証します。このようなセットアップでは、アプリケーションとユーザは直接データベーステーブルにアクセス権は無く、特定のストアドルーチンのみ実行することができます。

サーバとクライアント間の通信を減らすことができるので、ストアドルーチンは性能を向上させます。より多くの作業がサーバ側で実行され、クライアント（アプリケーション）側では、より少ない作業が実行されるので、欠点はこれがデータベースサーバ上の負荷を増やすということです。（Web サーバのような）多くのクライアントマシンに対して、1つあるいは少数のデータベースサーバによってメンテナンスされる場合、これを考慮に入れてください。

ストアドルーチンはユーザがデータベースサーバの中にファンクションライブラリを持つことも許します。内部にこのようなデザインを許容する（例えば、クラスを使うことによって）最近のアプリケーション言語が共有している特徴です。これらのクライアントアプリケーション言語の特徴を使用することは、データベースの使用範囲の外でもプログラマーにとって有益です。

MySQLはストアドルーチンに対して、IBMのDB2にも使われているSQL:2003構文に準じています。

ストアドルーチンのMySQLへの実装は進行途中です。この章で述べたすべての構文はサポートされており、全ての制約や拡張は適切文書化されます。ストアドルーチンの使用に対する制限については、[ストアドルーチンとトリガの規制](#)を参照してください。

ストアドルーチンのバイナリ ログ実行については、[ストアドルーチンとトリガのバイナリログ](#)を参照してください。

再帰的なストアドプロシージャは、デフォルトで無効化されていますが、[max_sp_recursion_depth](#)サーバシステム変数をゼロの値に設定することによって、サーバ上で有効化することができます。詳細については、[システム変数](#)をご参照してください。

保存されたファンクションは再帰的にはなり得ません。[ストアドルーチンとトリガの規制](#)を参照してください。

ストアドルーチンとgrantテーブル

ストアドルーチンでは、[mysql](#)データベース中に[proc](#)テーブルが要求されます。このテーブルはMySQL 5.1をインストールしている最中に生成されます。旧バージョンのMySQLからMySQL 5.1にアップグレードする場合、ユーザのgrantテーブルが更新され、[proc](#)テーブルが存在しているか確認してください。[mysql_upgrade — MySQL アップグレードのテーブルチェック](#)を参照してください。

サーバはストアドルーチンを生成、変更もしくは撤去するステートメントに対して、[mysql.proc](#)テーブルを操作します。このテーブルの手動操作のサポートは、サーバに通知されません。

MySQL grantシステムはストアドルーチンを以下の通り取り扱います。

- スタドルーチンを生成するため、[CREATE ROUTINE](#)特権が必要です。
- スタドルーチンの変更・撤去には、[ALTER ROUTINE](#)権限が必要です。必要な場合、この権限はルーチン生成者に自動的に与えられますが、生成者がルーチンを廃止すると、権限も消滅します。
- スタドルーチンを実行するため、[EXECUTE](#)権限が要求されます。必要な場合、この権限はルーチン生成者に自動的に与えられます。(生成者がルーチンを撤去すると、権限も消滅します)ルーチンのデフォルト設定[SQL SECURITY](#)も[DEFINER](#)です。これは、ルーチンに関連するデータベースにアクセス可能なユーザがルーチンを実行できるようにします。
- [automatic_sp_privileges](#)システム変数がゼロである場合、[EXECUTE](#)および[ALTER ROUTINE](#)権限は自動的に供与・除去されません。

ストアドルーチン構文

ストアドルーチンはプロシージャかファンクションのいずれかです。ストアドルーチンは[CREATE PROCEDURE](#)および[CREATE FUNCTION](#)ステートメントを使って作成されます。プロシージャは[CALL](#)ステートメントを使って起動し、アウトプット変数を使ってのみ値を返すことができます。関数(ファンクション)は(関数名を呼び出す方法を採用している)他の関数のように、ステートメントの内側から呼び出して、スカラー値を返すことができます。ストアドルーチンは他のストアドルーチンを呼び出すことができます。

ストアドルーチンもしくはファンクションは特定データベースに関連します。これは複数の意味を含んでいます。

- ルーチンを呼び出すと、必然的に [USE db_name](#)が実行されます(ルーチンの実行が終了すると停止します)。ストアドルーチンの中で[USE](#)ステートメントの使用は禁止されています。
- データベース名を使ってルーチン名を認定することができます。これは現在データベース中に含まれていないルーチンを参照するのに使用することができます。例えば、[test](#)データベースに関連するストアドルーチン [p](#) またはファンクション [f](#) を呼び出すため、[CALL test.p\(\)](#)または[test.f\(\)](#)とすることができます。
- データベースを撤去すると、それに関連する一切のストアドルーチンも撤去されます。

MySQLは、ストアドルーチンの中に含まれるレギュラー[SELECT](#)ステートメントの(カーソルまたはローカル変数なしで)の使用を可能にする非常に有用な拡張機能をサポートしています。このようなクエリーに対する結果セットは、簡単に直接クライアントに送られます。複数の[SELECT](#)ステートメントは複数の結果セットを生成するので、クライアントは複数の結果セットをサポートしているMySQLクライアント・ライブラリーを使用しなければなりません。これは、クライアントは少なくとも4.1より新しいバージョンのMySQLから取得したクライアント・ライブラリーを使用しなければならないことを意味します。クライアントは接続時、[CLIENT_MULTI_RESULTS](#)オプションも特定しなければなりません。Cプログラムに対しては、[mysql_real_connect\(\)](#)C API関数を使って実施することができます。[mysql_real_connect\(\)](#)、[マルチプルステートメントを実行するC APIハンドリング](#)を参照して下さい。

以下のセクションでは、ストアドルーチンとファンクションを生成、変更、並びに起動を実行するのに使用する構文について説明します。

CREATE PROCEDUREおよびCREATE FUNCTION 構文

CREATE


```

[DEFINER = { user | CURRENT_USER }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body

CREATE
[DEFINER = { user | CURRENT_USER }]
FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body

proc_parameter:
[ IN | OUT | INOUT ] param_name type

func_parameter:
param_name type

type:
Any valid MySQL data type

characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'

routine_body:
Valid SQL procedure statement

```

これらのステートメントはストアドルーチンを生成します。これらを使用するには[CREATE ROUTINE](#)権限を持つていることが必要です。バイナリログが有効化されている場合、[CREATE FUNCTION](#)ステートメントにも[SUPER](#)権限が**ストアドルーチンとトリガのバイナリログ**で述べた通り、要求されます。MySQLは[ALTER ROUTINE](#)権限と[EXECUTE](#)権限をルーチン生成者に自動的に供与します。

デフォルトで、そのルーチンは初期設定データベースに関連されます。あるデータベース中にルーチンを明確に関連させるには、生成時その名称を`db_name.sp_name`と特定してください。

ルーチン名が組み込まれているSQL機能と同じである場合、ルーチンを定義する時、名称とその後のかっこの間にスペースを使用しなければなりません。これを怠ると、構文エラーが発生します。これは、後にルーチンを呼び出す場合にも当てはまります。この理由によって、保存されているユーザ自身のルーチンに対してSQLの既存のファンクション名称を再使用しないようお勧めします。

[IGNORE_SPACE SQL](#)モードは、ストアドルーチンでなく、組み込まれているファンクションに適用され、[IGNORE_SPACE](#)が有効になっているか否かにかかわらず、ルーチン名の後にスペースを含むことは常に許容されています。

かっこの中に含めたパラメータリストは常に存在していなければいけません。パラメータがない場合、空欄のパラメータリスト`()`を使用すべきです。

各パラメータには、[COLLATE](#)属性は使用できないことを除けば、有効なデータタイプを使用していると宣言できます。

デフォルトで各パラメータは**IN**パラメータです。上記とは別にパラメータの属性を特定する場合、**OUT** または **INOUT** キーワードをパラメータ名の前で使用してください。

注:パラメータに**IN**、**OUT**しくは**INOUT**と特定することは**PROCEDURE**に対してのみ有効です。(b>FUNCTIONパラメータは常に**IN**パラメータと見なされます。

INパラメータはプロシージャにある値を渡します。プロシージャはその値を修正しなければならない場合もありますが、プロシージャが返されても、発信側にはその改良を閲覧することができません。**OUT**パラメータは手順からある値を発信側に返します。プロシージャ内の初期値は**NULL**で、発信側にプロシージャが返されるとき、その値を閲覧することができません。発信側は**INOUT**パラメータを初期化することができ、プロシージャはそれを改良することができ、プロシージャによる変更はプロシージャが返されたとき発信側で閲覧することができません。

各**OUT**パラメータもしくは**INOUT**パラメータは、ユーザに特定された変数を渡すことで、プロシージャが返されたときにその値を取得できます。(例については、[CALLステートメント構文](#)を参照してください。)保存された他のプロシージャの中からプロシージャまたはファンクションを呼び出す場合、あなたはルーチンパラメータまたはローカルルーチン変数を、**IN**パラメータまたは**INOUT**パラメータとして渡すことができます。

FUNCTIONだけに対して、遵守する義務のある**RETURNS**節を特定することができます。それは、ファンクションのリターンタイプとファンクション本体には、**RETURN value**ステートメントが含まれていなければならないことを示します。保存されたプロシージャとファンクションの**RETURN**ステートメントがタイプの異なる値を戻した場合、その値は正しい値に強制的に修正されます。例えば、ファンクションがその**RETURN**節の中に**ENUM**値または**SET**値を特定しますが、**RETURN**ステートメントが整数を戻す場合、ファンクションから返された値は、**SET**メンバーのセットに対応する**ENUM**メンバーに対する文字列となります。

routine_bodyは有効なSQLプロシージャステートメントから成り立っています。これを**SELECT**または**INSERT**のような簡単なステートメントもしくは**BEGIN**や**END**を使って書かれた複合ステートメントにすることができます。複合ステートメントの構文については、**BEGIN ... END 複合ステートメント構文**を参照してください。複合ステートメントには、宣言、ループ並びにその他の制御構造ステートメントを含むことができます。これらのステートメントに対する構文については、この章の後半部分で説明します。例えば、**DECLAREステートメント用構文**並びに**フローコントロール・コンストラクト**を参照してください。いくつかのステートメントはストアドルーチン内で使用することはできません (**ストアドルーチンとトリガの規制**を参照してください)。

ルーチンが生成されたとき、MySQLは有効化されていた**sql_mode**システム変数設定を保存し、現サーバのSQL モードに関係なく、必ずこの設定でルーチンを実行します。

CREATE FUNCTIONステートメントはUDF(ユーザ定義機能)をサポートするため、旧バージョンのMySQLで使用されています。**Adding New Functions to MySQL**を参照してください。UDFは保存されたファンクションが存在していてもサポートされ続けます。UDFは記憶された外部機能であるとなすことができます。ただし、保存されたファンクションは自身の名称スペースをUDFと共有していることに注意してください。サーバが異なった種類のファンクションに対するリファレンスを解釈する方法を述べた規則については、**関数名の構文解析と名前解決**を参照してください。

プロシージャあるいはファンクションは、それが同じインプットパラメータに対して常に同じ結果をもたらす場合、「決定論的」であるとみなされるが、同じ結果をもたらさない場合には、「非決定論的」であるとみなされます。ルーチンの定義に**DETERMINISTIC**も**NOT DETERMINISTIC**も附与しない場合、初期設定は**NOT DETERMINISTIC**となります。

NOW()関数(またはその同義語)または**RAND()**を含むルーチンは非決定論的であるが、複製に対して耐性を保持していることがあります。**NOW()**の場合、バイナリ ログはタイムスタンプを含み、正しく複製します。**RAND()**はルーチンの中で唯一回起動しただけで、正しく複製します。(ルーチン実行のタイムスタンプと乱数種を、マスタとスレーブが同じインプットとみなすことができます。)

現在、**DETERMINISTIC**特性は容認されていますが、まだオプチマイザによって使用されていません。ただし、バイナリログが有効化されている場合、この特徴はMySQLがどのルーチン定義を受け入れるかに影響します。**ストアドルーチンとトリガのバイナリログ**を参照してください。

幾つの特徴は、ルーチンによるデータ使用の性質に関する情報を提供します。MySQLでは、これらの特性は助言のみです。サーバはルーチンに実行が許されるステートメントの種類を制限するために、それらを使用しません。

- **CONTAINS SQL**はルーチンにはデータを読み書きするステートメントは含まれていないことを示しています。これらの特性が明確に附与されていない場合、これがデフォルトとなります。このようなステートメントの例は、**SET @x = 1**または**DO RELEASE_LOCK('abc')**です。これは、データの実行はしても読み書きを行いません。
- **NO SQL**はルーチンにSQLステートメントが含まれていないことを示します。
- **READS SQL DATA**は、ルーチンには(例えば、**SELECT**のように)データを読み取るが、書き取らないステートメントが含まれていることを示します。
- **MODIFIES SQL DATA**は、ルーチンには(例えば、**INSERT**もしくは**DELETE**のように)データを書き取ることができるステートメントが含まれていることを示します。

SQL SECURITY特徴はルーチンを生成させるユーザあるいはそれを呼び出すユーザの許可を使って、ルーチンが実行されるべきか否かを明示するために使うことができます。そのデフォルトは**DEFINER**です。この特徴はSQL:2003の新機能です。その生成者や利用者は、ルーチンが属するデータベースにアクセスできる許可を取得していなければなりません。ルーチンを実行することができる**EXECUTE**権限を持つ必要があります。この権限を持たなければいけないユーザは、**SQL SECURITY**機能を設定する方法によって、規定者が利用者のいずれかになります。

オプションの**DEFINER**節は**SQL SECURITY DEFINER**特徴を有するルーチンに対して、実行中にアクセス権限をチェックする時使用するべきMySQLアカウントを特定します。**DEFINER**節はMySQL 5.1.8.で追加されました。

`user` 値を附与する場合、それを '`user_name`'@'`host_name`' フォーマット (`GRANT` ステートメントに使用したと同じフォーマット) の中にあるMySQLアカウントにすべきです。 `user_name` の値と `host_name` の値が両方共必要です。 `CURRENT_USER`を`CURRENT_USER()`として附与することもできます。 `DEFINER`の初期値は`CREATE PROCEDURE`もしくは`CREATE FUNCTION`もしくはステートメントを実行するユーザです。(これは`DEFINER = CURRENT_USER`と同じです。)

`DEFINER`節を特定する場合、`SUPER`権限を保持していない限り、自分の値を除くいかなるアカウントにも値をセットすることはできません。これらの規則は`DEFINER` ユーザの法定値を定義します。

- `SUPER`権限を保持していない場合、文字によるか、`CURRENT_USER`を使って規定されている`user`法定値のみがユーザのアカウントとなります。デファイナーを別のアカウントに設定することはできません。
- `SUPER`権限を保持している場合、構文的に規定した有効なアカウントネームを特定することができます。そのアカウントが実在しない場合、警告が生成されます。

架空の`DEFINER`値を使ってルーチンを生成させることは可能ですが、ルーチンを`DEFINER`権限を使って実行すると、エラーが発生します。しかし定義者は実行中には存在しません。

ルーチンを起動すると、必然的に `USE db_name`が実行(ルーチンの実行が終わると自然に停止)されます。ストアードルーチン内で`USE`ステートメントを使用することは禁止されています。

サーバはルーチンパラメータのデータタイプまたはファンクションリターン値を以下の通り使用します：これらの規則は`DECLARE`ステートメント(`DECLARE ローカル変数`)で生成されたルーチン変数にも適用します。

- 割り当てたデータにミスマッチおよびオーバーフローがないかチェックします。警告の中に変換やオーバーフローの問題が、またストリクトモードにエラーがそれぞれもたらされます。
- 文字データタイプに対して、宣言文中に`CHARACTER SET`節がある場合、指定されたキャラクタセットとそのデフォルト照合順序が使用されます。このような節がない場合、ルーチンが生成される時有効であったデータベースキャラクタセットと照合順序が使用されます。(これらは`character_set_database`システム変数および`collation_database`システム変数の値によって附与されます。) `COLLATE`属性はサポートされていません。(このコンテキスト`BINARY`はキャラクタセットのバイナリー照合順序を規定するので、これには`BINARY`の使用が含まれます。)
- パラメータや変数にはスカラー値のみ割り当てることができます。例えば、`SET x = (SELECT 1, 2)`のようなステートメントは無効です。

`COMMENT`節はMySQLの拡張に含まれ、これはストアードルーチンの説明に使われます。この情報は`SHOW CREATE PROCEDURE`ステートメントと`SHOW CREATE FUNCTION`ステートメントによって表示されます。

MySQLはルーチンに`CREATE`および`DROP`のようなDDLステートメントを含めることを許します。MySQLはストアードプロシージャ (保存されたファンクションではない)に`COMMIT`のようなSQLトランザクションステートメントを含めることも許容します。保存されたファンクションに明示、黙示、コミットもしくは反論を行うステートメントを含めることは許容されません。これらのステートメントに対するサポートはSQLの基準によって要求されません。当該基準はこれについて、各DBMSベンダーはこれらを許すか否かを決定することができると述べています。

ストアードルーチンは`LOAD DATA INFILE`を使用することができません。

結果のセットを返すステートメントを保存されたファンクション内で使うことができません。これには、カラム値を変数に取り込むために`INTO`を使わない`SELECT`ステートメント、`SHOW`ステートメント並びに`EXPLAIN`のようなその他のステートメントが含まれています。ファンクションを規定する時、結果セットを戻すことを定義できるステートメントに対して、`Not allowed to return a result set from a function`エラーが発生します(`ER_SP_NO_RETSET_IN_FUNC`)。稼動中にだけ、結果セットを返すことを決めることが出来るステートメントに対して、`PROCEDURE %s can't return a result set in the given context`エラーが発生します(`ER_SP_BADSELECT`)。

以下は、`OUT`パラメータを使用する簡単なストアードプロシージャの例を示したものです。この例は、プロシージャを定義しながら、`mysql`クライアント`delimiter`コマンドを使用して、ステートメントデリミタを;`から//`に変更するのに使用します。これによって、プロシージャ本体の中で使用された;`デリミタが、mysql自身によって解釈されないで、サーバに転送されることが許容されます。`

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
```

```

-> SELECT COUNT(*) INTO param1 FROM t;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+-----+
| @a    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)

```

`delimiter` コマンドを使用する時、MySQL に対してエスケープキャラクタとなるので、バックスラッシュ (`\`) キャラクタの使用を避けてください。

パラメータを取り込み、SQL 機能を使ってオペレーションを行って結果を返すファンクションの例を次に紹介します。この場合、ファンクションの定義に内部ステートメントデリミタは含まれていないので、[デリミタ](#)を使う必要はありません。

```

mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
-> RETURN CONCAT('Hello, 's, '!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world!  |
+-----+
1 row in set (0.00 sec)

```

MySQL とのインターフェースを持つ言語で書かれたプログラムの中からストアドプロシージャを起動する方法については、[CALL ステートメント構文](#)を参照してください。

ALTER PROCEDURE および ALTER FUNCTION 構文

```

ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic:
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'

```

このステートメントはストアドプロシージャもしくはファンクションの特徴を変更するのに使用することができます。ルーチンに対して、[ALTER ROUTINE](#) 権限を持っていない限りなりません。(この権限はルーチン生成者に自動的に供与されます。)バイナリ ログが有効化されている場合、[ALTER FUNCTION](#) ステートメントにも [SUPER](#) 権限が [ストアドルーチンとトリガのバイナリログ](#) に述べた通り、要求されます。

[ALTER PROCEDURE](#) または [ALTER FUNCTION](#) ステートメントに複数の変更を施すことができます。

DROP PROCEDURE および DROP FUNCTION 構文

```

DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name

```

このステートメントはストアドプロシージャもしくはファンクションを撤去するのに使用されます。即ち、特定されたルーチンがサーバから撤去されます。ルーチンに対して、[ALTER ROUTINE](#) 権限を持っていない限りなりません。(この権限はルーチン生成者に自動的に供与されます。)

`IF EXISTS` 節は MySQL の拡張子です。それは、プロシージャもしくはファンクションが存在しない場合にエラーが発生するのを阻止します。[SHOW WARNINGS](#) を使用して閲覧することができる警告が生成されます。

CALL ステートメント構文

```
CALL sp_name([parameter[,...]])
CALL sp_name([])
```

CALLステートメントによって、CREATE PROCEDURE.を使用して以前に定義したプロシージャが起動されます。

CALLは、OUTまたはINOUTパラメータであると宣言されているパラメータを使って、値を発信側に返すことができます。またこれはROW_COUNT() ファンクションをコールする事で クライアントプログラムがSQLレベルで取得する事ができ、またmysql_affected_rows() C API 機能をコールする事によって Cから取得する事ができる、影響を受けた行を「返します」。

MySQL 5.1.13では現在、アーギュメントを取り込んでいないストアードプロシージャに、かつことをつけることなく取り出すことができるようになっています。即ち、CALL p()とCALL pは等価です。

OUTまたはINOUTパラメータを使って、値をプロシージャから戻すには、パラメータを、ユーザ変数を使って渡し、プロシージャが戻した後、変数の値をチェックします。ユーザが保存された他のプロシージャの中からプロシージャまたはファンクションを呼び出す場合、ユーザはルーチンパラメータまたはローカルルーチン変数を、INパラメータまたはINOUTパラメータとして渡すことができます。INOUTパラメータの場合、それをプロシージャに渡す前に値を初期化してください。以下のプロシージャには、そのプロシージャがサーバの現バージョンにセットするOUT パラメータおよびそのプロシージャがその現在値から1だけ増やすINOUT値が含まれています。

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

プロシージャを呼び出す前に、INOUTパラメータとして渡すべき変数を初期化してください。プロシージャを呼び出した後には、2つの変数はセットまたは改良されています。

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+-----+-----+
| @version | @increment |
+-----+-----+
| 5.1.12-beta-log | 11 |
+-----+-----+
```

CALLSQLステートメントを使用するCプログラムを書き込んで、結果セットを生成させるストアードプロシージャを実行する場合、ユーザはmysql_real_connect()を呼び出す時、CLIENT_MULTI_STATEMENTSをセットすることによって、CLIENT_MULTI_RESULTSフラグを明確または暗黙にセットする必要があります。このようなストアードプロシージャはそれぞれ、複数の結果を生成します。プロシージャ内で実行されたステートメントによって返された結果のセット、並びに呼び出しのステータスを示す結果。CALLステートメントの結果を処理するには、mysql_next_result()を呼び出すループを使用してさらに多くの結果があるが特定してください。(例については、マルチプルステートメントを実行するC APIハンドリングを参照してください。)

MySQLインターフェースを提供する言語で書かれたプログラムに対して、OUTパラメータやINOUTパラメータの結果をCALLステートメントから直接複製するネイティブ方法は存在しません。パラメータ値を取得するには、CALLステートメント中のプロシージャに、ユーザが規定した変数を渡し、その後、SELECTステートメントを実行して、変数値を含む結果セットを生成させてください。以下の例は、2つのOUTパラメータを含むストアードプロシージャ p1 に対するテクニック(エラーチェックを除く)を例示したものです。

```
mysql_query(mysql, "CALL p1(@param1, @param2)");
mysql_query(mysql, "SELECT @param1, @param2");
result = mysql_store_result(mysql);
row = mysql_fetch_row(result);
mysql_free_result(result);
```

先行するコードを実行した後、row[0]並びにrow[1]に、@param1および@param2の値をそれぞれ含ませます。

INOUTパラメータを扱う為に、ユーザ変数をその値にセットしてプロシージャに渡すCALLの前に、ステートメントを実行してください。

BEGIN ... END 複合ステートメント構文


```
[begin_label:] BEGIN
[statement_list]
END [end_label]
```

BEGIN ... END構文は、ストアドルーチンおよびトリガの中に表示することができる複合ステートメントを書くのに使用します。複合ステートメントには、**BEGIN**なるキーワードと**END**なるキーワードによって封じ込める方法を使って、複数のステートメントを含めることができます。**statement_list**は複数のステートメントのリストを意味します。**statement_list**中の各ステートメントは、セミコロン(;)ステートメントデリミタで終了させなければなりません。**statement_list**はオプションであることに注意してください。これは、空の合成ステートメント (**BEGIN END**)は有効であることを意味します。

複数のステートメントを使用する場合、クライアントには;ステートメントデリミタを含むステートメント文字列を送ることが要求されます。これは、**mysql**コマンド・ライン クライアントの中で、**delimiter**コマンドを使って扱われます。ステートメントの最後のデリミタである ; を変更すると、(例えば //に変更) ; をルーチンボディの中で使用する事ができます。(例については、**CREATE PROCEDURE**および**CREATE FUNCTION 構文**を参照してください。)

複合ステートメントにはラベルを貼ることができます。**begin_label** がなければ、**end_label** を付与する事はできません。両方が存在する場合、これらは同じでなければなりません。

オプションの[**NOT**] **ATOMIC**節はまだサポートされていません。これは、トランザクションのセーブポイントがインストラクションブロックの始めにセットされていなく、この文脈中で使用する**BEGIN**節は現在のトランザクションに対して効果が無いことを意味します。

DECLAREステートメント用構文

DECLAREステートメントはルーチンに付属する様々なアイテムを定義するのに使用します。

- ローカル変数。**ストアドルーチン内の変数** を参照してください。
- 条件とハンドラ。**条件とハンドラ** を参照してください。
- カーソル。**カーソル** を参照してください。

SIGNALステートメント並びに**RESIGNAL**ステートメントは現在サポートされていません。

DECLAREは**BEGIN ...**の内部でのみ許容されています。**END**はステートメントを合成し、他のステートメントの前にその始めがなくてはなりません。

宣言は決まったオーダーを遵守しなければなりません。カーソルは、ハンドラを宣言する前に宣言されなければなりません。また、変数と条件はカーソルかハンドラのいずれかを宣言する前に宣言されなければなりません。

ストアドルーチン内の変数

ユーザはルーチンの中で変数を宣言して使用することができます。

DECLARE ローカル変数

```
DECLARE var_name[,...] type [DEFAULT value]
```

このステートメントはローカル変数を宣言するのに使用します。**DEFAULT**節を含ませて、その変数に対するデフォルト値を提供してください。その値は表現として規定することができます。それは定数である必要はありません。**DEFAULT**節が含まれていない場合、初期値は**NULL**となります。

ローカル変数は、データタイプとオーバーフローチェックに関して、ルーチンパラメータと同じように処理されます。**CREATE PROCEDURE**および**CREATE FUNCTION 構文** を参照してください。

ローカル変数の範囲は、それが宣言されている **BEGIN ... END** ブロックの範囲内です。変数は、同じ名称を使って変数を宣言するこれらのブロックを除く、宣言ブロック内の入れ子を作っているブロックの中に引用することができます。

変数 SET ステートメント

```
SET var_name = expr [, var_name = expr] ...
```


ストアルーチン中のSETステートメントは一般SETステートメントの拡張されたバージョンです。引用された変数は、ルーチンもしくはグローバルシステム変数の内側に宣言されたものにすることができます。

ストアルーチン中のSETステートメントは、既存のSET構文の一部として実行されます。これは、異なった変数タイプ(ローカルに宣言された変数およびグローバル変数並びにセッション・サーバ変数)を混在させることができるSET `a=x, b=y, ...`の拡張された構文を容認します。これは、ローカル変数の組み合わせおよびシステム変数に対してだけ意味を持つ幾つかのオプションも認めます。この場合、オプションは認識されますが無視されます。

SELECT ... INTO ステートメント

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

このSELECT 構文は選択されたカラムを直接変数の中に保存します。従って、1本の横列のみ取り出すことが許されています。

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

ユーザ変数名に対して、大文字小文字を区別されません。[ユーザによって定義された変数](#)を参照してください。

重要SQL変数の名称をカラム名称と同じにすべきではありません。SELECT ... INTO のようなSQLステートメントが、カラムとローカル変数のリファレンスを同名で含んでいると、現在のMySQLはそのリファレンスを変数名だと認識します。例えば、次のステートメントの中では、`xname`は`xname`カラムではなく、[変数](#)カラムに対するリファレンスと解釈されます：

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname,id INTO newname,xid
  FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

このプロシージャを呼び出すとき、`newname`変数は、`table1.xname`カラムに関係なく、値'`bob`'を返します。

[ストアルーチンとトリガの規制](#)も参照してください。

条件とハンドラ

条件によっては、特別な扱いが求められます。これらの条件は、エラー並びにルーチンの内側で行われている一般フロー制御に関連している場合もあります。

DECLARE 条件

```
DECLARE condition_name CONDITION FOR condition_value
```

```
condition_value:
  SQLSTATE [VALUE] sqlstate_value
| mysql_error_code
```

このステートメントは特別な扱いが必要な条件を規定します。それには規定されたエラーの条件を含む名称が関連付けられます。その名称は後にDECLARE HANDLERステートメントの中で使われます。[DECLARE ハンドラ](#)を参照してください。

`condition_value`をSQLSTATE値もしくはMySQLエラーコードにすることができます。

DECLARE ハンドラ

```
DECLARE handler_type HANDLER FOR condition_value[,...] statement
```

```
handler_type:
  CONTINUE
| EXIT
| UNDO
```

```
condition_value:
  SQLSTATE [VALUE] sqlstate_value
| condition_name
| SQLWARNING
| NOT FOUND
| SQLEXCEPTION
| mysql_error_code
```

DECLARE ... HANDLERステートメントは各々が複数の条件で処理することができるハンドラを規定します。もし、これらの条件の1つが起った場合、**ステートメント**が実行されます。この場合、**ステートメント**を単純なものにすることができます。(例えば、**SET var_name = value**)、もしくは、**BEGIN** と **END** を使って書いた複合ステートメントにすることができます。(BEGIN ... END 複合ステートメント構文参照)

CONTINUEハンドラに対して、現ルーチンの実行が、ハンドラステートメントの実行の後に続きます。**EXIT** ハンドラに関しては、ハンドラが宣言された **BEGIN ... END** コンパウンドステートメントの中で実行が終了します。(これは、条件が内側にあるブロックの中に発生する場合でも同じです。) **UNDO** ハンドラタイプのステートメントはまだサポートされていません。

ハンドラがまだ宣言されていない条件がしている場合、デフォルトアクションは**EXIT**となります。

A **condition_value**は以下の値のいずれかにすることができます：

- **SQLSTATE**値もしくはMySQLエラーコード。
- 既に **DECLARE ... CONDITION**で指定されている条件名。 **DECLARE 条件** を参照してください。
- **SQLWARNING**は01で始まる全てのSQLSTATEコードに対する速記文字です。
- **NOT FOUND**は02で始まる全てのSQLSTATEコードに対する速記文字です。
- **SQLEXCEPTION**は**SQLWARNING**または**NOT FOUND**によって捕らえられなかった全てのSQLSTATEコードの速記文字です。

例:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
-> BEGIN
-> DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
-> SET @x = 1;
-> INSERT INTO test.t VALUES (1);
-> SET @x = 2;
-> INSERT INTO test.t VALUES (1);
-> SET @x = 3;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo();//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x |
+-----+
| 3 |
+-----+
1 row in set (0.00 sec)
```

その例は、重複キーエラーに対して発生するSQLSTATE 23000を持つハンドラに関連するものです。**@x** は 3です。MySQLがプロシージャの最後まで実行されたことを示しています。もし**DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;**ラインが存在していなかった場合、MySQLは(**EXIT**)のデフォルトパスを、2番目の**INSERT**が**PRIMARY KEY**制限によって失敗したとき取り、そして**SELECT @x** は2を返しています。

条件を無視したい場合、ユーザはそれに対して、**CONTINUE**ハンドラと宣言して、それを空のブロックと関連させることができます。例:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

カーソル

ストアドプロシージャとファンクションの内側に単純なカーソルがサポートされています。その構文は埋め込まれているSQLの中のものと同じです。カーソルは現在、アセンシティブ、読み取り専用、そしてスクロール機能はついていません。アセンシティブはサーバがその結果テーブルの複製を作ることができるか、できないかを意味します。

カーソルは、ハンドラを宣言する前に宣言されなければなりません。また、変数と条件はカーソルがハンドラのいずれかを宣言する前に宣言されなければなりません。

例:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
      IF b < c THEN
        INSERT INTO test.t3 VALUES (a,b);
      ELSE
        INSERT INTO test.t3 VALUES (a,c);
      END IF;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
  CLOSE cur2;
END
```

宣言用カーソル

```
DECLARE cursor_name CURSOR FOR select_statement
```

このステートメントはカーソルを宣言します。複数のカーソルを一つのルーチンの中に宣言することができますが、各カーソルは附与されたブロックの中に、ユニークな名称を持っていなければなりません。

SELECTステートメントに**INTO**節を含めることはできません。

カーソルOPENステートメント

```
OPEN cursor_name
```

このステートメントは以前に宣言したカーソルを開きます。

カーソルFETCHステートメント

```
FETCH cursor_name INTO var_name [, var_name] ...
```

このステートメントは、規定されたオープンカーソルを使って、次の行 (存在している場合)を取り込んで、ここにカーソルポインタを進めます。

行が得られなくなると、SQLSTATE値02000を使用したNo Data条件が発生します。この条件を検出するため、ハンドラをセットすることができます。例が[カーソル](#)で紹介されています。

カーソルCLOSEステートメント

```
CLOSE cursor_name
```

このステートメントは以前開いたカーソルを閉じます。

明らかに閉じていない場合、それは、それが宣言された複合ステートメントの最後で閉じられます。

フローコントロール・コンストラクト

IF、CASE、LOOP、WHILE、REPLACE ITERATEおよびLEAVE コンストラクトは完全に実装されます。

これらのコンストラクトの多くには、以下セクションの文法仕様に示すようなその他のステートメントが含まれています。このようなコンストラクトを入れ子とすることができます。例えば、IFステートメントにはそれ自体がCASEステートメントを含むWHILEループが含まれているかもしれません。

FORループは現在サポートされていません。

IF ステートメント

```
IF search_condition THEN statement_list
[ELSEIF search_condition THEN statement_list] ...
[ELSE statement_list]
END IF
```

IFは基本条件コンストラクトを施行します。search_conditionが真の場合、該当するSQLステートメントが実行されます。search_conditionが合致しない場合、ELSE節内のステートメントリストが実行されます。各statement_listは複数のステートメントから成り立っています。

注:ここで述べたIFステートメントとは異なるIFファンクションもあります。制御フロー関数を参照してください。

CASEステートメント

```
CASE case_value
  WHEN when_value THEN statement_list
[WHEN when_value THEN statement_list] ...
[ELSE statement_list]
END CASE
```

または

```
CASE
  WHEN search_condition THEN statement_list
[WHEN search_condition THEN statement_list] ...
[ELSE statement_list]
END CASE
```

ストアルーチンに対するCASEステートメントは複雑な条件コンストラクトを実装します。search_conditionが真の場合、該当するSQLステートメントが実行されます。検索条件が合致しない場合、ELSE節内のステートメントリストが実行されます。各statement_listは複数のステートメントから成り立っています。

注:ストアルーチンの内部で使用する目的でここに示したCASEステートメントの構文は、(制御フロー関数)で説明されたCASE表現とは若干異なっています。CASEステートメントにELSE NULL節を含めることはできません。これを持たせると、それは、ENDの代わりにEND CASEを使って消去されます。

LOOP ステートメント

```
[begin_label:] LOOP
  statement_list
END LOOP [end_label]
```

LOOPは単純なループコンストラクトを実装します。これによって、複数のステートメントからなるステートメントリストを繰り返して使用することが可能になります。ステートメントのループ内での実行は、ループが閉じられるまで繰り返されます。これは一般的にLEAVEステートメントを使って達成されます。

LOOPステートメントにはラベルを貼ることができます。begin_labelも存在していない限り、end_labelを附与することはできません。両方が存在する場合、これらは同じでなければなりません。

LEAVE ステートメント

```
LEAVE label
```

このステートメントは、ラベルを貼ったフローコントロールコンストラクトを閉じるために使用します。それは、**BEGIN END**もしくはループコンストラクト(**LOOP**、**REPEAT**、**WHILE**)の中で使うことができます。

ITERATE ステートメント

```
ITERATE label
```

ITERATEは**LOOP**ステートメント、**REPEAT**ステートメント並びに**WHILE**ステートメントの中にだけ現れます。**ITERATE**は「再びループを実行」を意味します。

例:

```
CREATE PROCEDURE doitrate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

REPEAT ステートメント

```
[begin_label:] REPEAT
  statement_list
UNTIL search_condition
END REPEAT [end_label]
```

REPEATステートメント中のステートメントリストは、**search_condition**が真になるまで繰り返されます。このようにして、**REPEAT** は常に、少なくとも1回入ループを書き込みます。**statement_list** は複数のステートメントから成り立っています。

REPEATステートメントにはラベルを貼ることができます。**begin_label** がなければ、**end_label** を与える事はできません。両方が存在する場合、これらは同じでなければなりません。

例:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
-> SET @x = 0;
-> REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x |
+-----+
| 1001 |
+-----+
1 row in set (0.00 sec)
```

WHILE ステートメント

```
[begin_label:] WHILE search_condition DO
  statement_list
END WHILE [end_label]
```

WHILEステートメント中のステートメントリストは、**search_condition**が真になるまで繰り返されます。**statement_list**は複数のステートメントから成り立っています。

CHECK **WHILE**ステートメントにはラベルを貼ることができます。**begin_label**も存在していない限り、**end_label**を附与することはできません。両方が存在する場合、これらは同じでなければなりません。

例:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END
```

ストアドプロシージャ、ファンクション、トリガ並びにLAST_INSERT_ID()

ストアドルーチン(プロシージャまたはファンクション)もしくはトリガの中で、[LAST_INSERT_ID\(\)](#)の値は同じ方法で、ステートメントがオブジェクトのこれらの種類の本体の外側で実行させるように変更されます。([情報関数](#) を参照してください)。ストアドルーチンまたはトリガの以下のステートメントに基づいて現れる[LAST_INSERT_ID\(\)](#)の値に与える効果は、ルーチンの種類によって変わります。

- ストアドプロシージャが、[LAST_INSERT_ID\(\)](#)の値を変えるステートメントを実行する場合、変更された値はプロシージャコールに従うステートメントによって現れます。
- 値を変える保存されたファンクションとトリガの場合、ファンクションまたはトリガが終わる時、値が元に戻るので、以下のステートメントには変更された値は現れません。

ストアドルーチンとトリガのバイナリログ

バイナリログには、データベースの中身を修正するSQLステートメントに関する情報が含まれています。この情報は改良を説明する「イベント」の形で記憶されます。バイナリログは2つの重要な目的を持っています。

- 複製の場合、マスターサーバはそのバイナリログに含まれているイベントを自分のスレーブに送ります。これによって、これらのイベントが実行されて、マスターが行ったと同じデータ変更が行われます。[レプリケーションの実装](#)を参照してください。
- ある種のデータリカバリには、バイナリログの使用が必要です。バックアップファイルが修復された後、バックアップ後に記録されたバイナリログ中のイベントは再実行されます。これらのイベントは、データベースをバックアップの点からデットまで持って行きます。[バックアップファイルでリカバリ](#)を参照してください。

このセクションで、MySQL 5.1がストアドルーチン(プロシージャとファンクション)およびトリガの為のバイナリログを取り扱う方法について説明します。ここで、その実装がストアドルーチンの使用上に置かれる現在の条件も説明し、これらの条件が必要とされる理由に関する追加情報を提供します。

一般に、ここで述べた問題はバイナリログがSQLステートメントレベルにおいて起こる時生じます。ユーザが行をベースとするバイナリログを使用する場合、ログには、SQLステートメントを実行した結果として個別の行に施した変更が含まれます。行をベースとするログに関する一般情報については、[レプリケーション フォーマット](#)を参照してください。

行をベースとするログを使用する時、ストアドルーチンとトリガの定義がステートメントとして複製されます。ルーチンやトリガを実行する時、行に施した変更は登録されますが、これらを実行するステートメントは登録されません。ストアドプロシージャに対して、これはCALLステートメントは登録されないことを意味します。保存されたファンクションに対して、ファンクション内で行に施した変更は登録されますが、ファンクション起動は登録されません。トリガの場合、トリガが行に対して行った変更は登録されます。スレーブ側では、行の変更のみ現れ、ルーチンまたはトリガ起動は現れません。

特に注記しない限り、ここに示した備考はユーザが既に`--log-bin`オプションを使ってサーバを立ち上げることによって、バイナリログを有効化しているものと見なします。([バイナリ ログ](#) を参照してください。) バイナリログが有効化されていない場合、複製は不可能なばかりでなく、データリカバリ用に、バイナリログも利用できません。

MySQL5.1は以下の通り総括することができます：これらの条件はストアドプロシージャには適用されず、これらはバイナリログが有効化されない限り、適用されません。

- 保存されたファンクションを生成もしくは変更するには、ユーザは通常要求される **CREATE ROUTINE** 権限もしくは **ALTER ROUTINE** 権限に加え、**SUPER** 権限を保持していなければなりません。
- 保存されたファンクションを生成する時、それが決定論的なものであるか、データを改良しないものであるかを宣言しなければなりません。これを怠ると、データリレカバリやデータの複製が安全にできなくなる場合があります。
- ファンクション生成に対する (**SUPER** 権限を持たなければならず、決定論的か、データを修正しないかの別を宣言しなければならない) 前の規制を緩和するには、グローバル **log_bin_trust_function_creators** システム変数を 1 に設定します。デフォルトで、これは 0 に設定されていますが、ユーザはこのようにして変更することができます。

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

この変数を、サーバを立ち上げる時 **--log-bin-trust-function-creators=1** オプションを使って設定することもできます。

バイナリログが有効化されていない場合、**log_bin_trust_function_creators** は適用されず、ファンクション生成に対して **SUPER** は要求されません。

トリガは保存されたファンクションと同等であるため、ファンクションに関する前の備考は、以下の場合を除き、トリガには適用されません。**CREATE TRIGGER** には、オプションの **DETERMINISTIC** 特徴は含まれていないので、トリガは常に決定論的であると見なします。しかし、この仮定は場合によっては無効です。例えば、**UUID()** 機能は非決定論的です (複製されません)。このような機能のトリガ中での使用に関して、注意すべきです。

CREATE TRIGGER トリガはテーブルを更新することができるので、ユーザが **TRIGGER** 権限 (MySQL 5.1.6 より前の版では **SUPER**) を持っておらず、**log_bin_trust_function_creators** が 0 である場合、保存されたファンクションに起こるこれらと同等なエラーメッセージが発生します。(スレーブ側では、スレーブはトリガ **DEFINER** 属性を使って、どのユーザがトリガ生成者であるか査定します。)

以下のディスカッションで、ログの実装とその意味に関する追加詳細を提供します。このディスカッションは最初のアイテムを除き、ステートメントをベースとするログだけを対象とし、行をベースとするログには適用されません。**CREATE** ステートメントおよび **DROP** ステートメントはログモードと関係なく、ステートメントとして登録されます。

- サーバは **CREATE PROCEDURE**、**CREATE FUNCTION**、**ALTER PROCEDURE**、**ALTER FUNCTION**、**DROP PROCEDURE** および **DROP FUNCTION** ステートメントをバイナリログに書き込みます。
- 保存されたファンクションの利用は、ファンクションがデータを変更し、これ以外登録しないステートメントの中に起こる場合、**SELECT** ステートメントとして登録されます。これによって、未登録ステートメントの中で保存されたファンクションを使用したことにより、データの変更が複製されない問題が防止されます。例えば、**SELECT** ステートメントはバイナリログに書き込まれないが、**SELECT** は変更を施す保存されたファンクションを使用しなければならない場合があります。これを扱うため、**SELECT func_name()** ステートメントは、あるファンクションが変更を実行する時、バイナリログに書き込まれます。以下のステートメントがマスターの上で実行されると仮定すると：

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

SELECT ステートメントが実行される時、ファンクション **f1()** は三回起動されます。これらの起動の内、2回の起動で、行を挿入し、MySQL はそれらの各々に対して **SELECT** ステートメントを登録します。即ち、MySQL は以下のステートメントをバイナリログに書き込みます。

```
SELECT f1(1);
SELECT f1(2);
```

サーバは、ファンクションがエラーを引き起こすストアプロシージャを取り出す時、保存されたファンクションの取り出しに対する **SELECT** ステートメントも登録します。この場合、サーバは、**SELECT** ステートメントを期待

エラーコードと一緒にログに書き込みます。スレーブに同じエラーが発生しても、それは期待された結果で、複製は継続されます。さもないと、複製はストップします。

注:MySQL 5.1.7の前に、`DO func_name()`として登録されたこれらの`SELECT func_name()`ステートメントが見えます。`SELECT`に対する変更は、`DO`を使用してのエラーコードチェックに十分な管理が得られないことが判明した結果行われています。

- ファンクションによって実行されたステートメントを除く、保存されたファンクションの利用に対する登録には、複製に対するセキュリティ問題が関与します。当該問題は、以下からなる2つの要因によって起こります。
 - ファンクションがマスタサーバとスレーブサーバ上にある異なった実行パスに従うことは可能です。
 - スレーブサーバ上で実行されたステートメントはフル権限を持つスレーブSQLスレッドによって処理されます。

ユーザはファンクションを生成させる`CREATE ROUTINE`権限を持っていないなければならないが、そのユーザは、フル権限を持つSQLスレッドによって処理されるスレーブ上でのみ実行される危険なステートメントを含むファンクションを書き込むことができます。例えば、マスタサーバとスレーブサーバがそれぞれ1と2のID値を持っている場合、マスタサーバ上のユーザは安全でない関数`unsafe_func()`を以下の通り生成して取り出すことができる場合があります。

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
-> BEGIN
-> IF @@server_id=2 THEN dangerous_statement; END IF;
-> RETURN 1;
-> END;
-> //
mysql> delimiter ;
mysql> INSERT INTO t VALUES(unsafe_func());
```

`CREATE FUNCTION`ステートメントおよび`INSERT`ステートメントはバイナリログに書き込まれるので、スレーブサーバはそれらを実行します。スレーブSQLスレッドはフル権限を持っているので、それは危険なステートメントを実行します。このようにして、ファンクションの取り出しはマスタとスレーブに異なった効果を与え、複製は安全でなくなります。

バイナリログを有効化したサーバに対するこの危険からサーバを守るには、保存されたファンクションの生成者は、要求された通常の`CREATE ROUTINE`権限に加え、`SUPER`権限も持っていなければなりません。同様に、`ALTER FUNCTION`を使用するため、ユーザは`ALTER ROUTINE`権限に加え、`SUPER`権限を持っていなければなりません。`SUPER`権限がないと、エラーが起こります：

```
ERROR 1419 (HY000): You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
```

ファンクションの生成者に`SUPER`権限を持つよう要求したくない場合(例えば、ユーザのシステム上に`CREATE ROUTINE`権限を持つすべてのユーザが経験豊かなアプリケーションデベロッパーである場合、グローバル`log_bin_trust_function_creators`システム変数を1にセットしてください。この場合、ユーザはこの値を`--log-bin-trust-function-creators=1` オプションを使って、サーバを立ち上げる時セットすることができます。バイナリログが有効化されていない場合、`log_bin_trust_function_creators`は適用されず、ファンクション生成に対して`SUPER`は要求されません。

- 更新を実行するファンクションが非決定論的ある場合、それは反復可能ではありません。これは2つの望ましくない効果をもたらす場合があります。
 - それがスレーブをマスタと違ったものにします。
 - 修復されたデータが元のデータと異なります。

これらの問題を処理するため、MySQLは以下の要件を実施しています：マスタサーバ上で、ユーザがファンクションに対して決定論的か、データを修正しないかの別を宣言しない限り、ファンクションの生成と変更は拒否されます。以下からなるファンクション特性の2つのセットが適用されます：

- `DETERMINISTIC`特性と`NOT DETERMINISTIC`特性は、あるインプットに対して、いつも同じ結果を生成するか否かを示します。いずれかの特性を附与しない場合、デフォルト設定は`NOT DETERMINISTIC`となります。ファンクションが決定論的であると宣言するには、`DETERMINISTIC`を明確に規定しなければなりません。

NOW() 関数(またはその同義語)または**RAND()** を使用しても、必ずしもファンクションが決定論的になるとは限りません。**NOW()** の場合、バイナリログはタイムスタンプを含み、正しく複製されます。**RAND()** はファンクション内で 1 回呼び出した場合のみ、正しく複製されます。(ファンクション実行のタイムスタンプと乱数種を、マスタとスレーブ上で同等であるとする暗黙のインプットとみなすことができます。)

SYSDATE() はバイナリログの中でのタイムスタンプによって影響されないの、ステートメントをベースとするログが使われる場合、それは、ストアルーチンを非決定論的になるようにします。行をベースとするログが使用されるか、サーバを `--sysdate-is-now` オプション使って立ち上げ、**SYSDATE()** が **NOW()** と別名とする場合、これは起こりません。

- **CONTAINS SQL**, **NO SQL**、**READS SQL DATA** および **MODIFIES SQL DATA** 特徴はデータの読み取りか、書き込みかに関する情報を提供します。**NO SQL** または **READS SQL DATA** は、特徴が附与されていない場合、ファンクションはデータを変更しないが、デフォルト設定は **CONTAINS SQL** となっているので、ユーザはこれらの中からいずれか 1 つを選んで明確に規定しなければなりません。

デフォルト設定で、**CREATE PROCEDURE** または **CREATE FUNCTION** ステートメントが受け入れられるようにするには、**DETERMINISTIC** または、**NO SQL** が **READS SQL DATA** のどちらかを明示的に指定しなければなりません。指定しないと、エラーが発生します。

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

`log_bin_trust_function_creators` を 1 にセットした場合、ファンクションを決定論的であるようにするか、データを修正しないにすべき要件は撤去されます。

ファンクションの性質は、生成者の「誠意」に基づき評価されます。MySQL はファンクションが宣言した **DETERMINISTIC** が非決定論的結果結果を生成しないことがないかチェックしません。

- ストアドプロシージャの呼び出しは **CALL** レベルでなく、ステートメントレベルで登録されます。即ち、サーバは **CALL** ステートメントを登録せず、実際に実行されたプロシージャ中にこれらのステートメントを登録します。結果として、マスタサーバに起こったと同じ変更がスレーブサーバに見られます。これは、異なった実行パス上に異なった機械を持つプロシージャから結果が得られる恐れがある問題を提供します。

一般に、ストアドプロシージャ内の実行されたステートメントは、独立ファクションで実行すべきでステートメントに適用されたと同じ規則を使ってバイナリログに書き込まれます。プロシージャステートメントを登録する時、プロシージャ内でのステートメントの実行は、非プロシージャコンテキスト内での実行と全く同じではないので、注意が必要です。

- 登録すべきステートメントにはローカルプロシージャ変数に対するリファレンスを含めなければならない場合があります。これらの変数はストアドプロシージャコンテキストの外側に存在しないので、当該変数を引用したステートメントを文章で登録することができません。これにも係わらず、ローカル変数に対する各リファレンスはログを目的として、これに置き換えされます:

```
NAME_CONST(var_name, var_value)
```

`var_name` ローカル変数の名称で、`var_value` はステートメントを登録する時、その変数が持つ値を示す定数です。**NAME_CONST()** は `var_value` および「`var_name`」の `name` の値を持っています。このようにして、ユーザこのファンクションを直接起動する場合、このような結果が得られます。

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
| 14 |
+-----+
```

NAME_CONST() は、登録された独立ステートメントがストアドプロシージャの中でマスタ上で実行された元のステートメントと同じ効果を持つように、スレーブ上で実行されることを許容します。

- 登録すべきステートメントにはユーザが規定した変数に対するリファレンスを含めなければならない場合があります。これを扱うため、MySQL は **SET** ステートメントをバイナリログに書き込んで、マスタ上にあると同じ値を

持つスレーブ上にその変数が存在することを確認します。例えば、ステートメントが変数@my_varを引用する場合、そのステートメントは以下のステートメントによって、バイナリログの中で先行します。この場合、valueはマスタ上@my_varの値です。

```
SET @my_var = value;
```

- プロシージャの呼び出しは、コミットされているもしくはロールバックトランザクション中で発生します。以前には、CALLステートメントは、それらがロールバックトランザクション中に発生した場合も、登録されました。MySQL 5.0.12より、プロシージャ実行のトランザクションが正しく複製されるように、トランザクションコンテキストが考慮されます。即ち、サーバは、これらのステートメントを、実際に実行し、データを修正するプロシージャ中に登録し更に、BEGIN, COMMITステートメントおよびROLLBACKステートメントも必要に応じて登録します。例えば、プロシージャがトランザクションテーブルだけを更新し、ロールバックされる取引の中で実行される場合、これらの更新は登録されません。プロシージャがコミットされたトランザクション中で起こる場合、BEGINステートメントとCOMMITステートメントはその更新を使って登録されます。ロールバックトランザクション中で実行されるプロシージャに対して、そのステートメントは、取引が独立してで実行された場合に適用されたと同じ規則を使って登録されます。
- トランザクションテーブルの更新は登録されません。
- 非トランザクションテーブルの更新は、ロールバックがこれらをキャンセルしないので、登録されます。
- トランザクションテーブルと非トランザクションテーブルを混ぜたものの更新は、スレーブがマスタ上で行われたと同じ変更とロールバックを行うように、BEGINとROLLBACKの間に登録されます。
- ストアドプロシージャの呼び出しは、保存されたファンクション中から呼び出す場合、ステートメントレベルのバイナリログに書き込まれません。この場合、登録される唯一つのものは、(それが登録されたステートメントの中で起こる場合)機能を取り出すステートメントまたは(それが登録されていないステートメントの中で起こる場合)DOステートメントです。よって、プロシージャを呼び出す保存されたファンクションの利用には、プロシージャそれ自体が安全でない限り、注意してください。

第18章 トリガ

目次

CREATE TRIGGER 構文	1071
DROP TRIGGER 構文	1074
トリガの使用	1074

トリガは名称を持つ、テーブルに付属するデータベース オブジェクトで、テーブルに特定イベントが発生すると有効化されます。例えば、次のステートメントを入力するとテーブル並びに **INSERT** トリガが生成されます。トリガはテーブル カラムの 1 つに挿入された値を合計します。

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

この章で、トリガを生成させ、除去する構文について説明し、それを使う方法に対して、幾つかの例を紹介します。トリガの使用に対する制限については、[ストアド ルーチンとトリガの規制](#) で説明されています。[ストアド ルーチンとトリガのバイナリログ](#) でトリガに適用されるバイナリ ログングについて補足説明しています。

MySQL 5.1 中のトリガに関する幾つかの主な質問の答えについては [MySQL 5.1 FAQ — Triggers](#) をご覧ください。

CREATE TRIGGER 構文

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

このステートメントによって、新しいトリガが生成されます。トリガは名称を持つ、テーブルに付属するデータベース オブジェクトで、テーブルに対して特定イベントが発生すると有効化されます。現在、**CREATE TRIGGER** には、そのトリガに添付したテーブルに対する **TRIGGER** 権限が必要です。(このステートメントは MySQL 5.1.6 より前のバージョンに **SUPER** 権限を求めるものです。)

トリガは、tbl_name なる名称を持つテーブルと連携するようになり、これによって、トリガはパーマネント テーブルを参照しなくなくなります。トリガを **TEMPORARY** テーブルあるいはビューと連携させる事はできません。

トリガが有効化されると、このセクションで後に述べるように **DEFINER** 条項によって、適用すべき権限が規定されます。

trigger_time はトリガのアクション タイムです。それを有効化するステートメントの前か後にトリガが有効化される事を示す **BEFORE** または **AFTER** となる事ができます。

trigger_event はトリガを有効化するステートメントの種類を示します。trigger_event では、以下の中から1つを選ぶ事ができます：

- **INSERT**:トリガは、新しい行がテーブルに挿入されると必ず、例えば **INSERT**、**LOAD DATA** 並びに **REPLACE** の各ステートメントを通して有効化されます。
- **UPDATE**:トリガは、行が修正されると必ず、例えば **UPDATE** ステートメントを通して有効化されます。
- **DELETE**:トリガは、新しい行がテーブルから削除されると必ず、例えば **DELETE** ステートメント並びに **REPLACE** ステートメントを通して有効化されます。しかし、テーブル上の **DROP TABLE** ステートメントおよび **TRUNCATE** ステートメントは **DELETE** を使用していないので、このトリガを有効化しません。パーティションをドロップすると、**DELETE** トリガも有効化されません。詳しくは [TRUNCATE 構文](#) を参照してください。

`trigger_event` は、それがテーブル 操作のタイプを表している場合には、トリガを有効化する SQL ステートメントのリテラル タイプを表さないと理解する事が重要です。例えば、ステートメントは両方共行をテーブルに挿入するので、`INSERT`トリガは、`INSERT` ステートメントのみならず、`LOAD DATA` ステートメントによっても有効化されます。

この潜在的に紛らわしい例は、`INSERT INTO ... ON DUPLICATE KEY UPDATE ...` という構文です： `BEFORE INSERT` トリガは、全ての行に対して有効化され、その後、行に対して重複キーがあるか否かによって、`AFTER INSERT` トリガもしくは `BEFORE UPDATE` トリガと `AFTER UPDATE` トリガの両方が有効化されます。

同じトリガ アクション タイムとトリガ イベントを持つテーブルに対して、2つのトリガは存在する事はできません。例えば、1個のテーブルに対して2つの `BEFORE UPDATE` トリガを持つ事はできません。しかし、`BEFORE UPDATE` トリガと `BEFORE INSERT` トリガもしくは `BEFORE UPDATE` トリガと `AFTER UPDATE` トリガを持つ事ができます。

`trigger_stmt` はトリガを有効化する時に実行させるステートメントです。複数の命令を実行させたい場合には、`BEGIN ... END` 複合文コンストラクトを使ってください。これは、ストアド ルーチン内で許されているのと同じステートメントも使う事ができるようにします。`BEGIN ... END 複合ステートメント構文` を参照してください。トリガ中に同じステートメントを使用する事は許されません。(`ストアド ルーチンとトリガの規制` を参照)。

MySQL は `sql_mode` システム中に、トリガが生成された時に有効であった変数設定を記憶し、現サーバの SQL モードに関係なく、この設定でトリガを実際に実行します。

注:トリガは現在、転送された外部キー アクションによって有効化されません。この制限はできるだけ早く撤廃されるでしょう。

MySQL 5.1 で、この例に示す `testref` という名前のトリガのような、直接の参照を含むトリガを名称別テーブルに書き込む事ができます：

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
  INSERT INTO test2 SET a2 = NEW.a1;
  DELETE FROM test3 WHERE a3 = NEW.a1;
  UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END;
|

DELIMITER ;

INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
(0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

以下の値をテーブル `test1` に、ここに示すように挿入すると仮定します：

```
mysql> INSERT INTO test1 VALUES
-> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8 Duplicates: 0 Warnings: 0
```

結果として、4つのテーブル中のデータは以下の通りになります：

```
mysql> SELECT * FROM test1;
+-----+
| a1 |
+-----+
| 1 |
| 3 |
```



```

| 1 |
| 7 |
| 1 |
| 8 |
| 4 |
| 4 |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+-----+
| a2 |
+-----+
| 1 |
| 3 |
| 1 |
| 7 |
| 1 |
| 8 |
| 4 |
| 4 |
+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+-----+
| a3 |
+-----+
| 2 |
| 5 |
| 6 |
| 9 |
| 10 |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+-----+
| a4 | b4 |
+-----+
| 1 | 3 |
| 2 | 0 |
| 3 | 1 |
| 4 | 2 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 1 |
| 9 | 0 |
| 10 | 0 |
+-----+
10 rows in set (0.00 sec)

```

エイリアス **OLD** とエイリアス **NEW** を使う事によって、問題のテーブル (トリガに添付されたテーブル) 中のカラムを参照する事ができます。**OLD.col_name** は既存の行のカラムが更新または削除される前にチェックします。**NEW.col_name** は挿入すべき新しい行あるいは更新された既存の行のカラムを参照します。

DEFINER 条項は、トリガの有効化においてアクセス権をチェックする時に使用すべき MySQL アカウントを指定します。**user** 値を附与する場合、それを '**user_name**'@'**host_name**' フォーマット (**GRANT** ステートメントに使用したのと同じフォーマット) の中にある MySQL アカウントとすべきです。**user_name** の値と **host_name** の値が両方共必要です。**CURRENT_USER** を **CURRENT_USER()** として附与する事もできます。**DEFINER** のデフォルト値は **CREATE TRIGGER** ステートメントを実行するユーザです。(これは **DEFINER = CURRENT_USER** と同じです。)

DEFINER 条項を特定する場合、**SUPER** 権限を保持していない限り、自分の値を除くいかなるアカウントにも値をセットする事はできません。これらの規則は有効な **DEFINER** ユーザ値を決定します:

- **SUPER** 権限を持っていない場合、文字によるか、**CURRENT_USER** を使って規定されている **user** 値だけが有効なユーザ アカウントとなります。デファイナを他のアカウントに設定する事はできません。
- **SUPER** 権限を持っている場合、構文的に有効なアカウント ネームを規定する事ができます。そのアカウントが実在しない場合、警告が生成されます。

実在しない **DEFINER** 値を使ってトリガを生成させる事は可能ですが、デファイナが実在するようになるまで、このようなトリガを有効化しておく事は決して良いアイデアではありません。さもないと、権限チェックに関する挙動パターンは定義されません。

注意:MySQL 5.1.6以前のバージョンでは、MySQL は **CREATE TRIGGER** を使用する為に **SUPER**権限を要求するので、前の規則の2番目の規定だけが適用されます。5.1.6以降のバージョンからは、**CREATE TRIGGER** は **TRIGGER** 権限を要求し、**SUPER** 権限は、**DEFINER**に自身のアカウント値以外の値をセットする事を可能にする為だけに要求されます。

MySQL は、このようなトリガ権限をチェックします:

- **CREATE TRIGGER** 時に、ステートメントを発行するユーザは **TRIGGER** 権限を持つ必要があります。(MySQL 5.1.6.前の **SUPER**)
- トリガを有効化する時、権限が **DEFINER** ユーザの物であるかチェックされます。このユーザは、これらの権限を持っていない限りなりません:
 - **TRIGGER** 権限。(MySQL 5.1.6.前の **SUPER**)
 - トリガの定義の中にある **OLD.col_name** あるいは **NEW.col_name** を経由して照合が行われる場合に得られる問題のテーブルに対する **SELECT** 権限。
 - テーブル カラムが、**SET NEW.col_name = value** トリガの定義中の値の割り当てターゲットである場合に得られる問題のテーブルに対する **UPDATE** 権限。
- トリガによってステートメントが実行される為に一般に要求されるその他一切の権限。

DROP TRIGGER 構文

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

このステートメントはトリガをドロップさせます。そのスキーマ(データベース)には、名称を自由につけることができます。スキーマを撤去すると、トリガは初期スキーマから除去されます。MySQL 5.0.2の中に **DROP TRIGGER** が追加されました。それを使用するには、トリガに関連したテーブルに対して **TRIGGER** 権限が必要となります。(このステートメントは MySQL 5.1.6.より前のバージョンで **SUPER** 権限を求めるものです。)

IF EXISTSを使用して、存在していないトリガに対してエラーが発生するのを防止してください。**IF EXISTS**を使用すると、実在していないトリガに対して **NOTE** が生成されます。**SHOW WARNINGS** 構文を参照してください。**IF EXISTS** 条項が MySQL 5.1.14の中に追加されました。

注:全ての MySQL — MySQL 5.1 を含め — 5.0.10以前の古いバージョンの MySQL を、5.0.10以後の新しいバージョンの物に更新する時、更新する前に 一切のトリガをドロップし、その後で再生させなければなりません。これを怠ると、更新後 **DROP TRIGGER** が作動しなくなります。更新手順に対するヒントについては、[MySQL 5.0 から 5.1 へのアップグレード](#) をご参照ください。

トリガの使用

このセクションで、MySQL 5.1 の中でトリガを使用する方法並びにこれらの使用に対する幾つかの制限について説明します。トリガに対する制約に関する追加情報を、[ストアド ルーチンとトリガの規制](#) に掲載します。

トリガは名称を持つ、テーブルに付属するデータベース オブジェクトで、テーブルに対して特定イベントが発生すると有効化されます。幾つかは、トリガに対して、テーブルに挿入すべき値を対象にチェックを実施するか、更新に含まれる値に関して、計算を実行するのに使用されます。

INSERT ステートメント、**DELETE** ステートメントもしくは **UPDATE** ステートメントがテーブルに対して実行されると、トリガがテーブルに関連付けられ、定義されて有効化されます。トリガはそのステートメントの前か後にセットして有効化する事ができます。例えば、トリガを、各行をテーブルから削除する前または各行を更新した後に有効化させる事ができます。

CREATE TRIGGER ステートメントまたは **DROP TRIGGER** ステートメントを使ってトリガを生成させるか除去してください。これらのステートメントに対する構文は、[CREATE TRIGGER 構文](#) および [DROP TRIGGER 構文](#) で説明されています。

INSERT ステートメントの為にテーブルをトリガに関連付けた例がここにあります。それはそのテーブルのカラムの1つに挿入された値を合計する加算器の役を果たします。

以下のステートメントによって、テーブルとそれに対するトリガが生成されます：

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

CREATE TRIGGER ステートメントは、**account** と関連している **ins_sum** なる名称のトリガを生成させます。それに、トリガに対する有効化の時期、トリガ イベント並びにトリガの有効化に要するその他を規定する条項も含まれています。

- ・ キーワード **BEFORE** はトリガ アクションの時期を示します。この場合、トリガを各行がテーブルに挿入される前に有効化すべきです。ここで許容されるその他のキーワードは **AFTER** です。
- ・ キーワード **INSERT** はトリガを有効化するイベントを示します。例では、**INSERT** ステートメントによってトリガの有効化が引き起こされます。**DELETE** ステートメントおよび **UPDATE** ステートメントに対してトリガを生成させる事ができます。
- ・ **FOR EACH ROW** の後に連なるステートメントは、トリガを有効化するたびに実行すべきステートメントを規定します。これは、トリガに対するステートメントによって影響を被る各行毎に一回発生します。この例では、トリガステートメントは、**amount** 欄に挿入された値を集計する簡単な **SET** です。そのステートメントは、「新しい行に挿入される **amount** カラムの値」を意味する **NEW.amount** として参照されます。

トリガを利用する為には、加算器変数をゼロにセットし、**INSERT** ステートメントを実行し、その後変数がどんな値になったかを調べます：

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----+
| Total amount inserted |
+-----+
| 1852.48               |
+-----+
```

この場合、**INSERT** ステートメントが実行された後の **@sum** の値は $14.98 + 1937.50 - 100$ あるいは **1852.48** です。

トリガを破壊する為には、**DROP TRIGGER** ステートメントを利用してください。そのトリガが初期設定スキーマに含まれていない場合には、スキーマ名を規定しなければなりません。

```
mysql> DROP TRIGGER test.ins_sum;
```

スキーマの名称欄にトリガ名が存在していますが、これは全てのトリガはスキーマの中に固有の名称を持っていないべきではない事を意味します。異なるスキーマの中にあるトリガには同じ名称を付ける事ができます。

スキーマ毎にトリガ名を固有な物としなければいけないという要件に加え、生成させる事ができるトリガのタイプに他の制限があります。特に、有効化の時期と有効化イベントが同じ1つのテーブルに対して、2個のトリガを持つ事はできません。例えば、1つのテーブルに対して、2つの **BEFORE INSERT** トリガもしくは2つの **AFTER UPDATE** トリガを定義する事はできません。(このセクションの後の部分に述べる) **BEGIN ... END** 合成ステートメントの構築を **FOR EACH ROW** の後に使う事によって、複数のステートメントを実行するトリガを定義する事ができるので、これを重要な制限とみなすべきではありません。

OLD なるキーワードと **NEW** なるキーワードを使用すると、トリガによって影響を被る行中のカラムにアクセスする事が可能となります。(**OLD** と **NEW** は大文字でも小文字でも入力する事ができます。) **INSERT** トリガの場合、**NEW.col_name** だけが使用可能です。古い行は存在しません。 **DELETE** トリガの場合、**OLD.col_name** だけが使用可能です。新しい行は存在しません。 **UPDATE** トリガの場合、**OLD.col_name** を使って更新前の行のカラムを、また **NEW.col_name** を使用して、アップデート後の行のカラムをそれぞれ参照する事ができます。

OLD の名称を持つカラムは読み取り専用です。(**SELECT** 権限をお持ちである場合)、それを調べる事ができますが、改訂する事はできません。 **SELECT** 権限をお持ちの場合、**NEW** の名称を持つカラムを調べる事ができます。 **BEFORE** トリガでは、**UPDATE** 権限をお持ちの場合、**SET NEW.col_name = value** を使ってその値を変更する事ができます。これは、トリガを使って、新しい行に挿入される、または行を更新する為に利用される値を変更する事ができるという事を意味します。

BEFORE トリガでは、**AUTO_INCREMENT** カラムに対する **NEW** 値は、新しい記録が実際に挿入される時に自動的に生成されるシーケンス番号ではなく、0となります。

OLD と **NEW** はトリガに対する MySQL 拡張子です。

BEGIN ... END 構築を使用する事によって、複数のステートメントを実行するトリガを定義する事ができます。**BEGIN** ブロックの中で、条件文やループのようなストアド ルーチンの中で許容されているその他の構文を使用する事もできます。しかし、ストアド ルーチンだけに対して、複数のステートメントを実行する1個のトリガを定義する **mysql** プログラムを使用した場合、**mysql** ステートメント デリミタを再定義して、トリガの定義の中で ; ステートメント デリミタを使用可能にする必要があります。次の例はこれらの点を例示しています。それは、各行を更新するのに使用すべき新しい値をチェックする **UPDATE** トリガを定義し、その値を0から100までの範囲に収まるように変更します。その値は、行を更新するのに使用する前にチェックする必要があるため、**BEFORE**トリガでなければなりません。

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
-> FOR EACH ROW
-> BEGIN
-> IF NEW.amount < 0 THEN
-> SET NEW.amount = 0;
-> ELSEIF NEW.amount > 100 THEN
-> SET NEW.amount = 100;
-> END IF;
-> END//
mysql> delimiter ;
```

ストアド プロシージャを別々に定義して、次に単純な **CALL** ステートメントを使ってそれを呼び出して、簡素化することができます。これは、幾つかのトリガから同じルーチンを呼び出したい場合にも有益です。

有効化する時トリガが実行するステートメントに記載する事ができる物に対して、幾つかの制限があります。

- トリガは、データをクライアントに戻すかダイナミック SQL を使用する、ストアド プロシージャを呼び出す **CALL** ステートメントを使用する事ができません。(ストアド プロシージャは、**OUT** または **INOUT** パラメータを通してトリガにデータを返す事を許します。)
- トリガは、**START TRANSACTION**、**COMMIT** や **ROLLBACK** のようなトランザクションを明示的にもしくは暗黙に開始または終了させるステートメントを使用する事ができません。

MySQL はトリガを実行している最中に発生したエラーを以下の通りに処理します：

- **BEFORE**トリガの機能が停止した場合、対応する行の操作が実施されなくなる。
- **BEFORE** トリガが、行を挿入するか改訂する attempt によって、当該試みがその後成功するか否かに関係なく有効化される。
- **AFTER** トリガが、**BEFORE** トリガ(存在している場合) 並びに行操作が両方共うまく実行された場合に限り実行される。
- **BEFORE** トリガあるいは **AFTER** トリガを実行している最中に発生したエラーが、トリガに狂いを引き起こし、これによって、全ステートメントを機能停止にする。
- トランザクション テーブルに対するステートメントの機能停止は、そのステートメントによって実施された一切の変更のロールバックを引き起こすべきです。トリガの機能停止は、ステートメントに機能停止をもたらし、これによって、トリガの機能停止がロールバックを引き起こす。非トランザクション テーブルに対してこのようなロールバックを行う事ができないので、ステートメントが機能停止しても、エラーが発生した時点より前に実施された一切の変更は有効なまま維持されます。

第19章 Event Scheduler

目次

Event Scheduler Overview	1077
Event Scheduler Syntax	1080
CREATE EVENT Syntax	1080
ALTER EVENT Syntax	1083
DROP EVENT Syntax	1085
Event Metadata	1085
Event Scheduler Status	1085
The Event Scheduler and MySQL Privileges	1086
Event Scheduler Limitations and Restrictions	1088

This chapter describes the MySQL Event Scheduler, for which support was added in MySQL 5.1.6, and is divided into the following sections:

- [Event Scheduler Overview](#) provides an introduction to and conceptual overview of MySQL Events.
- [Event Scheduler Syntax](#) discusses the SQL commands introduced in MySQL 5.1.6 for creating, altering, and dropping MySQL Events.
- [Event Metadata](#) shows how to obtain information about events and how this information is stored by the MySQL Server.
- [The Event Scheduler and MySQL Privileges](#) discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.
- [Event Scheduler Limitations and Restrictions](#) describes the restrictions and limitations of MySQL's Event Scheduler implementation.

Additional Resources:

- You may find the [MySQL Event Scheduler User Forum](#) of use when working with events. Here you can discuss the MySQL Event Scheduler with other MySQL users and the MySQL developers.

Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as scheduled events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix [crontab](#) (also known as a 「cron job」) or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as 「temporal triggers」, implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term events in order to avoid confusion with triggers of the type discussed in [18章トリガ](#). Events should more specifically not be confused with 「temporary triggers」. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

- In MySQL 5.1.12 and later, an event is uniquely identified by its name and the schema to which it is assigned. (Previously, an event was also unique to its definer.)
- An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a [BEGIN ... END](#) block if desired (see [BEGIN ... END 複合ステートメント構](#)

文). An event's timing can be either one-time or recurrent. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

- Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. A user may include statements in an event's action which require privileges that the user does not actually have. The event creation or modification statement succeeds but the event's action fails. See [The Event Scheduler and MySQL Privileges](#) for details.
- Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See [ALTER EVENT Syntax](#).

The definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last [ALTER EVENT](#) statement effecting that event. An event can be modified by any user having the [EVENT](#) privilege on the database for which the event is defined. (Prior to MySQL 5.1.12, only an event's definer, or a user having privileges on the `mysql.event` table, could modify a given event.) See [The Event Scheduler and MySQL Privileges](#).

- An event's action statement may include most SQL statements permitted within stored routines.

Events are executed by a special event scheduler thread; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the [SUPER](#) privilege in the output of [SHOW PROCESSLIST](#), as shown in the discussion that follows.

The global variable `event_scheduler` determines whether the Event Scheduler is enabled and running on the server. Beginning with MySQL 5.1.12, it has one of these 3 values, which affect event scheduling as described here:

- **OFF**: The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of [SHOW PROCESSLIST](#), and no scheduled events are executed. **OFF** is the default value for `event_scheduler`.

When the Event Scheduler is stopped (`event_scheduler` is **OFF**), it can be started by setting the value of `event_scheduler` to **ON**. (See next item.)

- **ON**: The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

When the Event Scheduler is **ON**, the event scheduler thread is listed in the output of [SHOW PROCESSLIST](#) as a daemon process, and its state is represented as shown here:

```
mysql> SHOW PROCESSLIST;
***** 1. row *****
  Id: 1
  User: root
  Host: localhost
  db: NULL
  Command: Query
  Time: 0
  State: NULL
  Info: show processlist
***** 2. row *****
  Id: 2
  User: event_scheduler
  Host: localhost
  db: NULL
  Command: Daemon
  Time: 3
  State: Waiting for next activation
  Info: NULL
2 rows in set (0.00 sec)
```

Event scheduling can be stopped by setting the value of `event_scheduler` to **OFF**.

- **DISABLED**: This value renders the Event Scheduler non-operational. When the Event Scheduler is **DISABLED**, the event scheduler thread does not run (and so does not appear in the output of [SHOW PROCESSLIST](#)).

When the server is running `event_scheduler` can be toggled between `ON` and `OFF` (using `SET`). It is also possible to use `0` for `OFF`, and `1` for `ON` when setting this variable. Thus, any of the following 4 statements can be used in the `mysql` client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@global.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@global.event_scheduler = 0;
```

Although `ON` and `OFF` have numeric equivalents, the value displayed for `event_scheduler` by `SELECT` or `SHOW VARIABLES` is always one of `OFF`, `ON`, or `DISABLED`. `DISABLED` has no numeric equivalent. For this reason, `ON` and `OFF` are usually preferred over `1` and `0` when setting this variable.

Note that attempting to set `event_scheduler` without specifying it as a global variable causes an error:

```
mysql> SET @@event_scheduler = OFF;
ERROR 1229 (HY000): Variable 'event_scheduler' is a GLOBAL
variable and should be set with SET GLOBAL
```

Important: It is not possible to enable or disable the Event Scheduler when the server is running. That is, you can change the value of `event_scheduler` to `DISABLED` — or from `DISABLED` to one of the other permitted values for this option — only when the server is stopped. Attempting to do so when the server is running fails with an error.

To disable the event scheduler, use one of the following two methods:

- As a command-line option when starting the server:

```
--event-scheduler=DISABLED
```

- In the server configuration file (`my.cnf`, or `my.ini` on Windows systems), include the line where it will be read by the server (for example, in a `[mysqld]` section):

```
event_scheduler=DISABLED
```

To enable the Event Scheduler, restart the server without the `--event-scheduler=DISABLED` command line option, or after removing or commenting out the line containing `event_scheduler=DISABLED` in the server configuration file, as appropriate. Alternatively, you can use `ON` (or `1`) or `OFF` (or `0`) in place of the `DISABLED` value when starting the server.

Note: You can issue event-manipulation statements when `event_scheduler` is set to `DISABLED`. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to `ON` (or `1`). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

In MySQL 5.1.11, `event_scheduler` behaved as follows: this variable could take one of the values `0` (or `OFF`), `1` (or `ON`), or `2`. Setting it to `0` turned event scheduling off, so that the event scheduler thread did not run; the `event_scheduler` variable could not be set to this value while the server was running. Setting it to `1` so that the event scheduler thread ran and executed scheduled events. In this state, the event scheduler thread appeared to be sleeping when viewed with `SHOW PROCESSLIST`. When `event_scheduler` was set to `2` (which was the default value), the Event Scheduler was considered to be 「suspended」; the event scheduler thread ran and could be seen in the output of `SHOW PROCESSLIST` (where `Suspended` was displayed in the `State` column), but did not execute any scheduled events. The value of `event_scheduler` could be changed only between `1` (or `ON`) and `2` while the server was running. Setting it to `0` (or `OFF`) required a server restart, as did changing its value from `0` (or `OFF`) to `1` (or `ON`) or `2`.

Prior to MySQL 5.1.11, `event_scheduler` could take one of only the 2 values `0|OFF` or `1|ON`, and the default value was `0|OFF`. It was also possible to start and stop the event scheduler thread while the MySQL server was running.

For more information concerning the reasons for these changes in behaviour, see Bug #17619.

For SQL statements used to create, alter, and drop events, see [Event Scheduler Syntax](#).

MySQL 5.1.6 and later provides an [EVENTS](#) table in the [INFORMATION_SCHEMA](#) database. This table can be queried to obtain information about scheduled events which have been defined on the server. See [Event Metadata](#), and [INFORMATION_SCHEMA EVENTS テーブル](#), for more information.

For information regarding event scheduling and the MySQL privilege system, see [The Event Scheduler and MySQL Privileges](#).

Event Scheduler Syntax

MySQL 5.1.6 and later provides several SQL statements for working with scheduled events:

- New events are defined using the [CREATE EVENT](#) statement. See [CREATE EVENT Syntax](#).
- The definition of an existing event can be changed by means of the [ALTER EVENT](#) statement. See [ALTER EVENT Syntax](#).
- When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the [DROP EVENT](#) statement. See [DROP EVENT Syntax](#). (Whether an event persists past the end of its schedule also depends on its [ON COMPLETION](#) clause, if it has one. See [CREATE EVENT Syntax](#).)

An event can be deleted by any user having the [EVENT](#) privilege for the database on which the event is defined. Prior to MySQL 5.12, a user other than the definer required privileges on the `mysql.event` table. See [The Event Scheduler and MySQL Privileges](#).

CREATE EVENT Syntax

```
CREATE EVENT [IF NOT EXISTS] event_name
  ON SCHEDULE schedule
  [ON COMPLETION [NOT] PRESERVE]
  [ENABLE | DISABLE]
  [COMMENT 'comment']
  DO sql_statement;

schedule:
  AT timestamp [+ INTERVAL interval]
  | EVERY interval [STARTS timestamp] [ENDS timestamp]

interval:
  quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
            WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
            DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

This statement creates and schedules a new event. The minimum requirements for a valid [CREATE EVENT](#) statement are as follows:

- The keywords [CREATE EVENT](#) plus an event name, which uniquely identifies the event in the current schema.
(Prior to MySQL 5.1.12, the event name needed to be unique only among events created by the same user on a given database.)
- An [ON SCHEDULE](#) clause, which determines when and how often the event executes.
- A [DO](#) clause, which contains the SQL statement to be executed by an event.

This is an example of a minimal [CREATE EVENT](#) statement:

```
CREATE EVENT myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;
```

The previous statement creates an event named `myevent`. This event executes once — one hour following its creation — by running an SQL statement that increments the value of the `myschema.mytable` table's `mycol` column by 1.

The [event_name](#) must be a valid MySQL identifier with a maximum length of 64 characters. It may be delimited using back ticks, and may be qualified with the name of a database schema. An event is associated with both a MySQL user

(the definer) and a schema, and its name must be unique among names of events within that schema. In general, the rules governing event names are the same as those for names of stored routines. See [識別子](#).

If no schema is indicated as part of `event_name`, then the default (current) schema is assumed. The definer is always the current MySQL user.

(Prior to MySQL 5.1.12, it was possible for two different users to create different events having the same name on the same database schema.)

Note: MySQL uses case-insensitive comparisons when checking for the uniqueness of event names. This means that, for example, you cannot have two events named `myevent` and `MyEvent` in the same database schema.

`IF NOT EXISTS` functions in the much the same fashion with `CREATE EVENT` as it does when used with a `CREATE TABLE` statement; if an event named `event_name` already exists in the same schema, no action is taken, and no error results. (However, a warning is generated.)

The `ON SCHEDULE` clause determines when, how often, and for how long the `sql_statement` defined for the event repeats. This clause takes one of two forms:

- `AT timestamp` is used for a one-time event. It specifies that the event executes one time only at the date and time, given as the `timestamp`, which must include both the date and time, or must be an expression that resolves to a datetime value. You may use a value which is of either the `DATETIME` or `TIMESTAMP` type for this purpose. The `timestamp` must also be in the future — you cannot schedule an event to take place in the past. Trying to do so fails with an error, as shown here:

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2006-02-10 23:59:01 |
+-----+
1 row in set (0.04 sec)

mysql> CREATE EVENT e_totals
-> ON SCHEDULE AT '2006-02-10 23:59:00'
-> DO INSERT INTO test.totals VALUES (NOW());
ERROR 1522 (HY000): Activation (AT) time is in the past
```

`CREATE EVENT` statements which are themselves invalid — for whatever reason — fail with an error.

You may use `CURRENT_TIMESTAMP` to specify the current date and time. In such a case, the event acts as soon as it is created.

In order to create an event which occurs at some point in the future relative to the current date and time — such as that expressed by the phrase 「three weeks from now」 — you can use the optional clause `+ INTERVAL interval`. The `interval` portion consists of two parts, a quantity and a unit of time, and follows the same syntax rules that govern intervals used in the `DATE_ADD()` function (see [日付時刻関数](#)). The units keywords are also the same, except that you cannot use any units involving microseconds when defining an event.

You can also combine intervals. For example, `AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY` is equivalent to 「three weeks and two days from now」. Each portion of such a clause must begin with `+ INTERVAL`.

- For actions which are to be repeated at a regular interval, you can use an `EVERY` clause. The `EVERY` keyword is followed by an `interval` as described in the previous discussion of the `AT` keyword. (`+ INTERVAL` is not used with `EVERY`.) For example, `EVERY 6 WEEK` means 「every six weeks」.

It is not possible to combine `+ INTERVAL` clauses in a single `EVERY` clause; however, you can use the same complex time units allowed in a `+ INTERVAL`. For example, 「every two minutes and ten seconds」 can be expressed as `EVERY '2:10' MINUTE_SECOND`.

An `EVERY` clause may also contain an optional `STARTS` clause. `STARTS` is followed by a `timestamp` value which indicates when the action should begin repeating, and may also use `+ INTERVAL interval` in order to specify an amount of time 「from now」. For example, `EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + 1 WEEK` means 「every three months, beginning one week from now」. Similarly, you can express 「every two weeks,

beginning six hours and fifteen minutes from now」 as `EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + '6:15' HOUR_MINUTE`. Not specifying `STARTS` is the same as using `STARTS CURRENT_TIMESTAMP` — that is, the action specified for the event begins repeating immediately upon creation of the event.

An `EVERY` clause may also contain an optional `ENDS` clause. The `ENDS` keyword is followed by a `timestamp` value which tells MySQL when the event should stop repeating. You may also use `+ INTERVAL interval` with `ENDS`; for instance, `EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK` is equivalent to 「every twelve hours, beginning thirty minutes from now, and ending four weeks from now」. Not using `ENDS` means that the event continues executing indefinitely.

`ENDS` supports the same syntax for complex time units as `STARTS` does.

You may use `STARTS`, `ENDS`, both, or neither in an `EVERY` clause.

Note: Where `STARTS` or `ENDS` is given as a datetime value, it is taken to mean local time on the server. However, the values for both of these are currently reported using Universal Time in the `INFORMATION_SCHEMA.EVENTS` and `mysql.event` tables, as well as in the output from `SHOW EVENTS`. This is not intended behaviour and your application should not rely on it, as it is subject to change (Bug #16420). For additional information, see [INFORMATION_SCHEMA.EVENTS テーブル](#).

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the `timestamp` or `interval` values which it contains. You may not use stored routines or user-defined functions in such expressions, nor may you use any table references; however, you may use `SELECT FROM DUAL`. This is true for both `CREATE EVENT` and `ALTER EVENT` statements. Beginning with MySQL 5.1.13, references to stored routines, user-defined functions, and tables in such cases is specifically disallowed, and fail with an error (see Bug #22830).

Normally, once an event has expired, it is immediately dropped. You can override this behavior by specifying `ON COMPLETION PRESERVE`. Using `ON COMPLETION NOT PRESERVE` merely makes the default non-persistent behavior explicit.

You can create an event but keep it from being active using the `DISABLE` keyword. Alternatively, you may use `ENABLE` to make explicit the default status, which is active. This is most useful in conjunction with `ALTER EVENT` (see [ALTER EVENT Syntax](#)).

You may supply a comment for an event using a `COMMENT` clause. `comment` may be any string of up to 64 characters that you wish to use for describing the event. The comment text, being a string literal, must be surrounded by quotation marks.

The `DO` clause specifies an action carried by the event, and consists of an SQL statement. Nearly any valid MySQL statement which can be used in a stored routine can also be used as the action statement for a scheduled event. (See [ストアド ルーチンとトリガの規制](#).) For example, the following event `e_hourly` deletes all rows from the `sessions` table once per hour, where this table is part of the `site_activity` schema:

```
CREATE EVENT e_hourly
ON SCHEDULE
  EVERY 1 HOUR
COMMENT 'Clears out sessions table each hour.'
DO
  DELETE FROM site_activity.sessions;
```

MySQL stores the `sql_mode` system variable setting that is in effect at the time an event is created, and always executes the event with this setting in force, regardless of the current server SQL mode.

A `CREATE EVENT` statement that contains an `ALTER EVENT` statement in its `DO` clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

Note: The `SHOW` statement and `SELECT` statements that merely return a result set have no effect when used in an event; the output from these is not sent to the MySQL Monitor, nor is it stored anywhere. However, you can use statements such as `SELECT INTO` and `INSERT ... SELECT` that store a result. (See the next example in this section for an instance of the latter.)

Any reference in the `DO` clause to a table in other than the same database schema to which the event belongs must be qualified with the name of the schema in which the table occurs. (In MySQL 5.1.6, all tables referenced in event `DO` clauses had to include a reference to the database.)

As with stored routines, you can use multiple statements in the **DO** clause by bracketing them with the **BEGIN** and **END** keywords, as shown here:

```
DELIMITER |

CREATE EVENT e_daily
ON SCHEDULE
EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day.'
DO
BEGIN
INSERT INTO site_activity.totals (when, total)
SELECT CURRENT_TIMESTAMP, COUNT(*)
FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END |

DELIMITER ;
```

Note the use of the **DELIMITER** statement to change the statement delimiter, as with stored routines. See [CREATE PROCEDURE](#) [および](#) [CREATE FUNCTION](#) [構文](#).

More complex compound statements, such as those used in stored routines, are possible in an event. This example uses local variables, an error handler, and a flow control construct:

```
DELIMITER |

CREATE EVENT e
ON SCHEDULE
EVERY 5 SECOND
DO
BEGIN
DECLARE v INTEGER;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

SET v = 0;

WHILE v < 5 DO
INSERT INTO t1 VALUES (0);
UPDATE t2 SET s1 = s1 + 1;
SET v = v + 1;
END WHILE;
END |

DELIMITER ;
```

There is no way to pass parameters directly to or from events; however, it is possible to invoke a stored routine with parameters:

```
CREATE EVENT e_call_myproc
ON SCHEDULE
AT CURRENT_TIMESTAMP + 1 DAY
DO CALL myproc(5, 27);
```

In addition, if the event's definer has the **SUPER** privilege, that event may read and write global variables. As granting this privilege entails a potential for abuse, extreme care must be taken in doing so.

Generally, any statements which are valid in stored routines may be used for action statements executed by events. For more information about statements allowable within stored routines, see [ストアドルーチン構文](#). You can create an event as part of a stored routine, but an event cannot be created by another event.

ALTER EVENT Syntax

```
ALTER EVENT event_name
[ON SCHEDULE schedule]
[RENAME TO new_event_name]
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE]
[COMMENT 'comment']
[DO sql_statement]
```

The **ALTER EVENT** statement is used to change one or more of the characteristics of an existing event without the need to drop and recreate it. The syntax for each of the **ON SCHEDULE**, **ON COMPLETION**, **COMMENT**, **ENABLE / DISABLE**, and **DO** clauses is exactly the same as when used with **CREATE EVENT**. (See **CREATE EVENT Syntax**.)

Beginning with MySQL 5.1.12, any user can alter an event defined on a database for which that user has the **EVENT** privilege. When a user executes a successful **ALTER EVENT** statement, that user becomes the definer for the effected event.

(In MySQL 5.1.11 and earlier, an event could be altered only by its definer, or by a user having the **SUPER** privilege.)

ALTER EVENT works only with an existing event:

```
mysql> ALTER EVENT no_such_event
> ON SCHEDULE
> EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): Unknown event 'no_such_event'
```

In each of the following examples, assume that the event named **myevent** is defined as shown here:

```
CREATE EVENT myevent
ON SCHEDULE
  EVERY 6 HOUR
COMMENT 'A sample comment.'
DO
  UPDATE myschema.mytable SET mycol = mycol + 1;
```

The following statement changes the schedule for **myevent** from once every six hours starting immediately to once every twelve hours, starting four hours from the time the statement is run:

```
ALTER EVENT myevent
ON SCHEDULE
  EVERY 12 HOUR
STARTS CURRENT_TIMESTAMP + 4 HOUR;
```

To disable **myevent**, use this **ALTER EVENT** statement:

```
ALTER EVENT myevent
DISABLE;
```

The **ON SCHEDULE** clause may use expressions involving built-in MySQL functions and user variables to obtain any of the **timestamp** or **interval** values which it contains. You may not use stored routines or user-defined functions in such expressions, nor may you use any table references; however, you may use **SELECT FROM DUAL**. This is true for both **ALTER EVENT** and **CREATE EVENT** statements. Beginning with MySQL 5.1.13, references to stored routines, user-defined functions, and tables in such cases is specifically disallowed, and fail with an error (see Bug #22830).

An **ALTER EVENT** statement that contains another **ALTER EVENT** statement in its **DO** clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

It is possible to change multiple characteristics of an event in a single statement. This example changes the SQL statement executed by **myevent** to one that deletes all records from **mytable**; it also changes the schedule for the event such that it executes once, one day after this **ALTER EVENT** statement is run.

```
ALTER TABLE myevent
ON SCHEDULE
  AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO
  TRUNCATE TABLE myschema.mytable;
```

To rename an event, use the **ALTER EVENT** statement's **RENAME TO** clause, as shown here:

```
ALTER EVENT myevent
RENAME TO yourevent;
```

The previous statement renames the event **myevent** to **yourevent**. (Note: There is no **RENAME EVENT** statement.)

You can also move an event to a different schema using **ALTER EVENT ... RENAME TO ...** and **schema_name.table_name** notation, as shown here:


```
ALTER EVENT oldschema.myevent
RENAME TO newschema.myevent;
```

In order to execute the previous statement, the user executing it must have the [EVENT](#) privilege on both the [oldschema](#) and [newschema](#) database schemas.

It is necessary to include only those options in an [ALTER EVENT](#) statement which correspond to characteristics that you actually wish to change; options which are omitted retain their existing values. This includes any default values for [CREATE EVENT](#) such as [ENABLE](#).

DROP EVENT Syntax

```
DROP EVENT [IF EXISTS] event_name
```

This statement drops the event named [event_name](#). The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error **ERROR 1517 (HY000): Unknown event 'event_name'** results. You can override this and cause the statement to fail silently by using [IF EXISTS](#).

Beginning with MySQL 5.1.12, an event can be dropped by any user having the [EVENT](#) privilege on the database schema to which the event to be dropped belongs. (In MySQL 5.1.11 and earlier, an event could be dropped only by its definer, or by a user having the [SUPER](#) privilege.)

Event Metadata

Information about events can be obtained as follows:

- Querying the [EVENTS](#) table of the [INFORMATION_SCHEMA](#) database. See [INFORMATION_SCHEMA EVENTS テーブル](#).
- Using the [SHOW EVENTS](#) statement. See [SHOW EVENTS](#).
- Using the [SHOW CREATE EVENT](#) statement. See [SHOW CREATE EVENT](#).
- A record of events executed on the server can be read from the MySQL Server's error log (see [The Event Scheduler and MySQL Privileges](#) for an example).

Event Scheduler Status

Information about the state of the Event Scheduler for debugging and troubleshooting purposes can be obtained as follows:

- In MySQL 5.1.11 [-debug](#) builds, you can use the [SHOW SCHEDULER STATUS](#) statement; see [SHOW SCHEDULER STATUS 構文](#).

Important: This statement was removed in MySQL 5.1.12. We intend to implement an SQL statement providing similar functionality in a future MySQL release.

- Beginning with MySQL 5.1.12, event scheduler status information can be obtained by running the [mysqladmin debug](#) (see [mysqladmin — MySQL サーバの管理を行うクライアント](#)); after running this command, the error log contains output relating to the Event Scheduler, similar to what is shown here:

```
Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA        : init_scheduler:313
LUA        : init_scheduler:318
WOC        : NO
Workers    : 0
```

```

Executed : 0
Data locked: NO

Event queue status:
Element count : 1
Data locked : NO
Attempting lock : NO
LLA : init_queue:148
LUA : init_queue:168
WOC : NO
Next activation : 0000-00-00 00:00:00

```

The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global `event_scheduler` variable. This requires the `SUPER` privilege.

MySQL 5.1.6 introduces a privilege governing the creation, modification, and deletion of events, the `EVENT` privilege. This privilege can be bestowed using `GRANT`. For example, this `GRANT` statement confers the `EVENT` privilege for the schema named `myschema` on the user `jon@ghidora`:

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the `EVENT` privilege on all schemas would require the following statement:

```
GRANT EVENT ON *.* TO jon@ghidora;
```

The `EVENT` privilege has schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

```

mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): Illegal GRANT/REVOKE command; please
consult the manual to see which privileges can be used

```

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that `jon@ghidora` has the `EVENT` privilege for `myschema`. Suppose also that this user has the `SELECT` privilege for `myschema`, but no other privileges for this schema. It is possible for `jon@ghidora` to create a new event such as this one:

```

CREATE EVENT e_store_ts
ON SCHEDULE
EVERY 10 SECOND
DO
INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());

```

The user waits for a minute or so, and then performs a `SELECT * FROM mytable;` query, expecting to see several new rows in the table. Instead, he finds that the table is empty. Since he does not have the `INSERT` privilege for the table in question, the event has no effect.

If you inspect the MySQL error log (`hostname.err`), you can see that the event is executing, but the action it is attempting to perform fails, as indicated by `RetCode=0`:

```

060209 22:39:44 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:44 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
060209 22:39:54 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:54 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0
060209 22:40:04 [Note] EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:40:04 [Note] EVEX EXECUTED event newdb.e [EXPR:10]. RetCode=0

```

Since this user very likely does not have access to the error log, he can verify whether the event's action statement is valid by running it himself:

```

mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
ERROR 1142 (42000): INSERT command denied to user
'jon'@'ghidora' for table 'mytable'

```

Inspection of the `INFORMATION_SCHEMA.EVENTS` table shows that `e_store_ts` exists and is enabled, but its `LAST_EXECUTED` column is `NULL`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME='e_store_ts'
> AND EVENT_SCHEMA='myschema'\G
***** 1. row *****
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_store_ts
DEFINER: jon@ghidora
EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 5
INTERVAL_FIELD: INTERVAL_SECOND
SQL_MODE: NULL
STARTS: 0000-00-00 00:00:00
ENDS: 0000-00-00 00:00:00
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2006-02-09 22:36:06
LAST_ALTERED: 2006-02-09 22:36:06
LAST_EXECUTED: NULL
EVENT_COMMENT:
1 row in set (0.00 sec)
```

(Note: Prior to MySQL 5.1.12, there was no `EVENT_DEFINITION` column, and `EVENT_BODY` contained the SQL statement or statements to be executed. See [INFORMATION_SCHEMA EVENTS テーブル](#), for more information.)

To rescind the `EVENT` privilege, use the `REVOKE` statement. In this example, the `EVENT` privilege on the schema `myschema` is removed from the `jon@ghidora` user account:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

重要

Revoking the `EVENT` privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of the renaming or dropping of the user who created it.

For example, suppose that that user `jon@ghidora` has been granted the `EVENT` and `INSERT` privileges on the `myschema` schema. This user then creates the following event:

```
CREATE EVENT e_insert
ON SCHEDULE
EVERY 7 SECOND
DO
INSERT INTO myschema.mytable;
```

After this event has been created, `root` revokes the `EVENT` privilege for `jon@ghidora`. However, `e_insert` continues to execute, inserting a new row into `mytable` each seven seconds. The same would be true if `root` had issued either of these statements:

- `DROP USER jon@ghidora;`
- `RENAME USER jon@ghidora TO someotherguy@ghidora;`

You can verify that this is true by examining the `mysql.event` table (discussed later in this section) or the `INFORMATION_SCHEMA.EVENTS` table (see [INFORMATION_SCHEMA EVENTS テーブル](#)) before and after issuing a `DROP USER` or `RENAME USER` statement.

Event definitions are stored in the `mysql.event` table, which was added in MySQL 5.1.6. To drop an event created by another user account, the MySQL `root` user (or another user with the necessary privileges) can delete rows from this table. For example, to remove the event `e_insert` shown previously, `root` can use the following statement:

```
DELETE FROM mysql.event
WHERE db = 'myschema'
AND definer = 'jon@ghidora'
AND name = 'e_insert';
```

It is very important to match the event name, database schema name, and user account when deleting rows from the `mysql.event` table. This is because the same user can create different events of the same name in different schemas.

Note: The namespace for scheduled events changed in MySQL 5.1.12. Prior to that MySQL version, different users could create different events having the same name in the same database; in MySQL 5.1.12 and later, that is no longer the case. When upgrading to MySQL 5.1.12 or later from MySQL 5.1.11 or earlier, it is extremely important to make sure that no events in the same database share the same name, prior to performing the upgrade.

Users' `EVENT` privileges are stored in the `Event_priv` columns of the `mysql.user` and `mysql.db` tables. In both cases, this column holds one of the values 'Y' or 'N'. 'N' is the default. `mysql.user.Event_priv` is set to 'Y' for a given user only if that user has the global `EVENT` privilege (that is, if the privilege was bestowed using `GRANT EVENT ON *.*`). For a schema-level `EVENT` privilege, `GRANT` creates a row in `mysql.db` and sets that row's `Db` column to the name of the schema, the `User` column to the name of the user, and the `Event_priv` column to 'Y'. There should never be any need to manipulate these tables directly, since the `GRANT EVENT` and `REVOKE EVENT` statement perform the required operations on them.

MySQL 5.1.6 introduces five status variables providing counts of event-related operations (but not of statements executed by events — see [Event Scheduler Limitations and Restrictions](#)). These are:

- `Com_create_event`: The number of `CREATE EVENT` statements executed since the last server restart.
- `Com_alter_event`: The number of `ALTER EVENT` statements executed since the last server restart.
- `Com_drop_event`: The number of `DROP EVENT` statements executed since the last server restart.
- `Com_show_create_event`: The number of `SHOW CREATE EVENT` statements executed since the last server restart.
- `Com_show_events`: The number of `SHOW EVENTS` statements executed since the last server restart.

You can view current values for all of these at one time by running the statement `SHOW STATUS LIKE '%event%';`.

Event Scheduler Limitations and Restrictions

This section lists restrictions and limitations applying to event scheduling in MySQL.

In MySQL 5.1.6, any table referenced in an event's action statement must be fully qualified with the name of the schema in which it occurs (that is, as `schema_name.table_name`).

An event may not be created, altered, or dropped by a trigger, stored routine, or another event. An event also may not create, alter, or drop triggers or stored routines. (Bug #16409, Bug #18896)

Event timings using the intervals `YEAR`, `QUARTER`, `MONTH`, and `YEAR_MONTH` are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition — due to rounding, the nature of threaded applications, and the fact that a non-zero length of time is required to create events and to signal their execution — events may be delayed by as much as 1 or 2 seconds. However, the time shown in the `INFORMATION_SCHEMA.EVENTS` table's `LAST_EXECUTED` column or the `mysql.event` table's `last_executed` column is always accurate to within one second of the time the event was actually executed. (See also Bug #16522.)

Execution of event statements has no effect on the server's statement counts such as `Com_select` and `Com_insert` that are displayed by `SHOW STATUS`.

Prior to MySQL 5.1.12, you could not view another user's events in the `INFORMATION_SCHEMA.EVENTS` table. In other words, any query made against this table was treated as though it contained the condition `DEFINER = CURRENT_USER()` in the `WHERE` clause.

Events cannot be created with a start time that is in the past.

Events do not support times later than the end of the Unix Epoch; this is approximately the end of the year 2037. Prior to MySQL 5.1.8, handling in scheduled events of dates later than this was buggy; starting with MySQL 5.1.8, such dates are specifically disallowed by the Event Scheduler. (Bug #16396)

In MySQL 5.1.6, [INFORMATION_SCHEMA.EVENTS](#) shows [NULL](#) in the [SQL_MODE](#) column. Beginning with MySQL 5.1.7, the [SQL_MODE](#) displayed is that in effect when the event was created.

In MySQL 5.1.6, the only way to drop or alter an event created by a user who was not the definer of that event was by manipulation of the [mysql.event](#) system table by the MySQL [root](#) user or by another user with privileges on this table. Beginning with MySQL 5.1.7, [DROP USER](#) drops all events for which that user was the definer; also beginning with MySQL 5.1.7 [DROP SCHEMA](#) drops all events associated with the dropped schema.

As with stored routines, events are not migrated to the new schema by the [RENAME SCHEMA](#) (or [RENAME DATABASE](#)) statement. See [RENAME DATABASE](#) [構文](#).

Beginning with MySQL 5.1.8, event names are handled in case-insensitive fashion. For example, this means that you cannot have two events in the same database (and — prior to MySQL 5.1.12 — with the same definer) with the names [anEvent](#) and [AnEvent](#). Important: If you have events created in MySQL 5.1.7 or earlier, which are assigned to the same database and have the same definer, and whose names differ only with respect to lettercase, then you must rename these events to respect case-sensitive handling before upgrading to MySQL 5.1.8 or later.

References to stored routines, user-defined functions, and tables in the [ON SCHEDULE](#) clauses of [CREATE EVENT](#) and [ALTER EVENT](#) statements are not supported. Beginning with MySQL 5.1.13, these sorts of references are disallowed. (See Bug #22830 for more information.)

第20章 ビュー

目次

ALTER VIEW 構文	1091
CREATE VIEW 構文	1091
DROP VIEW 構文	1097

ビュー(更新可能なビューを含む)はMySQL Server 5.1から入手することができます。

MySQL 5.1 中のビューに関する一般的な質問に対する答えについては、[MySQL 5.1 FAQ — Views](#)を参照してください。

この章では以下のトピックについて説明します。

- [CREATE VIEW](#) または [ALTER VIEW](#) を使用して行うビューの生成と変更
- [DROP VIEW](#) を使用して行うビューの破壊

ビューの使用制限に関する説明については [ビューの規制](#) を参照してください。

旧バージョンMySQLを5.1にアップグレードした場合、ビューの使用はビュー関連の権限を含むようにグラントテーブルもアップグレードしてください。[mysql_upgrade — MySQL アップグレードのテーブル チェック](#) を参照してください。

ビューに関するメタデータは、[SHOW CREATE VIEW](#) ステートメントを使用することによって、[INFORMATION_SCHEMA.VIEWS](#) テーブルから取得することができます。[INFORMATION_SCHEMA VIEWS テーブル](#)、[SHOW CREATE VIEW 構文](#) を参照して下さい。

ALTER VIEW 構文

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

このステートメントは存在しているはずのビューの定義を変更します。構文は[CREATE VIEW](#)のためのそれと類似していて、その効果は[CREATE OR REPLACE VIEW](#)のためのものと同じです。[CREATE VIEW 構文](#)を参照してください。このステートメントには、ビューに対して[CREATE VIEW](#)権限と[DROP](#)権限が要求され、[SELECT](#)ステートメントに引用された各コラムに対して、幾つかの権限が要求されます。

CREATE VIEW 構文

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

このステートメントは新しいビューを生成させるか、[OR REPLACE](#)節を附与すると、既存のビューを他のものと交換します。そのビューが存在しない場合、[CREATE OR REPLACE VIEW](#)は[CREATE VIEW](#)と同じになります。ビューが存在する場合、[CREATE OR REPLACE VIEW](#)は[ALTER VIEW](#)と同じになります。[select_statement](#)はビューの定義を提供する[SELECT](#)ステートメントです。ステートメントはベーステーブルまたはその他のビューから選択することができます。

このステートメントには、ビューに対するCREATE VIEW権限とSELECTステートメントによって選択された各カラムに対して幾つかの権限が要求されます。SELECTステートメントの中で使用されている他のカラムに対して、SELECT権限を所持していなければなりません。OR REPLACE節が存在している場合、ビューのDROP権限を所持していなければなりません。

ビューはデータベースに付随します。デフォルト設定によって、デフォルトデータベースの中に新しいビューが生成されます。あるデータベース中にビューを明確に生成させるには、ビューを生成するとき、その名称をdb_name.view_nameと規定してください。

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

ベーステーブルとビューはデータベース中で同じ名称スペースを共有しているので、データベースに同じ名称のベーステーブルとビューを含めることはできません。

ビューには、ベーステーブルと同じように、ユニークなカラム名を重複することなく持たせなければなりません。デフォルト設定に基づき、SELECTステートメントによって復元されたカラムの名称がビューカラム名に対して使用されます。ビューカラムに対して明確な名称を規定するため、オプションのcolumn_list節をコンマで区切りをつけた識別子のリストとして附与することができます。column_listの中に入れる名称の数はSELECTステートメントによって復元されたカラムの数と同じでなければなりません。

SELECTステートメントによって復元されたカラムはテーブルカラムを引用するシンプルなものにすることができま。これらは関数、定数値、オペレータ等を使用した表現にすることもできます。

SELECTステートメント中の不適切なテーブル名あるいはビュー名はデフォルトデータベースに対して解釈されます。ビューには、適切なデータベース名を使ってそのテーブルまたはビュー名に資格を附与することによって、他のデータベース中のテーブルまたはビューを引用することができます。

ビューは多くの種類のSELECTステートメントから生成することができます。そこから、ベーステーブルまたはビューを参照することができます。ジョイン、UNIONおよびサブクエリーを使用することができます。SELECTはテーブルさえ参照する必要がありません。次の例は他のテーブルから2つのカラム並びにそれらのカラムから計算された表現を選択したビューを定義します。

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+-----+-----+-----+
| qty | price | value |
+-----+-----+-----+
| 3 | 50 | 150 |
+-----+-----+-----+
```

ビュー定義は以下の制限に規定されます。

- SELECTステートメントはFROM節の中にサブクエリーを含めることができません。
- SELECTステートメントはシステム変数もしくはユーザー変数を参照することができません。
- SELECTステートメントは準備されたステートメントパラメータを参照することができません。
- ストアドルーチン内で、定義はルーチンパラメータもしくはローカル変数を参照することができません。
- 定義で参照したテーブルもしくはビューは存在しなければいけません。ただし、ビューを生成し終えた後に、定義が参照するテーブルまたはビューを撤去することができます。この場合、ビューの使用はエラーとなります。この類の問題に対してビュー定義をチェックするには、CHECK TABLEステートメントを使用してください。
- 定義はTEMPORARYテーブルを参照できない上、TEMPORARYビューを生成させることができません。
- ビュー定義内で名称を持つテーブルは存在していなければいけません。
- トリガにビューに関連させることはできません。

ORDER BYはビュー定義の中で許容されていますが、それ自身ORDER BYを持つステートメントを使ってビューから選択すると無視されます。

定義中の他のオプションあるいは節に対して、オプションまたはビューを参照するステートメントの節が追加されましたが、その効果は定義されていません。例えば、ビュー定義にLIMIT節が含まれているとき、それ自身のLIMIT節を持つステートメントを使って選択すると、いずれの限界が適用されるかが定義されていません。SELECTキーワードに従うALL、DISTINCTまたはSQL_SMALL_RESULTのようなオプション並びにINTO、FOR UPDATE、LOCK IN SHARE MODEおよびPROCEDUREのような節に関しては、同じ原理が適用します。

ビューを生成させてから、システム変数を変えることによってクエリ処理環境を変えると、ビューから得る結果が影響されることがあります。

```
mysql> CREATE VIEW v AS SELECT CHARSET(CHAR(65)), COLLATION(CHAR(65));
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SET NAMES 'latin1';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM v;
+-----+-----+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-----+-----+
| latin1           | latin1_swedish_ci   |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM v;
+-----+-----+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-----+-----+
| utf8              | utf8_general_ci     |
+-----+-----+
1 row in set (0.00 sec)
```

DEFINER節およびSQL SECURITY節はビューの呼び出しにおいて、アクセス権限をチェックするとき使用すべきセキュリティコンテキストを規定します。これらはMySQL 5.1.2で追加されています。

CURRENT_USERをCURRENT_USER()として附与することもできます。

SQL SECURITY DEFINER特徴を使って定義されているストアド ルーチン内で、CURRENT_USERはルーチン生成者に返します。ビュー定義の中にCURRENT_USERのDEFINER値が含まれている場合、これは、当該ルーチン中で規定されたビューに影響を及ぼします。

DEFINERのデフォルト値はCREATE VIEWステートメントを実行するユーザです。(これはDEFINER = CURRENT_USERと同じです。)user値を附与する場合、それを「'user_name'@'host_name'」フォーマット (GRANTステートメントに使用したと同じフォーマット) の中にあるMySQLアカウントとすべきです。user_name の値とhost_name の値が両方共要求されます。

DEFINER節を規定する場合、SUPER権限を持っていない限り、ユーザの値を除くいかなるアカウントにも値を設定することはできません。これらの規則は有効なDEFINER ユーザ値を決定します。

- SUPER権限を持っていない場合、文字によるか、CURRENT_USERを使って規定されているuser値だけが有効なユーザアカウントとなります。デファイナーを他のアカウントに設定することはできません。
- SUPER権限を持っている場合、構文的に有効なアカウントネームを規定することができます。そのアカウントが実在しない場合、警告が生成されます。

SQL SECURITY特徴はビューの実行においてビューに対するアクセス権限をチェックする時、どのMySQLアカウントを使用すべきかを決定します。有効な特徴値はDEFINERとINVOKERです。これらはそれぞれ、ビューがそれを定義が起動したユーザによって実行可能でなければならないことを示します。SQL SECURITYのデフォルト値はDEFINERです。

(DEFINER 節とSQL SECURITY節が実装された時)MySQL 5.1.2 以降、ビュー権限はこのようにしてチェックされます。

- ビューを定義するとき、ビュー作成者は、ビューがアクセスしたトップレベルのオブジェクトの使用に要する権限を持っていない限りなりません。例えば、ビュー定義が保存されているファンクションを参照する時、ファンク

ションを起動するために必要な権限だけをチェックすればよい場合があります。ファンクションを作動させるに要する権限は、それを実行するときにだけチェックすることができます。ファンクションを呼び出す方法が異なると、ファンクション中にある別な実行経路を使用しなければならない場合があります。

- ビューの実行において、**SQL SECURITY**特性が**DEFINER**あるいは**INVOKER**であるか否かによって、ビューがアクセスしたオブジェクトに対する権限が、ビュー生成者あるいはインボーカーが保持する権限と照合してそれぞれチェックされます。
- ビューの実行が保存されているファンクションの実行を引き起こす場合、ファンクションの中で実行されたステートメントに対して権限チェックを実行するか否かは、ファンクションが**DEFINER**あるいは**INVOKER**の**SQL SECURITY**ファンクションを使用して規定されているか否かによって決まります。セキュリティー特徴が**DEFINER**である場合、ファンクションはその生成者の権限を使って作動します。その特徴が**INVOKER**である場合、ファンクションはビューの**SQL SECURITY**特徴によって決定された権限を使用して作動します。

(**DEFINER**および**SQL SECURITY**節が実装される前の)5.1.2以前のMySQLの場合、ビュー中でオブジェクトの使用に要する権限はビュー生成時にチェックされます。

例:ビューは保存されているファンクションに依存し、そのファンクションは他ストアルーチンを起動する場合があります。例えば、以下のビューは保存されているファンクション**f()**を起動します。

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

f()にこのようなステートメントが含まれているとします。

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

f()を実行するとき、**f()**の中でステートメントの実行に要する権限をチェックする必要があります。これは、**f()**中の実行経路によって、**p1()**もしくは**p2()**に対する権限をチェックする必要があることを意味します。それらの権限はランタイム時にチェックする必要があります。権限を所持していなければならないユーザであるか否かはファンクション**f()**の**SQL SECURITY**値とビュー**v**によって定義されます。

ビューに対する**DEFINER**節および**SQL SECURITY**節は標準SQLの拡張子です。標準SQLでは、ビューは**SQL SECURITY INVOKER**に対する規則を使って処理されます。

5.0.13/5.1.2以前のMySQLで生成されたビューを呼び出す場合、それは、**SQL SECURITY DEFINER** 節並びにユーザのアカウントと同じ**DEFINER**値を使って生成されたものとして処理されます。しかし、実際のデファイナーが未知なので、MySQLは警告を発行します。警告を撤去するには、ビューを再び生成させて、ビュー定義に**DEFINER**節を含めれば十分です。

オプションの**ALGORITHM**節は標準MySQL の拡張子です。**ALGORITHM**には3つの値が付いています:**MERGE**、**TEMPTABLE**または**UNDEFINED**。**ALGORITHM**節がある場合、デフォルトアルゴリズムは**UNDEFINED**となります。アルゴリズムはMySQL がビューを処理する方法に影響を及ぼします。

MERGEの場合、ビューが参照するステートメントの本文とビュー定義が併合され、ビュー定義の部分が対応するステートメントの部分と取り替えられます。

TEMPTABLEの場合、ビューの結果がテンポラリーテーブルの中に復元され、その後、ステートメントを実行するために使用されます。

UNDEFINEDの場合、MySQLは使用すべきアルゴリズムを選択します。それは出来るだけ**TEMPTABLE**より**MERGE**を優先します。これは、**MERGE**は通常より効率的で、ビューはテンポラリーテーブルを使用すると更新可能ではなくなるためです。

明確に**TEMPTABLE**を選択する理由は、テンポラリーテーブルを選んだ後ステートメントの処理終了に使用する前に、内在するテーブルのロックを解放することができるからです。その結果、ロックを**MERGE**アルゴリズムよりも速やかに解除し、ビューを使う他のクライアントが長時間ブロックされないようにします。

以下に示す3つの理由によって、ビューアルゴリズムを**UNDEFINED**にすることができます。

- **CREATE VIEW**ステートメントの中に**ALGORITHM**節が現れない。

- **CREATE VIEW**ステートメントに**ALGORITHM = UNDEFINED**節が明確に含まれている。
- テンポラリテーブルだけを使って処理できるビューに対して、**ALGORITHM = MERGE**が規定される。この場合、MySQL は警告を発し、アルゴリズムを**UNDEFINED**にセットします。

前に述べたように、**MERGE**は、ビューを参照するステートメントの一部を該当するビュー定義の部分と併合することによって処理されます。次の例で、**MERGE**アルゴリズムが作動する方法を簡単に図解説明します。例にこの定義が含まれているビュー**v_merge**が存在していると見なすと：

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

例 1。我々がこのステートメントを発行すると仮定すると：

```
SELECT * FROM v_merge;
```

MySQLはステートメントを以下の通りに処理します：

- **v_merge**は**t**となる
- *****は**vc1, vc2** となり、**c1, c2**と一致する
- ビュー**WHERE**節が追加される

実行すべき結果ステートメントは以下の通りとなります。

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

例2。このステートメントを発行すると仮定します：

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

このステートメントは、**vc1 < 100**が**c1 < 100**になり、接続詞**AND**を使ってビュー**WHERE**節がステートメント**WHERE**節に追加され(更に、かっこを追加して、その節の部分が前例を正しく使って実行されていることを確かめる)以外、前のステートメントと同様に処理されます。実行すべき結果ステートメントは以下の通りとなります。

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

実行されるステートメントは結果的にこの形式の **WHERE**節を持ちます。

```
WHERE (select WHERE) AND (view WHERE)
```

MERGEアルゴリズムは、ビュー中の行と基礎テーブル内の行の間に 1 対 1 の関係が要求されます。この関係が保持されない場合、代わりにテンポラリテーブルを使用しなければなりません。ビューに多くの生成子が含まれると、一対一の関係に不足が生じます。

- 集約ファンクション (**SUM()**、**MIN()**、**MAX()**、**COUNT()**等)
- **DISTINCT**
- **GROUP BY**
- **HAVING**
- **UNION** もしくは**UNION ALL**
- 選択リスト中のサブ・クエリ
- 文字値だけを参照(この場合、基礎となるテーブルは存在しません)

幾つかのビューは更新可能です。すなわち、基礎をなすテーブルの内容を更新するため、**UPDATE**、**DELETE**もしくは**INSERT**のようなステートメントの中でそれらを使うことができます。ビューを更新可能にするため、ビュー中の行と基礎テーブル中の行の間に1対1の関係が存在しなければなりません。ビューを更新不能にするその他の生成子もあります。もっと具体的に言うと、それが以下のいずれかを含んでいるとビューは更新可能となりません。

- 集約ファンクション(**SUM()**、**MIN()**、**MAX()**、**COUNT()**等)

- **DISTINCT**
- **GROUP BY**
- **HAVING**
- **UNION** もしくは **UNION ALL**
- 選択リスト中のサブ・クエリ
- 特定結合 (このセクション中の後の部分に追加した結合の説明参照)
- **FROM**節中の更新不能ビュー
- **FROM**節中のテーブルを参照する**WHERE** 節中のサブ・クエリ
- 文字値だけを参照(この場合、更新する基礎となるテーブルは存在しません)
- **ALGORITHM = TEMPTABLE** (テンポラリテーブルの使用は常にビューを更新不能にする)

(**INSERT**ステートメントで更新不能となる)挿入性に関して、それがビューカラムに対するこれらの追加条件も満たすと、更新不能ビューが挿入可能になります。

- ビューカラム名に重複があってはなりません。
- ビューには、デフォルト値を持っていないベーステーブル内にあるすべてのカラムを含んでいなくてはなりません。
- ビューカラムは派生カラムではなく、単純なカラムリファレンスでなければなりません。派生カラムは単純なカラムリファレンスでなく、表現から派生したものです。これらは派生カラムの例です。

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

単純なカラムリファレンスと派生カラムを混合して持つビューは挿入できません。しかし、当該ビューは、派生したものでないこれらのカラムだけをアップデートする場合に限り更新することができます。このビューを想定すると：

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

このビューは、**col2**が表現から派生しているので挿入できません。しかし、**col2**を更新しようとしていない場合、それはアップデートすることができます。このアップデートは許容されます：

```
UPDATE v SET col1 = 0;
```

このアップデートは、それが派生カラムをアップデートしようとしているので許容されません。

```
UPDATE v SET col2 = 0;
```

MERGEアルゴリズムで処理することができると仮定すると、場合によってマルチ・テーブルビューをアップデートすることが可能です。これを実現するには、ビューに(外部結合または**UNION**でなく)内部結合を使用しなければなりません。また、ビュー定義に含まれている1つのテーブルだけがアップデート可能です。よって、**SET**節に基づき、ビュー中の1つのテーブルからカラムだけ選択して名前をつけなくてはなりません。**UNION ALL**を使用しているビューは、理論的に更新可能かもしれませんが、実装は処理にテンポラリテーブルを使用するので拒絶されます。

更新可能なマルチテーブルビューの場合、それを1つのテーブルに挿入すると、**INSERT**を作動させることができます。**DELETE**はサポートされません。

INSERT DELAYEDはビューでサポートされていません。

テーブルが**AUTO_INCREMENT**カラムを含んでいないテーブル上にある挿入可能なビューに挿入する**AUTO_INCREMENT**カラムを含んでいる場合、ビューの一部でないカラムにデフォルト値を挿入した副作用が見えないので、カラムは**LAST_INSERT_ID()**の値を変更しません。

それに対する`select_statement`中の`WHERE`節が真実であるものを除く行に、更新不能なビューが挿入されるか、当該行が更新されるのを回避するため、`WITH CHECK OPTION`節を附与することができます。

更新可能なビューに対する`WITH CHECK OPTION`節に基づき、`LOCAL`と`CASCADE`キーワードはビューが他のビューに対して定義される場合、チェックテストの範囲を決めます。定義されているビューだけに対して、`LOCAL`キーワードは`CHECK OPTION`を制限します。`CASCADE`は同様に基礎ビューを評価するチェックを起動させます。キーワードが附与されない場合、デフォルト設定は`CASCADE`となります。以下のテーブル並びにビューのセットを考慮すると：

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
-> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
-> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
-> WITH CASCADE CHECK OPTION;
```

ここでは、`v2`ビューと`v3`ビューが他のビューに対して定義され、`v1`、`v2`には`LOCAL` チェックオプションが含まれています。従って、挿入は`v2`チェックだけに対してテストされます。`v3`には`CASCADE` チェックオプションが含まれているので、挿入は、自身のチェックのみならず、基礎ビューに対してもテストされます。以下のステートメントはこれらの違いを例示したものです。

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

ビューの更新は`updatable_views_with_limit` システム変数の値に影響されます。[システム変数](#) を参照してください。

DROP VIEW 構文

```
DROP VIEW [IF EXISTS]
view_name [, view_name] ...
[RESTRICT | CASCADE]
```

`DROP VIEW`は複数のビューを除去します。各ビューごとに`DROP`権限を所持していなければなりません。アーギュメントリストの中に名前を登録したビューのいずれかが存在しない場合、MySQLは存在していなかったため除去できなかったビューを名称別に示して、エラーを返します。しかし、MySQLはリスト中に存在するビューもすべて除去します。

`IF EXISTS`節は存在しないビューに対してエラーが発生するのを回避します。この節を附与すると、存在しない各ビューに対して`NOTE`が生成されます。[SHOW WARNINGS 構文](#) を参照してください。

`RESTRICT`と`CASCADE`は、附与しても構文解析されて無視されます。

第21章 INFORMATION_SCHEMA データベース

目次

INFORMATION_SCHEMA SCHEMATA テーブル	1101
INFORMATION_SCHEMA TABLES テーブル	1101
INFORMATION_SCHEMA COLUMNS テーブル	1102
INFORMATION_SCHEMA STATISTICS テーブル	1103
INFORMATION_SCHEMA USER_PRIVILEGES テーブル	1104
INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル	1104
INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル	1105
INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル	1105
INFORMATION_SCHEMA CHARACTER_SETS テーブル	1106
INFORMATION_SCHEMA COLLATIONS テーブル	1106
INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル	1106
INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル	1106
INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル	1107
INFORMATION_SCHEMA ROUTINES テーブル	1108
INFORMATION_SCHEMA VIEWS テーブル	1109
INFORMATION_SCHEMA TRIGGERS テーブル	1109
INFORMATION_SCHEMA PLUGINS テーブル	1111
INFORMATION_SCHEMA ENGINES テーブル	1111
INFORMATION_SCHEMA PARTITIONS テーブル	1111
INFORMATION_SCHEMA EVENTS テーブル	1114
INFORMATION_SCHEMA FILES テーブル	1117
INFORMATION_SCHEMA PROCESSLIST テーブル	1121
INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル	1122
INFORMATION_SCHEMA GLOBAL_STATUS および SESSION_STATUS テーブル	1123
INFORMATION_SCHEMA GLOBAL_VARIABLES および SESSION_VARIABLES テーブル	1123
その他の INFORMATION_SCHEMA テーブル	1123
SHOW ステートメントへの拡張	1123

INFORMATION_SCHEMA はデータベース メタデータへのアクセスを提供します。

メタデータ は、データベース名またはテーブル名、カラムのデータタイプ、あるいはアクセス権限などのデータに関するデータです。この情報に時々使用される他の用語にはデータ ディクショナリおよびシステム カタログがあります。

INFORMATION_SCHEMA は情報のデータベースで、MySQL サーバーが保持する他のすべてのデータベースに関する情報を保存しています。INFORMATION_SCHEMA の中にはいくつかの読み出し専用テーブルがあります。それらは実際はベーステーブルではなく表示ですので、それらに関連付けされたファイルはありません。

実際には INFORMATION_SCHEMA という名前のデータベースがありますが、その名前ではサーバーはデータベースディレクトリを作成しません。INFORMATION_SCHEMA を USE ステートメントでデフォルトのデータベースとして選択できますが、それはテーブルのコンテンツを読むことしかできません。挿入、更新、および削除はできません。

ここに INFORMATION_SCHEMA から情報を取り出すステートメントの例を示します。

```
mysql> SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name DESC;
+-----+-----+-----+
| table_name | table_type | engine |
+-----+-----+-----+
| v56       | VIEW      | NULL   |
| v3        | VIEW      | NULL   |
| v2        | VIEW      | NULL   |
| v         | VIEW      | NULL   |
| tables    | BASE TABLE | MyISAM |
```

```

| t7      | BASE TABLE | MyISAM |
| t3      | BASE TABLE | MyISAM |
| t2      | BASE TABLE | MyISAM |
| t       | BASE TABLE | MyISAM |
| pk      | BASE TABLE | InnoDB |
| loop    | BASE TABLE | MyISAM |
| kurs    | BASE TABLE | MyISAM |
| k       | BASE TABLE | MyISAM |
| into    | BASE TABLE | MyISAM |
| goto    | BASE TABLE | MyISAM |
| fk2     | BASE TABLE | InnoDB |
| fk      | BASE TABLE | InnoDB |
+-----+-----+
17 rows in set (0.01 sec)

```

説明：ステートメントはデータベース `db5` 内のすべてのテーブルのリストを要求し、アルファベットの逆の順序で、次の 3 項目の情報を表示します。テーブル名、テーブルタイプ、およびそのストレージ エンジン。

各 MySQL ユーザーはこれらのテーブルへのアクセス権限がありますが、ユーザーが適切なアクセス権限を持つオブジェクトに一致するテーブルの行のみ表示することができます。しかし、場合によっては (例えば、`INFORMATION_SCHEMA.ROUTINES` テーブルの `ROUTINE_DEFINITION` カラム)、不十分な権限を有するユーザーには `NULL` が表示される場合があります。

`SELECT ...FROM INFORMATION_SCHEMA` ステートメントは様々な `SHOW` ステートメント、つまり MySQL がサポートする (`SHOW DATABASES`、`SHOW TABLES`、など) により提供された情報へのアクセスをさらに一貫した手法を意図したものです。 `SELECT` は、`SHOW` に比べて有利な点が 3 つあります。

- それは Codd の規則に合致していることです。つまり、すべてのアクセスはテーブルで行われます。
- 新しいステートメント構文を学ぶ必要はありません。なぜなら既に `SELECT` の機能を理解されており、オブジェクト名を知るだけでいいからです。
- インプリメンターはキーワードを加える心配する必要がありません。
- たった 1 つの出力に代わり、可能な出力は無数にあります。これによりメタデータに対する多様な要件を持つアプリケーションにさらに柔軟性を提供します。
- すべての他の DBMS が同じ手法を用いていますので移行は容易です。

しかし、`SHOW` は MySQL の従業員やユーザーに評判がよく、また無くなったら混乱することが考えられたため、従来の構文の利点だけでは `SHOW` を無くすのに十分な理由とはいえません。実際のところ、`INFORMATION_SCHEMA` を実装することによって、`SHOW` もまた同様に強化されます。これらのことは [SHOW ステートメントへの拡張](#) で説明しています。

`SHOW` ステートメントに必要な権限と `INFORMATION_SCHEMA` から情報を選択する権限の間には違いはありません。どちらの場合でも、オブジェクトに関する情報を表示するにはいくつかの権限を有する必要があります。

MySQL での `INFORMATION_SCHEMA` テーブル構成のインプリメンテーションは ANSI/ISO SQL:2003 標準 パート 11 の Schemata に準拠しています。SQL:2003 コア機能 F021 基本情報スキーマに最大限準拠することを意図しています。

SQL サーバ 2000 (標準に準拠) のユーザーは非常に近い類似性を認める事でしょう。しかし、MySQL ではインプリメンテーションに関連しない多くのカラムを割愛し、MySQL 特化のカラムを追加しています。それらの追加されたカラムの 1 つが `ENGINE` カラムで `INFORMATION_SCHEMA.TABLES` テーブルにあります。

他の DBMS は `syscat` あるいは `システム` などの様々な名前を使用していますが、標準の名前は `INFORMATION_SCHEMA` です。

以下の項で `INFORMATION_SCHEMA` の各テーブルおよびカラムについて説明します。各カラムに対し、3 項目の情報がります。

- 「`INFORMATION_SCHEMA` 名」は `INFORMATION_SCHEMA` テーブルのカラム名を意味します。これは「備考」欄で「MySQL 拡張」に触れない限り標準の SQL 名に一致します。
- 「`SHOW` 名」は `SHOW` 名がある場合に近接する `SHOW` ステートメントの相当するフィールド名を意味します。

- 「備考」は必要に応じて追加の情報を提供します。この領域が **NULL** の場合、カラムの値は常に **NULL** をであることを意味します。この領域に「MySQL 拡張」の事が書かれている場合、そのカラムは標準の SQL に対する MySQL 拡張です。

標準あるいは DB2、SQL サーバ、または Oracle に保持されている名前の使用しないように、「MySQL 拡張」の印の付いたカラムの名前を変更しています。(例えば、**TABLES** のテーブルでは **COLLATION** を **TABLE_COLLATION** に変更しています。)本件の最後にある予約した単語のリストを参照してください。 <http://www.dbazine.com/gulutzan5.shtml>.

文字列の定義 (例えば、**TABLES.TABLE_NAME**) は一般的には **VARCHAR(N) CHARACTER SET utf8** で、そこでは **N** は少なくとも 64 です。MySQL はこの文字セット (**utf8_general_ci**) をそのようなカラムのすべての検索、分類、比較、および他の文字列の操作に使用しています。デフォルトの照合が二重を満たさない場合、**COLLATE** 節 (**SQL ステートメント COLLATE 節を使用する**) で適切な照合を使用することができます。

各セクションはそのようなステートメントがある場合どの **SHOW** ステートメントが **INFORMATION_SCHEMA** から情報を取り出す **SELECT** に一致するかを示します。

注:現在、いくつかの不明なカラムおよび適切でないカラムがあります。現在この作業に携わっており変更があり次第変更を加えて文書を更新しています。

INFORMATION_SCHEMA データベースに関するよく出される質問の答えに関しては、[MySQL 5.0 FAQ — INFORMATION_SCHEMA](#) を参照してください。

INFORMATION_SCHEMA SCHEMATA テーブル

スキーマはデータベースで、**SCHEMATA** テーブルはデータベースに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
CATALOG_NAME		NULL
SCHEMA_NAME		データベース
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

以下のステートメントは等価です。

```
SELECT SCHEMA_NAME AS `Database`
FROM INFORMATION_SCHEMA.SCHEMATA
[WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
[LIKE 'wild']
```

INFORMATION_SCHEMA TABLES テーブル

TABLES テーブルはデータベースのテーブルに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
TABLE_CATALOG		NULL
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	MySQL 拡張
VERSION	Version	MySQL 拡張
ROW_FORMAT	Row_format	MySQL 拡張
TABLE_ROWS	Rows	MySQL 拡張

AVG_ROW_LENGTH	Avg_row_length	MySQL 拡張
DATA_LENGTH	Data_length	MySQL 拡張
MAX_DATA_LENGTH	Max_data_length	MySQL 拡張
INDEX_LENGTH	Index_length	MySQL 拡張
DATA_FREE	Data_free	MySQL 拡張
AUTO_INCREMENT	Auto_increment	MySQL 拡張
CREATE_TIME	Create_time	MySQL 拡張
UPDATE_TIME	Update_time	MySQL 拡張
CHECK_TIME	Check_time	MySQL 拡張
TABLE_COLLATION	Collation	MySQL 拡張
CHECKSUM	Checksum	MySQL 拡張
CREATE_OPTIONS	Create_options	MySQL 拡張
TABLE_COMMENT	Comment	MySQL 拡張

注：

- **TABLE_SCHEMA** および **TABLE_NAME** は **SHOW** ディスプレーの単一の領域で、例えば **Table_in_db1** のようになります。
- **TABLE_TYPE** は **BASE TABLE** あるいは **VIEW** になります。テーブルがテンポラリーの場合、**TABLE_TYPE = TEMPORARY** になります。(テンポラリーな表示はないため、不明瞭になることはありません。)
- 分割したテーブルの場合、MySQL 5.1.9 で始めると、**ENGINE** カラムはすべての分割で使用されるストレージ エンジン名を表示します。(以前は、このカラムはそのようなテーブルに **PARTITION** を表示していました。)
- **TABLE_ROWS** カラムはテーブルが **INFORMATION_SCHEMA** のデータベースにある場合は **NULL** です。InnoDB テーブルは、行カウントは SQL の最適化で使用される単なる大雑把な予測です。
- というのは、**NDBCLUSTER** ストレージ エンジンを使用しているテーブルは、MySQL 5.1.12 で始まり、**DATA_LENGTH** カラムが可変幅カラムの真のストレージ量を表します。(Bug #18413 参照)

注:なぜなら MySQL クラスタは可変幅カラムにストレージを各 32Kバイトで 10 ページ割り当てるため、そのようなカラムのスペースの使用は 320 KB の増分になります。

- テーブルのデフォルトの文字セットには何もありません。**TABLE_COLLATION** は閉じています。なぜなら照合名は文字セット名で始まるからです。
- MySQL 5.1.9 を起動すると、**CREATE_OPTIONS** カラムはテーブルが分割されている場合 **partitioned** を表示します。

以下のステートメントは等価です。

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
[WHERE table_schema = 'db_name']
[WHERE]AND table_name LIKE 'wild']

SHOW TABLES
[FROM db_name]
[LIKE 'wild']
```

INFORMATION_SCHEMA COLUMNS テーブル

COLUMNS テーブルはテーブルのカラムに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
TABLE_CATALOG		NULL
TABLE_SCHEMA		

TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		注を参照
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL 拡張
COLUMN_KEY	Key	MySQL 拡張
EXTRA	Extra	MySQL 拡張
COLUMN_COMMENT	Comment	MySQL 拡張

注：

- `SHOW` では、`Type` 表示は異なるいくつかの `COLUMNS` カラムの値を含んでいます。
- `ORDINAL_POSITION` は `ORDER BY ORDINAL_POSITION` をいう場合がありますので必要です。`SHOW` とは異なり、`SELECT` には自動オーダーリングはありません。
- `CHARACTER_OCTET_LENGTH` は、マルチバイトの文字セットを除いては `CHARACTER_MAXIMUM_LENGTH` と同じでなければなりません。
- `CHARACTER_SET_NAME` は `Collation` から得られます。例えば、`SHOW FULL COLUMNS FROM t` という場合、`Collation` カラムに `latin1_swedish_ci` の値が表示されます。文字セットは最初のアンダースコアの前の部分、つまり `latin1` です。

以下のステートメントはほぼ等価です。

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']
[AND column_name LIKE 'wild']

SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```

INFORMATION_SCHEMA STATISTICS テーブル

`STATISTICS` テーブルはテーブル インデックスの情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
TABLE_CATALOG		NULL
TABLE_SCHEMA		= データベース
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database

INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL 拡張
PACKED	Packed	MySQL 拡張
NULLABLE	Null	MySQL 拡張
INDEX_TYPE	Index_type	MySQL 拡張
COMMENT	Comment	MySQL 拡張

注：

- インデックスには標準のテーブルはありません。前のリストは SQL サーバ 2000 が [sp_statistics](#) に返すものと、[QUALIFIER](#) を [CATALOG](#) に、[OWNER](#) を [SCHEMA](#) に置き換えた事を除いては類似しています。

明らかに、前のテーブルおよび [SHOW INDEX](#) の出力は同じ親に由来しています。ですから相関関係は既に閉じています。

以下のステートメントは等価です。

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']

SHOW INDEX
FROM tbl_name
[FROM db_name]
```

INFORMATION_SCHEMA USER_PRIVILEGES テーブル

[USER_PRIVILEGES](#) テーブルはグローバル権限に関する情報を提供します。この情報は [mysql.user](#) グラントテーブルにあります。

INFORMATION_SCHEMA 名	SHOW 名	備考
GRANTEE		'user_name'@'host_name' 値、MySQL 拡張
TABLE_CATALOG		NULL, MySQL 拡張
PRIVILEGE_TYPE		MySQL 拡張
IS_GRANTABLE		MySQL 拡張

注：

- これは非標準のテーブルです。その値は [mysql.user](#) のテーブルにあります。

INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル

[SCHEMA_PRIVILEGES](#) テーブルはスキーマ（データベース）権限に関する情報を提供します。この情報は [mysql.user](#) グラントテーブルにあります。

INFORMATION_SCHEMA 名	SHOW 名	備考
GRANTEE		'user_name'@'host_name' 値、MySQL 拡張
TABLE_CATALOG		NULL, MySQL 拡張
TABLE_SCHEMA		MySQL 拡張

PRIVILEGE_TYPE		MySQL 拡張
IS_GRANTABLE		MySQL 拡張

注：

- これは非標準のテーブルです。その値は `mysql.db` テーブルにあります。

INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル

`TABLE_PRIVILEGES` テーブルはテーブル権限に関する情報を提供します。この情報は `mysql.tables_priv` グラントテーブルにあります。

INFORMATION_SCHEMA 名	SHOW 名	備考
GRANTEE		'user_name'@'host_name' 値
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

注：

- `PRIVILEGE_TYPE` はこれらの値の 1 つ (1 つのみ) を含むことができます。SELECT、INSERT、UPDATE、REFERENCES、ALTER、INDEX、DROP、CREATE VIEW。

以下のステートメントは等価ではありません。

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
```

```
SHOW GRANTS ...
```

INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル

`COLUMN_PRIVILEGES` テーブルはカラムの権限に関する情報を提供します。この情報は `mysql.columns_priv` グラントテーブルにあります。

INFORMATION_SCHEMA 名	SHOW 名	備考
GRANTEE		'user_name'@'host_name' 値
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

注：

- `SHOW FULL COLUMNS` の出力では、権限はすべて 1 つの領域にあり小文字です。例えば、`select,insert,update,references`。`COLUMN_PRIVILEGES` では、行ごとに 1 つの権限があり、大文字です。
- `PRIVILEGE_TYPE` はこれらの値の 1 つ (1 つのみ) を含むことができます。SELECT、INSERT、UPDATE、REFERENCES。
- ユーザーに `GRANT OPTION` の権限がある場合、`IS_GRANTABLE` は YES になります。権限が無い場合、`IS_GRANTABLE` は NO になります。その出力は `GRANT OPTION` を個別の権限としてリストしません。

以下のステートメントは等価ではありません。

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
```

INFORMATION_SCHEMA CHARACTER_SETS テーブル

CHARACTER_SETS テーブルは利用できる文字セットに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPTION	Description	MySQL 拡張
MAXLEN	Maxlen	MySQL 拡張

以下のステートメントは等価です。

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
[WHERE name LIKE 'wild']
SHOW CHARACTER SET
[LIKE 'wild']
```

INFORMATION_SCHEMA COLLATIONS テーブル

COLLATIONS テーブルは各文字セットの照合に関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	MySQL 拡張
ID	Id	MySQL 拡張
IS_DEFAULT	Default	MySQL 拡張
IS_COMPILED	Compiled	MySQL 拡張
SORTLEN	Sortlen	MySQL 拡張

以下のステートメントは等価です。

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
[WHERE collation_name LIKE 'wild']
SHOW COLLATION
[LIKE 'wild']
```

INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル

COLLATION_CHARACTER_SET_APPLICABILITY テーブルはどの文字セットがどの照合に適用できるかを示します。カラムは SHOW COLLATION から取得する最初の 2 つの表示領域に等価です。

INFORMATION_SCHEMA 名	SHOW 名	備考
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル

TABLE_CONSTRAINTS テーブルはどのテーブルに制約があるかを説明します。

INFORMATION_SCHEMA 名	SHOW 名	備考

CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

注：

- **CONSTRAINT_TYPE** 値は **UNIQUE**、**PRIMARY KEY**、あるいは **FOREIGN KEY** になります。
- **UNIQUE** および **PRIMARY KEY** 情報は **Non_unique** 領域が 0 のとき **SHOW INDEX** の出力の **Key_name** 領域から得られる情報とほぼ同じになります。
- **CONSTRAINT_TYPE** カラムはこれらの値のどれか 1 つを含みます。**UNIQUE**、**PRIMARY KEY**、**FOREIGN KEY**、**CHECK**。これは **CHAR** (非 **ENUM**) カラムです。**CHECK** 値は **CHECK** をサポートするまでは利用できません。

INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル

KEY_COLUMN_USAGE テーブルはどのキーカラムに制約があるかを説明します。

INFORMATION_SCHEMA 名	SHOW 名	備考
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

注：

- 制約が外部キーの場合、これは外部キーのカラムで、外部キーが参照するカラムではありません。
- **ORDINAL_POSITION** の値は制約内のカラムの位置で、テーブル内のカラムの位置ではありません。カラムの位置には 1 から始まる番号が付いています。
- **POSITION_IN_UNIQUE_CONSTRAINT** の値は一意およびプライマリ キー制約に対し **NULL** です。外部キーの制約では、参照されるテーブルのキーの順序です。

例えば、以下の定義を有する 2 つのテーブル名 **t1** と **t3** あるとします。

```
CREATE TABLE t1
(
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
) ENGINE=InnoDB;

CREATE TABLE t3
```

```
(
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

それらの 2 つのテーブルに対し、`KEY_COLUMN_USAGE` テーブルには 2 つの行があります。

- 1 つの行の `CONSTRAINT_NAME = 'PRIMARY'`, `TABLE_NAME = 't1'`, `COLUMN_NAME = 's3'`, `ORDINAL_POSITION = 1`, `POSITION_IN_UNIQUE_CONSTRAINT = NULL`.
- 1 つの行の `CONSTRAINT_NAME = 'CO'`, `TABLE_NAME = 't3'`, `COLUMN_NAME = 's2'`, `ORDINAL_POSITION = 1`, `POSITION_IN_UNIQUE_CONSTRAINT = 1`.

INFORMATION_SCHEMA ROUTINES テーブル

`ROUTINES` テーブルは保存されたルーチン (プロシージャおよび関数の両方) に関する情報を提供します。`ROUTINES` テーブルはこの段階ではユーザー定義の関数 (UDF) を含みません。

「`mysql.proc` 名」の名前のカラムは `INFORMATION_SCHEMA.ROUTINES` テーブル カラムに相当する `mysql.proc` テーブル カラムを意味します。

INFORMATION_SCHEMA 名	mysql.proc 名	備考
<code>SPECIFIC_NAME</code>	<code>specific_name</code>	
<code>ROUTINE_CATALOG</code>		NULL
<code>ROUTINE_SCHEMA</code>	<code>db</code>	
<code>ROUTINE_NAME</code>	<code>name</code>	
<code>ROUTINE_TYPE</code>	<code>type</code>	{PROCEDURE FUNCTION}
<code>DTD_IDENTIFIER</code>		(データタイプ デスクリプター)
<code>ROUTINE_BODY</code>		SQL
<code>ROUTINE_DEFINITION</code>	<code>body</code>	
<code>EXTERNAL_NAME</code>		NULL
<code>EXTERNAL_LANGUAGE</code>	<code>language</code>	NULL
<code>PARAMETER_STYLE</code>		SQL
<code>IS_DETERMINISTIC</code>	<code>is_deterministic</code>	
<code>SQL_DATA_ACCESS</code>	<code>sql_data_access</code>	
<code>SQL_PATH</code>		NULL
<code>SECURITY_TYPE</code>	<code>security_type</code>	
<code>CREATED</code>	<code>created</code>	
<code>LAST_ALTERED</code>	<code>modified</code>	
<code>SQL_MODE</code>	<code>sql_mode</code>	MySQL 拡張
<code>ROUTINE_COMMENT</code>	<code>comment</code>	MySQL 拡張
<code>DEFINER</code>	<code>definer</code>	MySQL 拡張

注：

- MySQL は `EXTERNAL_LANGUAGE` をこのように計算します。
 - `mysql.proc.language='SQL'` の場合、`EXTERNAL_LANGUAGE` は NULL
 - そうでない場合、`EXTERNAL_LANGUAGE` は `mysql.proc.language` にあります。しかし、まだ外部言語がないので、それは常に NULL です。

INFORMATION_SCHEMA VIEWS テーブル

VIEWS テーブルはデータベースの表示に関する情報を提供します。このテーブルにアクセスするには **SHOW VIEW** の権限が必要です。

INFORMATION_SCHEMA 名	SHOW 名	備考
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		

注：

- VIEW_DEFINITION カラムには **SHOW CREATE VIEW** が生成する **Create Table** 領域で表示されるそのほとんどがあります。**SELECT** の前の単語および **WITH CHECK OPTION** の前の単語をスキップします。元のステートメントは以下のものであったと想定します。

```
CREATE VIEW v AS
SELECT s2,s1 FROM t
WHERE s1 > 5
ORDER BY s1
WITH CHECK OPTION;
```

その際表示の定義はこうになります。

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- CHECK_OPTION カラムは常には **NONE** の値があります。
- IS_UPDATABLE カラムは表示が更新可能な場合は **YES**、更新できない場合は **NO** になります。
- DEFINER カラムは誰がその表示を定義したかを示します。SECURITY_TYPE には **DEFINER** あるいは **INVOKER** の値があります。

INFORMATION_SCHEMA TRIGGERS テーブル

TRIGGERS テーブルはトリガに関する情報を提供します。このテーブルにアクセスするには **SUPER** 権限が必要です。

INFORMATION_SCHEMA 名	SHOW 名	備考
TRIGGER_CATALOG		NULL
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		NULL
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		0
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	

ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED		NULL (0)
SQL_MODE		MySQL 拡張
DEFINER		MySQL 拡張

注：

- **TRIGGER_SCHEMA** および **TRIGGER_NAME** カラムはトリガが発生するデータベースおよびトリガ名をそれぞれ含みます。
- **EVENT_MANIPULATION** カラムは 'INSERT'、'DELETE'、あるいは 'UPDATE' のいずれかの値を含みます。
- **18章トリガ** で説明したように、すべてのトリガは正確に 1 つのテーブルに関連付けられます。**EVENT_OBJECT_SCHEMA** および **EVENT_OBJECT_TABLE** カラムはこのテーブルが発生するデータベース、およびテーブル名を含みます。
- **ACTION_ORDER** ステートメントは同じテーブルのすべての類似トリガ リストのトリガの実行順序を含みます。現在この値は常に 0 です。というのは同じテーブルで同じ **EVENT_MANIPULATION** および **ACTION_TIMING** で 1 つ以上のトリガを持ってないからです。
- **ACTION_STATEMENT** カラムはトリガが呼び出されたときに実行されるステートメントを含みます。これは **SHOW TRIGGERS** から出力される **Statement** カラムに表示されるテキストと同じです。このテキストは UTF-8 エンコーディングを使用していることを留意してください。
- **ACTION_ORIENTATION** カラムは常に 'ROW' の値を含みます。
- **ACTION_TIMING** カラムは 'BEFORE' あるいは 'AFTER' のいずれかの値を含みます。
- **ACTION_REFERENCE_OLD_ROW** および **ACTION_REFERENCE_NEW_ROW** はそれぞれ新旧のカラム識別子を含みます。このことは **ACTION_REFERENCE_OLD_ROW** は常に 'OLD' の値を含み **ACTION_REFERENCE_NEW_ROW** は常に 'NEW' の値を含みます。
- **SQL_MODE** カラムはトリガが作成されたとき（そしてこのように現在のサーバーの SQL モードに関係なくトリガが呼び出されたときにはいつでも有効です）有効だったサーバーの SQL モードを表示します。このカラムの可能な範囲の値は **sql_mode** システム変数と同じです。**SQL モード** を参照してください。
- **DEFINER** カラムが MySQL 5.1.2 に追加されました。**DEFINER** は誰がトリガを定義したかを示します。
- 以下のカラムは常に NULL を含みます。**TRIGGER_CATALOG**、**EVENT_OBJECT_CATALOG**、**ACTION_CONDITION**、**ACTION_REFERENCE_OLD_TABLE**、**ACTION_REFERENCE_NEW_TABLE** および **CREATED**。

トリガの使用 で定義された **ins_sum** の使用例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\G
***** 1. row *****
TRIGGER_CATALOG: NULL
TRIGGER_SCHEMA: test
TRIGGER_NAME: ins_sum
EVENT_MANIPULATION: INSERT
EVENT_OBJECT_CATALOG: NULL
EVENT_OBJECT_SCHEMA: test
EVENT_OBJECT_TABLE: account
ACTION_ORDER: 0
ACTION_CONDITION: NULL
ACTION_STATEMENT: SET @sum = @sum + NEW.amount
ACTION_ORIENTATION: ROW
```

```

ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
ACTION_REFERENCE_OLD_ROW: OLD
ACTION_REFERENCE_NEW_ROW: NEW
    CREATED: NULL
    SQL_MODE:
    DEFINER: me@localhost

```

SHOW TRIGGERS [構文](#) も参照してください。

INFORMATION_SCHEMA PLUGINS テーブル

PLUGINS テーブルはサーバーのプラグインに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
PLUGIN_NAME	Name	MySQL 拡張
PLUGIN_VERSION		MySQL 拡張
PLUGIN_STATUS	Status	MySQL 拡張
PLUGIN_TYPE	Type	MySQL 拡張
PLUGIN_TYPE_VERSION		MySQL 拡張
PLUGIN_LIBRARY	Library	MySQL 拡張
PLUGIN_LIBRARY_VERSION		MySQL 拡張
PLUGIN_AUTHOR		MySQL 拡張
PLUGIN_DESCRIPTION		MySQL 拡張

注：

- PLUGINS テーブルは非標準のテーブルです。MySQL 5.1.5 に追加されました。

SHOW PLUGINS [構文](#) も参照してください。

INFORMATION_SCHEMA ENGINES テーブル

PLUGINS テーブルはストレージ エンジンに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
ENGINE	Engine	MySQL 拡張
SUPPORT	Support	MySQL 拡張
COMMENT	Comment	MySQL 拡張
TRANSACTIONS	Transactions	MySQL 拡張
XA	XA	MySQL 拡張
SAVEPOINTS	Savepoints	MySQL 拡張

注：

- ENGINES テーブルは非標準のテーブルです。MySQL 5.1.5 に追加されました。

SHOW ENGINES [構文](#) も参照してください。

INFORMATION_SCHEMA PARTITIONS テーブル

PARTITIONS テーブルはテーブルの分割に関する情報を提供します。テーブルの分割に関する詳細は [15章パーティショニング](#) を参照してください。

INFORMATION_SCHEMA 名	SHOW 名	備考

TABLE_CATALOG		MySQL 拡張
TABLE_SCHEMA		MySQL 拡張
TABLE_NAME		MySQL 拡張
PARTITION_NAME		MySQL 拡張
SUBPARTITION_NAME		MySQL 拡張
PARTITION_ORDINAL_POSITION		MySQL 拡張
SUBPARTITION_ORDINAL_POSITION		MySQL 拡張
PARTITION_METHOD		MySQL 拡張
SUBPARTITION_METHOD		MySQL 拡張
PARTITION_EXPRESSION		MySQL 拡張
SUBPARTITION_EXPRESSION		MySQL 拡張
PARTITION_DESCRIPTION		MySQL 拡張
TABLE_ROWS		MySQL 拡張
AVG_ROW_LENGTH		MySQL 拡張
DATA_LENGTH		MySQL 拡張
MAX_DATA_LENGTH		MySQL 拡張
INDEX_LENGTH		MySQL 拡張
DATA_FREE		MySQL 拡張
CREATE_TIME		MySQL 拡張
UPDATE_TIME		MySQL 拡張
CHECK_TIME		MySQL 拡張
CHECKSUM		MySQL 拡張
PARTITION_COMMENT		MySQL 拡張
NODEGROUP		MySQL 拡張
TABLESPACE_NAME		MySQL 拡張

注：

- **PARTITIONS** テーブルは非標準のテーブルです。それは MySQL 5.1.6 に追加されています。
このテーブルの各レコードは個々の分割あるいは分割されたテーブルのサブ分割に一致します。
- **TABLE_CATALOG**:このカラムは常に **NULL** です。
- **TABLE_SCHEMA**:このカラムはテーブルが属すデータベース名を含みます。
- **TABLE_NAME**:このカラムは分割を含むテーブル名を含みます。
- **PARTITION_NAME**:分割の名前です。
- **SUBPARTITION_NAME**:**PARTITIONS** テーブルのレコードがサブ分割を表す場合、このカラムはサブ分割名を含みます。そうでない場合 **NULL** になります。
- **PARTITION_ORDINAL_POSITION**:すべての分割はそれらが定義されたときの順序でインデックスが付けられます。最初の分割に割り当てられた番号は **1** になります。インデックスは分割が追加、削除、再手配されることによって変わります。このカラムに表示された番号はインデックスの変更を考慮した現在の順序を表します。
- **SUBPARTITION_ORDINAL_POSITION**:所定の分割のサブ分割はテーブル内で分割がインデックスされるの同様にインデックスまたは再インデックスされます。
- **PARTITION_METHOD**:**RANGE**、**LIST**、**HASH**、**LINEAR HASH**、**KEY**、あるいは **LINEAR KEY** のいずれかの値です。つまり、**パーティショニングのタイプ** で説明した利用できる分割タイプの 1 つです。

- **SUBPARTITION_METHOD**:HASH、LINEAR HASH、KEY あるいは LINEAR KEY のいずれかの値の 1 つです。つまり、[サブパーティショニング](#) で説明したサブ利用できる分割タイプの 1 つです。
- **PARTITION_EXPRESSION**:これはテーブルの現在の分割スキーマを作成した CREATE TABLE あるいは ALTER TABLE に使用された分割関数の式です。

例として、このステートメントを使用して `test` データベースで作成された分割テーブルを考えてみます。

```
CREATE TABLE tp (
  c1 INT,
  c2 INT,
  c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

このテーブルの分割のPARTITIONS テーブル レコードのPARTITION_EXPRESSION カラムは以下のように `c1 + c2` を表示します。

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+-----+
| PARTITION_EXPRESSION |
+-----+
| c1 + c2              |
+-----+
1 row in set (0.09 sec)
```

- **SUBPARTITION_EXPRESSION**:これはテーブルの分割を定義するために使用された分割式に PARTITION_EXPRESSION が行っているのと同様にテーブルのサブ分割を定義するサブ分割式に対して同様に機能します。

テーブルにサブ分割がない場合、このカラムは `NULL` です。

- **PARTITION_DESCRIPTION**:このカラムは RANGE および LIST 分割に使用されます。RANGE 分割では、それは分割の VALUES LESS THAN 節で設定された値セットを含みます。それは整数あるいは MAXVALUE のいずれかになります。LIST 分割では、このカラムは分割 VALUES IN 節で定義された値を含みます。それは整数値のコンマ区切りのリストです。

PARTITION_METHOD が RANGE あるいは LIST 以外の分割の場合、このカラムは常に `NULL` になります。

- **TABLE_ROWS**:分割のテーブル行の数です。
- **AVG_ROW_LENGTH**:この分割あるいはサブ分割で保存された行の平均の長さでバイトで表します。
これは TABLE_ROWS で分割された DATA_LENGTH と同じです。
- **DATA_LENGTH**:この分割あるいはサブ分割に保存されたすべての行の全長でバイトで表します。— つまり分割およびサブ分割に保存されたバイトの総数です。
- **MAX_DATA_LENGTH**:この分割およびサブ分割に保存される最大のバイト数です。
- **INDEX_LENGTH**:この分割およびサブ分割のインデックス ファイルの長さでバイトで表します。
- **DATA_FREE**:分割あるいはサブ分割に割り当てられたバイト数で使用されていないものです。
- **CREATE_TIME**:分割およびサブ分割の作成に要する時間です。
- **UPDATE_TIME**:分割あるいはサブ分割が最後に変更された時間です。
- **CHECK_TIME**:分割あるいはサブ分割が属すテーブルが最後にチェックされた時間です。

注:ストレージ エンジンの中には今回更新されないものがあります。というのはこれらのストレージ エンジンを使用しているテーブルに対し、この値が常に `NULL` だからです。

- **CHECKSUM**:チェックサムの値 (もしあれば)、そうでない場合、このカラムは `NULL` です。

- **PARTITION_COMMENT**:このカラムは分割に対するコメントのテキストを含んでいます。

このカラムのデフォルト値は空の文字列です。

- **NODEGROUP**:これは分割が属すノードグループです。これは MySQL クラスタテーブルにのみ関連します。そうでない場合このカラムの値は常に **0** です。
- **TABLESPACE_NAME**:このカラムは分割が属すテーブルスペース名を含んでいます。MySQL 5.1 では、このカラムの値は常に **DEFAULT** です。
- 重要MySQL 5.1.6 以前の MySQL バージョンで作成された分割テーブルがある場合、MySQL 5.1.6 あるいはそれ以降にアップグレードする場合、**SHOW**、あるいは **DESCRIBE the PARTITIONS** から **SELECT** することはできません。MySQL 5.1.5 あるいはそれ以前のバージョンから MySQL 5.1.6 およびそれ以降にアップグレードする前に [Changes in release 5.1.6 \(01 February 2006\)](#) を参照してください。
- 非分割のテーブルには **INFORMATION_SCHEMA.PARTITIONS** にレコードが 1 つあります。しかし、**PARTITION_NAME**、**SUBPARTITION_NAME**、**PARTITION_ORDINAL_POSITION**、**SUBPARTITION_ORDINAL_POSITION** および **PARTITION_DESCRIPTION** カラムの値はすべて **NULL** です。(この場合の **PARTITION_COMMENT** のカラムは空白です。)

MySQL 5.1 には、**PARTITIONS** テーブルに **NDBCluster** ストレージ エンジンを使用したテーブルに対しレコードが 1 つだけあります。同じカラムはまた非分割テーブルに対しては **NULL** (あるいは空) です。

INFORMATION_SCHEMA EVENTS テーブル

EVENTS テーブルは [19章Event Scheduler](#) で説明した計画したイベントに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
EVENT_CATALOG		NULL , MySQL 拡張
EVENT_SCHEMA	Db	MySQL 拡張
EVENT_NAME	Name	MySQL 拡張
DEFINER	Definer	MySQL 拡張
EVENT_BODY		MySQL 拡張
EVENT_DEFINITION		MySQL 拡張
EVENT_TYPE	Type	MySQL 拡張
EXECUTE_AT	Execute at	MySQL 拡張
INTERVAL_VALUE	Interval value	MySQL 拡張
INTERVAL_FIELD	Interval field	MySQL 拡張
SQL_MODE		MySQL 拡張
STARTS	Starts	MySQL 拡張
ENDS	Ends	MySQL 拡張
STATUS	Status	MySQL 拡張
ON_COMPLETION		MySQL 拡張
CREATED		MySQL 拡張
LAST_ALTERED		MySQL 拡張
LAST_EXECUTED		MySQL 拡張
EVENT_COMMENT		MySQL 拡張

注 :

- **EVENTS** テーブルは非標準のテーブルです。それは MySQL 5.1.6 に追加されています。
- **EVENT_CATALOG**:このカラムの値は常に **NULL** です。

- **EVENT_SCHEMA**: イベントが属するスキーマ (データベース) の名前です。
- **EVENT_NAME**: イベントの名前です。
- **DEFINER**: イベントを作成したユーザーです。常に 'user_name'@'host_name' フォーマットに表示されます。
- **EVENT_BODY**: イベントの DO 節のステートメントに使用された言語です。MySQL 5.1 では、これは常に SQL です。

このカラムは MySQL 5.1.12 に追加されています。MySQL の以前のバージョンの同じ名前のカラム (今の名前は **EVENT_DEFINITION**) と混同することはありません。
- **EVENT_DEFINITION**: イベント DO 節を構成する SQL ステートメントのテキストです。換言すれば、このイベントで実行されたステートメントです。

注: MySQL 5.1.12 以前のバージョンでは、このカラムは **EVENT_BODY** という名前でした。
- **EVENT_TYPE**: **ONE TIME** あるいは **RECURRING** のいずれかの値です。
- **EXECUTE_AT**: 1 回のイベントでは、これはイベントの作成に使用された **CREATE EVENT** ステートメントの **AT** 節で指定された **DATETIME** 値、あるいはイベントを変更した最後の **ALTER EVENT** ステートメントです。このカラムに表示された値はイベント **AT** 節に含まれる **INTERNAL** 値の加算あるいは減算を意味しています。例えば、イベントが **ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR** を使用して作成された場合、およびイベントが 2006-02-09 の 14:05:30 に作成された場合、カラムに表示される値は '2006-02-10 20:05:30' になります。

そのイベントのタイミングが **AT** 節 (すなわち、イベントは再帰的な場合) ではなく **EVERY** 節で決定される場合、このカラムの値は **NULL** になります。
- **INTERVAL_VALUE**: 再帰的なイベントの場合、このカラムはイベント **EVERY** 節の数値的な部分を含みます。

1 回のイベントの場合 (つまり、イベントのタイミングが **AT** 節で決まるイベントの場合)、このカラムの値は **NULL** になります。
- **INTERVAL_FIELD**: 再帰的なイベントの場合、このカラムはそのイベントを管理する **EVERY** 節の単位部分を含みます。そのイベントは 'INTERVAL_' のプリフィックスが付きます。このように、このカラムは 'INTERVAL_YEAR'、'INTERVAL_QUARTER'、'INTERVAL_DAY' などの値を含みます。

1 回のイベントの場合 (つまり、そのタイミングが **AT** 節で決まるイベントの場合)、このカラムの値は **NULL** になります。
- **SQL_MODE**: イベントが作成されたあるいは変更された時に実行中の SQL モード。
- **STARTS**: 定義に **STARTS** 節を含む再帰的なイベントでは、このカラムは相当する **DATETIME** 値を含みます。 **EXECUTE_AT** カラムで、この値は使用されている式を解きます。

イベントのタイミングに影響を与えるような **STARTS** 節がない場合、このカラムは空です。 (MySQL 5.1.8 以前の場合は、そのような場合それは **NULL** が含まれていました。)
- **ENDS**: 定義に **ENDS** 節を含む再帰的なイベントでは、このカラムは相当する **DATETIME** 値を含みます。 **EXECUTE_AT** カラム (前の例を参照) では、この値は使用されている式を解きます。

イベントのタイミングに影響を与える **ENDS** 節がない場合、このカラムは **NULL** を含みます。
- **STATUS**: **ENABLED** あるいは **DISABLED** の 2 つの値のいずれかの値です。
- **ON_COMPLETION**: **PRESERVE** あるいは **NOT PRESERVE** の 2 つの値のいずれかの値です。
- **CREATED**: イベントが作成された日時です。これは **DATETIME** 値です。
- **LAST_ALTERED**: イベントが最後に変更された日時です。これは **DATETIME** 値です。イベントが作成されてから変更されなかった場合、このカラムは **CREATED** カラムと同じ値を保持します。
- **LAST_EXECUTED**: イベントが最後に実行された日時です。 **DATETIME** 値。イベントが実行されなかった場合、このカラムの値は **NULL** です。

- **EVENT_COMMENT**: イベントにコメントがある場合のコメントのテキストです。コメントが無い場合、このカラムの値は空の文字列になります。

例: ユーザー `jon@ghidora` が `e_daily` というイベントを作成したとします。それを数分後に以下の **ALTER EVENT** ステートメントを使用して変更します。

```
DELIMITER |

CREATE EVENT e_daily
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
DISABLE
COMMENT 'Saves total number of sessions and
        clears the table once per day.'
DO
BEGIN
  INSERT INTO site_activity.totals (when, total)
  SELECT CURRENT_TIMESTAMP, COUNT(*)
  FROM site_activity.sessions;
  DELETE FROM site_activity.sessions;
END |

DELIMITER ;

ALTER EVENT e_daily
ENABLED;
```

(コメントは複数の行にわたって展開できます。)

このユーザーは次に以下の **SELECT** ステートメントを実行し、下記の出力を得ます。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME = 'e_daily'
> AND EVENT_SCHEMA = 'myschema'\G

***** 1. row *****
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_daily
DEFINER: jon@ghidora
EVENT_BODY: BEGIN
  INSERT INTO site_activity.totals (when, total)
  SELECT CURRENT_TIMESTAMP, COUNT(*)
  FROM site_activity.sessions;
  DELETE FROM site_activity.sessions;
END
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 1
INTERVAL_FIELD: INTERVAL_DAY
SQL_MODE: NULL
STARTS: 2006-02-09 10:41:23
ENDS: NULL
STATUS: ENABLED
ON_COMPLETION: DROP
CREATED: 2006-02-09 14:35:35
LAST_ALTERED: 2006-02-09 14:41:23
LAST_EXECUTED: NULL
EVENT_COMMENT: Saves total number of sessions and
               clears the table once per day.
1 row in set (0.50 sec)
```

重要**STARTS**、**ENDS**、および **LAST_EXECUTED** カラムで表示された時間はサーバーのタイムゾーンの設定にかかわらず現在の世界時 (GMT あるいは UTC) で表示されます。`mysql.event` テーブルの **starts**、**ends**、および **last_executed** カラム並びに **SHOW [FULL] EVENTS** の出力の **Starts** および **ENDs** カラムも同様に世界時を使用しています。) **CREATED** および **LAST_ALTERED** カラムはサーバーのタイムゾーン (`mysql.event` テーブルの **created** および **last_altered** カラムも同様) を使用しています。。

例えば、前に例示した `e_daily` イベントはオーストラリアのブリスベンにあるコンピュータで、東部オーストラリア時間の2006年2月9^日の14:35:35に作成されたとすると、それはタイムゾーンでは **GMT+10.00** となります。イベントの定義は (このセクションの以前の例示の **ALTER EVENT** を使用して) 数分後に 14:41:23 に更新されました。

これらが `CREATED` および `LAST_ALTERED` に表示される値です。イベントは 6 時間後 — つまり、同日のローカルタイムの 20:41:23 に実行されるように計画されます。この時間から 10 時間を減算すると世界時の 10:41:23 が得られ、この値が `STARTS` に表示されます。

世界時の使用はアプリケーションでは当てに出来ません。というのは MySQL のリリースによってサーバーのローカルタイムが変更されることが予想されるためです。(Bug #16420 参照)

`SHOW EVENTS` も参照してください。

INFORMATION_SCHEMA FILES テーブル

`FILES` テーブルは MySQL NDB ディスクデータ テーブルが保存されるファイルに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
FILE_ID		MySQL 拡張
FILE_NAME		MySQL 拡張
FILE_TYPE		MySQL 拡張
TABLESPACE_NAME		MySQL 拡張
TABLE_CATALOG		MySQL 拡張
TABLE_SCHEMA		MySQL 拡張
TABLE_NAME		MySQL 拡張
LOGFILE_GROUP_NAME		MySQL 拡張
LOGFILE_GROUP_NUMBER		MySQL 拡張
ENGINE		MySQL 拡張
FULLTEXT_KEYS		MySQL 拡張
DELETED_ROWS		MySQL 拡張
UPDATE_COUNT		MySQL 拡張
FREE_EXTENTS		MySQL 拡張
TOTAL_EXTENTS		MySQL 拡張
EXTENT_SIZE		MySQL 拡張
INITIAL_SIZE		MySQL 拡張
MAXIMUM_SIZE		MySQL 拡張
AUTOEXTEND_SIZE		MySQL 拡張
CREATION_TIME		MySQL 拡張
LAST_UPDATE_TIME		MySQL 拡張
LAST_ACCESS_TIME		MySQL 拡張
RECOVER_TIME		MySQL 拡張
TRANSACTION_COUNTER		MySQL 拡張
VERSION		MySQL 拡張
ROW_FORMAT		MySQL 拡張
TABLE_ROWS		MySQL 拡張
AVG_ROW_LENGTH		MySQL 拡張
DATA_LENGTH		MySQL 拡張
MAX_DATA_LENGTH		MySQL 拡張
INDEX_LENGTH		MySQL 拡張
DATA_FREE		MySQL 拡張

CREATE_TIME		MySQL 拡張
UPDATE_TIME		MySQL 拡張
CHECK_TIME		MySQL 拡張
CHECKSUM		MySQL 拡張
STATUS		MySQL 拡張
EXTRA		MySQL 拡張

注：

- **FILE_ID** カラムの値は自動生成されます。
- **FILE_NAME** は **CREATE LOGFILE GROUP** あるいは **ALTER LOGFILE GRUOP** によって作成された **UNDO** ログファイルの名前です。または **CREATE TABLESPACE** あるいは **ALTER TABLESPACE** で作成されたデータファイルです。
- **FILE_TYPE** は **UNDOFILE** あるいは **DATAFILE** のいずれかの値です。
- **TABLESPACE_NAME** はファイルが関連付けられているテーブルスペースの名前です。
- MySQL 5.1 では、**TABLESPACE_CATALOG** カラムの値は常に **NULL** です。
- **TABLE_NAME** は関連するファイルがある場合にファイルに関連付けられたディスク データ テーブルの名前です。
- **LOGFILE_GROUP_NAME** カラムはログファイルあるいはデータファイルが属すログファイルのグループに名前を付けます。
- **UNDO** ログファイルでは、**LOGFILE_GROUP_NUMBER** はログファイルが属すログファイル グループの自動生成された ID を含みます。
- MySQL クラスタ ディスクデータのログファイルあるいはデータファイルに対し、**ENGINE** カラムの値は常に **NDB** あるいは **NDBCLUSTER** になります。
- MySQL クラスタ ディスクデータのログファイルに対し、**FULLTEXT_KEYS** カラムの値は常に空です。
- **FREE EXTENTS** カラムはファイルが使用していない拡張の番号を表示します。**TOTAL EXTENTS** カラムはファイルに割り当てられた拡張の総数を表示します。

これらの 2 つのカラムの違いは現在ファイルが使用している拡張の数です。

```
SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

使用中のディスク容量を **EXTENT_SIZE** カラムの値によってこの違いを乗算したファイルによって最大化することができます。それによりファイルにバイトで拡張のサイズを与えます。

```
SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

同様に、所定のファイルに残された利用できるスペースの容量を **FREE_EXTENTS** に **EXTENT_SIZE** 乗算することで予測できます。

```
SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

重要上記のクエリで生成されたバイトの値は概算のみで、その正確性は **EXTENT_SIZE** の値に逆比例します。つまり、**EXTENT_SIZE** が大きくなれば、概算の正確性は落ちます。

拡張が一度使用されるとそれが属すデータファイルをドロップせずにはそれを自由に出来ないことを覚えておく必要があります。このことは、ディスクデータのテーブルからの削除はディスクスペースを増やさないということです。

拡張サイズは `CREATE TABLESPACE` ステートメントで設定できます。詳細については、`CREATE TABLESPACE` 構文をご参照してください。

- `INITIAL_SIZE` カラムはファイルのサイズをバイトで表示します。これはファイルの作成に使用された `CREATE LOGFILE GROUP`、`ALTER LOGFILE GROUP`、`CREATE TABLESPACE`、あるいは `ALTER TABLESPACE` の `INITIAL_SIZE` 節に使用された値と同じです。

MySQL 5.1 クラスタ ディスク データ ファイルでは、`MAXIMUM_SIZE` カラムの値は常に `INITIAL_SIZE` と同じで、`AUTOEXTEND_SIZE` カラムは常に空です。

- `CREATION_TIME` カラムはファイルが作成された日時を表示します。`LAST_UPDATE_TIME` カラムはファイルが最後に変更された日時を表示します。`LAST_ACCESSED` カラムはファイルが最後にサーバーによってアクセスされた日時を表示します。

現在は、これらのカラムの値はオペレーティング システムから知らされたものであり、`NDB` ストレージ エンジンにより供給されたものではありません。オペレーティング システムが何の値も提供していない場合には、これらのカラムには `0000-00-00 00:00:00`が表示されます。

- MySQL クラスタ ディスク データ ファイルでは、`RECOVER_TIME` および `TRANSACTION_COUNTER` カラムは常に `0` です。
- MySQL 5.1 クラスタ ディスク データ ファイルでは、以下のカラムは常に `NULL` です。

- `VERSION`
- `ROW_FORMAT`
- `TABLE_ROWS`
- `AVG_ROW_LENGTH`
- `DATA_LENGTH`
- `MAX_DATA_LENGTH`
- `INDEX_LENGTH`
- `DATA_FREE`
- `CREATE_TIME`
- `UPDATE_TIME`
- `CHECK_TIME`
- `CHECKSUM`

- MySQL クラスタ ディスク データ ファイルでは、`STATUS` カラムの値は常に `NORMAL` です。
- MySQL クラスタ ディスク データ ファイルでは、`EXTRA` カラムは、各データノードがファイルのコピーを持っているためどのデータノードにファイルが属するかを表示します。例えば、このステートメントを MySQL クラスタの 4 つのデータノードで使用するものとします。

```
CREATE LOGFILE GROUP mygroup
  ADD UNDOFILE 'new_undo.dat'
  INITIAL_SIZE 2G
  ENGINE NDB;
```

`CREATE LOGFILE GROUP` ステートメントの実行を完了すると、`FILES` テーブルに対するこのクエリの以下の結果に類似した結果が表示されます。

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'new_undo.dat';
+-----+-----+-----+
```

```
| LOGFILE_GROUP_NAME | FILE_TYPE | EXTRA |
+-----+-----+-----+
| mygroup            | UNDO FILE | CLUSTER_NODE=3 |
| mygroup            | UNDO FILE | CLUSTER_NODE=4 |
| mygroup            | UNDO FILE | CLUSTER_NODE=5 |
| mygroup            | UNDO FILE | CLUSTER_NODE=6 |
+-----+-----+-----+
4 rows in set (0.01 sec)
```

- **FILES** テーブルは非標準のテーブルです。それは MySQL 5.1.6 に追加されています。
- MySQL 5.1.14 を開始すると、ログファイル グループの作成に続いて **FILES** テーブルに追加の行が表示されます。この行は **FILE_NAME** カラムの値に対して **NULL** です。というのはこの行に対し、**FILE_ID** カラムの値は常に **0** で、**FILE_TYPE** カラムは常に **UNDO FILE** で、**STATUS** カラムは常に **NORMAL** になります。MySQL 5.1 では、**ENGINE** カラムの値は常に **ndbcluster** です。

この行は **FREE_EXTENTS** カラムで **LOGFILE_GROUP_NAME** および **LOGFILE_GROUP_NUMBER** カラムにそれぞれその名前と数が表示された所定のログファイル グループに属すすべての undo ファイルに利用できるフリー拡張の総数を表示します。

MySQL クラスタに既存のログファイルが無いものとして、以下のステートメントを使用してログファイルを 1 つ作成するとします。

```
mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'undofile.dat'
-> INITIAL_SIZE = 16M
-> UNDO_BUFFER_SIZE = 1M
-> ENGINE = NDB;
Query OK, 0 rows affected (3.81 sec)
```

FILES テーブルにクエリすると **NULL** 行が表示されます。

```
mysql> SELECT DISTINCT
-> FILE_NAME AS File,
-> FREE_EXTENTS AS Free,
-> TOTAL_EXTENTS AS Total,
-> EXTENT_SIZE AS Size,
-> INITIAL_SIZE AS Initial
-> FROM INFORMATION_SCHEMA.FILES;
+-----+-----+-----+-----+-----+
| File      | Free | Total | Size | Initial |
+-----+-----+-----+-----+-----+
| undofile.dat | NULL | 4194304 | 4 | 16777216 |
| NULL      | 4184068 | NULL | 4 | NULL |
+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

undo ロギングに利用できるフリー拡張の総数は常にログファイルのすべての undo ファイルに対して undo ファイルを維持に必要なオーバーヘッドにより **TOTAL_EXTENTS** カラムの値の合計より幾分少なくなります。これはログファイル グループに 2 番目の undo ファイルを追加すると表示され、**FILES** テーブルに対して前のクエリを繰り返します。

```
mysql> ALTER LOGFILE GROUP lg1
-> ADD UNDOFILE 'undofile02.dat'
-> INITIAL_SIZE = 4M
-> ENGINE = NDB;
Query OK, 0 rows affected (1.02 sec)

mysql> SELECT DISTINCT
-> FILE_NAME AS File,
-> FREE_EXTENTS AS Free,
-> TOTAL_EXTENTS AS Total,
-> EXTENT_SIZE AS Size,
-> INITIAL_SIZE AS Initial
-> FROM INFORMATION_SCHEMA.FILES;
+-----+-----+-----+-----+-----+
| File      | Free | Total | Size | Initial |
+-----+-----+-----+-----+-----+
| undofile.dat | NULL | 4194304 | 4 | 16777216 |
| undofile02.dat | NULL | 1048576 | 4 | 4194304 |
| NULL      | 5223944 | NULL | 4 | NULL |
+-----+-----+-----+-----+-----+
```



```
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

このログファイル グループを使用したディスク データ テーブルによる undo ロギングに利用されるバイトでのフリースペースの最大化はフリー拡張に初期サイズを乗算することで最大化できます。

```
mysql> SELECT
-> FREE_EXTENTS AS 'Free Extents',
-> FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg1'
-> AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
| 5223944 | 20895776 |
+-----+-----+
1 row in set (0.02 sec)
```

ディスク データ テーブルを作成してその中にいくつか行を挿入すると、undo のロギング用に残されたスペースを大まかに知ることができます。

```
mysql> CREATE TABLESPACE ts1
-> ADD DATAFILE 'data1.dat'
-> USE LOGFILE GROUP lg1
-> INITIAL_SIZE 512M
-> ENGINE = NDB;
Query OK, 0 rows affected (8.71 sec)

mysql> CREATE TABLE dd (
-> c1 INT NOT NULL PRIMARY KEY,
-> c2 INT,
-> c3 DATE
-> )
-> TABLESPACE ts1 STORAGE DISK
-> ENGINE = NDB;
Query OK, 0 rows affected (2.11 sec)

mysql> INSERT INTO dd VALUES
-> (NULL, 1234567890, '2007-02-02'),
-> (NULL, 1126789005, '2007-02-03'),
-> (NULL, 1357924680, '2007-02-04'),
-> (NULL, 1642097531, '2007-02-05');
Query OK, 4 rows affected (0.01 sec)

mysql> SELECT
-> FREE_EXTENTS AS 'Free Extents',
-> FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg1'
-> AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
| 5207565 | 20830260 |
+-----+-----+
1 row in set (0.01 sec)
```

- **FILES** テーブルに関連付けられた **SHOW** コマンドはありません。
- **FILES** テーブルを使用してクラスタ ディスク データ テーブルに関する情報の取得に関する他の例については [MySQL Cluster ディスク データ ストレージ](#) を参照してください。

INFORMATION_SCHEMA PROCESSLIST テーブル

PROCESSLIST テーブルは動作しているスレッドに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
ID	Id	MySQL 拡張

USER	User	MySQL 拡張
HOST	Host	MySQL 拡張
DB	db	MySQL 拡張
COMMAND	Command	MySQL 拡張
TIME	Time	MySQL 拡張
STATE	State	MySQL 拡張
INFO	Info	MySQL 拡張

テーブル カラムの全体的な説明については [SHOW PROCESSLIST 構文](#) を参照してください。

注：

- [PROCESSLIST](#) テーブルは非標準のテーブルです。それは MySQL 5.1.7 に追加されています。
- 相当する [SHOW](#) ステートメントの出力と同様、[PROCESSLIST](#) テーブルは [PROCESS](#) の権限が無い場合は、お客様のスレッドに関する情報のみ表示されます。権限がある場合には他のスレッドに関する情報も表示できます。匿名のユーザーはどの行も見ることができません。
- SQL ステートメントが [INFORMATION_SCHEMA.PROCESSLIST](#) を参照している場合、ステートメントの実行が開始されると全体のテーブルを一度表示します。それでステートメント中の読み取りの一貫性が保たれます。複数のステートメントのトランザクションではリードの一貫性はありません。

以下のステートメントは等価です。

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW PROCESSLIST
```

INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル

[REFERENTIAL_CONSTRAINTS](#) テーブルは外部キーに関する情報を提供します。

INFORMATION_SCHEMA 名	SHOW 名	備考
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
UNIQUE_CONSTRAINT_CATALOG		NULL
UNIQUE_CONSTRAINT_SCHEMA		
UNIQUE_CONSTRAINT_NAME		
MATCH_OPTION		
UPDATE_RULE		
DELETE_RULE		
TABLE_NAME		
REFERENCED_TABLE_NAME		

注：

- [REFERENTIAL_CONSTRAINTS](#) テーブルは MySQL 5.1.10 に追加されています。[REFERENCED_TABLE_NAME](#) カラムは MySQL 5.1.16 に追加されています。
- [TABLE_NAME](#) は [INFORMATION_SCHEMA.TABLE_CONSTRAINTS](#) の [TABLE_NAME](#) と同じ値を持っています。
- [CONSTRAINT_SCHEMA](#) および [CONSTRAINT_NAME](#) は外部キーを認識します。

- [UNIQUE_CONSTRAINT_SCHEMA](#)、[UNIQUE_CONSTRAINT_NAME](#)、および [REFERENCED_TABLE_NAME](#) は参照キーを認識します。(注:MySQL 5.1.16 以前のバージョンは、[UNIQUE_CONSTRAINT_NAME](#) は制約ではなく間違っって参照テーブルを指定しています。
- この段階の [MATCH_OPTION](#) の唯一有効な値は [NONE](#) です。
- [UPDATE_RULE](#) あるいは [DELETE_RULE](#) の可能な値は [CASCADE](#)、[SET NULL](#)、[SET DEFAULT](#)、[RESTRICT](#)、[NO ACTION](#) です。

INFORMATION_SCHEMA GLOBAL_STATUS および SESSION_STATUS テーブル

[GLOBAL_STATUS](#) および [SESSION_STATUS](#) テーブルはサーバーのステータス変数に関する情報を提供します。それらの中身は [SHOW GLOBAL STATUS](#) および [SHOW SESSION STATUS](#) ステートメント ([SHOW STATUS](#) [構文参照](#)) が生成する情報に一致します。

INFORMATION_SCHEMA 名	SHOW 名	備考
VARIABLE_NAME	Variable_name	
VARIABLE_VALUE	Value	

注：

- [GLOBAL_STATUS](#) および [SESSION_STATUS](#) テーブルは MySQL 5.1.12 に追加されています。
- [VARIABLE_VALUE](#) カラムには [BIGINT](#) タイプがあります。非整数値を持ついくつかのステータス変数は [BIGINT](#) 値に組み込まれています。

INFORMATION_SCHEMA GLOBAL_VARIABLES および SESSION_VARIABLES テーブル

[GLOBAL_VARIABLES](#) および [SESSION_VARIABLES](#) テーブルはサーバーのステータス変数に関する情報を提供します。それらの中身は [SHOW GLOBAL VARIABLES](#) および [SHOW SESSION VARIABLES](#) ステートメント ([SHOW VARIABLES](#) [構文参照](#)) が生成した情報に一致します。

INFORMATION_SCHEMA 名	SHOW 名	備考
VARIABLE_NAME	Variable_name	
VARIABLE_VALUE	Value	

注：

- [GLOBAL_VARIABLES](#) および [SESSION_VARIABLES](#) テーブルは MySQL 5.1.12 に追加されています。

その他の INFORMATION_SCHEMA テーブル

今後さらに [INFORMATION_SCHEMA](#) テーブルを追加する予定です。特に、[PARAMETERS](#) テーブルの必要性を強く感じています。

SHOW ステートメントへの拡張

[SHOW](#) ステートメントのいくつかの拡張は [INFORMATION_SCHEMA](#) の実装を伴います。

- [SHOW](#) を [INFORMATION_SCHEMA](#) そのものの構成に関する情報を取得するために使用できます。
- いくつかの [SHOW](#) ステートメントにはどの行を表示するかを指定する際に柔軟性を提供する [WHERE](#) 節を使用できます。

INFORMATION_SCHEMA は情報のデータベースで、その名前は **SHOW DATABASES** の出力に含まれています。同様に、**SHOW TABLES** を **INFORMATION_SCHEMA** と一緒に使用してそのテーブルのリストを取得できます。

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
```

```
+-----+-----+
| Tables_in_information_schema |
+-----+-----+
| CHARACTER_SETS              |
| COLLATIONS                  |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                    |
| COLUMN_PRIVILEGES           |
| ENGINES                     |
| EVENTS                      |
| FILES                       |
| KEY_COLUMN_USAGE            |
| PARTITIONS                  |
| PLUGINS                     |
| PROCESSLIST                 |
| ROUTINES                    |
| SCHEMATA                    |
| SCHEMA_PRIVILEGES           |
| STATISTICS                  |
| TABLES                     |
| TABLE_CONSTRAINTS          |
| TABLE_PRIVILEGES           |
| TRIGGERS                    |
| USER_PRIVILEGES             |
| VIEWS                       |
+-----+-----+
22 rows in set (0.04 sec)
```

SHOW COLUMNS および **DESCRIBE** は個々の **INFORMATION_SCHEMA** テーブルのカラムに関する情報を表示できます。

いくつかの **SHOW** ステートメントは **WHERE** 節を使用できるように拡張されています。

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW KEYS
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW VARIABLES
```

WHERE 節がある場合、それは **SHOW** ステートメントで表示されるカラム名で評価されます。例えば、**SHOW CHARACTER SET** ステートメントはこれらの出力カラムを生成します。

```
mysql> SHOW CHARACTER SET;
```

```
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci   | 2      |
| dec8    | DEC West European      | dec8_swedish_ci   | 1      |
| cp850   | DOS West European      | cp850_general_ci  | 1      |
| hp8     | HP West European       | hp8_english_ci    | 1      |
| koi8r   | KOI8-R Relcom Russian  | koi8r_general_ci  | 1      |
| latin1   | cp1252 West European   | latin1_swedish_ci | 1      |
| latin2   | ISO 8859-2 Central European | latin2_general_ci | 1      |
| ...
```

WHERE 節を **SHOW CHARACTER SET** と一緒に使用する場合、それらのカラム名を参照します。例えば、以下のステートメントはデフォルトの照合が文字列 **'japanese'** を含む文字セットに関する情報を表示します。

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
```

```
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
```

```

+-----+-----+-----+
| ujis | EUC-JP Japanese | ujis_japanese_ci | 3 |
| sjis | Shift-JIS Japanese | sjis_japanese_ci | 2 |
| cp932 | SJIS for Windows Japanese | cp932_japanese_ci | 2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci | 3 |
+-----+-----+-----+

```

このステートメントは複数バイトの文字セットを表示します。

```

mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+
| big5 | Big5 Traditional Chinese | big5_chinese_ci | 2 |
| ujis | EUC-JP Japanese | ujis_japanese_ci | 3 |
| sjis | Shift-JIS Japanese | sjis_japanese_ci | 2 |
| euckr | EUC-KR Korean | euckr_korean_ci | 2 |
| gb2312 | GB2312 Simplified Chinese | gb2312_chinese_ci | 2 |
| gbk | GBK Simplified Chinese | gbk_chinese_ci | 2 |
| utf8 | UTF-8 Unicode | utf8_general_ci | 3 |
| ucs2 | UCS-2 Unicode | ucs2_general_ci | 2 |
| cp932 | SJIS for Windows Japanese | cp932_japanese_ci | 2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci | 3 |
+-----+-----+-----+

```


第22章 精密計算

目次

数値のタイプ	1127
DECIMAL データ タイプの変更	1128
式の取り扱い	1129
丸め挙動	1130
精密計算の例	1131

MySQL 5.1 は精密計算に対するサポートを提供します。非常に正確な結果をもたらす数値処理と無効な値に対する高度なコントロール。精密計算はこれら2つの特徴に基づいています:

- どの程度厳格に無効なデータを容認するか、拒否しようとしているかを管理する SQL モード。
- 固定小数点演算に対する MySQL ライブラリ。

これらの特徴は数値演算に対していくつかの帰結的意味を持っています:

- 精密計算:正確な値の数に対して、計算は浮動小数点エラーを導入しません。その代わりに、精密計算が使用されています。例えば、.0001のような数は近似値としてより、むしろ正確な値として取り扱われ、10,000回それを合計する事は、単に 1 に「近い」値ではなく、正確に1とする結果を生みます。
- 明確な四捨五入挙動:正確な価値を持つ数に対する ROUND() の結果は、基礎をなす C ライブラリを動作させる方法のような環境要因ではなく、その引数に依存します。
- プラットフォームの独立:正確な数値に関する演算は Windows や Unix のような異なったプラットフォームを使ってもすべて同じです。
- 無効な値の取り扱いコントロール:ゼロによるオーバーフローと分割は探知できますが、エラーとして扱う事もできます。例えば、カラムに対して大きすぎる値の場合、切り捨てて、データ タイプの範囲内に丸めるのではなく、エラーとして扱う事ができます。同様に、ゼロによる分割を NULL の結果を生む演算としてよりむしろエラーとして扱う事ができます。採用すべき方法の選択は sql_mode システム変数の設定によって決まります。

これらの特徴による重要な結果は、MySQL 5.1 はスタンダード SQL に高度に準拠した物を提供するという事です。

以下の部分には、(古いアプリケーションに不適合しない可能性を含めて)精密計算がどのように機能するのかに関するいくつかの局面が含まれています。最後に、MySQL 5.1 がいかに正確に演算を処理するかを明らかにする若干の例を掲載しましたのでご覧ください。SQL モードをコントロールする為の sql_mode システム変数の使用に関する情報については、SQL モード をご参照ください。

数値のタイプ

正確に演算する為の精密計算の範囲には、正確値データ タイプ (DECIMALタイプと整数タイプ) 並びに正確値数値文字が含まれます。近似値データ タイプと数値文字は浮動小数点数として扱われます。

正確な値の数値文字には、整数部分あるいは端数部分もしくは両方が含まれています。それらにサインする事ができます。例: 1, .2, 3.4, -5, -6.78, +9.10.

近似値の数値文字は仮数と指数の付いた科学的な記号で表示されます。いずれかもしくは両方にサインする事ができます。例: 1.2E3、1.2E-3、-1.2E3、-1.2E-3。

同じに見える2つの数は両方とも正確な値か近似値である必要はありません。例えば、2.34 は(固定点の)正確な値ですが、2.34E0 は(浮動点の)近似値です。

DECIMAL データ タイプは固定点タイプで、計算は正確です。MySQL では、DECIMAL タイプは、幾つかの同義語を含んでいます:NUMERIC、DEC、FIXED。整数タイプも正確値タイプです。

FLOAT データ タイプおよび **DOUBLE** データ タイプは浮動点タイプで、計算によって近似値が得られます。MySQL では、**FLOAT** または **DOUBLE** を使った同義語のタイプは **DOUBLE PRECISION** および **REAL** です。

DECIMAL データ タイプの変更

このセクションは、以下の主題を中心にして、MySQL 5.1 内における **DECIMAL** データ タイプ(並びにその同義語)の持つ特徴について説明しています。

- 最大桁数
- ストレージ フォーマット
- ストレージ要件
- **DECIMAL** カラムの上部に対する非スタンダード MySQL 拡張子

MySQL の古いバージョン用に書かれたアプリケーションに適合しない可能性のある問題については、このセクション全体で指摘されます。

DECIMAL カラムに対する宣言構文は **DECIMAL(M,D)** です。MySQL 5.1 における引数値の範囲は次の通りです:

- **M** は最大桁数(精度)です。それには1から65までの範囲が含まれています。(古いバージョンの MySQL では、1から254までの範囲が許容されています。)
- **D** は少数点 (スケール) の右側にある数字の桁数です。それは0から30までの範囲であり、**M** より広くてはいけません。

M に対する最大値が65である事は、**DECIMAL** 値に関する計算が65桁まで正確である事を意味します。精度を65桁とするこの限界は正確値数値リテラルにも適用されるので、このようなりテラルの最大範囲は以前とは異なります。(古いバージョンの MySQL では、10進法の値を最高254桁まで持つ事ができました。しかし、計算は浮動小数点を使って実行されていたので、結果は正確でなく、近似値でした。)

MySQL 5.1 では、**DECIMAL** カラムの値は、4バイトの中に9個の10進数をパックするバイナリ フォーマットを使って格納されます。各値の整数部と端数部に対する格納要件は別々に決定されます。9桁の倍ごとに4バイト、使い残しの桁には4バイトの分数分の容量がそれぞれ必要です。例えば、**DECIMAL(18,9)** カラムは少数点のいずれかの側に9桁のスペースを持っているので、整数部分と端数部分には各々4バイト必要です。**DECIMAL(20,10)** カラムは少数点のいずれかの側に10桁のスペースを持っています。各々の部分には、9桁に対して4バイト、残りの桁に対して1バイトがそれぞれ必要です。

使い残しの桁に必要な格納容量を以下のテーブルに列記します:

使い残り桁数	バイト数
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

(5.0.3より前の)古いバージョンの MySQL とは異なり、MySQL 5.1 内の **DECIMAL** カラムは主要な + 文字または主要な 0 桁数を格納しません。**DECIMAL(5,1)** カラムの中に +0003.1 を挿入すると、それは 3.1 と格納されます。古い性能に依存するアプリケーションはこの変更を考慮するように改良を施さなければなりません。

MySQL 5.1 における **DECIMAL** カラムは、カラムの定義に規定する範囲を超える大きい値を容認しません。例えば、**DECIMAL(3,0)** カラムは -999 から 999 までの範囲をサポートしています。**DECIMAL(M,D)** カラムは小数点の左側に最大 $M - D$ 桁を容認します。これは + サインの代わりに余分な桁を格納する事を許容している MySQL のもっと古いバージョンのアプリケーションには対応できません。

SQL スタンダードは、**NUMERIC(M,D)** の精度が 正確に M 桁である事を要求しています。**DECIMAL(M, D)** に対して、スタンダードは少なくとも M 桁の精度を要求していますが、それ以上でも容認されます。MySQL の場合、**DECIMAL(M,D)** 並びに **NUMERIC (M,D)** は同じで、両方共正確に M 桁の精度を持っています。

DECIMAL データ タイプの古い処理に依存しているアプリケーションのポーテングに関する詳細情報については、MySQL 5.0 Reference Manual をご参照ください。

式の取り扱い

精密計算では、出来るだけ正確な値の数を附与して使用します。例えば、比較における数字を、値を変更しないで附与して正確に使用します。厳密な SQL モードでは、カラムの中に正確なデータ タイプ(**DECIMAL** あるいは整数)を利用してカラム内に **INSERT** する為に、もしカラム範囲内に収まる場合は、その数値が正確な値を使って挿入されます。検索された場合は、挿入した時と同じに値である必要があります。(ストリクト モードなしで、**INSERT** の為の切り捨てが許容されています。)

数値表現の取り扱い方法は表現にどんな種類の値が含まれているかによって異なります。

- 近似値が存在している場合、式は近似であり、これを、浮動小数点算数を使って評価します。
- 近似値が存在していない場合、式は正確な値だけを含みます。正確な値に端数部分 (小数点の後に続く値) が含まれている場合、式は **DECIMAL** 正確算数を使って評価され、式は 6 5 桁の精度を持ちます。(「exact」という言葉は、バイナリで表現できる、物の限界を条件と使用しています。例えば、**1.0/3.0** は小数表記法で **.333...** と近似させる事ができますが、正確な数として書かれないので、**(1.0/3.0)*3.0** は正確に **1.0** であると評価しません。)
- そうでなければ、式に整数値だけが含まれます。式は正確で、整数算数を使って評価され、**BIGINT** (64 ビット) と同じ精度を持っています。

数値表現に文字列が含まれている場合、それらはダブル精度浮動小数点の値に換えられ、その式は近似となります。

数値カラムへの挿入は SQL モードによって影響され、**sql_mode** システム変数によって制御されます。(**SQL モード** を参照してください。) 以下の部分で、(**STRICT_ALL_TABLES** モード値もしくは **STRICT_TRANS_TABLES** モード値によって選択された) ストリクト モード並びに **ERROR_FOR_DIVISION_BY_ZERO** について述べます。すべての制約をオンにするには、ストリクト モード値と **ERROR_FOR_DIVISION_BY_ZERO** を両方共含む **TRADITIONAL** モードを簡単に使用する事ができます。

```
mysql> SET sql_mode='TRADITIONAL';
```

正確タイプ カラム(**DECIMAL** もしくは整数)に数を挿入すると、カラムの範囲に収まる場合、それは正確な値と一緒に挿入されます。

値に桁数の多い端数部分が含まれている場合には、切り捨てが起こり、警告が生成されます。切り捨ては、セクション **丸め挙動** で述べた通りに行われます。

整数部分の桁数が多すぎ、値が大き過ぎる場合、以下の通り処理されます。

- ストリクト モードが有効でない場合、値は法定値に最も近い値に切り下げられ、警告が生成されます。
- ストリクト モードが有効である場合、オーバーフロー エラーが発生します。

アンダーフローは検出されないので、アンダーフローの処理は定義されていません。

デフォルトによって、ゼロでの分割は **NULL** の結果をもたらしますが、警告は生成されません。**ERROR_FOR_DIVISION_BY_ZERO** を使って SQL モードを有効にすると、MySQL はゼロでの分割を別途処理します。

- ストリクト モードが有効でないと、警告が生成されます。
- ストリクト モードが有効である場合、ゼロでの分割を含む挿入と更新は禁止され、エラーが発生します。

言い換えると、ゼロでの分割を実施する式を含む挿入と更新はエラーとして処理する事ができますが、これには、ストリクト モードの他に、`ERROR_FOR_DIVISION_BY_ZERO` が必要です。

我々がこのステートメントを発行すると仮定して下さい：

```
INSERT INTO t SET i = 1/0;
```

これは、ストリクト モードに `ERROR_FOR_DIVISION_BY_ZERO` モードを組み合わせると発生するものです：

sql_mode 値	結果
" (デフォルト)	警告・ エラーなし; <code>i</code> は <code>NULL</code> にセット。
ストリクト	警告・ エラーなし; <code>i</code> は <code>NULL</code> にセット。
<code>ERROR_FOR_DIVISION_BY_ZERO</code>	警告です。エラーがありません、 <code>i</code> は <code>NULL</code> にセット。
ストリクト、 <code>ERROR_FOR_DIVISION_BY_ZERO</code>	エラー ;行が挿入されていません。

数値カラムへの文字列の挿入に関しては、もし文字列が非数値内容を含んでいたら文字列から数値への変換は次のように行われます。

- 数で始まっていない文字列は数として使用する事はできず、これを使用すると、ストリクト モードの中にエラーが発生するか、警報が生成されます。これは空の文字列を含んでいます。
- 数で始まる文字列は変換する事ができますが、添付されている非数値部は切り捨てられます。切り捨て部分にスペース以外の何かが含まれている場合、これによって、ストリクト モードの中にエラーが発生するか、警報が生成されます。

丸め挙動

このセクションでは、`ROUND()` 関数、そして正確値タイプ (`DECIMAL` と整数) を利用したのカラムへの挿入の為の正確計算の丸めについて説明します。

`ROUND()` 関数は、引数が正確なものであるか、近似値であるかによって異なった丸めを実行します。

- 正確な値の数に対して、`ROUND()` は「round half up」規則を使用します:.5以上の端数を持つ値は、次の整数がポジティブである場合、それに対して切り上げ、ネガティブである場合、それに対して切り下げます。(言い換えると、それはゼロから切り捨てられるという事です。) .5以下の端数を持つ値は、次の整数がポジティブである場合、それに対して切り下げ、ネガティブである場合、それに対して切り上げます。
- 近似値の数に対する結果は C ライブラリによって異なります。多くのシステム上では、これは、`ROUND()` は「round to nearest even」規則を使用する事を意味します:端数部分を持つ値は最も近い偶整数に丸められます。

以下の例は、正確な値の丸めと近似値の丸めの違いを示しています。

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3         | 2         |
+-----+-----+
```

`DECIMAL` または整数カラムへの挿入の場合、目標は正確なデータ タイプであるので、挿入すべき値が正確か近似であるかを問わず、丸めには「round half up,」が使われます。

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 2

mysql> SELECT d FROM t;
+-----+
| d     |
+-----+
| 3     |
```

```
| 3 |
+---+
```

精密計算の例

このセクションでは、MySQL 5.1 における精密計算に対するクエリ結果を示す例を紹介します。

例1。数には、出来るだけ、与えられた通りの正確な値が使われている:

```
mysql> SELECT .1 + .2 = .3;
+-----+
|.1 + .2 = .3 |
+-----+
|          1 |
+-----+
```

浮動点値に対する結果は不正確です。

```
mysql> SELECT .1E0 + .2E0 = .3E0;
+-----+
|.1E0 + .2E0 = .3E0 |
+-----+
|          0 |
+-----+
```

正確値の扱いと近似値の扱いの違いを調べる別の方法は、合計に何回も小さい数字を加える事です。変数に `.0001` を 1,000回加える、以下のストアード プロシージャを考慮してみてください。

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

`d` と `f` 両方に対する合計は論理的には 1 であるべきですが、それは小数計算についてだけ言える事です。浮動点計算は小さなエラーを引き起こします:

```
+-----+
| d | f |
+-----+
| 1.0000 | 0.999999999999991 |
+-----+
```

例2。乗算はスタンダード SQL によって要求されたスケールで実施されます。すなわち、スケール `S1` と `S2` を持つ 2 つの数、`X1` と `X2` に対して、結果のスケールは `S1 + S2` です。

```
mysql> SELECT .01 * .01;
+-----+
|.01 * .01 |
+-----+
| 0.0001 |
+-----+
```

例3。丸め挙動はよく定義されています。

丸め挙動 (例えば、`ROUND()` 関数を使用)は基礎をなしている C ライブラリのインプリメンテーションから独立していて、それは、結果はプラットフォームが変わっても一貫している事を意味します。

正確な値のカラム(`DECIMAL`と整数)と正確な値の数に対する丸めには、「round half up」規則が使われます。`.5` 以上の端数を持つ値は、ここに示すように、ゼロから最も近い整数に丸められます。

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+-----+
```

```
| ROUND(2.5) | ROUND(-2.5) |
+-----+-----+
| 3         | -3         |
+-----+-----+
```

しかしながら、浮動小数点値の丸めには、多くのシステム上で「round to nearest even」規則を使用している C ライブラリが使用されます。このようなシステム上の端数付きの値は最も近い偶整数に丸められます。

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+-----+-----+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+-----+-----+
|          2   |          -2   |
+-----+-----+
```

例4。ストリクト モードでは、大き過ぎる値を挿入すると、結果はオーバーフローとなり、法的な値に切り下げる事なく、エラーを引き起こします。

MySQL がストリクト モードで運転されていない時には、法的な値への切り捨てが起こります:

```
mysql> SET sql_mode="";
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| 127   |
+-----+
1 row in set (0.00 sec)
```

しかし、ストリクト モードが有効であると、オーバーフロー条件が発生します。

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

例 5。ストリクト モードにして `ERROR_FOR_DIVISION_BY_ZERO` をセットすると、ゼロによる除算がエラーを引き起こし、`NULL` の結果は得られません。

非ストリクト モードにすると、ゼロによる除算が `NULL` の結果をもたらします:

```
mysql> SET sql_mode="";
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| NULL  |
+-----+
1 row in set (0.03 sec)
```


しかし、適当な SQL モードを有効にすると、ゼロによる除算はエラーとなります。

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

例6。MySQL 5.0.3より前のバージョン(精密計算が導入の前)では、正確値リテラルと近似値リテラルは両方共、ダブル精密浮動点値に変換されます。

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 4.1.18-log |
+-----+
1 row in set (0.01 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.07 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| a      | double(3,1) |      |      | 0.0      |       |
| b      | double      |      |      | 0         |       |
+-----+-----+-----+-----+-----+
2 rows in set (0.04 sec)
```

MySQL 5.0.3以降のバージョンでは、近似値リテラルは浮動点に変換されますが、正確値リテラルは **DECIMAL** として処理されます。

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 5.1.6-alpha-log |
+-----+
1 row in set (0.11 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+
| Field | Type              | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| a      | decimal(2,1) unsigned | NO   |      | 0.0      |       |
| b      | double              | NO   |      | 0         |       |
+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

例7。総計関数に対する引数が正確な数値タイプの物である場合、その結果も、少なくとも引数に等しいスケールの正確な数値タイプの物となります。

これらのステートメントを考慮してみてください:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

MySQL 5.0.3 (精密計算が MySQL の中に導入される以前の)での結果:

```
mysql> DESCRIBE y;
```

Field	Type	Null	Key	Default	Extra
AVG(i)	double(17,4)	YES		NULL	
AVG(d)	double(17,4)	YES		NULL	
AVG(f)	double	YES		NULL	

引数のタイプに関係なく、結果はダブルとなります。

MySQL 5.0.3による結果:

```
mysql> DESCRIBE y;
```

Field	Type	Null	Key	Default	Extra
AVG(i)	decimal(14,4)	YES		NULL	
AVG(d)	decimal(14,4)	YES		NULL	
AVG(f)	double	YES		NULL	

結果は、浮動点引数に対してだけダブルとなります。正確なタイプの引数の場合、結果も正確なタイプの物となります。

第23章 APIとライブラリー

目次

埋め込まれたMySQLサーバライブラリ、libmysqld	1135
埋め込まれたMySQLサーバライブラリーの概括	1135
libmysqldを使って行うプログラムの翻訳	1136
埋め込まれたMySQLサーバの使用に対する規制	1136
埋め込まれたサーバに対するオプション	1137
埋め込まれたサーバの例	1137
埋め込まれたサーバに対するのライセンスの供与	1140
MySQL C API	1140
C APIデータタイプ	1141
C API機能の概要。	1144
C API機能の説明	1148
準備されたC APIステートメント。	1193
準備されたC APIステートメントデータタイプ	1193
準備されたC APIステートメント機能の概要	1198
準備されたC APIステートメント機能の詳細	1201
準備されたC API ステートメントの問題	1220
マルチプルステートメントを実行するC APIハンドリング	1221
日付とタイム値のC API式取り扱い	1223
C APIスレッド機能の説明	1224
埋め込まれたC API機能の説明	1225
自動再接続挙動の管理	1225
C APIを使うときよく尋ねられる質問と問題	1226
クライアントプログラムの構築	1228
スレッド付きクライアントを作る方法	1228
MySQL PHP API	1229
MySQLとPHPに対する共通問題	1230
mysql と mysqli の両方を PHP内で可能にする	1230
MySQL Perl API	1230
MySQL C++ API	1231
MySQL Python API	1231
MySQL Tcl API	1231
MySQLエッフェルラッパー	1231
MySQLプログラム開発ユーティリティー	1231
mysql2mysqldb — MySQLと一緒に使うため、mSQLプログラムを変換してください。	1232
mysql_config — コンパイルオプションをコンパイルクライアントのために (用) 取得してください。 ..	1232

この章で、MySQLに利用できるAPIを入手する場所およびそれを使用する方法について説明します。CAPIは、MySQLチームによって開発されたので、最も適用範囲の広い内容を含み、他のAPIの基礎をなすものです。この章には、libmysqldライブラリー(埋め込みのサーバ)並びにアプリケーション・デベロッパーに有用な若干のプログラムも含まれています。

埋め込まれたMySQLサーバライブラリ、libmysqld

埋め込まれたMySQLサーバライブラリーの概括

埋め込まれたMySQLサーバライブラリーは、フル機能のMySQLサーバをクライアント・アプリケーションの中で運転することを可能にします。主な利益は埋め込まれたアプリケーションに対するスピードの増加とマネージメントの単純化です。

埋め込みのサーバライブラリーは、C/C++で書かれているMySQLのクライアント/サーバ用のバージョンに基づいています。従って、埋め込まれたサーバもC/C++で書かれます。他の言語を使って利用可能な埋め込みサーバは存在しません。

APIは埋め込みのMySQLバージョンとクライアント/サーバ用バージョン対するものと同じです。使い古しのアプリケーションを変えて、埋め込まれたライブラリを使うには、あなたは通常、次の機能に対するコールを加えなければなりません。

機能	呼び出し時期
<code>mysql_library_init()</code>	は他のMySQL機能が呼び出される前のなるべく早期に、 <code>main()</code> 機能の中に呼び出すべきです。
<code>mysql_library_end()</code>	を、あなたのプログラムが終了する前に、呼び出すべきです。
<code>mysql_thread_init()</code>	を、MySQLにアクセスするよう、あなたが生成する各スレッドの中に呼び出すべきです。
<code>mysql_thread_end()</code>	を、 <code>pthread_exit()</code> を呼び出す前に呼び出すべきです。

その後あなたは、自分のコードをlibmysqlclient.aの代わりに、libmysqld.aとリンクさせなければなりません。

`mysql_library_xxx()`関数もlibmysqlclient.aに含まれているので、あなたのアプリケーションを正しいライブラリにリンクさせることによって、埋め込まれているバージョンとクライアント/サーバ用のバージョンの間に変更を施すことが許されます。詳しくは`mysql_library_init()`を参照してください。

埋め込みサーバと独立サーバの間の1つ違いは、埋め込みサーバの場合、接続のための認証が初期設定によって無効にされていることです。埋め込みサーバに対して認証を使用するには、`configure`を呼び出して、あなたのMySQLデストリビューションを設定する時、`--with-embedded-privilege-control`を規定してください。

libmysqld使って行うプログラムの翻訳

libmysqldライブラリを取得するには、`--with-embedded-server`オプションを使って、MySQLを配置すべきです。詳しくは典型的な `configure` オプションを参照してください。

あなたのプログラムをlibmysqldとリングさせるとき、MySQLサーバが使用する、システムに固有なpthreadライブラリも含めなければなりません。`mysql_config --libmysqld-libs`を実行することによって、ライブラリのフルリストを取得することができます。

スレッド機能を自分のコードの中に直接呼び出さない場合でも、スレッド付きプログラムを翻訳し且つリンクさせるための正しいフラグを使用しなければなりません。

Cプログラムを翻訳して、埋め込まれたMySQLサーバライブラリに対して必要なファイルをプログラムの翻訳バージョンの中に含ませるため、GNU C コンパイラ(`gcc`)を使ってください。コンパイラには、様々なファイルを見つける場所を知る必要があります。彼にはプログラムを翻訳する方法に関するインストラクションも必要です。次の例は、プログラムをコマンドラインから翻訳出来る方法を示したものです。

```
gcc mysql_test.c -o mysql_test -lz \
/usr/local/mysql/bin/mysql_config --include --libmysqld-libs
```

`gcc`コマンドの直ぐ後に、Cプログラムファイルの名称を記載します。その後に、`-o`オプションを附与して、後に続くファイル名称は、コンパイラがアウトプットファイルに附与すべき名称であることを示します。コードの次のラインは、含めるファイルとライブラリおよび、そこでそれが翻訳されるシステムに対するその他の設定を告げます。`mysql_config`に発生した問題に起因して、(圧縮するための)オプション`-lz`がここに追加されます。`mysql_config`ピースは、一重引用符でなく、バックチェックの中に含められます。

埋め込まれたMySQLサーバの使用に対する規制

埋め込まれたサーバには次の制限が適用されます。

- ・ユーザが定義した機能(UDF)の使用禁止。
- ・コア・ダンプ上でのスタックトレースの禁止。
- ・これを親もしくは(複製が禁止されている)子供として設定することはできません。
- ・非常に大きい結果セットはメモリー性能の低いシステム上で使用することができない恐れがあります。
- ・あなたは埋め込まれたサーバに、外部プロセスからソケットまたはTCP/IPを使って接続することができません。しかし、あなたは中間アプリケーションに接続することはできませんが、その代わり、遠隔クライアントまたは外部プロセスのために、埋め込みサーバに接続することができます。

これらの制限の幾つかを、`mysql_embed.h`を編集して、ファイルと再翻訳MySQLを含めることによって、変更することができます。

埋め込まれたサーバに対するオプション

`mysql`サーバ・デーモンを使って附与することができるオプションを、埋め込まれたサーバ・ライブラリと一緒に使うことができます。サーバオプションは、`mysql_library_init()`に対する引数として、配列の中に附与することができます。これによって、サーバが初期化されます。それらを`my.cnf`のようなオプションファイルの中に附与することができます。Cプログラムのためにオプションファイルを規定するため、`--defaults-file`オプションを`mysql_library_init()`の2番目の引数要素の1つとして使ってください。`mysql_library_init()`機能に関する明細については、`mysql_library_init()`をご参照ください。

オプションファイルを使用すると、クライアント/サーバアプリケーションとMySQLが埋め込まれる場所にあるものの間で行う切り替えを容易にすることができます。共通オプションを`[server]`グループの下に置いてください。これらは両方のMySQLバージョンによって読み取られます。クライアント/サーバに固有なオプションは`[mysqld]`セクションの下に持ってくるべきです。埋め込まれたMySQLサーバライブラリに固有なオプションを`[embedded]`セクションの中に配置してください。アプリケーションに固有なオプションは、`[ApplicationName_SERVER]`と書いたラベルを貼ったセクションの下に配置されます。詳しくは[オプションファイルの使用](#)を参照してください。

埋め込まれたサーバの例

これら2つのエクザンプルプログラムはLinuxまたはFreeBSDシステムに変更を施すことなく、実行すべきです。その他のオペレーティングシステムに対して、大抵ファイルパスを使って、マイナーな変更を施すことが必要です。これらの例は、問題を把握するに十分な明細を、実のアプリケーションに対する必要部分であるクラッターなしで、あなたに附与するようにデザインされています。最初の例は、非常に直接的なものです。2番目の例は、幾つかのエラーチェックを伴う若干進歩したものです。最初の例の後には、プログラムを編集するため、コマンドライン・エントリが続きます。第2の例の後には、代わりに編集するために使うことができるGNUメークファイルが続きます。

例 1。

`test1_libmysqld.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = \
    { "mysql_test", "--defaults-file=my.cnf", NULL };
int num_elements = (sizeof(server_options) / sizeof(char *)) - 1;

static char *server_groups[] = { "libmysqld_server",
    "libmysqld_client", NULL };

int main(void)
{
    mysql_library_init(num_elements, server_options, server_groups);
    mysql = mysql_init(NULL);
    mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
    mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

    mysql_real_connect(mysql, NULL, NULL, NULL, "database1", 0, NULL, 0);

    mysql_query(mysql, "SELECT column1, column2 FROM table1");

    results = mysql_store_result(mysql);

    while((record = mysql_fetch_row(results))) {
        printf("%s - %s\n", record[0], record[1]);
    }

    mysql_free_result(results);
    mysql_close(mysql);
}
```

```
mysql_library_end();

return 0;
}
```

上のプログラムを翻訳するためのコマンドラインがここにあります：

```
gcc test1_libmysqld.c -o test1_libmysqld -lz \
`usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

例2。

その例を実際に試してみるために、MySQL ソースディレクトリと同じレベルに `test2_libmysqld` ディレクトリを生成させてください。 `test2_libmysqld.c` ソースと `GNUmakefile` をディレクトリーの中にセーブし、GNU `make` を `test2_libmysqld` ディレクトリーの内側から運転してください。

test2_libmysqld.c

```
/*
 * A simple example client, using the embedded MySQL server library
 */

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
    "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
    MYSQL *one, *two;

    /* mysql_library_init() must be called before any other mysql
     * functions.
     *
     * You can use mysql_library_init(0, NULL, NULL), and it
     * initializes the server using groups = {
     *   "server", "embedded", NULL
     * }.
     *
     * In your $HOME/.my.cnf file, you probably want to put:
     [test2_libmysqld_SERVER]
     language = /path/to/source/of/mysql/sql/share/english

     * You could, of course, modify argc and argv before passing
     * them to this function. Or you could create new ones in any
     * way you like. But all of the arguments in argv (except for
     * argv[0], which is the program name) should be valid options
     * for the MySQL server.
     *
     * If you link this client against the normal mysqlclient
     * library, this function is just a stub that does nothing.
     */
    mysql_library_init(argc, argv, (char **)server_groups);

    one = db_connect("test");
    two = db_connect(NULL);

    db_do_query(one, "SHOW TABLE STATUS");
    db_do_query(two, "SHOW DATABASES");

    mysql_close(two);
    mysql_close(one);
}
```



```
/* This must be called after all other mysql functions */
mysql_library_end();

exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    (void)putc('\n', stderr);
    if (db)
        db_disconnect(db);
    exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
    MYSQL *db = mysql_init(NULL);
    if (!db)
        die(db, "mysql_init failed: no memory");
    /*
     * Notice that the client and server use separate group names.
     * This is critical, because the server does not accept the
     * client's options, and vice versa.
     */
    mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
    if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
        die(db, "mysql_real_connect failed: %s", mysql_error(db));

    return db;
}

void
db_disconnect(MYSQL *db)
{
    mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
    if (mysql_query(db, query) != 0)
        goto err;

    if (mysql_field_count(db) > 0)
    {
        MYSQL_RES *res;
        MYSQL_ROW row, end_row;
        int num_fields;

        if (!(res = mysql_store_result(db)))
            goto err;
        num_fields = mysql_num_fields(res);
        while ((row = mysql_fetch_row(res)))
        {
            (void)fputs(">> ", stdout);
            for (end_row = row + num_fields; row < end_row; ++row)
                (void)printf("%s\t", row ? (char*)row : "NULL");
            (void)putc('\n', stdout);
        }
        (void)putc('\n', stdout);
        mysql_free_result(res);
    }
    else
        (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

    return;
err:
    die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

}

GNUmakefile

```
# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc     := $(HOME)/mysql-5.1/include
#lib     := $(HOME)/mysql-5.1/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS   = -L$(lib) -lmysqld -lz -lm -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
    rm -f $(targets) $(objects) *.core
```

埋め込まれたサーバに対するのライセンスの供与

我々は皆さんに、多目的言語(GPL)に基づくコードもしくは互換性のあるライセンスを解放することによって、フリーソフトウェアを昇進させること奨励します。これを行うことができない人々に対するその他のオプションは、MySQLコードのための商業ライセンスをMySQL ABから買うことです。明細については、<http://www.mysql.com/company/legal/licensing/>をご参照ください。

MySQL C API

C APIコードはMySQLを使って配付されます。それは[mysqlclient](#)ライブラリの中に含まれ、これによって、データベースにアクセスすることが許されます。

ソース・デストリビューション中のクライアントの多くはC言語で書かれています。C APIを使用する方法を示す例を探す場合、これらのクライアントを調べてください。MySQLソースデストリビューション中の[clients](#)ディレクトリーの中でこれらを見つけることができます。

他のクライアントAPIの殆ど(Connector/JおよびConnector/NETを除く全て)は[mysqlclient](#)ライブラリを使ってMySQLサーバと交信します。これは、例えば、他のクライアントプログラムによって使われると同じ環境変数の多くから利益を得ることができることを意味します。これらの変数のリストについては、[7章クライアントプログラムとユーティリティプログラム](#)をご参照ください。

クライアントは最大のコミュニケーションバッファサイズを持っています。最初に割り当てられたバッファのサイズ(16KB)は自動的に最大サイズまで増やされます。(この場合の最大は16MB)バッファサイズは需要保証としてだけ増やされるので、初期設定最大限度を単純に増加させても、使用すべき資源の本質的增加を引き起こしません。このサイズチェックは殆どの場合、エラーステートメントとコミュニケーションパケットに対するチェックです。

コミュニケーションバッファは、(クライアントからサーバへのトラフィックに対する)シングルSQLステートメント並びに(サーバからクライアントへのトラフィックに対する)返還データの1本の列を含めるに十分な大きさを持っていなければなりません。各スレッドのコミュニケーションバッファはクエリーもしくは列を最大限度まで扱

うため、ダイナミックに拡大されます。例えば、サイズが16MB未満のデータを含む `BLOB` 値を持っている場合、(サーバとクライアントの両方の中に)少なくとも16MBのコミュニケーション・バッファ・リミットを持っていないければなりません。クライアントのデフォルト最大値は16MBですが、サーバのそれは1MBです。サーバを立ち上げる時、`max_allowed_packet` パラメータの値を変更することによってこれを増やすことができます。[サーバパラメータのチューニング](#)を参照してください。

MySQLサーバは各クエリーの後、各コミュニケーションバッファを `net_buffer_length` バイトに縮小させます。クライアントの場合、接続に関するバッファのサイズは接続が閉鎖されるまで減少されません。その時、クライアントのメモリーは改善されます。

詳しくは[スレッド付きクライアントを作る方法](#)をご参照ください。同じプログラムの中にサーバとクライアントを含み(外部MySQLサーバと交信しない)独立したアプリケーションを生成するには、[埋め込まれたMySQLサーバライブラリ](#)、`libmysqld`をご参照ください。

C APIデータタイプ

- `MYSQL`

この構造は 1 データベース接続に対するハンドルを表します。それは殆ど全てのMySQL機能に対して使われます。`MYSQL` 構造のコピーを作ろうとすべきではありません。このようなコピーが使用可能である保証はありません。

- `MYSQL_RES`

この構造は横列(`SELECT`、`SHOW`、`DESCRIBE`、`EXPLAIN`)を戻すクエリーの結果を表します。クエリーから返された情報はこのセクションの残りの部分の中で結果セットと呼ばれます。

- `MYSQL_ROW`

これはデータの 1 本の横列に関して、タイプに安全な表現です。それは現在カウントされたバイトストリングのアーレイとして搭載されています。(フィールド値にバイナリーデータが含まれている場合、内部に無効なバイトが含まれている恐れがあるので、これらをゼロで終わるストリングとして扱うことはできません。)横列は `mysql_fetch_row()` を呼び出すことによって得られます。

- `MYSQL_FIELD`

この構造には、フィールド名、タイプ並びにサイズのようなフィールドに関する情報が含まれています。ここで、その中身 (メンバー) について明細に説明します。繰り返して `mysql_fetch_field()` を呼び出すことによって、各フィールドごとに、`MYSQL_FIELD` 構造を得ることができます。フィールド値はこの構造の一部ではありません; これらは `MYSQL_ROW` 構造の中に含まれています。

- `MYSQL_FIELD_OFFSET`

これは、MySQL フィールドリストの中に入力してもタイプに安全なオフセットの表示です。(`mysql_field_seek()`) によって使われた。) オフセットは列の中でゼロで始まるフィールドナンバーです。

- `my_ulonglong`

横列の数および `mysql_affected_rows()`、`mysql_num_rows()` と `mysql_insert_id()` に使ったタイプ。このタイプは、0 から `1.84e19` までの範囲を提供します。

幾つかのシステム上では、タイプ `my_ulonglong` の値をプリントしようと試みる動作は作動しません。このような値をプリントするには、それを `unsigned long` に変換して、`%lu` 印刷フォーマットを使ってください。例:

```
printf ("Number of rows: %lu\n",
        (unsigned long) mysql_num_rows(result));
```

- `my_bool`

(ゼロでない)真または(ゼロの)虚の値に対するブーリアンタイプ。

`MYSQL_FIELD` 構造には、ここで列記したメンバーが含まれています:

- `char * name`

ゼロで終わるストリングを含むフィールドの名称。フィールドに [AS](#) クローズを含むアリアスを附与すると、[name](#) はアリアスとなります。

- [char * org_name](#)

ゼロで終わるストリングを含むフィールドの名称。アリアスは無視されます。

- [char * table](#)

それが計算フィールドでない場合、このフィールドを含むテーブルの名称。計算されたフィールドの場合、[table](#) 値は空のストリングとなります。カラムが画面から選択される場合、[table](#) は画面を指定します。テーブルあるいは画面に [AS](#) クローズを使ってエイリアスを附与すると、[table](#) の値はエイリアスとなります。

- [char * org_table](#)

ゼロで終わるストリングを含むテーブルの名称。エイリアスは無視されます。カラムを画面から選ぶと、[org_table](#) が基盤となるテーブルを指定します。

- [char * db](#)

ゼロで終わるストリングとして、フィールドを生むデータベースの名称。フィールドが計算フィールドである場合、[db](#) は空のストリングとなります。

- [char * catalog](#)

カタログ名。この値は常に "def" となります。

- [char * def](#)

ゼロで終わるストリングを含むこのフィールドのデフォルト値。[mysql_list_fields\(\)](#) を使う場合に限り、これはセットされます。

- [unsigned long length](#)

テーブルの定義に規定したフィールドの幅。

- [unsigned long max_length](#)

結果セットに対するフィールドの最大幅(結果セット中に実在する列に対する最長フィールド値の長さ)。[mysql_store_result\(\)](#) あるいは [mysql_list_fields\(\)](#) を使う場合、これにはフィールドに対する最大長さが含まれます。[mysql_use_result\(\)](#) を使うと、この変数の値はゼロになります。

[max_length](#) の値は、結果セット中の値のストリング表示の長さとなります。例えば、[FLOAT](#)カラムを復元する場合にあって、「widest」値が-12.345ある場合、[max_length](#)は('12.345'の長さに等しい) 7となります。

準備されたステートメントを使っている場合、バイナリのプロトコルに対して、値の長さは結果セット中の値のタイプによって変わるので、[max_length](#)は初期設定されません。([準備されたC APIステートメントデータタイプ](#)を参照してください。) [max_length](#)値がとにかく欲しい場合、[mysql_stmt_attr_set\(\)](#)を使って、[STMT_ATTR_UPDATE_MAX_LENGTH](#)オプションを有効化すると、[mysql_stmt_store_result\(\)](#)を呼び出す時、その長さがセットされます。([mysql_stmt_attr_set\(\)](#)と[mysql_stmt_store_result\(\)](#)をご参照ください。)

- [unsigned int name_length](#)

[name](#)の長さ。

- [unsigned int org_name_length](#)

[org_name](#)の長さ。

- [unsigned int table_length](#)

[table](#)の長さ。

- [unsigned int org_table_length](#)

`org_table`の長さ。

- `unsigned int db_length`

`db`の長さ。

- `unsigned int catalog_length`

`catalog`の長さ。

- `unsigned int def_length`

`def`の長さ。

- `unsigned int flags`

フィールドに異なるビットフラグ。`flags`値を次のビットセットに関してゼロ以上にすることができます:

フラグ値	フラグの説明
<code>NOT_NULL_FLAG</code>	フィールドは <code>NULL</code>
であることができません <code>PRI_KEY_FLAG</code>	フィールドはプライマリーキーの一部です
<code>UNIQUE_KEY_FLAG</code>	フィールドはユニークキーの一部です
<code>MULTIPLE_KEY_FLAG</code>	フィールドは非ユニークキーの一部です
<code>UNSIGNED_FLAG</code>	フィールドには <code>UNSIGNED</code> 属性が含まれています
<code>ZEROFILL_FLAG</code>	フィールドには <code>ZEROFILL</code> 属性が含まれています
<code>BINARY_FLAG</code>	フィールドには <code>BINARY</code> 属性が含まれています
<code>AUTO_INCREMENT_FLAG</code>	フィールドには <code>AUTO_INCREMENT</code> 属性が含まれています
<code>ENUM_FLAG</code>	フィールドは <code>ENUM</code> (deprecated)です
<code>SET_FLAG</code>	フィールドは <code>SET</code> (deprecated)です
<code>BLOB_FLAG</code>	フィールドは <code>BLOB</code> または <code>TEXT</code> (deprecated)です
<code>TIMESTAMP_FLAG</code>	フィールドは <code>TIMESTAMP</code> (deprecated)です

`BLOB_FLAG`フラグ、`ENUM_FLAG`フラグ、`SET_FLAG`フラグおよび`TIMESTAMP_FLAG`フラグの使用は、それらはそのタイプではなく、フィールドのタイプを表示するので、けなされます。`field->type`を、`MYSQL_TYPE_BLOB`、`MYSQL_TYPE_ENUM`、`MYSQL_TYPE_SET`もしくは代わりに`MYSQL_TYPE_TIMESTAMP`に対して出来るだけテストしてください。

次の例は`flags`値の典型的な利用を図解したものです:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

`flags`値のブーリアン・ステータス決定するため、次の便利なマクロを使うことができます:

フラグのステータス	摘要
<code>IS_NOT_NULL(フラグ)</code>	このフィールドが <code>NOT NULL</code>
<code>IS_PRI_KEY(フラグ)</code> であると規定されている場合、真	このフィールドがプライマリーキーである場合、真
<code>IS_BLOB(フラグ)</code> であると規定されている場合、真	このフィールドが <code>BLOB</code> または <code>TEXT</code> (deprecated; test <code>field->type</code> instead)である場合、真

- `unsigned int decimals`

数値フィールドに対する小数の数。

- `unsigned int charsetnr`

フィールドに対するキャラクター・セットナンバー。

- `enum enum_field_types` タイプ

フィールドのタイプ。type値は次のテーブルに示すMySQL_TYPE_シンボルの1つにすることができます。

タイプの値	タイプの説明
MYSQL_TYPE_TINY	TINYINT フィールド
MYSQL_TYPE_SHORT	SMALLINT フィールド
MYSQL_TYPE_LONG	INTEGER フィールド
MYSQL_TYPE_INT24	MEDIUMINT フィールド
MYSQL_TYPE_LONGLONG	BIGINT フィールド
MYSQL_TYPE_DECIMAL	DECIMAL または NUMERIC フィールド
MYSQL_TYPE_NEWDECIMAL	精密計算 DECIMAL または NUMERIC
MYSQL_TYPE_FLOAT	FLOAT フィールド
MYSQL_TYPE_DOUBLE	DOUBLE or REAL フィールド
MYSQL_TYPE_BIT	BIT フィールド
MYSQL_TYPE_TIMESTAMP	TIMESTAMP フィールド
MYSQL_TYPE_DATE	DATE フィールド
MYSQL_TYPE_TIME	TIME フィールド
MYSQL_TYPE_DATETIME	DATETIME フィールド
MYSQL_TYPE_YEAR	YEAR フィールド
MYSQL_TYPE_STRING	CHAR または BINARY フィールド
MYSQL_TYPE_VAR_STRING	VARCHAR または VARBINARY フィールド
MYSQL_TYPE_BLOB	BLOB or TEXT フィールド (大長を決めるためにmax_lengthを使用)
MYSQL_TYPE_SET	SET フィールド
MYSQL_TYPE_ENUM	ENUM フィールド
MYSQL_TYPE_GEOMETRY	空間フィールド
MYSQL_TYPE_NULL	NULL-type field

フィールドが数値のタイプを持っているかどうか試すためにIS_NUM()マクロを使うことができます。type値をIS_NUM()に渡してください。そうすると、フィールドが数値である場合、それが真であると判定します。

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

ストリングデータタイプに対して、バイナリーデータであるか、非バイナリーデータであるかを区別するには、charsetnr値が63であるかどうか調べてください。もしそうであるなら、キャラクターセットはbinaryで、それは非バイナリーデータでなくバイナリーデータであることを示します。これが、BINARYとCHAR、VARBINARYとVARCHAR並びにBLOBとTEXTを区別する方法です。

C API機能の概要。

C API中で利用可能な機能の概要をここで一括説明し、後のセクションで各機能について詳細に説明します。C API機能の説明を参照してください。

機能	摘要
<code>my_init()</code>	グローバル変数およびスレッドに安全なプログラム中のスレッドハンドラーを初期化する。

<code>mysql_affected_rows()</code>	最後のUPDATE、DELETEまたはINSERTクエリーによって、負荷/削除/挿入された行の数を返す。
<code>mysql_autocommit()</code>	トグルスイッチを操作して、オートコミットモードをオンまたはオフにする。
<code>mysql_change_user()</code>	オープン接続上のユーザーとデータベースを変更する。
<code>mysql_close()</code>	サーバ接続を閉にする。
<code>mysql_commit()</code>	>取引を確約する。
<code>mysql_connect()</code>	MySQLサーバに接続する。この機能は忌避されるので、代わりに、 <code>mysql_real_connect()</code> を使用してください。
<code>mysql_create_db()</code> に	データベースを生成してください。この機能は忌避されるので、代わりにSQLステートメントCREATE DATABASEを使ってください。
<code>mysql_data_seek()</code>	は、クエリー結果セット中の任意の列ナンバーを探し求めます。
<code>mysql_debug()</code>	は、附与されたストリングを使ってDEBUG_PUSHを実行します。
<code>mysql_drop_db()</code>	はデータベースをドロップします。この機能は忌避されるので、代わりに、SQLステートメントDROP DATABASEを使ってください。
<code>mysql_dump_debug_info()</code>	はサーバにデバッグ情報をログに書き込ませます。
<code>mysql_eof()</code>	は最後の結果セットが読み込まれたか否か査定します。この機能は忌避されるので、代わりに <code>mysql_erro()</code> または <code>mysql_error()</code> を使うことができます。
<code>mysql_errno()</code>	は最近取り出したMySQL機能に対するエラーのナンバーを返します。
<code>mysql_error()</code>	は最近取り出したMySQL機能に対するエラーのメッセージを返します。
<code>mysql_escape_string()</code>	SQLステートメントの中で使用するため、ストリング中で行う特別キャラクターの使用を避けるてください。
<code>mysql_fetch_field()</code>	は次のテーブルフィールドのタイプを返します。
<code>mysql_fetch_field_direct()</code>	は、フィールドナンバーを附与すると、テーブルフィールドのタイプを返します。
<code>mysql_fetch_fields()</code>	は全フィールド構造のアレーを返します。
<code>mysql_fetch_lengths()</code>	は現列中に全カラムの長さを返します。
<code>mysql_fetch_row()</code>	は結果セットから次列を持ってきます。
<code>mysql_field_seek()</code>	はカラムカーソルを規定カラム上に移動させます。
<code>mysql_field_count()</code>	最近のステートメントに対する結果のナンバーを返します。
<code>mysql_field_tell()</code>	は最後の <code>mysql_fetch_field()</code> に対して使用したフィールドカーソルの位置に戻します。
<code>mysql_free_result()</code>	は結果セットによって使用されたメモリーを解放します。
<code>mysql_get_client_info()</code>	はクライアントバージョン情報をすとストリングとして返します。
<code>mysql_get_client_version()</code>	はクライアントバージョン情報をすと整数として返します。
<code>mysql_get_host_info()</code>	は接続を述べたストリングを返します。
<code>mysql_get_server_version()</code>	はサーバのナンバーを整数として返します。
<code>mysql_get_proto_info()</code>	は接続に使用したプロトコルバージョンを返します。
<code>mysql_get_server_info()</code>	はサーバのバージョンナンバーを返します。
<code>mysql_info()</code>	は最近実効したクエリーに関する情報を返します。
<code>mysql_init()</code>	はMYSQL構造を取得するか、初期化します。
<code>mysql_insert_id()</code>	は前のクエリーによってAUTO_INCREMENTカラムに対して生成されたIDを返します。
<code>mysql_kill()</code>	は附与したスレッドを破壊します。
<code>mysql_library_end()</code>	はMySQL C APIライブラリーを最終承認します。
<code>mysql_library_init()</code>	はMySQL C APIライブラリーを初期化します。

mysql_list_dbs()	はシンプルなレギュラー表現にマッチするデータベースの名称を返します。
mysql_list_fields()	はシンプルなレギュラー表現にマッチするフィールドの名称を返します。
mysql_list_processes()	は現サーバレッドのリストを返します。
mysql_list_tables()	はシンプルなレギュラー表現にマッチするテーブルの名称を返します。
mysql_more_results()	は更に結果が存在しているかチェックします。
mysql_next_result()	は複数ステートメントの実効において、次結果を返し、初期化します。
mysql_num_fields()	は結果セット中のカラムのナンバーを返します。
mysql_num_rows()	は結果セット中の列のナンバーを返します。
mysql_options()	は mysql_connect() に対する接続オプションをセットします。
mysql_ping()	はサーバへの接続が作動しているかチェックし、必要に応じて再接続します。
mysql_query()	はゼロで終わるストリングとして規定されているSQLクエリーを実効します。
mysql_real_connect()	はMySQLサーバに接続します。 Connects to a MySQL server.
mysql_real_escape_string()	は、接続の現キャラクターセットを考慮して、SQLステートメント中で使用するストリング中に特別なキャラクターを使用することを避けます。
mysql_real_query()	はカウントストリングとして規定されているSQLクエリーを実効します。
mysql_refresh()	はテーブルとキャッシュをフラッシュするリセットする。
mysql_reload()	は供与されたテーブルを再び取り込むよう告げます。
mysql_rollback()	は取引を撤回させます。
mysql_row_seek()	は mysql_row_tell() から戻った値を使って、結果セット中から列オフセットを模索します。
mysql_row_tell()	は列カーソルの位置を返します。
mysql_select_db()	はデータベースを選択します。
mysql_server_end()	はMySQL C APIライブラリーを最終承認します。
mysql_server_init()	はMySQL C APIライブラリーを初期化します。
mysql_set_local_infile_default()	はLOAD DATA LOCAL INFILEハンドラーをその初期設定値にコールバックするようセットします。
mysql_set_local_infile_handler()	アプリケーションに固有なLOAD DATA LOCAL INFILEハンドラーコールバックをインストールします。
mysql_set_server_option()	は接続用オプション(multi-statements を含む)をセットします。
mysql_sqlstate()	は最後のエラーに対するSQLSTATEエラーコードを返します。
mysql_shutdown()	はデータベースサーバの操業を停止させます。
mysql_stat()	はサーバステータスをストリングとして返します。
mysql_store_result()	はクライアントに対する結果セットを一式復元します。
mysql_thread_end()	はスレッドハンドラーを最終承認します。
mysql_thread_id()	は現スレッドのIDを返します。
mysql_thread_init()	はスレッドハンドラーを初期化します。
mysql_thread_safe()	は、クライアントがスレッドに対して安全になるよう編集されている場合、1を返します。
mysql_use_result()	は、列ごとに得られた結果セットを各々復元させる作業を開始させます。
mysql_warning_count()	は以前のSQLステートメントに対する警告カウントを返します。

アプリケーションプログラムには、MySQLとの相互対話のために、この一般アウトラインを使うべきです:

1. [mysql_library_init\(\)](#)を呼び出すことによって、MySQLライブラリーを初期化してください。この機能は、[mysqlclient](#) C クライアントライブラリーと埋め込まれた[mysqld](#)サーバライブラリーの両方に存在するので、それは、-

`libmysqlclient` フラグとリンクすることによって、レギュラークライアントプログラムを構築するか、または `libmysqld` フラグとリンクすることによって、埋め込みサーバアプリケーションを構築するのに使用されます。

2. `mysql_init()` を呼び出して、接続ハンドラーを初期化し、更に `mysql_real_connect()` を呼び出して、サーバに接続してください。
3. SQLステートメントを発行して、それらの結果を処理してください。(次の講義で、これを行う方法に関する明細な情報を提供します。)
4. `mysql_close()` を呼び出すことによって、MySQLサーバに対する接続を閉じてください。
5. `mysql_library_end()` を呼び出すことによって、MySQL ライブラリの使用を終えてください。

`mysql_library_init()` と `mysql_library_end()` の呼び出しは、MySQL ライブラリの適正な初期化と最終承認を確保する目的で実効します。クライアントライブラリとリンクされるアプリケーションのために、それらは改善されたメモリ管理を提供します。`mysql_library_end()` を呼び出さない場合、メモリブロックが割り当てられたままになります。(これによって、アプリケーションによって使われたメモリの量は増えませんが、若干のメモリ漏れ探知器がそれについて不平を言います。)埋め込まれたサーバとリンクされるアプリケーションに対して、これらのコールはサーバを立ち上げたり、立ち下げたりします。

非マルチスレッド環境では、`mysql_library_init()` へのコールは除かれる恐れがあります。なぜなら `mysql_init()` は必要に応じてそれを自動的に呼び出すからです。しかし、マルチスレッド環境では、`mysql_library_init()` が `mysql_init()` によって呼び出されると、レース条件が可能となります：`mysql_library_init()` はスレッドに対して安全でないので、他のクライアントライブラリを呼び出す前に、それを呼び出すべきです。

サーバに接続するには、`mysql_init()` を呼び出して接続ハンドラーを初期化してから、そのハンドラーを(ホストネーム、ユーザーネームおよびパスワード等の他の情報と一緒に)使って、`mysql_real_connect()` を呼び出してください。接続と同時に、`mysql_real_connect()` は、`reconnect` フラグ(MYSQL 構造の一部)を、バージョンが 5.0.3 より古い API 中にある 1 または新しいバージョンの API 中にある 0 の値にセットします。このフラグに対する 1 の値は、ステートメントが失われた再接続のために実効できない場合、実効を諦める前にサーバに再び接続しようとすることを示します。`MYSQL_OPT_RECONNECT` オプションを `mysql_options()` に使用して、再接続行動を制御することができます。接続を止めたい場合、`mysql_close()` を呼び出して、それを修了させてください。

クライアントは接続がアクティブである間に、`mysql_query()` あるいは `mysql_real_query()` を使って、SQLステートメントをサーバに送ることができます。2つの違いは、`mysql_query()` クエリーにゼロで終わるストリングを規定するよう期待しますが、`mysql_real_query()` はカウントストリングを規定するよう期待します。ストリングが(空のバイトを含む恐れのある)バイナリーデータを含んでいる場合、`mysql_real_query()` を使わなければなりません。

各非 `SELECT` クエリー(例えば、`INSERT`、`UPDATE`、`DELETE`)に対して、`mysql_affected_rows()` を呼び出すことによって、幾つの列が変更(影響)されたかを調べることができます。

`SELECT` クエリーに対して、選択された列を結果セットとして取り出します。(幾つかのステートメントは、列を戻す場合のように、`SELECT` となることにご注目ください。これらには `SHOW`、`DESCRIBE` 及び `EXPLAIN` が含まれています。これらを `SELECT` ステートメントの場合と同じ方法で処理すべきです。)

クライアントが結果セットを処理する 2 つの方法があります。1 つの方法は、`mysql_store_result()` を呼び出すことによって、全結果セットを直ちに回収する方法です。この機能はサーバからクエリーによって戻された全ての列を取得して、それらをクライアントの中に記憶します。第2の方法は、`mysql_use_result()` を呼び出すことによって、クライアントが列別結果セットの回収を開始させる方法です。この機能は復元したものを初期化しますが、実際にサーバから横列を取得しません。

両ケースで、`mysql_fetch_row()` を呼び出すことによって、横列にアクセスします。`mysql_store_result()`、`mysql_fetch_row()` を使って、前にサーバからフェッチされた横列にアクセスします。`mysql_use_result()` を使って、`mysql_fetch_row()` はサーバから実際に列を復元させます。`mysql_fetch_lengths()` を呼び出すことによって、各横列中にあるデータのサイズに関する情報を入手することができます。

結果セットを閉じた後、そめに使用したメモリーを解放するため、`mysql_free_result()` を呼び出してください。

2 つの復元メカニズムは相補的關係を持っています。クライアントプログラムに、それらの要件を満たす最も適切なアプローチを選択すべきです。実務においてクライアントはより共通して、`mysql_store_result()` を使う傾向があります。

`mysql_store_result()`を呼び出すと、横列はすべて、クライアントにフェッチされているので、単に連続的に横列にアクセスするだけでなく、`mysql_data_seek()` または `mysql_row_seek()` を使って、結果セットの中を前後に移動して、結果セット中の現横列の位置を変更することができる利益が生まれます。`mysql_num_rows()`を呼び出すことによって、そこに存在している横列の数を検知することもできます。一方、`mysql_store_result()`に対するメモリー要件は結果セットに対して非常に厳しいので、記憶不足が起こる恐れがあります。

`mysql_use_result()`を利用すると、それは1回に1列しか維持せず、(割り当て間接容量も低くて済む)ので、クライアントには結果セットに対してより低い記憶容量が要求され、`mysql_use_result()`をより高速にすることができる利益が得られます)。この場合、サーバを縛り付けるのを避けるため、速やかに各横列を処理しなければならず、それらを全部復元するまで、結果セット中に幾列あるか知ることができず(ただ順次横列にアクセスすることができるだけである)という不利益が生まれます。さらに、復元の途中で探していた情報を情報が発見できたとしても、必ずすべての横列を復元してください。

APIは、クライアントがステートメントがSELECTであるか否かを知ることなく、適切に(必要に応じて横列だけを復元する)ステートメントに返答することを可能にします。各`mysql_query()`(または`mysql_real_query()`)の後に、`mysql_store_result()`を呼び出すことによって、これを行うことができます。結果セットの呼び出しが成功すると、ステートメントはSELECTであったので、横列を読むことができます。結果セットの呼び出しが失敗した場合には、`mysql_field_count()`を呼び出して、結果が実際に期待できたはずか否か査定してください。`mysql_field_count()`がゼロを返す場合、戻されたステートメントは戻ったが、(それがINSERT、UPDATE、DELETE等であったことを示す)データがないので、列を戻すことが期待できなかったこととなります。`mysql_field_count()`がゼロでない場合、ステートメントは列を戻していたはずですが、戻しませんでした。これは、ステートメントが機能を停止したSELECTであったことを示します。これを実行できる方法を示す例については、`mysql_field_count()`に対する説明をご参照ください。

`mysql_store_result()`と`mysql_use_result()`は両方共、結果セットを作り出すフィールドに関する情報(フィールドの数と名称およびタイプ等)を取得することを可能にします。`mysql_fetch_field()`を繰り返し呼び出すか、もしくは列の中のフィールドナンバー別に`mysql_fetch_field_direct()`を呼び出すことによって、フィールド情報に列の中で連続してアクセスすることができます。`mysql_field_seek()`を呼び出すことによって、現在のカーソルポジションを変更することができます。フィールドカーソルの設定は`mysql_fetch_field()`に対するその後の呼び出しに影響を及ぼします。`mysql_fetch_fields()`を呼び出すことによって、フィールドに対する情報を全て直ちに取得することもできます。

エラーを検出して報告するために、MySQLは`mysql_errno()`機能および`mysql_error()`機能を使ってエラー情報にアクセスする手段を提供します。これらは、どんなエラーがいつ起こったかを査定することを可能にして、最近呼び出した、成功するか失敗することもあり得る機能に対するエラーコードあるいはエラーメッセージを返します。

C API機能の説明

この説明で使ったNULLパラメータまたは戻り値は、MySQLNULL値ではなく、Cプログラミング言語のセンスに含まれるNULLを意味します。

値を返す機能は一般にポインタあるいは整数を返します。特に規定しない限り、ポインタを返す機能は、非NULL値を返して成功を示すか、NULL値を返してエラーを示し、整数を返す機能は、ゼロを返して成功を示すか、非ゼロを返してエラーを示します。「non-zero」は文字通りの意味を持っていることに注目ください。機能説明に別な指示がない限り、ゼロ以外の値に対してテストをしてはなりません：

```
if (result)           /* correct */
... error ...

if (result < 0)        /* incorrect */
... error ...

if (result == -1)     /* incorrect */
... error ...
```

機能がエラーを返すとき、機能説明のサブセクションにあるエラーが起こる恐れのあるエラーのタイプを列記します。`mysql_errno()`を呼び出すことによって、これらのどれが起こったかを見つけることができます。`mysql_error()`を呼び出すことによって、エラーのストリング表示を得ることができます。

`mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

説明

`mysql_query()`または`mysql_real_query()`を使ってステートメントを実行した後、`UPDATE`のために変更されたか、`DELETE`のために削除されたか、または`INSERT`のために挿入された行の数を返します。`SELECT`ステートメントに対して、`mysql_affected_rows()`は`mysql_num_rows()`と同じように作動します。

戻り値

ゼロより大きい整数は影響を与えられたか、復元された横列の数を示します。ゼロは、`UPDATE`ステートメントに対する記録が更新されなかったか、クエリー中の`WHERE`クローズにマッチした列が存在していなかったか、クエリーがまだ実行されていないことを示します。`-1`は、クエリーがエラーを戻したか、`SELECT`クエリーに対して、`mysql_store_result()`を呼び出す前に、`mysql_affected_rows()`が呼び出されたことを示します。`mysql_affected_rows()`が未サイン値を返すので、戻り値を`(my_ulonglong)-1` (または同等な`(my_ulonglong)~0`)と比べることによって、`-1`をチェックすることができます。

エラー

なし。

例

```
char *stmt = "UPDATE products SET cost=cost*1.25 WHERE group=10";
mysql_query(&mysql,stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

`UPDATE`ステートメントに対して、`CLIENT_FOUND_ROWS`フラグを規定する場合、`mysql`に接続するとき、`mysql_affected_rows()`は、`WHERE`クローズの規定に基づき、マッチさせられた行の数を返します。そうならないと、デフォルト行動は実際に変えられた横列の数を返すはずです。

`REPLACE` コマンドを使うとき、古い列を新しい列と入れ替えた場合、この例では、重複するものを削除した後、1本の列が挿入されたので、`mysql_affected_rows()`は2を返すことにご注目ください。

`INSERT ... ON DUPLICATE KEY UPDATE`を使って1本の列を挿入すると、その列が新しい列である場合、`mysql_affected_rows()`は1を返し、既存の列を更新すると2を返します。

`mysql_autocommit()`

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

説明

`mode`が1の場合、オートコミット モードをオンに、`mode`が0の場合、オフにそれぞれセットしてください。

戻り値

成功している場合ゼロ。エラーが起こった場合、ゼロ以外。

エラー

なし。

`mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password, const char *db)
```

説明

ユーザーを変えて、`db`によって規定されたデータベースを、`mysql`によって規定された接続上の(現)デフォルトデータベースになるように仕向けてください。次のクエリーの中では、このデータベースは明確なデータベース規定者を含まないテーブルリファレンスのためのデフォルトとなります。

接続されたユーザーを認証することができないか、当該ユーザーがデータベースを使う許可を得ていない場合、`mysql_change_user()`は失敗します。この場合、ユーザーとデータベースは変更されません。

デフォルトデータベースを持ちたくない場合、`db`パラメータを`NULL`にセットすることができます。

このコマンドは状態を新しい接続をしたかのようにリセットします。([自動再接続挙動の管理](#) を参照してください。) それは常に、アクティブな取引の`ROLLBACK`を実施し、すべての一時テーブル閉じてドロップし、ロックされているすべてのテーブルをアンロックします。セッションシステム変数が対応するグローバルシステム変数の値にリセットされます。準備されたステートメントがリリースされ、`HANDLER`変数が閉じられます。`GET_LOCK()`を使って取得されたロックが解放されます。たとえユーザーが変わらなかったとしても、これらの効果は起こります。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

`mysql_real_connect()`から取得出来る同じもの。

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

- `ER_UNKNOWN_COM_ERROR`

MySQLサーバはこのコマンドを実行しません。(サーバが多分高齢のため)。

- `ER_ACCESS_DENIED_ERROR`

ユーザー名あるいはパスワードが間違っています。

- `ER_BAD_DB_ERROR`

データベースが存在していませんでした。

- `ER_DBACCESS_DENIED_ERROR`

ユーザーはデータベースに対するアクセス権利を持っていませんでした。

- `ER_WRONG_DB_NAME`

データベース名が長すぎます。

例

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
    fprintf(stderr, "Failed to change user. Error: %s\n",
        mysql_error(&mysql));
}
```

`mysql_character_set_name()`

```
const char *mysql_character_set_name(MYSQL *mysql)
```


説明

現在の接続に対するデフォルト文字セットの戻り。

戻り値

デフォルト文字セット

エラー

なし。

mysql_close()

```
void mysql_close(MYSQL *mysql)
```

説明

前に開かれた接続を閉じてください。ハンドルがmysql_init()またはliteral>mysql_connect()によって自動的に割り当てられたものである場合、mysql_close()は、mysqlに合わせた接続ハンドルの割り当ても解除します。

戻り値

なし。

エラー

なし。

mysql_commit()

```
my_bool mysql_commit(MYSQL *mysql)
```

説明

現在の取引を確定してください。

この機能のアクションはcompletion_typeシステム変数の値の適用を条件としています。特に、completion_typeの値が2である場合、取引を終えた後に、サーバはリリースを行って、クライアント接続を閉じます。クライアントプログラムはクライアント側から接続を閉じるために、mysql_close()を呼び出すべきです。

戻り値

成功している場合ゼロ。エラーが起こった場合、ゼロ以外。

エラー

なし。

mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)
```

説明

この機能はけなされます。代わりに、出来るだけmysql_real_connect()を使ってください。

mysql_connect()は、host上で運転されるMySQLデータベースエンジンに接続を確立しようと努めます。mysql_connect()は、他のAPI機能のいずれかをが実行可能になる前に、mysql_get_client_info()を除き、うまく完了されなければなりません。

パラメータの意味は、接続パラメータをNULLにすることができる点を除き、`mysql_real_connect()`に対する該当パラメータの場合と同じです。この場合、C APIは、メモリーを自動的に接続構造に割り当て、`mysql_close()`が呼び出されると、それを解放します。この方法には、接続に失敗すると、エラーの修復ができない欠点があります。(mysql_errno()またはmysql_error()から、エラー情報を得るには、有効なMYSQLポインターを用意しなければなりません。)

戻り値

`mysql_real_connect()`に対するものと同じ。

エラー

`mysql_real_connect()`に対するものと同じ。

mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

説明

dbパラメータと名付けたデータベースを生成させてください。

この機能はけなされます。SQLCREATE DATABASEステートメントを発行するために、代わりに、出来るだけmysql_query()を使用してください。

戻り値

データベースがうまく生成された場合には、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

コマンドが妥当でないオーダーで実行されました。

- CR_SERVER_GONE_ERROR

MySQLサーバが立ち去りました。

- CR_SERVER_LOST

サーバへの接続がクエリー中に失われました。

- CR_UNKNOWN_ERROR

未知のエラーが起きました。

例

```
if(mysql_create_db(&mysql, "my_database"))
{
    fprintf(stderr, "Failed to create new database. Error: %s\n",
        mysql_error(&mysql));
}
```

mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

説明

クエリー結果セットの中で任意の横列を探してください。offset値は列ナンバーなので、0からmysql_num_rows(result)-1までの範囲に収めるべきです。

この機能は、結果セット構造にクエリーの全結果を含めることを求めるので、`mysql_data_seek()`は、`mysql_use_result()`と一緒に、`mysql_use_result()`と併用する場合に限り使用することができます。

戻り値

なし。

エラー

なし。

mysql_debug()

```
void mysql_debug(const char *debug)
```

説明

`DBUG_PUSH`が或るストリングを処理する場合。`mysql_debug()`はFred Fish デバッグ ライブラリを使います。この機能を使うには、デバッグをサポートするよう、クライアントライブラリを編集しなければなりません。[Debugging a MySQL Server](#)、[Debugging a MySQL Client](#) を参照して下さい。

戻り値

なし。

エラー

なし。

例

ここに示す呼び出しは、クライアントライブラリにクライアントマシン上の/tmp/client.traceの中にトレースファイルを生成させます:

```
mysql_debug("d:t:O:/tmp/client.trace");
```

mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

説明

`db`パラメータと名付けたデータベースをドロップさせてください。

この機能はけなされます。SQL `CREATE DATABASE` ステートメントを発行するために、代わりに、出来るだけ `mysql_query()` を使用してください。

戻り値

データベースがうまくドロップされた場合には、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

例

```
if(mysql_drop_db(&mysql, "my_database"))
    fprintf(stderr, "Failed to drop the database: Error: %s\n",
        mysql_error(&mysql));
```

mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

説明

サーバに幾つかのデバッグ情報をログに書き込むよう指示してください。これを働かせるには、接続されたユーザーは[SUPER](#)特権を持っていないけません。

戻り値

コマンドの附与が成功した場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

説明

この機能はけなされます。代わりに[mysql_errno\(\)](#)または[mysql_error\(\)](#)を使うことができます。

[mysql_eof\(\)](#)は結果セットの最後の列が読み込まれたか否かを査定します。

[mysql_store_result\(\)](#)に対して成功した呼び出しから結果セットを取得する場合、クライアントは1回のオペレーションで全セットを受け取ります。この場合、[NULL](#)の[mysql_fetch_row\(\)](#)からの戻りは、結果セットの終わりが届いているので、[mysql_eof\(\)](#)を呼び出す必要はないことを常に意味します。[mysql_store_result\(\)](#)と一緒に使うとき、[mysql_eof\(\)](#)は常に真を返します。

一方、[mysql_use_result\(\)](#)を使って結果セットの復元を開始させる場合、セットの列は、[mysql_fetch_row\(\)](#)を繰り返して呼び出すと、サーバから一つずつ得られます。この処理の途中に接続にエラーが発生した恐れがあるので、[mysql_fetch_row\(\)](#)からの[NULL](#)戻り値は、結果セットの終わりが正常に届いたことを必ずしも意味しません。

この場合、`mysql_eof()`を使用して、何が起こったかを査定することができます。結果セットが終わりまで届いた場合、`mysql_eof()`はゼロ以外の値を返し、エラーが発生した場合にはゼロを返します。

歴史的に、`mysql_eof()`は標準MySQLエラー機能`mysql_errno()`および`mysql_error()`より先行されます。それらのエラー機能が同じ情報を提供するので、それらの使用は`mysql_eof()`より優先され、けなされます。(実際、エラー機能は、エラーが起こるとエラーの理由を示すのに対して、`mysql_eof()`はブール値だけを返すので、それらはもっと多くの情報を提供します。)

戻り値

エラーが発生しなかった場合、ゼロ。結果セットが終わりまで届いた場合、非ゼロ。

エラー

なし。

例

次の例は、`mysql_eof()`の使用方法を示したものです：

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

しかし、標準MySQLエラー機能で同じ効果を達成することができます：

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

mysql_errno()

`unsigned int mysql_errno(MYSQL *mysql)`

説明

`mysql`によって規定された接続に対して、`mysql_errno()`は最近使用したものを選んでを返します、成功/失敗のあるAPI機能のエラーコードを返します。ゼロの戻り値はエラーが起こらなかったことを意味します。クライアントのエラーメッセージナンバーは、`MySQLerrmsg.h`ヘッダーファイル中に列記されています。サーバのエラーメッセージナンバーは、`mysqlid_error.h`の中に列記されています。[Errors, Error Codes, and Common Problems](#)にはエラーも列記されています。

`mysql_fetch_row()`を含む幾つかの機能は、これらが成功した場合、`mysql_errno()`をセットしない点にご注目ください。

経験則に従うと、サーバに情報を求めなければならない全ての機能は成功した場合、(必ず)`mysql_errno()`をリセットします。

`mysql_errno()`が戻したMySQLに固有なエラーナンバーは、`mysql_sqlstate()`が戻したSQLSTATE値とは異なっています。例えば、`mysql`クライアントプログラムは以下のフォーマットを使ってエラーを表示します。この場合、1146は`mysql_errno()`値で、'42S02'は対応する`mysql_sqlstate()`値です：

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

戻り値

最後に行った`mysql_xxx()`の呼び出しに対するエラーコード、それが失敗した場合、ゼロはエラーがゼロはエラーが発生しなかったことを意味します。

エラー

なし。

mysql_error()

```
const char *mysql_error(MYSQL *mysql)
```

説明

`mysql`が規定した接続に対して、`mysql_error()`は、最近呼び出したが失敗したAPI機能に対するエラーを含む、ゼロで終わる文字列を返します。機能が失敗しなかった場合、`mysql_error()`の戻り値は前のエラーかもしくはエラーがないことを示す空の文字列となる場合があります。

経験則に従うと、サーバに情報を求めなければならない全ての機能は成功した場合、(必ず)`mysql_errno()`をリセットします。

`mysql_errno()`をリセットする機能に対して、次の2つのテストは同等です：

```
if(*mysql_errno(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0])
{
    // an error occurred
}
```

クライアントエラーメッセージの言語は、MySQLクライアントライブラリを再編集することによって変更することができます。現在、幾つかの異なった言語で書かれたエラーメッセージの中から、好みのものを選ぶことができます。[英語以外のエラーメッセージ](#)を参照してください。

戻り値

エラーを述べたゼロで終わる文字列。エラーが発生しなかった場合の空文字列。

エラー

なし。

mysql_escape_string()

代わりに、`mysql_real_escape_string()`を使うべきです！

この機能は、`mysql_real_escape_string()`と全く同じです。`mysql_real_escape_string()`は接続ハンドラーをその第一引数として取り込み、現在の文字セットに従う文字列を回避します。`mysql_escape_string()`は接続引数を取り込まず、現在の文字セットを尊重しません。

mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

説明

結果セットの定義を**MYSQL_FIELD**構造として規定したカラムに戻ってください。この機能を繰り返して呼び出して、結果セット中のすべてのカラムに関する情報を復元してください。フィールドがまだ残っているときには、**mysql_fetch_field()**は**NULL**を返します。

mysql_fetch_field()はリセットされて、新しい**SELECT**クエリーを実行する度に、最初のフィールドに関する情報を返します。**mysql_fetch_field()**によって戻されたフィールドは、**mysql_field_seek()**に対する呼び出しによっても影響を受けます。

mysql_query()を呼び出して、テーブル上で**SELECT**を実施したが、**mysql_store_result()**を呼び出さなかった場合、MySQLは、**mysql_fetch_field()**を呼び出して、**BLOB**フィールドの長さを求めると、デフォルトblobの長さ (8KB) を返します。(MySQLは**BLOB**に対する最大長さを知らないので、8KBのサイズが選ばれます。いつか、これを構成可能にすべきです。)結果セットを復元したと同時に、**field->max_length**には、特定クエリー中のこのカラムに対する最大値の長さが含まれます。

戻り値

現在のカラムのための**MYSQL_FIELD**構造。**NULL**カラムが残っていない場合。

エラー

なし。

例

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

mysql_fetch_field_direct()

MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)

説明

結果セット内のカラムに対して、フィールドナンバー**fieldnr**を附与すると、そのカラムのフィールドの定義が**MYSQL_FIELD**構造として戻ります。任意のカラムのために定義を復元する目的でこの機能を使うことができます。**fieldnr**の値を0から**mysql_num_fields(result)-1**までの範囲に収めるべきです。

戻り値

規定されたカラムのための**MYSQL_FIELD**構造。

エラー

なし。

例

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

mysql_fetch_fields()

`MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)`

説明

それは結果セットに対する全ての`MYSQL_FIELD`構造のアレーを返します。各構造は、フィールドの定義に結果セットの1コラムを規定します。

戻り値

結果セットの全てのコラムのための`MYSQL_FIELD`構造アレー。

エラー

なし。

例

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
    printf("Field %u is %s\n", i, fields[i].name);
}
```

mysql_fetch_lengths()

`unsigned long *mysql_fetch_lengths(MYSQL_RES *result)`

説明

それは結果セット内にある現在の列のコラムの長さを返します。フィールド値をコピーすることを計画する場合、この長さ情報は、`strlen()`の呼び出しを避けることができるので、最適化に有用です。以上に加え、結果セットにバイナリーデータが含まれている場合、`strlen()`はゼロの文字を含むフィールドに対して不正確な結果を返すので、この機能を利用してデータのサイズを査定しなければなりません。

空のコラムのため、および`NULL`値を含むコラムのための長さはゼロです。これら2つのケースを区別する方法を調べるには、`mysql_fetch_row()`の説明をご参照ください。

戻り値

(ゼロで終わる文字群を含まない)各コラムを代表する未サインの長い整数アレー)。`NULL`エラーが起こった場合。

エラー

`mysql_fetch_lengths()`は結果セットの現在列に対してのみ有効です。`mysql_fetch_row()`を呼び出す前もしくは結果セット中のすべての列を復元した後にそれを呼び出すと、それは`NULL`を返します。

例

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("Column %u is %lu bytes in length.\n",

```

```

        i, lengths[i]);
    }
}

```

mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

説明

それは結果セットの次列を復元します。mysql_store_result()の後で使用する、mysql_fetch_row()は、復元すべき列がないとき、NULLを戻します。mysql_use_result()の後で使用する、mysql_fetch_row()は、復元すべき列がないか、エラーが発生した場合、NULLを戻します。

行中の値の数はmysql_num_fields(result)によって与えられます。rowがmysql_fetch_row()を呼び出して得た戻り値を保持している場合、その値へのポインターは、row[0]から row[mysql_num_fields(result)-1]としてアクセスされます。列中のNULL値はNULLポインターによって示されます。

列中のフィールド値の長さは、mysql_fetch_lengths()を呼び出すことによって得ることができます。空のフィールドとNULLを含むフィールドは両方共length 0を含んでいます。フィールド値のためのポインターをチェックすることによって、これらを区別することができます。ポインターがNULLである場合、フィールドもNULLとなります。さもないと、フィールドは空になります。

戻り値

次列のためのMYSQL_ROW構造。NULL復元すべき列がないか、エラーが発生した場合。

エラー

mysql_fetch_row()の呼び出しと呼び出しの間に、エラーはリセットされないことにご注目ください。

- CR_SERVER_LOST

サーバへの接続がクエリー中に失われました。

- CR_UNKNOWN_ERROR

未知のエラーが起きました。

例

```

MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
    unsigned long *lengths;
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("[%s] ", (int) lengths[i],
            row[i] ? row[i] : "NULL");
    }
    printf("\n");
}

```

mysql_field_count()

unsigned int mysql_field_count(MYSQL *mysql)

説明

それは接続にある最近のクエリーに対するカラムの数を戻します。

この機能の正常な使用は、`mysql_store_result()`がNULLを戻した時(および、それによって、結果セットポインターを持っていなかった時)です。この場合、`mysql_field_count()`を呼び出して、`mysql_store_result()`が空でない結果を生成し終えているべきであったか否かを査定することができます。これによって、クライアントプログラムが、クエリーがSELECT(またはSELECTのような)ステートメントであったか否かを知ることなく、適正なアクションをとることが許されます。ここの例はこれを実行できる方法を示すものです。

詳しくは [なぜmysql_store_result\(\)時々返すNULLの後mysql_query\(\) 成功を返す](#) を参照してください。

戻り値

結果セット中にあるカラムの数を表す無署名の整数。

エラー

なし。

例

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

代案は`mysql_field_count(&mysql)`の呼び出しを`mysql_errno(&mysql)`の呼び出しに変えることです。この場合、`mysql_field_count()`の値からステートメントがSELECTであったか否かを推定しないで、`mysql_store_result()` から直接、エラーがないかチェックしています。

mysql_field_seek()

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)

説明

フィールドカーソルを附与附与されたオフセットにセットしてください。`mysql_fetch_field()`に対する次の呼び出しによって、オフセットに添付されているカラムの定義が復元されます。

列の始めを探すために、ゼロのオフセット値を渡してください。

戻り値

フィールドカーソルの前の値。

エラー

なし。

mysql_field_tell()

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

説明

最後のmysql_fetch_field()に使用したフィールドカーソルの位置に戻ってください。この値をmysql_field_seek()に対する引数として使うことができます。

戻り値

フィールドカーソルの現在のオフセット。

エラー

なし。

mysql_free_result()

```
void mysql_free_result(MYSQL_RES *result)
```

説明

mysql_store_result()、mysql_use_result()、mysql_list_dbs()等によって結果セットに割り当てられたメモリーを解放してください。結果セットを処分するとき、mysql_free_result()を呼び出すことによって、それが使用しているメモリーを解放しなければなりません。

それを解放した後、結果セットにアクセスしようと試みてはなりません。

戻り値

なし。

エラー

なし。

mysql_get_character_set_info()

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

説明

この機能はデフォルトクライアント文字セットに関する情報を提供します。デフォルト文字セットはmysql_set_character_set()機能を使って変更することができます。

例

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

```
}
```

mysql_get_client_info()

```
const char *mysql_get_client_info(void)
```

説明

それはクライアントライブラリ・バージョンを表す文字列を返します。

戻り値

MySQLクライアントライブラリ・バージョンを表す文字列。

エラー

なし。

mysql_get_client_version()

```
unsigned long mysql_get_client_version(void)
```

説明

それはクライアントライブラリ・バージョンを表す整数を返します。値には、**XYZZ**フォーマットが含まれています。ここでは、**X**はメジャーバージョンで、**YY**はリリースレベルで、**ZZ**はリリースレベル内のバージョンナンバーです。例えば、**40102**の値は、**4.1.2**のクライアントバージョンを表します。

戻り値

MySQLクライアントライブラリ・バージョンを表す整数。

エラー

なし。

mysql_get_host_info()

```
const char *mysql_get_host_info(MYSQL *mysql)
```

説明

それはサーバーホスト名を含む使用接続のタイプを述べた文字列を返します。

戻り値

サーバーホスト名と接続タイプを示す文字列。

エラー

なし。

mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

説明

それは現在の接続によって使われたプロトコルのバージョンを返します。

戻り値

現在の接続によって使われたプロトコルのバージョンを表す無署名の整数。

エラー

なし。

mysql_get_server_info()

```
const char *mysql_get_server_info(MYSQL *mysql)
```

説明

それはサーバのバージョンナンバーを表すストリングを返します。

戻り値

それはサーバのバージョンナンバーを表す文字ストリングを返します。

エラー

なし。

mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

説明

それはサーバのバージョンナンバーを整数として返します。

戻り値

このフォーマット中のMySQLサーバーバージョンを表すナンバー：

```
major_version*10000 + minor_version *100 + sub_version
```

例えば、5.1.5が50105として戻されます。

この機能は、クライアントプログラム中で、バージョンに固有なサーバー能力が幾つか存在するかどうかを速やかに査定するのに有用です。

エラー

なし。

mysql_get_ssl_cipher()

```
const char *mysql_get_ssl_cipher(MYSQL *mysql)
```

説明

`mysql_get_ssl_cipher()`は、或るサーバへの接続に使用したSSL暗号を返します。`mysql`は、`mysql_init()`から戻された接続ハンドラーです。

この機能はMySQL 5.1.9に追加されました。

戻り値

SSL暗号の名称を持つストリングで、接続に使われたもの、または、暗号が使われていない場合、`NULL`。

mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long length)
```

説明

この機能は、SQLステートメントの中に使うことができる法定SQLストリングを作るために使われます。文字列を参照してください。

from中のストリングは、2つの16進桁数としてコード化された各文字と一緒に、16進法のフォーマットにコード化されます。結果はtoの中に置かれ、その最後の部位にゼロバイトが付加されます。

fromによって指し示されたストリングは、lengthバイトの長さのもでなければなりません。toバッファに長さが少なくともlength*2+1バイトを割り当てなければなりません。mysql_hex_string()が戻るとき、toの中身はゼロで終わるストリングです。戻り値は、ゼロで終わる文字を含まないコード化されたストリングの長さに等しい大きさを持っています。

戻り値は、0xvalueまたはX'value'フォーマットを使ったSQLステートメントの中に置くことができます。しかし、戻り値には、0xまたはX'...'は含まれていません。呼び出し人は、どれを望まれても供給しなくてはなりません。

例

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What's this",11);
end = strmov(end,"0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
        mysql_error(&mysql));
}
```

サンプルの中に使用したstrmov()機能は、mysqlclientライブラリの中に含まれています。当該機能はstrcpy()と同じような働きをしますが、ポインターを最初のパラメータの終わりのゼロに戻します。

戻り値

toの中に置かれた、ゼロで終わる文字を含まない値の長さ。

エラー

なし。

mysql_info()

```
const char *mysql_info(MYSQL *mysql)
```

説明

ここに列記したステートメントを除く、最近実行されたステートメントに関する情報を復元してください。他の情報に対して、mysql_info()はNULLを戻します。ここで述べたように、ストリング変数のフォーマットはステートメントのタイプによって変わります。ナンバーは説明用のもので、ストリングには適切な値が含まれています。

- INSERT INTO ... SELECT ...

ストリングフォーマット: 記録: 100 Duplicates:0 Warnings: 0

- INSERT INTO ... VALUES (...),(...),(...)

ストリングフォーマット: 記録: 3 Duplicates:0 Warnings: 0

- LOAD DATA INFILE ...

ストリングフォーマット : 記録 : 1 削除:スキップ:0 Warnings: 0

- ALTER TABLE

ストリングフォーマット : 記録 : 3 Duplicates:0 Warnings: 0

- UPDATE:

ストリングフォーマット : マッチする列 : 40 変更済み:40 Warnings: 0

`mysql_info()`は非 `NULL` 値を`INSERT ... VALUES` ステートメントの複数列フォームに対してのみ、返すことにご注目ください (即ち、複数列のリストが指定されている場合のみ)。

戻り値

最近実行されたステートメントに関する追加情報を表す文字ストリング。`NULL`ステートメントに対する情報が得られない場合。

エラー

なし。

mysql_init()

`MYSQL *mysql_init(MYSQL *mysql)`

説明

`mysql_real_connect()`に適した`MYSQL`オブジェクトを割り当てるか初期化してください。`mysql`が`NULL`ポインタである場合、機能は、新しいオブジェクトを割り当て、初期化し且つ戻します。さもなければ、オブジェクトは初期化され、オブジェクトのアドレスが戻されます。`mysql_init()`が新しいオブジェクトを割り当てる場合、それは、接続を閉じるために`mysql_close()`が呼び出されるとき、解放されます。

戻り値

初期化された`MYSQL*` ハンドル。`NULL`新しいオブジェクトを割り当てるに十分なメモリーがない場合。

エラー

メモリーが不十分な場合、`NULL`が戻されます。

mysql_insert_id()

`my_ulonglong mysql_insert_id(MYSQL *mysql)`

説明

それは、`AUTO_INCREMENT`カラムのために、以前の`INSERT` ステートメントによって生成された値を戻します。`INSERT`ステートメントを、`AUTO_INCREMENT` フィールドを含むテーブルの中で実行した後、この機能を使ってください。

`mysql_insert_id()`の戻り値は、以下の条件が明確に更新されない限り、常にゼロです。

- `AUTO_INCREMENT`カラムの中に値を記憶する`INSERT`ステートメント。値が、`NULL`または0をカラムに収納することによって、自動的に生み出されるか、あるいは明示的な非スペシャル値であるか否かにかかわらず、これは真実です。
- マルチ列`INSERT`ステートメントの場合、`mysql_insert_id()`の戻り値は、MySQLのバージョンによって異なります。

バージョンナンバー5.1.12以降では、初めに自動的にかつ 首尾よく生成された `AUTO_INCREMENT` 値を返します。MySQLバージョンが5.1.12以前のMySQLでは、`mysql_insert_id()`は、値の挿入が成功したか否かにかかわらず、最初に自動生成された`AUTO_INCREMENT`値を返します。

列がうまく挿入されなかった時、`mysql_insert_id()`はゼロを返します。

- MySQL 5.1.12を立ち上げて、`INSERT ... SELECT` ステートメントを実行したが、自動生成された値がうまく挿入されない場合、`mysql_insert_id()` は最後に挿入した列のidを返します。
- MySQL 5.1.12を立ち上げて、`INSERT ... SELECT`ステートメントが`LAST_INSERT_ID(expr)`を使う場合、`mysql_insert_id()`は`expr`を返します。
- `LAST_INSERT_ID(expr)`をカラムに挿入することによって、`AUTO_INCREMENT`値を生成する`INSERT`ステートメント。
- `LAST_INSERT_ID(expr)`を更新することによって、`AUTO_INCREMENT`値を生成する`INSERT`ステートメント。
- `mysql_insert_id()`の値は、結果セットを返す`SELECT`のようなステートメントによって影響を受けません。
- 以前のステートメントがエラーを戻した場合、`mysql_insert_id()`の値は規定されません。

5.1.12以降のバージョンでは、`mysql_insert_id()`の戻り値を以下のシーケンスに単純化することができます：

- 自動インCREMENTカラムが存在し、自動生成された値がうまく挿入された場合、当該値の最初のものが戻されます。
- ステートメント中に`LAST_INSERT_ID(EXPR)`が含まれていた場合、影響されたテーブル中にオートインCREMENT・カラムが含まれていたとしても、`EXPR`が戻されます。
- 戻り値は使用したステートメントによって変わります。`INSERT INTO`ステートメントの後に呼び出されたとき：
 - テーブル中にオートインCREMENT・カラムが存在していて、テーブル中にうまく挿入されたこのカラムーに対して、明確な幾つかの値があった場合、当該明確な値の最後が戻されます。

`INSERT ...ON DUPLICATE KEY`ステートメントの後に呼び出されるとき：

- オートインCREMENT・カラムとテーブルが存在していて更に、うまく挿入された幾つかの明確な値もしくは幾つかの更新された列があった場合、挿入されたか更新された値の最後が戻されます。

注記

前のステートメントが`AUTO_INCREMENT`値を使用していない場合、`mysql_insert_id()`は0値を返します。後で使用するため、値をセーブする必要がある場合、`mysql_insert_id()`を、値を生成するステートメントの直後に確実に呼び出してください。

`mysql_insert_id()`の値は、現在のクライアント接続の中に発行されたステートメントのみによって影響を受けます。それは、他のクライアントによって発行されたステートメントによって影響を受けません。

詳しくは [情報関数](#) を参照してください。

`SQLLAST_INSERT_ID()`機能の値には、(5.1.12の立ち上げ後)最初に自動生成された値で、うまく挿入された値、または(5.1.12より前に)列がうまく挿入された場合、最初に自動生成された値が含まれてることにもご注目ください。他の違いは、`AUTO_INCREMENT`カラムを特別でない値にセットした場合、`LAST_INSERT_ID()`は更新されないことです。

`LAST_INSERT_ID()`と`mysql_insert_id()`の違いは、`LAST_INSERT_ID()`はスクリプト中で容易にされるが、`mysql_insert_id()`は`AUTO_INCREMENT`カラムに何が起こったかに関する少し正確な情報を提供しようとすることに起因して生まれます。

戻り値

戻り値。

エラー

なし。

mysql_kill()

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

説明

`pid`によって規定されたスレッドを抹消しようサーバに求めてください。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

mysql_library_end()

```
void mysql_library_end(void)
```

説明

この機能は MySQL ライブラリを完成させます。(例えば、サーバとの接続を断った後で)ライブラリの使用を行うとき、それを呼び出すべきです。呼び出しによって起こされるアクションは、あなたのアプリケーションがMySQLクライアント・ライブラリにリンクされているか、MySQL埋め込みサーバ・ライブラリにリンクされているかによって変わります。`-lmysqlclient`フラグを使うことによって、`libmysqlclient`ライブラリに対してリンクされたクライアントプログラムのために、`mysql_library_end()`は、クリーンアップするために、幾つかのメモリー管理を実施します。`-lmysqld`フラグを使うことによって、`ibmysqld`に対してリンクされた埋め込みサーバ・アプリケーションのために、`mysql_library_end()`は、埋め込みサーバをシャットダウンした後、クリーンアップを行います。

使用情報については、[C API機能の概要](#)。および[mysql_library_init\(\)](#)をご参照ください。

mysql_library_init()

```
int mysql_library_init(int argc, char **argv, char **groups)
```

説明

他のMySQL機能呼び出す前に、この機能をMySQLライブラリを初期化するために呼び出すべきです。アプリケーションが埋め込みサーバを使っている場合、この呼び出しはサーバを立ち上げ、サーバが使うサブシステム(`mysys`、`InnoDB`等)を初期化します。

アプリケーションがMySQLライブラリを使って実行された後、`mysql_library_end()`を呼び出してクリーンアップしてください。`mysql_library_init()`を参照してください。

非マルチスレッド環境では、`mysql_library_init()`へのコールは除かれる恐れがあります。なぜなら[mysql_init\(\)](#)は必要に応じてそれを自動的に呼び出すからです。しかしながら、マルチスレッドの環境では、もし[mysql_library_init\(\)](#)が、`mysql_library_init()`はスレッドにとって安全でないことを理由に、`mysql_library_init()`によ

て取り出される場合、競合条件が可能です。それ故に、他のクライアントライブラリを呼び出すの前に、それを呼びだすべきです。

`argc` 引数と `argv` 引数は、`main()` に対する引数と類似しています。`argv` の最初の要素は無視されます。(それには大抵プログラム名が含まれています)。便利にするため、もしサーバに対するコマンドライン引数がない場合、`argc` は 0 (ゼロ) であることが許されます。`mysql_library_init()` は引数のコピーを作るので、呼び出し後に起こる `argv` または `groups` の破壊に対して安全になります。

埋め込みサーバを立ち上げることなく、外部サーバに接続したい場合、`argc` に対してネガティブな値を規定しなければなりません。

`groups` 引数を、そこからオプションを読み取るべきオプションファイル内のグループを示すストリングのアーレイにすべきです。[オプションファイルの使用](#)を参照してください。配列への最終入力を `NULL` にすべきです。`groups` 引数が `NULL` である場合、便利にするため、`[server]` グループと `[embedded]` グループが初期設定によって使われます。

追加使用情報については、[C API機能の概要](#)。をご参照ください。

例

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
    "this_program", /* this string is not used */
    "--datadir=",
    "--key_buffer_size=32M"
};
static char *server_groups[] = {
    "embedded",
    "server",
    "this_program_SERVER",
    (char *)NULL
};

int main(void) {
    if (mysql_library_init(sizeof(server_args) / sizeof(char *),
        server_args, server_groups)) {
        fprintf(stderr, "could not initialize MySQL library\n");
        exit(1);
    }

    /* Use any MySQL API functions here */

    mysql_library_end();

    return EXIT_SUCCESS;
}
```

戻り値

Okなら 0、エラーが起これたら 1。

mysql_list_dbs()

`MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)`

説明

それは、`wild` パラメータによって規定された単純なレギュラー表現にマッチするサーバー上のデータベース名によって構成されている結果セットを戻します。`wild` には、ワイルドカード文字および `'%'` または `'_'` 含めるか、もしくはすべてのデータベースにマッチする `NULL` ポインターであることが許されます。`mysql_list_dbs()` の呼び出しは、クエリー `SHOW databases [LIKE wild]` の実行と同じです。

`mysql_free_result()` を使って結果セットを解放しなければなりません。

戻り値

成功するためのMYSQL_RES結果セット。NULLエラーが起こった場合。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_OUT_OF_MEMORY](#)

メモリ不足。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起こりました。

mysql_list_fields()

MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)

説明

それは、[wild](#)パラメータによって規定された単純なレギュラー表現にマッチする規定のテーブル中のフィールド名によって構成されている結果セットを戻します。[wild](#)には、ワイルドカード文字および '%'または '_'含めるか、これをすべてのデータベースにマッチするNULLポインターにすることが許されます。[mysql_list_fields\(\)](#)の呼び出しは、クエリーSHOW COLUMNS FROM tbl_name [LIKE wild]の実行と同じです。

[mysql_list_fields\(\)](#)の代わりに、SHOW COLUMNS FROMtbl_nameを使用することが推薦されていることにご注目ください。

[mysql_free_result\(\)](#)を使って結果セットを解放しなければなりません。

戻り値

成功するためのMYSQL_RES結果セット。NULLエラーが起こった場合。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起こりました。

mysql_list_processes()

MYSQL_RES *mysql_list_processes(MYSQL *mysql)

説明

それは、現在のサーブスレッドを述べた結果セットを返します。これは、`mysqladmin processlist`または`SHOW PROCESSLIST`クエリーによって報告されたものと種類が同じ情報です。

`mysql_free_result()`を使って結果セットを解放しなければなりません。

戻り値

成功するためのMYSQL_RES結果セット。NULLエラーが起こった場合。

エラー

- CR_COMMANDS_OUT_OF_SYNC

コマンドが妥当でないオーダーで実行されました。

- CR_SERVER_GONE_ERROR

MySQLサーバが立ち去りました。

- CR_SERVER_LOST

サーバへの接続がクエリー中に失われました。

- CR_UNKNOWN_ERROR

未知のエラーが起こりました。

mysql_list_tables()

MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)

説明

それは、`wild`パラメータによって規定された単純なレギュラー表現にマッチする現在のデータベース中のテーブル名によって構成された結果セットを返します。`wild`には、ワイルドカード文字`'%'`または`'_'`を含めるか、これをすべてのデータベースにマッチするNULLポインターにすることが許されます。`mysql_list_tables()`の呼び出しは、クエリー`SHOW tables [LIKE wild]`の実行と同じです。

`mysql_free_result()`を使って結果セットを解放しなければなりません。

戻り値

成功するためのMYSQL_RES結果セット。NULLエラーが起こった場合。

エラー

- CR_COMMANDS_OUT_OF_SYNC

コマンドが妥当でないオーダーで実行されました。

- CR_SERVER_GONE_ERROR

MySQLサーバが立ち去りました。

- CR_SERVER_LOST

サーバへの接続がクエリー中に失われました。

- CR_UNKNOWN_ERROR

未知のエラーが起きました。

mysql_more_results()

```
my_bool mysql_more_results(MYSQL *mysql)
```

説明

この機能は、シングルステートメントストリングとして規定された複数のステートメントを実行するとき、もしくは複数の結果セットを戻すCALLステートメントを実行するとき使用します。

現在実行したステートメントより、多くの結果が存在している場合、[mysql_more_results\(\)](#)は真となります。この場合、アプリケーションは[mysql_next_result\(\)](#)を呼び出して、結果をフェッチしなければなりません。

戻り値

以上の結果が存在する場合は、[TRUE](#)。上の結果が存在しない場合は、[FALSE](#) (0)。

殆どの場合、もっと多くの結果が存在しないかテストし、存在している場合には、これらの復元を開始する代わりに、[mysql_next_result\(\)](#)を呼び出すことができます。

[マルチプルステートメントを実行するC APIハンドリング](#) と [mysql_next_result\(\)](#)を参照して下さい。

エラー

なし。

mysql_next_result()

```
int mysql_next_result(MYSQL *mysql)
```

説明

この機能は、シングルステートメントストリングとして規定された複数のステートメントを実行するとき、もしくは複数の結果セットを戻すCALLステートメントを実行するとき使用します。

もっと多くの結果が存在している場合、[mysql_next_result\(\)](#)は次のステートメントを読み取り、そのステータスをアプリケーションに返送します。

それが結果セットを戻したクエリーである場合、[mysql_next_result\(\)](#)を呼び出す前に、先行ステートメントのために、[mysql_free_result\(\)](#)を呼び出さなければなりません。

[mysql_next_result\(\)](#)を呼び出した後の接続は、次のステートメントのために、[mysql_real_query\(\)](#)または[mysql_query\(\)](#)を呼び出したかのような状態になります。これは、[mysql_store_result\(\)](#)、[mysql_warning_count\(\)](#)、[mysql_affected_rows\(\)](#)等を呼び出すことができることを意味します。

[mysql_next_result\(\)](#)がエラーを戻す場合、他のステートメントは実行されず、フェッチすべき更なる結果も存在しません。

ユーザのプログラムが [CALL](#) SQL ステートメントで保存されたプロシージャを実行する場合、必ず [CLIENT_MULTI_RESULTS](#)フラグを明確に設定するか、[mysql_real_connect\(\)](#)に通信するとき [CLIENT_MULTI_STATEMENTS](#) にセットすることで必然的にフラグが設定されるようにしてください。これは、各CALLは、手順の中で実行されたステートメントによって戻すべき結果に加え、呼び出しのステータスを示す結果を戻すことに起因して起こります。更に、[CALL](#)は複数の結果を戻すことができるので、これらの結果を、[mysql_next_result\(\)](#)を呼び出して、更なる結果がないか査定するループを使って処理すべきです。

[mysql_next_result\(\)](#)を使用する方法を示す例については、[マルチプルステートメントを実行するC APIハンドリング](#)をご参照ください。

戻り値

戻り値	摘要
0	成功し且つ結果が更に存在する。
-1	成功し且つ結果が更に存在しない。
>0	エラーが発生した。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。例えば、前の結果セットに対して、[mysql_use_result\(\)](#)を呼び出さなかった場合。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

[mysql_num_fields\(\)](#)

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

代わりに [MYSQL*](#) アーギュメントを渡す場合、[unsigned int mysql_field_count\(MYSQL *mysql\)](#)を使用してください。

説明

それは、結果セット中のカラムの数を返します。

カラムの数をポインターから結果セットもしくは接続ハンドルに取り出すことができることに注目してください。接続ハンドルは、[mysql_store_result\(\)](#)もしくは[mysql_use_result\(\)](#)がNULLを返したが(しかし、結果セットポインターがない)場合に使われるでしょう。この場合、[mysql_field_count\(\)](#)を呼び出して、[mysql_store_result\(\)](#)が空でない結果を生成し終えているべきであったか否かを査定することができます。これによって、クライアントプログラムが、クエリーがSELECT(またはSELECTのような)ステートメントであったか否かを知ることなく、適正なアクションをとることが許されます。ここの例はこれを実行できる方法を示すものです。

詳しくは [なぜmysql_store_result\(\)時々返すNULLの後mysql_query\(\) 成功を返す](#) を参照してください。

戻り値

結果セット中にあるカラムの数を表す無署名の整数。

エラー

なし。

例

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
```

```

// error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
}

```

(自分のクエリーは結果セットを戻すべきであったと知ったときの)代案は、`mysql_errno(&mysql)`の呼び出しを、`mysql_field_count(&mysql)`が0であったか否かのチェックに変えて実行することです。これは何かが故障した場合に起こります。

mysql_num_rows()

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

説明

それは、結果セット中の行数を戻します。

`mysql_num_rows()`の使用法は、結果セットを戻すために`mysql_store_result()`を使用するか、`mysql_use_result()`を使用するかによって異なります。`mysql_store_result()`を使うと、`mysql_num_rows()`を直ちに呼び出すことができません。`mysql_use_result()`を使うと、結果セット中の全ての列が復元されるまで、`mysql_num_rows()`は正しい値を戻しません。

`mysql_num_rows()`はSELECTを含む結果セットを戻すステートメントと一緒に使用するよう意図されています。INSERT、UPDATEまたはDELETEのようなステートメントのために、影響された列の数を`mysql_affected_rows()`を使って取得することができます。

戻り値

結果セット中の行の数。

エラー

なし。

mysql_options()

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

説明

それは、余分な接続オプションをセットし、接続のための行動に影響を与えるために使うことができます。幾つかのオプションをセットするために、この機能を何回でも呼び出すことができます。

`mysql_options()`は`mysql_init()`の後および`mysql_connect()`もしくは`mysql_real_connect()`の前に呼び出されるべきです。

`option` 引数は、セットしたいオプションです。`arg` 引数はそのオプションのための値です。オプションが整数である場合、`arg` はその整数の値を指し示すべきです。

可能なオプション値：

オプション	アーギュメント型	ファンクション
<code>MYSQL_INIT_COMMAND</code>	<code>char *</code>	サーバに接続する際実行するコマンド。再接続すると、自動的に再実行されます。
<code>MYSQL_OPT_COMPRESS</code>	使用しない	圧縮されたクライアント/サーバプロトコルを使用。
<code>MYSQL_OPT_CONNECT_TIMEOUT</code>	<code>unsigned int *</code>	接続タイムアウトまで秒読み。
<code>MYSQL_OPT_GUESS_CONNECTION</code>	使用しない	<code>libmysqld</code> に対してリンクされたアプリケーションに対して、これは、ライブラリが埋込サーバを使うべきか、遠隔サーバを使うべきか憶測することを許します。「Guess」は、ホスト名がセットされたが、これが <code>localhost</code> でない場合、遠隔サーバを使うことを意味します。この行動はデフォルトです。 <code>MYSQL_OPT_USE_EMBEDDED_CONNECTION</code> および <code>MYSQL_OPT_USE_REMOTE_CONNECTION</code> はそれを無効にするのに使うことができます。このオプションは、 <code>libmysqlclient</code> に対してリンクされたアプリケーションの場合、無視されます。
<code>MYSQL_OPT_LOCAL_INFILE</code>	ユニットに対するオプションポインター	ポインターが附与されない場合もしくは <code>unsigned int != 0</code> を指さす場合、コマンド <code>LOAD LOCAL INFILE</code> は有効です。
<code>MYSQL_OPT_NAMED_PIPE</code>	使用しない	NT上のMySQLサーバに接続する指定パイプを使用してください。
<code>MYSQL_OPT_PROTOCOL</code>	<code>unsigned int *</code>	使用するプロトコルの種類。 <code>mysql.h</code> で定義され <code>mysql_protocol_type</code> のエナumber値のうちの1つであるべきです。
<code>MYSQL_OPT_READ_TIMEOUT</code>	<code>unsigned int *</code>	サーバからの読み込みタイムアウト (TCP/IPコネクションまたはMySQL 5.1.12以前のWindowsでのみ使用できます)。このオプションを使用すると、最後の接続を10分のTCP/IP <code>Close_Wait_Timeout</code> 値より早期に検出することができます。
<code>MYSQL_OPT_RECONNECT</code>	<code>my_bool *</code>	は、接続が失われたと判明した場合、は、サーバへの自動再接続を有効か無効にします。再接続は初期設定によってオフになっていますが、このオプションは再接続行動を明確にセットする方法を提供します。
<code>MYSQL_OPT_SET_CLIENT_IP</code>	<code>char *</code>	<code>libmysqld</code> に対して (認証サポートを使って編集された <code>libmysqld</code> 利用して) リンクされたアプリケーションの場合、これは、ユーザーは認証を目的に、(ストリングとして規定された) 規定IPアドレスから接続したと思われることを意味します。このオプションは、 <code>libmysqlclient</code> に対してリンクされたアプリケーションの場合、無視されます。
<code>MYSQL_OPT_SSL_VERIFY_SERVER_CERT</code>	<code>my_bool *</code>	は、サーバに接続するとき使用したホスト名に対する証明の中に登録されたサーバの

		固有名の認証を有効化したり、無効化したりします。ミスマッチがあった場合、接続は拒絶されます。この機能は、man-in-the-middleアタックを防止するために使用することができます。認証は初期設定では無効になっています。MySQL 5.1.11に追加。
<code>MYSQL_OPT_USE_EMBEDDED_CONNECTION</code>	使用しない	<code>libmysqld</code> に対してリンクされたアプリケーションの場合、これは、接続に埋込サーバを使用することを強要します。このオプションは、 <code>libmysqlclient</code> に対してリンクされたアプリケーションの場合、無視されます。
<code>MYSQL_OPT_USE_EMBEDDED_CONNECTION</code>	使用しない	<code>libmysqld</code> に対してリンクされたアプリケーションの場合、これは、接続に遠隔サーバを使用することを強要します。このオプションは、 <code>libmysqlclient</code> に対してリンクされたアプリケーションの場合、無視されます。
<code>MYSQL_OPT_USE_RESULT</code>	使用しない	このオプションは使用されません。
<code>MYSQL_OPT_WRITE_TIMEOUT</code>	<code>unsigned int *</code>	(現在、TCP/IP接続上のWindowsでのみ有効な)サーバへの書き込みに対するタイムアウト。
<code>MYSQL_READ_DEFAULT_FILE</code>	<code>char *</code>	<code>my.cnf</code> から行う代わりに、指定オプションファイルからオプションを読み取ってください。
<code>MYSQL_READ_DEFAULT_GROUP</code>	<code>char *</code>	指定グループ、 <code>my.cnf</code> または <code>MYSQL_READ_DEFAULT_FILE</code> を使って規定したファイルからオプションを読み取ってください。
<code>MYSQL_REPORT_DATA_TRUNCATION</code>	<code>my_bool *</code>	<code>MYSQL_BIND.error</code> を経由して行う準備されたステートメントに対するデータ・トラランケーションエラーの報告を有効化したり、無効化します。(初期設定:無効化。)
<code>MYSQL_SECURE_AUTH</code>	<code>my_bool*</code>	MySQL 4.1.1およびそれより新しいバージョンのMySQLに使用されているパスワードバッシングをサポートしていないサーバへの接続の可否。
<code>MYSQL_SET_CHARSET_DIR</code>	<code>char*</code>	文字セットの定義ファイルを含むディレクトリに対するパスネーム。
<code>MYSQL_SET_CHARSET_NAME</code>	<code>char*</code>	デフォルト文字セットとして使用すべき文字セットの名称。
<code>MYSQL_SHARED_MEMORY_BASE_NAME</code>	<code>char*</code>	サーバとのコミュニケーションに使用すべき共有メモリオブジェクトの名称。接続したい <code>mysqld</code> サーバのために使用したオプション <code>--shared-memory-base-name</code> と同じにすべきです。

`client`グループは、`MYSQL_READ_DEFAULT_FILE`または`MYSQL_READ_DEFAULT_GROUP`を使う場合、常に読み込まれることにご注目ください。

オプションファイル内の指定されたグループには、以下のオプションを含めることができます：

オプション	摘要
<code>connect-timeout</code>	接続タイムアウト(秒)。Linux上で、このタイムアウトはサーバから最初の答えを待つためにも使います。

<code>compress</code>	圧縮したクライアント/サーバプロトコルを使ってください。
<code>database</code>	接続コマンドの中に規定されなかった場合、このデータベースに接続してください。
<code>debug</code>	デバッグオプション。
<code>disable-local-infile</code>	<code>LOAD DATA LOCAL</code> の使用が無効。
<code>host</code>	デフォルトホストネーム。
<code>init-command</code>	MySQLサーバに接続する時実行すべきコマンド。再接続するとき、自動的に再実行されます。
<code>interactive-timeout</code>	<code>CLIENT_INTERACTIVE</code> を <code>mysql_real_connect()</code> に規定すると同じように。 <code>mysql_real_connect()</code> 参照。
<code>local-infile[=(0 1)]</code>	引数がないか、引数!= 0の場合、 <code>LOAD DATA LOCAL</code> の使用を有効にする。
<code>max_allowed_packet</code>	サーバから読み取ることができるパケットの最大サイズ。
<code>multi-results</code>	マルチステートメントまたは記憶された手順の実行からマルチ結果セットを取得することを許す。
<code>multi-statements</code>	クライアントが、マルチステートメントを(';'によって仕切られた)シングルスティングの中に送ることを許す。
<code>password</code>	デフォルトパスワード。
<code>pipe</code>	NT上のMySQLサーバに接続するため、指定パイプを使ってください。
<code>protocol={TCP SOCKET PIPE MEMORY}</code>	サーバに接続するとき使用すべきプロトコル。
<code>port</code>	デフォルトポートナンバー。
<code>return-found-rows</code>	<code>UPDATE</code> を使うとき、更新された列の代わりに見つかった列を戻すよう <code>mysql_info()</code> に教えてください。
<code>shared-memory-base-name=名称</code>	サーバに接続するとき使用すべき共有メモリー(デフォルトは"MYSQL")。
<code>socket</code>	デフォルトソケットファイル。
<code>user</code>	デフォルトユーザー。

`timeout`は`connect-timeout`に置き換えられましたが、`timeout`は、下位互換性について、まだMySQL 5.1.15-betaによってサポートされている点にご注目ください。

オプションファイルの明細については、[オプションファイルの使用](#)をご参照ください。

戻り値

成功のためのゼロ。未知のオプションを使う場合、非ゼロ。

例

```

MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}

```

このコードは、クライアントに圧縮されたクライアント/サーバプロトコルを使い、`my.cnf`ファイル中にある`odbc`セクションから追加オプションを読み取ることを求めるものです。

mysql_ping()

```
int mysql_ping(MYSQL *mysql)
```

説明

サーバへの接続が作動しているかどうか調べる。接続がダウンしたら、自動再接続が試みられます。

長い間働いていないままのクライアントサーバはこの機能を使って、接続が閉じたかどうか調べ、必要に応じて再接続することができます。

戻り値

サーバへの接続が作動しているかどうか調べるゼロ。エラーが起こった場合ゼロ以外。戻り値がゼロ以外であることは、MySQLサーバがダウンしたか、接続がネットワーク問題を含むその他の原因で破壊された恐れがあることを示します。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起こりました。

mysql_query()

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

説明

ゼロで終わる文字列`stmt_str`を指摘するSQLステートメントを実行してください。通常、文字列はシングルSQLステートメントによって構成されているので、セミicolonで終わる`;`や`\g`をステートメントに加えるべきではありません。マルチステートメントの実行が有効になっている場合、文字列には、セミicolonで仕切られた幾つかのステートメントを含めることができます。[マルチプルステートメントを実行するC APIハンドリング](#)を参照してください。

バイナリーデータを含むステートメントには、[mysql_query\(\)](#)を使うことはできません。この代わりに、[mysql_real_query\(\)](#)を使わなければなりません。(バイナリーデータには、`'\0'`キャラクターを含めることができます。その[mysql_query\(\)](#)はステートメント文字列の終わりであると解釈されます。

ステートメントが結果セットを返すべきであるかどうか知りたい場合、[mysql_field_count\(\)](#)を使ってこれをチェックすることができます。[mysql_field_count\(\)](#)を参照してください。

戻り値

コマンドの附与が成功した場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char *db,
unsigned int port, const char *unix_socket, unsigned long client_flag)
```

説明

`mysql_real_connect()`は`host`上で運転されるMySQLデータベースエンジンに接続を確立しようと努めます。`mysql_real_connect()`は、有効なMySQL接続ハンドル構造を必要とする他のAPI機能が実行可能になる前に、うまく完了されなければなりません。

パラメータは次のように規定されます：

- 最初のパラメータは既存のMySQL構造にすべきです。`mysql_real_connect()`を呼び出す前に、`mysql_init()`を呼び出してMySQL構造を初期化しなければなりません。`mysql_options()`を呼び出して、多くの接続オプションを変更することができます。`mysql_options()`を参照してください。
- (この場合)`host`の値をホスト名かIPアドレスにすることができます。`host`がNULLまたはストリング"`localhost`"である場合、ローカルホストへの接続と仮定します:Windowsに対して、サーバが共有メモリー接続を有効にしている場合、クライアントはメモリー接続を使って接続します。さもなければ、TCP/IPが使われます。Unixの場合、クライアントはUnix専用のソケットファイルを使って接続します。ローカル接続の場合、`mysql_options()`に対してMySQL_OPT_PROTOCOLオプションもしくはMySQL_OPT_NAMED_PIPEオプションと一緒に使う接続のタイプに影響を及ぼすこともできます。接続のタイプはサーバによってサポートされなければなりません。Windows上の"."の`host`値に対して、クライアントは指定パイプを使って接続します。指定パイプによる接続が有効化されていないと、エラーが発生します。
- `user`パラメータには、ユーザーのMySQLログインIDが含まれています。`user`がNULLまたは空ストリング""である場合、現在のユーザーを想定します。Unixでは、これは現在のログイン名です。Windows ODBCでは、現在のユーザー名を明確に規定しなければなりません。24章MySQL コネクタのMyODBCセクションをご参照ください。
- `passwd`パラメータには、`user`に対するパスワードが含まれています。`passwd`がNULLである場合、ブランク(空の)パスワードフィールドを持つユーザーのために行った`user`テーブルへの入力だけがマッチするかチェックされます。これは、データベース管理者がMySQL特権システムを、ユーザーがパスワードを規定したか否かによって、異なった特権を取得するようにセットすることを可能にします。

注:`mysql_real_connect()`を呼び出す前に、パスワードを暗号化しようとしてはなりません。パスワードの暗号化はクライアントAPIによって自動的に取り扱われます。

- `db`はデータベース名です。`db`がNULLでない場合、接続はこの値にデフォルトデータベースを設定します。
- `port`が0でない場合、値はTCP/IP接続のためのポートナンバーとして使用されます。`host`パラメータが接続のタイプを決めることにご注目ください。
- `unix_socket`がNULLでない場合、ストリングは使用すべきソケットまたは指定パイプを規定します。`host`パラメータが接続のタイプを決めることにご注目ください。
- `client_flag`の値は通常0ですが、次のフラグを組み合わせると、ある種の機能を有効にすることができます:

フラグ名	フラグの説明
CLIENT_COMPRESS	圧縮プロトコルの使用。
CLIENT_FOUND_ROWS	変更した行の数でなく、検出され(マッチした)行の数を戻す。
CLIENT_IGNORE_SPACE	機能名の後にスペースを取ることを許します。全ての機能名を予約されたワードにします。
CLIENT_INTERACTIVE	接続を閉じる前に(<code>wait_timeout</code> 秒の代わりに)、 <code>interactive_timeout</code> 秒間停止します。クライアントのセッション <code>wait_timeout</code> 変数は、セッション <code>interactive_timeout</code> 変数の値にセットされます。

CLIENT_LOCAL_FILES	は LOAD DATA LOCAL ハンドリングを有効化します。
CLIENT_MULTI_RESULTS	サーバに、クライアントはマルチ結果セットをマルチステートメントの実効から処理できることを通知します。 CLIENT_MULTI_STATEMENTS がセットされる場合、自動的にセットされます。フラグの詳細については、このテーブルの後にあるノートをご参照ください。
CLIENT_MULTI_STATEMENTS	サーバに、クライアントは、(『』によって隔離された)シングルスtringの中にマルチステートメントを送ることができることを通知します。このフラグがセットされないと、マルチステートメントの実効は不能になります。フラグの詳細については、このテーブルの後にあるノートをご参照ください。
CLIENT_NO_SCHEMA	db_name.tbl_name.col_name 構文の使用を許しません。これはODBCのためのものです。幾つかODBCプログラム中でバグを捕えるのに役立つあの構文を使うと、それが、パーサーがエラーを生成する原因を引き起こします。
CLIENT_ODBC	クライアントはODBCクライアントです。これは mysqlId をODBCにもっと優しくなるように変更します。
CLIENT_SSL	SSL(暗号化されたプロトコル)を使います。このオプションはアプリケーションプログラムによってセットされるべきではありません。これはクライアントライブラリ内部でセットされます。その代わりに、 mysql_real_connect() を呼び出す前に、 mysql_ssl_set() を使ってください。

プログラムが結果セットを生成する記憶された手順を実行するために、[CALLSQL](#)ステートメントを使用する場合、あなたは、[mysql_real_connect\(\)](#)を呼び出すとき、[CLIENT_MULTI_STATEMENTS](#)をセットすることによって、[CLIENT_MULTI_RESULTS](#)フラグを明示的もしくは黙示的にセットしなければなりません。このような記憶された手順は各々、複数の結果を生成するので、これを実行しなければなりません。手順の中で実行された命令文によって戻された結果の組並びに呼び出しのステータスを示す結果。

[CLIENT_MULTI_STATEMENTS](#)または[CLIENT_MULTI_RESULTS](#)を有効にした場合、[mysql_next_result\(\)](#)を呼び出して、更に結果があるか否かを査定するループを使用することによって、[mysql_query\(\)](#)または[mysql_real_query\(\)](#)に対する全ての呼び出しの結果を処理すべきです。(例については、[マルチプルステートメントを実行するC APIハンドリング](#)をご覧ください。)

幾つかのパラメータを[mysql_real_connect\(\)](#)の呼び出文中の明確な値からでなく、オプションファイルかた取らせせることが可能です。これを行うため、[mysql_real_connect\(\)](#)を呼び出す前に、[MYSQL_READ_DEFAULT_FILE](#)または[MYSQL_READ_DEFAULT_GROUP](#)と一緒に、[mysql_options\(\)](#)を呼び出してください。その後、各パラメータがオプションファイルから読み取られるように、[mysql_real_connect\(\)](#)の呼び出しに、「no-value」値を規定してください。

- [host](#)に対して、[NULL](#)または空のstring(″″)の値を規定してください。
- [user](#)に対して、[NULL](#)または空のstringの値を規定してください。
- [passwd](#)に対して、[NULL](#)の値を規定してください。(パスワードに対して、[mysql_real_connect\(\)](#)コール中の空stringの値がオプションファイルに入るのを禁止することはできません。なぜなら、空のstringは、MySQLアカウントには空のパスワードが含まれていなければならないことを明示しているからです。)
- [db](#)に対して、[NULL](#)または空のstringの値を規定してください。
- [port](#)に対して、ゼロの値を規定してください。
- [unix_socket](#)に対して、[NULL](#)の値を規定してください。

パラメータのためのオプションファイル中に値が見つからない場合、このセクションの始めの部分で説明したように、そのデフォルト値が使われます。

戻り値

接続が成功した場合[MYSQL*](#)接続ハンドル。接続が不成功であった場合 [NULL](#)。接続が成功した場合、戻り値は最初のパラメータの値と同じになります。

エラー

- [CR_CONN_HOST_ERROR](#)

MySQLサーバとの接続が失敗した。

- [CR_CONNECTION_ERROR](#)

ローカルMySQLサーバとの接続が失敗した。

- [CR_IPSOCK_ERROR](#)

IPソケットの生成に失敗した。

- [CR_OUT_OF_MEMORY](#)

メモリ不足。

- [CR_SOCKET_CREATE_ERROR](#)

Unixソケットの生成に失敗した。

- [CR_UNKNOWN_HOST](#)

ホスト名のIP アドレスの検出に失敗した。

- [CR_VERSION_ERROR](#)

異なったバージョンのクライアントライブラリを持つサーバに接続しようと試みたことから発生したプロトコルミスマッチ。これは、`--old-protocol` オプションを使って立ち上げなかった新しいサーバに接続する非常に古いクライアントライブラリを使った場合に起こることがあります。

- [CR_NAMEDPIPEOPEN_ERROR](#)

Windows上に指定パイプを生成することに失敗した。

- [CR_NAMEDPIPEWAIT_ERROR](#)

Windows上に指定パイプを待つことに失敗した。

- [CR_NAMEDPIPESETSTATE_ERROR](#)

Windows上にパイプハンドラーを取得することに失敗した。

- [CR_SERVER_LOST](#)

`connect_timeout`が>0で、サーバに接続するまでに`connect_timeout`秒未満の時間がかかるか、`init-command`を実行している間にサーバが死亡した場合。

例

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql, MYSQL_READ_DEFAULT_GROUP, "your_prog_name");
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, NULL, 0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}
```

`mysql_options()`を使用すると、MySQLライブラリは、誰かが非標準的な方法でMySQLをセットした場合でも、プログラムが作動することを確認する`my.cnf`ファイル中の`[client]`セクションおよび`[your_prog_name]`セクションを読み取ります。

接続と同時に、`mysql_real_connect()`は、`reconnect` フラグ(MYSQL構造の一部)を、バージョンが5.0.3より古いAPI中にある1の値または0の値にセットすることにご注目ください。このフラグに対する1の値は、ステートメントが失われた接続のために実効できない場合、実効を諦める前にサーバに再び接続しようとすることを示しま

す。[MYSQL_OPT_RECONNECT](#) オプションを[mysql_options\(\)](#) に対して、再接続行動を制御するのに使用することができます。

mysql_real_escape_string()

`unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned long length)`

[mysql](#) は有効なオープン接続でなければならないことにご注目ください。回避はサーバによって使用中の文字セットに依存しているのでこれが必要です。

説明

この機能は、SQL ステートメントの中に使うことができる法定 SQL スtring を作るために使われます。[文字列](#)を参照してください。

[from](#) 中の String は、現在の接続文字セットを考慮して、エスケープした SQL String に対してコード化されます。結果は [to](#) の中に置かれ、最後の部位にゼロバイトが付加されます。コード化された文字は [NUL](#) (ASCII 0)、['\n'](#)、['\r'](#)、['\'](#)、['"'](#) および Control-Z ([リテラル値](#) 参照) です。(厳密に言えば、MySQL はバックスラッシュおよびクエリー中に String を引用する引用文字を除外することだけを要求します。この機能はそれらをログファイルの中で読み取るを容易にするため、他の文字を引用します。)

[from](#) によって指定された String は、長さが [length](#) バイトでなければなりません。[to](#) バッファを割り当て、長さが少なくとも [length*2+1](#) バイトにしなければなりません。(最悪の場合、各文字は 2 バイトを使用するように、コード化する必要があるかもしれないので、端末ゼロバイトのための余裕が必要です。) [mysql_real_escape_string\(\)](#) が戻るとき、[to](#) の中身はゼロで終わる String です。戻り値は、ゼロで終わる文字を含まないコード化された String の長さに等しい大きさを持っています。

接続の文字セットに変更を施す必要がある場合、[SET NAMES](#) (または [SET CHARACTER SET](#) ステートメントではなく、[mysql_set_character_set\(\)](#) 機能を使うべきです。[mysql_set_character_set\(\)](#) は、[SET NAMES](#) のように働きますが、[mysql_real_escape_string\(\)](#) が使用した文字セットせつとも影響を及ぼします。この場合、[SET NAMES](#) には影響を及ぼしません。

例

```
char query[1000], *end;

end = strmov(query, "INSERT INTO test_table values('");
*end++ = '\';
end += mysql_real_escape_string(&mysql, end, "What's this", 11);
*end++ = '\';
*end++ = ',';
*end++ = '\';
end += mysql_real_escape_string(&mysql, end, "binary data: \0\r\n", 16);
*end++ = '\';
*end++ = ')';

if (mysql_real_query(&mysql, query, (unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
            mysql_error(&mysql));
}
```

サンプルの中に使用した [strmov\(\)](#) 機能は、[mysqlclient](#) ライブラリの中に含まれています。当該機能は [strcpy\(\)](#) と同じような働きをしますが、ポインターを最初のパラメータの終わりのゼロに戻します。

戻り値

[to](#) の中に置かれた、ゼロで終わる文字を含まない値の長さ。

エラー

なし。

mysql_real_query()

`int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)`

説明

`stmt_str`によって指定されたSQLステートメントを実行します。それは、`length`バイトの長さ持つべきです。通常、ストリングをシングルSQLステートメントで構成すべきですが、ステートメントには、最後の部分にセミコロン(';')または`\g`を追加すべきではありません。マルチステートメントの実行が有効になっている場合、ストリングには、セミコロンで仕切られた幾つかのステートメントを含めることができます。[マルチプルステートメントを実行するC APIハンドリング](#)を参照してください。

バイナリーデータを含むステートメントには、`mysql_query()`を使うことはできません。これの代わりに、`mysql_real_query()`を使わなければなりません。(バイナリーデータには、'\0'キャラクターを含めることができます。その`mysql_query()`はステートメントストリングの終わりであると解釈されます。)さらに、`mysql_real_query()`は`mysql_query()`より高速です。なぜなら、それはステートメントストリング上に`strlen()`を呼び出さないからです。

ステートメントが結果セットを返すべきであるかどうか知りたい場合、`mysql_field_count()`を使ってこれをチェックすることができます。[mysql_field_count\(\)](#)を参照してください。

戻り値

コマンドの附与が成功した場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

mysql_refresh()

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

説明

この機能はテーブルあるいはキャッシュをクリアするか、サーバ情報を模写します。接続されたユーザーは`RELOAD`特権を持っていないければなりません。

`options`引数は以下の値からなるビットマスクです。値複の値と一緒にORして、一回の呼び出しで複数のオペレーションを実施することができます。

- `REFRESH_GRANT`

`FLUSH PRIVILEGES`のような付与されたテーブルをリフレッシュしてください。

- `REFRESH_LOG`

`FLUSH LOGS`のようなログをフラッシュしてください。

- `REFRESH_TABLES`

`FLUSH TABLES`のようなテーブルキャッシュをフラッシュしてください。

- `REFRESH_HOSTS`

[FLUSH HOSTS](#)のようなホストキャッシュをフラッシュしてください。

- [REFRESH_STATUS](#)

[FLUSH STATUS](#)のようなステータス変数をリセットしてください。

- [REFRESH_THREADS](#)

スレッドキャッシュをフラッシュしてください。

- [REFRESH_SLAVE](#)

スレイベ模写サーバー上で、マスターサーバ情報をリセットし、[RESET SLAVE](#)のようなスレイベを再起動してください。

- [REFRESH_MASTER](#)

マスター模写サーバ上で、バイナリーログインデックスを除去して、[RESET MASTER](#)のようなインデックスファイルを切り捨ててください。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起こりました。

mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

説明

MySQLサーバに供与されたテーブルを再ロードするよう依頼してください。接続されたユーザーは[RELOAD](#)特権を持っていないかもしれません。

この機能はけなされます。代わりに、[mysql_query\(\)](#)を使って、[SQLFLUSH PRIVILEGES](#)ステートメントを発行することは望ましいことです。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

`mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

説明

現取引のロールバック。

この機能のアクションは`completion_type`システム変数の値の適用を条件としています。特に、`completion_type`の値が2である場合、取引を終えた後に、サーバはリリースを行って、クライアント接続を閉じます。クライアントプログラムはクライアント側から接続を閉じるために、`mysql_close()`を呼び出すべきです。

戻り値

成功している場合ゼロ。エラーが起こった場合、ゼロ以外。

エラー

なし。

`mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

説明

クエリー結果セット中の任意の横列に列カーソルをセットしてください。`offset`値は、`mysql_row_tell()`または`mysql_row_seek()`から戻った値であるべき列オフセットです。この値は列ナンバーではありません。結果セットの中で、列を番号別に探索する場合、代わりに`mysql_data_seek()`を使ってください。

この機能は、結果セット構造にクエリーの全結果を含めることを求めるので、`mysql_row_seek()`は、`mysql_use_result()`と一緒になく、`mysql_store_result()`と併用する場合に限り使用することができます。

戻り値

列カーソルの前の値。この値を`mysql_row_seek()`に対する次の呼び出しに渡すことができます。

エラー

なし。

`mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

説明

それは、最後の`mysql_fetch_field()`に使用した列カーソルの現位置をを戻します。この値を`mysql_row_seek()`に対する引数として使うことができます。

`mysql_row_tell()`を、`mysql_use_result()`の後でなく、`mysql_store_result()`の後にだけ使うべきです。

戻り値

列カーソルの現在のオフセット。

エラー

なし。

`mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

説明

`db`によって規定されたデータベースを、`mysql`によって規定された接続上の(現)デフォルトデータベースになるように仕向けてください。次のクエリーの中では、このデータベースは明確なデータベース規定者を含まないテーブルリファレンスのためのデフォルトとなります。

接続されたユーザーが認証されることができない限り、`mysql_select_db()`は失敗します。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

`mysql_set_character_set()`

```
int mysql_set_character_set(MYSQL *mysql, char *csname)
```

説明

この機能は現在の接続のためのデフォルト文字セットをセットするのに使います。ストリング`csname`は有効な文字セット名を規定します。接続の照合は文字セットのデフォルト照合となります。この機能は`SET NAMES`ステートメントと同じ働きをしますが、`mysql->charset`の値もセットし、これによって、`mysql_real_escape_string()`によって使用されている文字セットに影響を及ぼします。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

例

```
MYSQL mysql;
mysql_init(&mysql);
```

```

if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
        mysql_character_set_name(&mysql));
}

```

mysql_set_local_infile_default()

```

void
mysql_set_local_infile_default(MYSQL *mysql);

```

説明

LOAD LOCAL DATA INFILE ハンドラーコールバック機能を、Cクライアントライブラリによって内部で使用されているデフォルトにセットしてください。[mysql_set_local_infile_handler\(\)](#) が呼び出されなかったか、これがコールバックの各々に対して有効な機能を提供しない場合、ライブラリはこの機能を自動的に呼び出します。

MySQL 4.1.2. に [mysql_set_local_infile_default\(\)](#) 機能が追加されました。

戻り値

なし。

エラー

なし。

mysql_set_local_infile_handler()

```

void
mysql_set_local_infile_handler(MYSQL *mysql,
    int (*local_infile_init)(void **, const char *, void *),
    int (*local_infile_read)(void *, char *, unsigned int),
    void (*local_infile_end)(void *),
    int (*local_infile_error)(void *, char *, unsigned int),
    void *userdata);

```

説明

この機能は **LOAD DATA LOCAL INFILE** ステートメントの実行中に使用すべきコールバックをインストールします。それは、アプリケーションプログラムが(クライアント側の)ローカルデータファイルの読取りを制御することを可能にします。引数は、接続ハンドラー、コールバック機能に対するポインタの組並びにコールバックが情報を共有するのに使用出来るデータエリアに対するポインタです。

[mysql_set_local_infile_handler\(\)](#) を使うために、以下のコールバック機能を書き込まなければなりません:

```

int
local_infile_init(void **ptr, const char *filename, void *userdata);

```

初期化機能。これは、必要な設定を実行し、データファイルを開き、データ構造等を割り当てると同時に呼び出されます。最初の `void**` 引数はポインタ別のもとなります。コールバックの各々に渡す値に対するポインタ(即ち、`*ptr`)を(`void*`と同じように)セットすることができます。コールバックは、このポイントされた値を状態情報を維持するのに使用することができます。`userdata` 引数は [mysql_set_local_infile_handler\(\)](#) に渡したと同じ値のもです。

初期化機能は成功に対してゼロ、エラーに対して非ゼロをそれぞれ戻すべきです。

```

int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);

```

データ読取り機能。これはデータファイルを読み取るために、繰り返して呼び出されます。`buf` は読み取ったデータを記憶すべきバッファを指し、`buf_len` は、コールバックが読取り、バッファに記憶することができるバイトの

最大ナンバーとなります。(それはより少ないバイトを読み取ることができますが、より多くのバイトを読み取るべきではありません。)

戻り値は読み取ったバイトの数ですが、データを読み取ることが出来なかったデータがあった時には、これが(EOFを示す)ゼロとなります。エラーがあると、ゼロ未満の値を戻します。

```
void
local_infile_end(void *ptr)
```

停止機能。これは、[local_infile_read\(\)](#)がゼロ(EOF)またはエラーを戻した後に一度呼び出されます。この機能は、[local_infile_init\(\)](#)が割り当てたメモリの割り当てを解除し、必要なその他のクリーンアップを実施します。それは初期化機能がエラーを戻す場合でも呼び出されます。

```
int
local_infile_error(void *ptr,
                  char *error_msg,
                  unsigned int error_msg_len);
```

エラー処理機能。これは、テキストエラーメッセージを受け取って、他の機能がエラーを戻す場合、ユーザーに戻すために呼び出されます。[error_msg](#)は、メッセージを書き込むべきバッファを指し、[error_msg_len](#)はそのバッファの長さとなります。メッセージはゼロで終わるストリングとして書き込まれるべきなので、メッセージの長さはせいぜい[error_msg_len](#)-1 バイトとなります。

戻り値がエラーナンバーです。

大抵、その他のコールバックは[ptr](#)を指さすデータ構造中にエラーメッセージを記憶するので、[local_infile_error\(\)](#)はメッセージをそこから[error_msg](#)の中にコピーすることができます。

[mysql_set_local_infile_handler\(\)](#)をあなたのCコードの中に呼び出し、ポインタをコールバック機能に渡した後、[LOAD DATA LOCAL INFILE](#)ステートメントを、(例えば、[mysql_query\(\)](#)を使って)発行することができます。).クライアントライブラリは自動的にコールバックを呼び出します。[LOAD DATA LOCAL INFILE](#)中に規定されたファイル名は、第2パラメータとして[local_infile_init\(\)](#)コールバックに第2パラメータとして渡されます。

MySQL 4.1.2.に[mysql_set_local_infile_handler\(\)](#)機能が追加されました。

戻り値

なし。

エラー

なし。

mysql_set_server_option()

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

説明

接続の為のオプションを有効にしたり無効にしたりします。[option](#)に以下の値を持たせることができます。

MYSQL_OPTION_MULTI_STATEMENTS_ON	マルチステートメントサポートを有効化します。
MYSQL_OPTION_MULTI_STATEMENTS_OFF	マルチステートメントサポートを無効化します。

マルチ・ステートメントサポートを有効化する場合、呼び出し結果を、[mysql_next_result\(\)](#)を呼び出して、更なる結果が存在しないか査定するループを使って、[mysql_query\(\)](#)または[mysql_real_query\(\)](#)に複写すべきです。(例については、[マルチプルステートメントを実行するC APIハンドリング](#)をご覧ください。)

[MYSQL_OPTION_MULTI_STATEMENTS_ON](#)を使うマルチ・ステートメントの有効化は、[CLIENT_MULTI_STATEMENTS](#)フラグを[mysql_real_connect\(\)](#)に渡すことによって、それを有効化する場合と全く同じ効果を持つてはいません:[CLIENT_MULTI_STATEMENTS](#)も[CLIENT_MULTI_RESULTS](#)を有効化します。[CALL SQL](#)ステートメントをプログラムの中で使っている場合、マルチ結果サポートを有効化しなければなりません。これ

は、`MYSQL_OPTION_MULTI_STATEMENTS_ON`は、それだけで`CALL`の使用を許すには不十分であることを意味します。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `ER_UNKNOWN_COM_ERROR`

サーバが`mysql_set_server_option()`をサポートしなかった (そのサーバが4.1.1より古いかった場合)もしくはサーバがセットしようと努めたオプションをサポートしなかった。

mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

説明

データベースサーバにシャットダウンするように頼む。接続されたユーザーは`SHUTDOWN`特権を持っていなければならない。MySQL 5.1サーバは1つのタイプのシャットダウンしかサポートしていません。`shutdown_level`は`SHUTDOWN_DEFAULT`と等しくなければなりません。シャットダウンレベルの追加が、望みのレベルが選べるよう計画されています。旧バージョンの `libmysqlclient` のヘッダーや呼び出し `mysql_shutdown()` でコンパイルされたダイナミックリンク実行文 (ファイルまたはプログラム) は、旧 `libmysqlclient` のダイナミックライブラリと一緒に使用しなければいけません。

シャットダウンプロセスは [シャットダウン プロセス](#) で説明します。

戻り値

成功のためのゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

mysql_sqlstate()

```
const char *mysql_sqlstate(MYSQL *mysql)
```

説明

最近実行したSQLステートメントに対するSQLSTATEエラーを含む、ゼロで終わる文字列を返します。エラーコードは5つの文字から成り立っています。'00000'は「no error」を意味します。値はANSI SQLとODBCによって規定されています。可能な値のリストについては、[Errors, Error Codes, and Common Problems](#)をご参照ください。

`mysql_sqlstate()`が戻したMySQLに固有なエラーナンバーは、`mysql_errno()`が戻したSQLSTATE値とは異なっています。例えば、`mysql`クライアントプログラムは以下のフォーマットを使ってエラーを表示します。この場合、1146は`mysql_errno()`値で、'42S02'は対応する`mysql_sqlstate()`値です:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

すべてのMySQLエラーナンバーがSQLSTATEエラーコードにマップされるわけではありません。値'HY000'(一般エラー)がマップされていないエラーナンバー用に使われます。

戻り値

SQLSTATEエラーコードを含むゼロで終わる文字列。

もご参照ください。

`mysql_errno()`、`mysql_error()`および`mysql_stmt_sqlstate()`をご参照ください。

mysql_ssl_set()

```
int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const char *ca, const char *capath, const char *cipher)
```

説明

`mysql_ssl_set()`はSSLを使った安定した接続を確立するために使用します。それは`mysql_real_connect()`の前に呼び出されなければなりません。

`mysql_ssl_set()`はOpenSSLサポートがクライアントライブラリの中で有効でない限り、何も実行できません。

`mysql`は`mysql_init()`から戻された接続ハンドラーです。他のパラメータは次のように規定されます:

- `key`はキーファイルに対するパスネームです。
- `cert`は証明書ファイルに対するパスネームです。
- `ca`は証明証明官庁ファイルに対するパスネームです。
- `capath`はpemフォーマットの信頼されたSSL CA証明を含むディレクトリに対するパスネームです。
- `cipher`はSSL暗号化のために使用すべき許容暗号のリストです。

未使用SSLパラメータは全てNULLとして与えられます。

戻り値

この機能は常に0を返します。SSLの設定が不適當である場合、`mysql_real_connect()`は接続を試みるときに、エラーを返します。

mysql_stat()

```
const char *mysql_stat(MYSQL *mysql)
```

説明

`mysqladmin status` コマンドによって提供されたと同等な情報を含む文字ストリングを返します。これには、使用可能時間(秒)と運転スレッドのナンバー、疑問、再ロードならびにオープンテーブルが含まれています。

戻り値

サーバのステータスを述べた文字ストリング。NULLエラーが起こった場合。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

mysql_store_result()

`MYSQL_RES *mysql_store_result(MYSQL *mysql)`

説明

`mysql_query()` または `mysql_real_query()` を取り出した後、データ (SELECT、SHOW、DESCRIBE、EXPLAIN、CHECK TABLE等) をうまく復元するすべてのステートメントのために、`mysql_store_result()` または `mysql_use_result()` を呼び出すことができます。結果セットが必要となつてから、`mysql_free_result()` を呼び出さなければなりません。

他のステートメントのために、`mysql_store_result()` または `mysql_use_result()` を呼び出さなくてもよいが、全てのケースで `mysql_store_result()` を呼び出すと、被害を及ぼすか、もしくは顕著な性能の劣化を引き起こしません。`mysql_store_result()` が (後で詳しく述べる) ゼロ以外の値を戻すか否かを調べることによって、あなたはステートメントに結果セットが含まれているか否かを検出することができます。

マルチ・ステートメントサポートを有効化する場合、呼び出し結果を、`mysql_next_result()` を呼び出して、更なる結果が存在しないか査定するループを使って、`mysql_query()` または `mysql_real_query()` に複写すべきです。(例については、[マルチプルステートメントを実行するC APIハンドリング](#)をご覧ください。)

ステートメントが結果セットを返すべきかどうか知りたい場合、`mysql_field_count()` を使ってこれをチェックすることができます。`mysql_field_count()` を参照してください。

`mysql_store_result()` はクライアントに対するクエリーの全結果を読み取り、`MYSQL_RES` 構造を割り当て、この構造の中に結果を配置します。

ステートメントが結果セットを戻さない (例えば、INSERTステートメント) の場合、`mysql_store_result()` はゼロポインターを返します。

`mysql_store_result()` は結果セットが失敗した場合、ゼロポインターも返します。`mysql_error()` が空でないストリングを戻すか、`mysql_errno()` がゼロ以外を戻すか、`mysql_field_count()` がゼロを戻すかをチェックすることによって、エラーが起こったか否かをチェックすることができます。

戻された列がない場合、空の結果セットが戻されます。(空の結果セットは戻り値がゼロポインターと異なっています。)

`mysql_store_result()` を呼び出し、ゼロポインターでない結果を戻した後、`mysql_num_rows()` を呼び出して、何個の列が結果セットの中にあるかを検知することができます。

`mysql_fetch_row()`を呼び出して、結果セットから列をフェッチするか、`mysql_row_seek()`と`mysql_row_tell()`を呼び出して、現在の列の位置を取得するか、結果セットの中にセットすることができます。

詳しくは [なぜmysql_store_result\(\)時々返すNULLの後mysql_query\(\) 成功を返す](#) を参照してください。

戻り値

`MYSQL_RES`結果を含む結果構造。`NULL`エラーが起こった場合。

エラー

`mysql_store_result()`は成功すると、`mysql_error()`と`mysql_errno()`をリセットします。

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

`mysql_thread_id()`

`unsigned long mysql_thread_id(MYSQL *mysql)`

説明

現在接続のスレッドIDを戻します。この値をスレッドを殺す`mysql_kill()`に対する引数として使うことができます。

接続が失われたとき、`mysql_ping()`に接続すると、スレッドのIDが変わります。これは、スレッドIDを取得して、後のためにそれを記憶すべきではないことを意味します。要なとき、それを取得すべきです。

戻り値

現在接続のスレッドID。

エラー

なし。

`mysql_use_result()`

`MYSQL_RES *mysql_use_result(MYSQL *mysql)`

説明

データ(`SELECT`、`SHOW`、`DESCRIBE`、`EXPLAIN`)をうまく復元する全てのクエリーのために、`mysql_store_result`もしくは`mysql_use_result()`を呼び出さなければなりません。

`mysql_use_result()`は結果セットの複製を先導しますが、`mysql_store_result()`がするように、実際にクライアントの中に読み取りません。にもかかわらず、`mysql_fetch_row()`に対する呼び出しを行うことによって、各列を個別に復元し

なければなりません。これは、サーバから直接クエリーの結果を、それを一時テーブルまたはローカルバッファに記憶させることなく読み取ります。これは幾らか高速で、`mysql_store_result()`より少ないメモリーを使います。クライアントは現在の列および`max_allowed_packet`バイトに成長した通信バッファだけにメモリーを割り当てます。

一方、クライアント側にある各列に対して多くの処理を行う場合、もしくはユーザーがアウトプットを`^S`(スクロール停止)をタイプしてもよいスクリーンに送る場合、`mysql_use_result()`を使用すべきではありません。これはサーバを拘束し、他のスレッドが、そこからデータがフェッチされているテーブルをアップデートするのを阻止します。

`mysql_use_result()`を使うとき、`NULL`値が戻されるまで、`mysql_fetch_row()`を実行しなければなりません。C APIは、これを行うのを忘れた場合、エラー`Commands out of sync; you can't run this command now`を附与します。

`mysql_data_seek()`、`mysql_row_seek()`、`mysql_row_tell()`、`mysql_num_rows()`もしくは`mysql_affected_rows()`を`mysql_use_result()`から戻された結果と一緒に使ってはなりません。また、`mysql_use_result()`が終了するまで、他のクエリーを発行してなりません。(しかしながら、すべての列をフェッチした後、`mysql_num_rows()`はフェッチした行の数を正確に戻します。)

結果セットが必要となってから、`mysql_free_result()`を呼び出さなければなりません。

`libmysqld`埋込サーバを使うとき、`mysql_free_result()`が呼び出されるまで、メモリーの使用が戻された各列と共に徐々に増加するので、メモリー利益が本質的に失われます。

戻り値

`MYSQL_RES`結果構造。`NULL`エラーが起こった場合。

エラー

`mysql_store_result()`は成功すると、`mysql_error()`と`mysql_errno()`をリセットします。

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

`mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

説明

前のSQLステートメントの実行中に生成された警告の数を返します。

戻り値

警告アカウント。

エラー

なし。

準備されたC APIステートメント。

MySQLクライアント/サーバ・プロトコルは準備されたステートメントの使用を規定します。この能力には、`mysql_stmt_init()`初期化機能によって戻された**MYSQL_STMT**ステートメントハンドラー・データ構造が使われています。準備された実行は1回以上ステートメントを実行する効率的な方法です。ステートメントはその実行のために最初解析されます。その後、それは初期化機能によって戻されたシテートメントハンドルを使って複数回実行されます。

準備された実行は、基本的にクエリの解析が一度だけなので、一度以上実行されるステートメントの実行を、直接実行するより素早く行います。直接実行の場合、クエリは実行のたびに解析されます。準備された実行はまた、実行のたびにパラメータにデータを送るだけで済むので、ネットワークの通信料を減らす効果があります。

準備されたステートメントは状況によっては、性能の増加をもたらさない場合があります。最も良い結果を得るため、準備されたステートメントと準備されていないステートメントの両方を使って、自分のアプリアプリケーションをテストして、最もよい性能が得られる方を選んでください。

プリペアド ステートメントの他の利点は、クライアントとサーバ間のデータ伝達をより効果的にするバイナリ プロトコルを使用していることです。

次のステートメントは準備されたステートメントとして用いることができます：**CREATE TABLE**ステートメント、**DELETE**ステートメント、**DO**ステートメント、**INSERT**ステートメント、**REPLACE**ステートメント、**SELECT**ステートメント、**SET**ステートメント、**UPDATE**並びに殆どの**SHOW**ステートメント。

MySQL 5.1.10では、次のステートメントがサポートされています：

```
ANALYZE TABLE
OPTIMIZE TABLE
REPAIR TABLE
```

MySQL 5.1.12では、次のステートメントがサポートされています：

```
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN
```

他のステートメントはMySQL 5.1にサポートされていません。

準備されたC APIステートメントデータタイプ

準備されたステートメントにはいくつかのデータ構造が使われます：

- ステートメントを準備するため、ステートメントストリングを**mysql_stmt_init()**に渡してください。これによって、ポインタが**MYSQL_STMT**に戻されます。
- インプットパラメータを準備されたステートメントに規定するには、**MYSQL_BIND**構造を設定して、それを**mysql_stmt_bind_param()**に渡してください。アウトプットカラムを受け取るには、**MYSQL_BIND**を設定して、これを**mysql_stmt_bind_result()**に渡してください。
- MYSQL_TIME**構造は一時データを両方向に転送するために使われます。

次の講義で、準備されたステートメントのデータタイプを詳細に説明します。

- **MYSQL_STMT**

この構造は準備されたステートメントを表します。ステートメントは`mysql_stmt_init()`を呼び出すことによって生成され、これによって、ステートメントハンドル(即ち**MYSQL_STMT**)が戻されます。ハンドルは、それを`mysql_stmt_close()`を使って選択するまで、そのステートメントを使った後のオペレーションに使われます。

MYSQL_STMT構造には、アプリケーションに使うよう意図されたメンバーが含まれていません。また、**MYSQL_STMT**構造のコピーを作ろうとすべきではありません。このようなコピーが使用可能である保証はありません。

マルチステートメントハンドルには接続が1個して付いていません。ハンドルの数に対する限度は利用できる資源によって変わります。

- **MYSQL_BIND**

この構造は、ステートメント・インプット(サーバに送られたデータ値)とステートメント・アウトプット(サーバから戻された結果値)の両方のために使われます：

- インプットに対して、**MYSQL_BIND**は、`mysql_stmt_bind_param()`と一緒に、パラメータデータを、`mysql_stmt_execute()`が使う目的でバッファに固定するために使われます。
- インプットに対して、**MYSQL_BIND**は`mysql_stmt_bind_result()`と一緒に、列をフェッチするのに使用するため、結果セットバッファを`mysql_stmt_fetch()`に固定するのに使われます。

MYSQL_BIND構造を使うために、中身をゼロにして、それを初期化してから、そのメンバーを適当にセットしてください。例えば、3つの**MYSQL_BIND**構造の配列を宣言し、初期化するために、このコードを使ってください：

```
MYSQL_BIND bind[3];
memset(bind, 0, sizeof(bind));
```

MYSQL_BIND構造はアプリケーションプログラムが使用する、次のメンバーを含んでいます。メンバーの幾つかに対して使用するマナーは、構造がインプットに使われるか、アウトプットに使われるかによって変わります。

- **enum enum_field_types buffer_type**

バッファのタイプ。このメンバーはあなたがステートメントパラメータに固定しているC言語変数のデータタイプを示します。許容される**buffer_type**値はこのセクションで後に列記されています。インプットに対して、**buffer_type**はサーバに送る値を含む変数のタイプを示します。アウトプットに対してそれは、サーバから受け取った値を記憶させたい変数のタイプを示します。

- **void *buffer**

データの転送に使用する、バッファに対するポインター。これは変数のアドレスです。

インプットの場合、**buffer**はステートメントのパラメータが記憶される変数に対するポインターです。`mysql_stmt_execute()`を呼び出すとき、MySQLは変数に記憶した値を取り込んでそれを、ステートメント中の対応するパラメータ・マーカーの場所に置きます。

アウトプットの場合、**buffer**はその中に結果セットの列値を戻す変数に対するポインターです。`mysql_stmt_fetch()`を呼び出すとき、MySQLは列値を戻し、それをこの変数の中に記憶します。呼び出しに戻って値にアクセスすることができます。

MySQLが実施する必要のあるクライアント側のC言語の値とサーバ側のSQLの値の間のタイプ変換を最小にするため、対応するSQLの値と同等なタイプを含む変数を使ってください。データが数値タイプである場合、**buffer**をCタイプの適当な数値に指定すべきです。(charまたは整数変数の場合、あなたは、後で述べる**unsigned**メンバーをこのリストの中にセットすることによって、変数に**is_unsigned**属性が含まれているか否かも示すべきです。文字(非バイナリー)タイプおよびバイナリーストリングデータタイプに対して、**buffer**をバッファに指定すべきです。日付データタイプおよび時間データタイプに対して、**buffer**を**MYSQL_TIME**構造に指すべきです。

後のセクションにあるタイプ変換に関する注をご参照ください。

- **unsigned long buffer_length**

`*buffer`の実サイズ(バイト)。これはバッファに記憶することができるデータの最大量を示します。文字とバイナリーCデータに対して、`buffer_length`値は、`mysql_stmt_bind_param()`と一緒に使ってインプット値を規定する`*buffer`の長さ、または`mysql_stmt_bind_result()`と一緒に使うとき、バッファにフェッチできるアウトプットデータバイトの最大数を規定します。

- `unsigned long *length`

`*buffer`中に記憶されたデータの実のバイト数を示す`*buffer`変数に対するポインター。`length`は文字またはバイナリーCデータに使われます。

インプットパラメータデータを結合する場合、`length`は、`*buffer`中に記憶されたパラメータ値の実際の長さを示す`unsigned long`変数を指します。これは`mysql_stmt_execute()`によって使われます。

アウトプット値を結合する場合、`mysql_stmt_fetch()`の戻り値は長さの解釈を決めます：

- `mysql_stmt_fetch()`がゼロを戻す場合、`*length`は、パラメータ値の実の長さを示します。
- `mysql_stmt_fetch()`が`MYSQL_DATA_TRUNCATED`を戻す場合、`*length`はパラメータ値の切り捨て前の長さを示します。この場合、`*length`と`buffer_length`の最小は、その値の実の値を示します。

データ値の長さは`buffer_type`値によって決まるので、`length`は数値タイプのデータと一時タイプのデータに対して無視されます。

- `my_bool *is_null`

このメンバーは、値が`NULL`である場合、真で、それが`NULL`でない場合、虚となる`my_bool`変数を指します。インプットに対して、`*is_null`を真にセットして、`NULL`値をステートメントパラメータとして渡すことを示してください。

`is_null`はブーリアンスカラーではないが、pointerの代わりにブーリアンスカラーをさす理由は、`NULL`値を規定する方法にフレキシビリティを提供することです：

- データ値がいつも`NULL`である場合、カラムを結束するとき、`MYSQL_TYPE_NULL`を`buffer_type`値として使ってください。他のメンバーは問題ではありません。
- データ値がいつも`NOT NULL`である場合、あなたが結束している変数に対して、他のメンバーを適当にセットし、`is_null = (my_bool*) 0`をセットしてください。
- その他の場合、他のメンバーを適当にセットし、`is_null`を`my_bool`変数のアドレスにセットしてください。実行と実行の間に変数の値を適当に真か虚にセットして、データ値が`NULL`であるか`NOT NULL`であるかを、それぞれ示してください。

アウトプットに対して、`is_null`をさす値は、ステートメントから戻された結果セットカラム値が`NULL`である場合、列をフェッチした後に真にセットされます。

- `my_bool is_unsigned`

このメンバーは、`unsigned (char, short int, int, long long int)`であることができるデータタイプと一緒にC変数のために使われます。`buffer`をさす変数が`unsigned`である場合、`is_unsigned`を真にセットし、そうでない場合には、虚にセットしてください。例えば、`signed char`変数を`buffer`に結びつける場合、`MYSQL_TYPE_TINY`のタイプコードを規定し、`is_unsigned`を虚にセットしてください。代わりに、`unsigned char`を結びつける場合、タイプコードは同じですが、`is_unsigned`を真にすべきです。(charに対して、それがサインされるか否かが規定されないため、`signed char`または`unsigned char`を使って、サインについて明確にすることがベストです。)

`is_unsigned`はクライアント側のC言語変数だけに適用されます。それはクライアント側の対応するSQL値に対するサインの必要性について何も示しません。例えば、`int`を使って`BIGINT UNSIGNED`カラムのために値を供給する場合、`int`はサインするタイプなので、`is_unsigned`を虚にすべきです。`unsigned int`を使って`BIGINT`カラムのために値を供給する場合、`unsigned int`はサインするタイプなので、`is_unsigned`を真にすべきです。MySQLは、結果が切り捨てられる場合警告が起こりますが、サイン済みの値と未サインの値の間の変換を両方向に実施します。

- `my_bool *error`

アウトプットに対して、このメンバーを`my_bool`変数をさして、列をフェッチした後、そこに記憶されたパラメータに対する切り捨て情報を持たせてください。(切り捨て報告はデフォルトによって有効化されますが、`MYSQL_REPORT_DATA_TRUNCATION`オプションを使って、`mysql_options()`を呼び出すことによって、これを制御することができます。)切り捨て報告を有効化するとき、`mysql_stmt_fetch()`は`MYSQL_DATA_TRUNCATED`を戻し、`*error`は、その中で切り捨てが起こる`MYSQL_BIND`構造中で真となります。切り捨ては、サインまたは重要な桁の喪失あるいは、ストリングはコラムに収めるには長すぎたことを示します。

- `MYSQL_TIME`

この構造は、`DATE`、`TIME`、`DATETIME`および`TIMESTAMP`データを直接サーバからサーバに送り且つ受け取るのに使われます。`MYSQL_BIND`構造の`buffer_type`メンバーを一時タイプ(`MYSQL_TYPE_TIME`、`MYSQL_TYPE_DATE`、`MYSQL_TYPE_DATETIME`、`MYSQL_TYPE_TIMESTAMP`)の一つにセットし、`buffer`メンバーを`MYSQL_TIME`構造をさすようにセットしてください。

`MYSQL_TIME`構造には、次のテーブルに列記したメンバーが含まれています：

メンバー	摘要
<code>unsigned int year</code>	年
<code>unsigned int month</code>	年の月
<code>unsigned int day</code>	月の日
<code>unsigned int hour</code>	日の時間
<code>unsigned int minute</code>	時間の分
<code>unsigned int second</code>	分の秒
<code>my_bool neg</code>	時間がネガティブか否かを示すブーリアンフラグ
<code>unsigned long second_part</code>	マイクロ秒中の秒の部分;現在使用していない

一時値の或るタイプに適用する`MYSQL_TIME`構造のこれらの部分だけが使われます。`year`、`month`および`day`エレメントは、`DATE`、`DATETIME`および`TIMESTAMP`値に対して使用されます。`hour`、`minute`および`second`エレメントは、`TIME`、`DATETIME`および`TIMESTAMP`値に対して使用されます。[日付とタイム値のC API式取り扱い](#)を参照してください。

以下のテーブルは、インプットのための`MYSQL_BIND`構造の`buffer_type`ナンバーの中に規定することができる許容値を示します。その値はあなたが結合しているC言語変数のデータタイプに従って選択されるべきです。その値が`unsigned`である場合、`is_unsigned`ナンバーも真にセットすべきです。テーブルは、使用できるC変数タイプ、対応するタイプコードおよび供給された値を変換することなく使用できるSQLデータタイプを示します。

入力変数C型	<code>buffer_type</code> 値	SQL型のあて先値
<code>signed char</code>	<code>MYSQL_TYPE_TINY</code>	<code>TINYINT</code>
<code>short int</code>	<code>MYSQL_TYPE_SHORT</code>	<code>SMALLINT</code>
<code>int</code>	<code>MYSQL_TYPE_LONG</code>	<code>INT</code>
<code>long long int</code>	<code>MYSQL_TYPE_LONGLONG</code>	<code>BIGINT</code>
<code>float</code>	<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT</code>
<code>double</code>	<code>MYSQL_TYPE_DOUBLE</code>	<code>DOUBLE</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_TIME</code>	<code>TIME</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_DATE</code>	<code>DATE</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_DATETIME</code>	<code>DATETIME</code>
<code>MYSQL_TIME</code>	<code>MYSQL_TYPE_TIMESTAMP</code>	<code>TIMESTAMP</code>
<code>char[]</code>	<code>MYSQL_TYPE_STRING</code> (非バイナリデータ用)	<code>TEXT</code> , <code>CHAR</code> , <code>VARCHAR</code>

char[]	MYSQL_TYPE_BLOB (バイナリデータ用)	BLOB, BINARY, VARBINARY
	MYSQL_TYPE_NULL	NULL

MYSQL_TYPE_NULLの使用は、`is_null`ナンバーと関連させて前に説明されています。

以下のテーブルは、インプットのためのMYSQL_BIND構造の`buffer_type`ナンバーの中に規定することができる許容値を示します。その値はあなたが結合しているC言語変数のデータタイプに従って選択されるべきです。その値が`unsigned`である場合、`is_unsigned`ナンバーも真にセットすべきです。テーブルは、受領値のSQLタイプ、このような値が結果セットメタデータ中に持つ対応するタイプコード並びにSQL値を変換することなく受け取るMYSQL_BIND構造と結合する推薦されたC言語データのタイプを示します。

SQL 型の受信値	buffer_type 値	出力変数C型
TINYINT	MYSQL_TYPE_TINY	signed char
SMALLINT	MYSQL_TYPE_SHORT	short int
MEDIUMINT	MYSQL_TYPE_INT24	int
INT	MYSQL_TYPE_LONG	int
BIGINT	MYSQL_TYPE_LONGLONG	long long int
FLOAT	MYSQL_TYPE_FLOAT	float
DOUBLE	MYSQL_TYPE_DOUBLE	double
DECIMAL	MYSQL_TYPE_NEWDECIMAL	char[]
YEAR	MYSQL_TYPE_SHORT	short int
TIME	MYSQL_TYPE_TIME	MYSQL_TIME
DATE	MYSQL_TYPE_DATE	MYSQL_TIME
DATETIME	MYSQL_TYPE_DATETIME	MYSQL_TIME
TIMESTAMP	MYSQL_TYPE_TIMESTAMP	MYSQL_TIME
CHAR, BINARY	MYSQL_TYPE_STRING	char[]
VARCHAR, VARBINARY	MYSQL_TYPE_VAR_STRING	char[]
TINYBLOB, TINYTEXT	MYSQL_TYPE_TINY_BLOB	char[]
BLOB, TEXT	MYSQL_TYPE_BLOB	char[]
MEDIUMBLOB, MEDIUMTEXT	MYSQL_TYPE_MEDIUM_BLOB	char[]
LOB, LONGTEXT	MYSQL_TYPE_LONG_BLOB	char[]
BIT	MYSQL_TYPE_BIT	char[]

タイプ変換を避けたい場合、C言語の変数タイプは推薦されたそれらです。クライアント側のC変数とサーバ側の対応するSQL値の間にミスマッチがある場合、MySQLは暗黙のタイプ変換を実施します。

MySQLはサーバ側のSQL値に対するタイプコードを知っています。`buffer_type`値は、クライアント側に値を保持する変数のタイプコードを示します。2つのコードは一緒になってMySQLにどんな変換を実施しなければならないかを告げます。以下に例を示します。

- `MYSQL_TYPE_LONG`を`int`変数と一緒に使用して、整数値を`FLOAT`カラムに記憶されるべきサーバに渡す場合、MySQLはその値を、それを記憶する前に浮動点フォーマットに変換します。
- SQL `MEDIUMINT`カラム値をフェッチするが、`MYSQL_TYPE_LONGLONG`の`buffer_type`値を規定し、タイプ`long long int`のC変数をバッファの行き先として使用する場合、MySQLは(8バイトより少ないメモリーを必要とする)`MEDIUMINT`値を、`long long int`(8バイトの変数)の中に記憶させるために変換します。
- 数値カラムを255の値を使って、`char[4]`文字アレーの中にフェッチし、`MYSQL_TYPE_STRING`の`buffer_type`値を規定する場合、アレー中の合成値は、`'255\0'`を含む4バイト・ストリングとなります。
- `DECIMAL`値はストリングとして戻されます。これが対応するCタイプが`char[]`である理由です。`DECIMAL`値は、最初のサーバ側の値のストリング表示に対応するサーバによって戻されます。例えば、`12.345`はクライアント

に'12.345'として戻されます。[MYSQL_TYPE_NEWDECIMAL](#)を規定し、ストリングバッファを[MYSQL_BIND](#)構造につなげる場合、[mysql_stmt_fetch\(\)](#)はバッファ中の値を、変換することなく記憶します。代わりに、数値変数とタイプコード規定する場合、[mysql_stmt_fetch\(\)](#)はストリングフォーマット[DECIMAL](#)値を数値フォームに変換します。

- [MYSQL_TYPE_BIT](#)タイプコードのために、[BIT](#)値はストリングバッファの中に戻されます。(これによって、対応するCタイプもこの[char\[\]](#)になります。) 値はクライアント側に関する解釈を必要とするビットストリングを表します。値を扱い易いタイプとして戻すため、[+0](#)を使う次のフォームのクエリーを使って、値が整数に計算されるようにすることができます。

```
SELECT bit_col + 0 FROM t
```

値を復元するため、整数をその値を保持するに十分な大きさの変数に結び付けて、対応する適当な整数タイプコードを規定してください。

変数をカラムをフェッチするのに使用すべき[MYSQL_BIND](#)構造に結びつける前に、結果セットの各カラムに対して、タイプコードをチェックすることができます。どの変数タイプがタイプ変換を避けるために使うのに最も良いか決めたい場合、これが望ましいかもしれません。タイプコードを得るため、準備されたステートメントを[mysql_stmt_execute\(\)](#)を使って実行した後、[mysql_stmt_result_metadata\(\)](#)を呼び出してください。メタデータは、[mysql_stmt_result_metadata\(\)](#)およびC APIデータタイプで述べた結果セットのためのタイプコードへのアクセス手段を提供します。

[MYSQL_FIELD](#)カラムメタデータ構造の[max_length](#)メンバーを([mysql_stmt_attr_set\(\)](#)メンバーを呼び出すことによって)セットさせるようにする場合、結果セットのための[max_length](#)値が、バイナリー画面の長さではなく、結果値の最も長いストリングを示すことにご注目ください。即ち、[max_length](#)は、準備されたステートメントのために使用されたバイナリープロトコルを使って値をフェッチするに要するバッファのサイズに必ずしも対応しません。バッファのサイズはその中に値をフェッチする変数のタイプに従って選択されるべきです。

[MYSQL_TYPE_STRING](#)によって示される)インプット文字(非バイナリー)ストリングの日付に対して、その値は[character_set_client](#)システム変数によって示される文字セットの中にあると見なします。値が異なった文字セットと一緒にカラムの中に記憶される場合、その文字セットに対して適切な変換が起こります。([MYSQL_TYPE_BLOB](#)によって示される)インプットバイナリーストリングデータに対して、その値は[binary](#)文字セットを含んでいるものとして処理されます。即ち、それはバイトストリングとして処理されるので、変換は起こりません。

サーバから戻されたアウトプットストリング値にバイナリーまたは非バイナリーデータが含まれているか否かを査定するには、結果セットメタデータの[charsetnr](#)値が 63であるか否かをチェックします。(C APIデータタイプ参照) そうである場合、文字セットは[binary](#)で、これは、非バイナリーデータでなく、バイナリーデータであることを示します。これが、[BINARY](#)と[CHAR](#)、[VARBINARY](#)と[VARCHAR](#)および[BLOB](#)並びに[TEXT](#)タイプを区別することを可能にします。

準備されたC APIステートメント機能の概要

準備されたステートメントの処理に利用可能な機能の概要をここで一括説明し、後のセクションで各機能について詳細に説明します。[準備されたC APIステートメント機能の詳細](#)を参照してください。

機能	説明
mysql_stmt_affected_rows()	準備された UPDATE 、 DELETE 、または INSERT ステートメントの行変更、消去、もしくは挿入の回数を返す。
mysql_stmt_attr_get()	準備されたステートメントの属性値を入手する。
mysql_stmt_attr_set()	準備されたステートメントの属性をセットする。
mysql_stmt_bind_param()	準備されたSQLステートメントのパラメータマーカとアプリケーションデータバッファを関連させる。
mysql_stmt_bind_result()	結果セットのカラムとアプリケーションデータバッファを関連させる。
mysql_stmt_close()	準備されたステートメントに使用されているメモリを開放する。
mysql_stmt_data_seek()	ステートメント結果セットの任意の行ナンバーをシークする。
mysql_stmt_errno()	最後に実行されたステートメントに対してエラーナンバーを返す。
mysql_stmt_error()	最後に実行されたステートメントに対してエラーメッセージを返す。

<code>mysql_stmt_execute()</code>	準備されたステートメントを実行する。
<code>mysql_stmt_fetch()</code>	結果セットから次のデータの行を入手し、全ての束縛されたカラムのデータを返す。
<code>mysql_stmt_fetch_column()</code>	結果セットの現在の行の 1 カラムからデータを入手する。
<code>mysql_stmt_field_count()</code>	最近のステートメントの結果カラムの数を返す。
<code>mysql_stmt_free_result()</code>	ステートメントハンドルに割り当てられたリソースを開放する。
<code>mysql_stmt_init()</code>	<code>MYSQL_STMT</code> 構成のためのメモリを確保し、初期化する。
<code>mysql_stmt_insert_id()</code>	準備されたプログラムに夜 <code>AUTO_INCREMENT</code> カラム用に生成された ID を返す。
<code>mysql_stmt_num_rows()</code>	ステートメントのバッファされた結果セットからトータル行を返す。
<code>mysql_stmt_param_count()</code>	準備された SQL ステートメント内のパラメータの数を返す。
<code>mysql_stmt_param_metadata()</code>	(結果セットの形でパラメータメタデータを返す。)現在、この機能は何もしません。
<code>mysql_stmt_prepare()</code>	は実行のため、SQL ストリングを用意します。
<code>mysql_stmt_reset()</code>	Reset the statement buffers in the server.
<code>mysql_stmt_result_metadata()</code>	Returns prepared statement metadata in the form of a result set.
<code>mysql_stmt_row_seek()</code>	Seeks to a row offset in a statement result set, using value returned from <code>mysql_stmt_row_tell()</code> .
<code>mysql_stmt_row_tell()</code>	Returns the statement row cursor position.
<code>mysql_stmt_send_long_data()</code>	Sends long data in chunks to server.
<code>mysql_stmt_sqlstate()</code>	Returns the SQLSTATE error code for the last statement execution.
<code>mysql_stmt_store_result()</code>	Retrieves the complete result set to the client.

ステートメントハンドルを作成するには `mysql_stmt_init()` を呼び出してください。準備するには `mysql_stmt_prepare` を呼び出してください。パラメータデータを提供するには `mysql_stmt_bind_param()` を呼び出してください。ステートメントを実行するには `mysql_stmt_execute()` を呼び出してください。 `mysql_stmt_bind_param()` を通して供給された各バッファ中のパラメータを変更することによって、 `mysql_stmt_execute()` を繰り返すことができます。 .

ステートメントが `SELECT` または結果セットを生成するその他のステートメントである場合、 `mysql_stmt_prepare()` は、 `MYSQL_RES` 結果セットのフォームの中に、 `mysql_stmt_result_metadata()` を通して、結果セットメタデータ情報も戻します。

`mysql_stmt_bind_result()` を使って、結果バッファを供給することができるので、 `mysql_stmt_fetch()` は自動的にデータをこれらのバッファに戻します。これが列別フェッチングです。

シャंक中のエキストまたはバイナリーデータを `mysql_stmt_send_long_data()` を使ってサーバに送ることもできます。 `mysql_stmt_data_seek()` を参照してください。

ステートメントの実行が終わったとき、それを `mysql_stmt_close()` を使って閉じなければならないので、それに付属する全ての資源を解放することができます。

`mysql_stmt_result_metadata()` を呼び出すことによって、 `SELECT` ステートメントの結果セットメタデータを取得した場合、 `mysql_free_result()` を使ってメタデータも解放すべきです。

実行ステップ

ステートメントをを準備して実行するために、アプリケーションはこれらのステップに従います：

1. `mysql_stmt_init()` を使って準備されたステートメントのハンドルを生成させてください。サーバ上にステートメントを準備するため、 `mysql_stmt_prepare()` を呼び出して、それを SQL ステートメントを含むストリングに渡してください。
2. ステートメントが結果セットを生成する場合、 `mysql_stmt_result_metadata()` を呼び出して、結果セットメタデータを取得してください。このメタデータは、クエリーによって戻された列を含むものとは違いますが、結果セット

中に含まれています。メタデータ結果セットは、カラムが結果中に幾つ存在しているかを示し、各カラムに関する情報を含んでいます。

3. `mysql_stmt_bind_param()`を使って、パラメータの値をセットしてください。すべてのパラメータをセットしなければなりません。さもなければ、ステートメントがエラーを戻すか、意外な結果を引き起こします。
4. `mysql_stmt_execute()`を呼び出してステートメントを実行してください。
5. ステートメントが結果セットを生成する場合、メタデータバッファを結束して、`mysql_stmt_bind_result()`を呼び出すことによって、列の値を復元するために使ってください。
6. 繰り返し`mysql_stmt_fetch()`を呼び出すことによって、列が見つからなくなるまで、データを列ごとにバッファの中フェッチしてください。
7. パラメータの値を変えてステートメントを再実行することによって、必要に応じて3回から6回ステップを繰り返し実行してください。

`mysql_stmt_prepare()`が呼び出されると、MySQLクライアント/サーバプロトコルはこれらのアクションを実施します。

- サーバはステートメントを解析し、ステートメントにIDを割り当てることによって、Okのステータスをクライアントに送り返します。それが結果セットを指向したステートメントである場合、サーバはパラメータの合計ナンバー、カラムカウントおよびそのメタデータも送ります。ステートメントのすべてのsyntaxとsemanticsがこの呼び出し中に、サーバによってチェックされます。
- クライアントは、そのステートメントを、ステートメントプールにあるその他のステートメントと区別することができるよう、このステートメントのIDを更なるオペレーションに使用します。

`mysql_stmt_execute()`が呼び出されると、MySQLクライアント/サーバプロトコルはこれらのアクションを実施します。

- クライアントはステートメントハンドルを使って、サーバにパラメータデータを送ります。
- サーバは、クライアントが提供したIDを使ってステートメントを識別し、パラメータ・マーカーを使って新しく供給されたデータと入れ替えて、ステートメントを実行します。ステートメントが結果セットを生成する場合、サーバはクライアントにデータを送り返します。さもなければ、それは変更、削除または挿入されたOkステータスと列の合計数を送ります。

`mysql_stmt_fetch()`が呼び出されると、MySQLクライアント/サーバプロトコルはこれらのアクションを実施します。

- クライアントは列別にパケット列からデータを読み取って、それを必要な変換をすることによって、アプリケーションデータバッファの中に置きます。アプリケーションバッファタイプが、サーバからタイプが戻されたフィールドのそれと同じであると、変換は簡単です。

エラーが発生した場合、`mysql_stmt_errno()`、`mysql_stmt_error()`および`mysql_stmt_sqlstate()`を繰り返し使って、ステートメントエラーコード、エラーメッセージ並びにSQLSTATE値を得ることができます。

準備されたステートメントのロギング

`mysql_stmt_prepare()`と`mysql_stmt_execute()`C API ファンクションで準備されたステートメントに対して、サーバは準備そして実行ラインを一般クエリログに送信します。これによりステートメントがいつ準備、実行されたかわかります。

次のようにステートメントを準備して実行すると考えてください：

1. `mysql_stmt_prepare()`を呼び出してステートメントストリング"`SELECT ?`"を用意します。
2. `mysql_stmt_bind_param()`を呼び出して、値3を準備されたステートメント中のパラメータに結びつけます。
3. `mysql_stmt_execute()`を呼び出して準備されたステートメントを実行してください。

前のコールの結果として、サーバは一般クエリログに次の行を書き込みます：

```
Prepare [1] SELECT ?
```

```
Execute [1] SELECT 3
```

ログ内の各**準備**と**実行**ラインは、**[N]**ステートメント識別子でタグされます。これはどの準備されたステートメントがログしているか確認するためです。**N**は正の整数です。複数の準備されたステートメントがクライアントに対して同時にアクティブである場合、**N**を1より大きくすることができます。各**Execute**ラインは、**?**パラメータのためにデータ値を取り替えた後の準備されたステートメントを示します。

バージョンの注意：**準備**ラインは、**[N]**なしで、MySQL 4.1.10前は表示されます。**実行**ラインはMySQL 4.1.10前では表示されません。

準備されたC APIステートメント機能の詳細

クエリーを用意して実行するために、次のセクションで詳細を述べる機能を使ってください。

MYSQL_STMT構造を使って運転する全ての機能は接頭語**mysql_stmt_**で始まることにご注目ください。

MYSQL_STMT ハンドルを作成するには、**mysql_stmt_init()** フังก์ションを使用してください。

mysql_stmt_affected_rows()

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

説明

最後に実行されたステートメントによって、変更、削除または挿入された列の合計数を返します。**mysql_stmt_execute()**は、**UPDATE**、**DELETE**、あるいは**INSERT**ステートメントの直後、呼び出されることがあります。**SELECT**ステートメントに対して、**mysql_stmt_affected_rows()**は**mysql_num_rows()**と同じように作動します。

戻り値

ゼロより大きい整数は影響を与えられたか、復元された横列の数を示します。ゼロは、**UPDATE**ステートメントに対する記録が更新されなかったか、クエリー中の**WHERE**クローズにマッチした列が存在していなかったか、クエリーがまだ実行されていないことを示します。**-1**は、クエリーがエラーを戻したか、**SELECT**クエリーに対して、**mysql_stmt_store_result()**を呼び出す前に、**mysql_affected_rows()**が呼び出されたことを示します。**mysql_affected_rows()**が未サイン値を返すので、戻り値を**(my_ulonglong)-1** (または同等な**(my_ulonglong)~0**)と比べることによって、**-1**をチェックすることができます。

戻り値に関する追加情報については、**mysql_affected_rows()**をご参照ください。

エラー

なし。

例

mysql_stmt_affected_rows()の使用については、**mysql_stmt_execute()**から例をご参照ください。

mysql_stmt_attr_get()

```
int mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, void *arg)
```

説明

ステートメントの持つ属性のために現在の値を得るために使うことができます。

option数は取得したいオプションです。**arg**は、オプション値を含むべき変数さすべきです。オプションが整数である場合、**arg**はその整数の値を指し示すべきです。

オプションとオプションタイプのリストについては、**mysql_stmt_attr_set()**をご参照ください。

戻り値

OKの場合**0option**が未知である場合、ゼロ以外。

エラー

なし。

mysql_stmt_attr_set()

```
int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, const void *arg)
```

説明

準備されたステートメントのために挙動に影響を付与するために使うことができます。幾つかのオプションをセットするために、この機能を何回でも呼び出すことができます。

option 引数は、セットしたいオプションです。**arg** 引数はそのオプションのための値です。オプションが整数である場合、**arg** はその整数の値を指し示すべきです。

可能な**オプション**値:

オプション	アーギュメント型	ファンクション
STMT_ATTR_UPDATE_MAX_LENGTH	my_bool *	1に設定している場合。mysql_stmt_store_result()中のメタデータMYSQL_FIELD->max_lengthを更新してください。
STMT_ATTR_CURSOR_TYPE	unsigned long *	mysql_stmt_execute()が起動されている場合開くステートメントのカーソル。 *arg はCURSOR_TYPE_NO_CURSOR (デフォルトの) あるいはCURSOR_TYPE_READ_ONLYになります。
STMT_ATTR_PREFETCH_ROWS	unsigned long *	カーソル使用時に、サーバから入手する行の数。 *arg は1から最大値 unsigned long. デフォルトは1です。

STMT_ATTR_CURSOR_TYPE オプションをCURSOR_TYPE_READ_ONLYと一緒に使う場合、mysql_stmt_execute()を取り出すとき、ステートメントのために、カーソルが開かれます。mysql_stmt_execute()の前の呼び出しから、カーソルが既に開かれて存在している場合、それは新しいものを開く前にカーソルを閉じます。mysql_stmt_reset()も再実行のためステートメントを準備する前に、閉じられます。mysql_stmt_free_result()はどんなカーソルでも開きます。

カーソルを準備されたステートメントのために開く場合、その機能はクライアント側に一時的に記憶させるようにするので、mysql_stmt_store_result()は不要です。

戻り値

OKの場合**0**optionが未知である場合、ゼロ以外。

エラー

なし。

例

以下の例は、準備されたステートメントのために、カーソルを開き、5になる時間にフェッチするように、列のナンバーをセットします。

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
```

```
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                        (void*) &prefetch_rows);
/* ... check return value ... */
```

mysql_stmt_bind_param()

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

説明

`mysql_stmt_bind_param()`は、`mysql_stmt_prepare()`に渡したSQLステートメント中のパラメータマーカーのためにインプットデータを束ねるのに使われます。それは、`MYSQL_BIND`構造をデータを提供するのに使います。`bind`は、`MYSQL_BIND`構造の配列のアドレスです。クライアントライブラリは配列に、各'?'クエリー中に存在するパラメータマーカーが含まれることを期待します。

次のステートメントを準備すると仮定してください：

```
INSERT INTO mytbl VALUES(?,?,?)
```

パラメータを束ねる場合、`MYSQL_BIND`構造の配列には、3つの要素が含まれていなければなりません。このことは、このように宣言することができます：

```
MYSQL_BIND bind[3];
```

準備されたC APIステートメントデータタイプは各`MYSQL_BIND`エレメントのメンバーおよび値を提供するためセットすべき方法を説明します。

戻り値

結束作業が成功していたらゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_UNSUPPORTED_PARAM_TYPE`

変換はサポートされません。`buffer_type`値は違法であるが、サポートされているタイプの一つでない可能性があります。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

例

`mysql_stmt_bind_param()`を使用するため、例を`mysql_stmt_execute()`から参照してください。

mysql_stmt_bind_result()

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

説明

`mysql_stmt_bind_result()`は、結果セット中のアウトプットカラムをデータバッファおよび長さバッファに添付(即ち結束)するのに使います。`mysql_stmt_fetch()`がデータをフェッチするために呼び出されたとき、MySQLクライアント/サーバプロトコルはポンドカラムのためのデータを規定されたバッファの中に置きます。

全てのカラムは、`mysql_stmt_fetch()`を呼び出す前に、バッファと結束されなければなりません。この場合、`bind`は`MYSQL_BIND`構造の配列のアドレスです。クライアントライブラリは結果セットの各カラムに対して、1つの要素が含まれることを期待します。カラムを`MYSQL_BIND`構造に結びつけないと、`mysql_stmt_fetch()`は簡単

にデータフェッチを怠ります。プロトコルはチャンクでデータ値を返さないで、バッファをデータ値が持てる十分な大きさにべきです。

結果セットが一部複製された後でも、カラムは一回で結束または再結束することができます。新しい結束は、次に`mysql_stmt_fetch()`が呼び出されたとき有効となります。アプリケーションがカラムを結果セットの中に結束し、`mysql_stmt_fetch()`を呼び出すと仮定する。クライアント/サーバ・プロトコルは結束されたバッファの中にデータを戻します。その後、アプリケーションがバッファの異なったセットにカラムをバインドすると考えてください。`mysql_stmt_fetch()`への次のコールが起こるとき、プロトコルは新たに選ばれたバッファの中にデータを置きます。

カラムをしぼるため、アプリケーションが`mysql_stmt_bind_result()`を呼び出し、その中に値を記憶すべきアウトプットバッファのタイプ、アドレス及び長さを渡します。[準備されたC APIステートメントデータタイプ](#)は各`MYSQL_BIND`エレメントのメンバーおよび値を受け取るためセットすべき方法を説明します。

戻り値

結束作業が成功していたらゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_UNSUPPORTED_PARAM_TYPE`

変換はサポートされません。`buffer_type`値は違法であるか、サポートされているタイプの一つでない可能性があります。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

例

`mysql_stmt_bind_result()`の使用には、`mysql_stmt_fetch()`から例を参照してください。

`mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

説明

準備されたステートメントを閉じてください。`mysql_stmt_close()`は、`stmt`をさすステートメントハンドルの割り当ても解除します。

もし現在のステートメントがペンディングの結果か未読の結果を含む場合、次のクエリーを実行可能にするため、この機能はこれらをキャンセルします。

戻り値

ステートメントが解放された場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起こりました。

例

`mysql_stmt_affected_rows()`を使用するため、`mysql_stmt_execute()`から例を参照してください。

`mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

説明

ステートメント結果セットの中で任意の横列を探してください。`offset`値は列ナンバーなので、0から`mysql_stmt_num_rows(stmt)-1`までの範囲に収めるべきです。

この機能は、結果セット構造にクエリーの全結果を含めることを求めるので、`mysql_stmt_data_seek()`は、`mysql_stmt_store_result()`と併用する場合に限り使用することができます。

戻り値

なし。

エラー

なし。

`mysql_stmt_errno()`

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

説明

`stmt`によって規定された接続に対して、`mysql_stmt_errno()`成功する場合もあり失敗する場合もあるAPI 機能に対するエラーコードから最近使用したものを返します。ゼロの戻り値はエラーが起らなかったことを意味します。クライアントのエラーメッセージナンバーは、`MySQLerrmsg.h`ヘッダーファイル中に列記されています。サーバのエラーメッセージナンバーは、`mysql_error.h`の中に列記されています。[Errors, Error Codes, and Common Problems](#)にはエラーも列記されています。

戻り値

エラーコード値。エラーが発生しなかった場合、ゼロ。

エラー

なし。

`mysql_stmt_error()`

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

説明

`stmt`によって規定された接続に対して、`mysql_stmt_errno()`は、成功する場合もあり失敗する場合もあるAPI 機能に対するエラーコードから最近使用したものを返します。エラーが発生しなかった場合、空ストリング(<"">)が返されます。これは次の2つのテストが同等であることを意味します：

```
if(*mysql_stmt_errno(stmt))
{
    // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
    // an error occurred
}
```

クライアントエラーメッセージの言語は、MySQLクライアントライブラリを再編集することによって変更することができます。現在、幾つかの異なった言語で書かれたエラーメッセージの中から、好みのものを選ぶことができます。

戻り値

エラーを述べた文字ストリング。エラーが発生しなかった場合の空ストリング。

エラー

なし。

mysql_stmt_execute()

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

説明

`mysql_stmt_execute()`はステートメントハンドルと結び付けられた準備されたクエリーを実行します。しばらくした現在のパラメータマーカー値はこのコールの間にサーバに送られ、サーバはそのマーカーをこの新たに供給されたデータと置き換えます。

ステートメントが、`UPDATE`、`DELETE`または`INSERT`である場合、変更、削除もしくは挿入された列の合計ナンバーは、`mysql_stmt_affected_rows()`を呼び出すことによって見つけることができます。これが結果セットを生成する`SELECT`のようなステートメントである場合、クエリー処理が起こるその他の機能を呼び出す前に、`mysql_stmt_fetch()`を呼び出してデータをフェッチしなければなりません。結果をフェッチする方法の詳細情報については、`mysql_stmt_fetch()`をご参照ください。

結果セットを生成するステートメントに対して、あなたは、`mysql_stmt_execute()`に、ステートメントを実行する前に、`mysql_stmt_attr_set()`を呼び出すことによって、ステートメントのためにカーソルを開くよう求めることができます。ステートメントを複数回実行する場合、新しいものを開く前に、`mysql_stmt_execute()`はオープンカーソルを閉じます。

戻り値

成功した場合ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

例

次の例

は、`mysql_stmt_init()`、`mysql_stmt_prepare()`、`mysql_stmt_param_count()`、`mysql_stmt_bind_param()`、`mysql_stmt_execute()`および`mysql_stmt_affected_rows()`を使って、テーブルを生成してポピュレートする方法を明示します。`mysql`変数は有効な接続ハンドルと見なされます。

```
#define STRING_SIZE 50
```

```

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                col2 VARCHAR(40),\
                                col3 SMALLINT,\
                                col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO \
    test_table(col1,col2,col3) \
    VALUES(?,?,?)"

MYSQL_STMT *stmt;
MYSQL_BIND bind[3];
my_ulonglong affected_rows;
int param_count;
short small_data;
int int_data;
char str_data[STRING_SIZE];
unsigned long str_length;
my_bool is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
    fprintf(stderr, " DROP TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
    fprintf(stderr, " CREATE TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/* sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;

```

```
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
    fprintf(stderr, "mysql_stmt_bind_param() failed\n");
    fprintf(stderr, "%s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Specify the data values for the first row */
int_data= 10;          /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, "mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, "%s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, "total affected rows(insert 1): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, "invalid affected rows by MySQL\n");
    exit(0);
}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
    The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;      /* smallint */
is_null= 0;            /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, "mysql_stmt_execute, 2 failed\n");
    fprintf(stderr, "%s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, "total affected rows(insert 2): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, "invalid affected rows by MySQL\n");
    exit(0);
}
```

```

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```

注:準備されたステートメント機能の使用に関する完全な例については、[tests/mysql_client_test.c](#)をご参照ください。このファイルはMySQL source distributionあるいはBitKeeper source repositoryから取得することができます。

mysql_stmt_fetch()

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

説明

[mysql_stmt_fetch\(\)](#)は結果セット中に次列を戻します。結果セットが存在している間だけ、それは呼び出されます：即ち、結果セットを生成する[mysql_stmt_execute\(\)](#)に対する呼び出し後または[mysql_stmt_store_result\(\)](#)後で、それは、全結果セットを一時的に蓄えるため[mysql_stmt_execute\(\)](#)後に呼び出されます。

[mysql_stmt_fetch\(\)](#)は、[mysql_stmt_bind_result\(\)](#)によって束ねられたバッファを使って列データを戻します。それは、現在の列セット中の全てのカラムのために、これらのバッファの中にデータを戻し、長さは[length](#)ポインターに戻されます。

[mysql_stmt_fetch\(\)](#)を呼び出す前に、すべてのカラムはアプリケーションによって束ねられなければなりません。

フェッチされたデータ値がNULL値である場合、対応するMYSQL_BIND構造の*is_null値にはTRUE (1)が含まれます。さもないと、データとその長さは、アプリケーションによって規定されたバッファータ입に基づき、*bufferエレメント及び*lengthエレメントの中に戻されます。次のテーブルの中に列記されているように、数値と時間の各タイプは一定の長さを持っています。data_lengthによって示されるように、ストリングタイプの長さは実際のデータ値の長さに依存します。

タイプ	長さ
MYSQL_TYPE_TINY	1
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8
MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

戻り値

戻り値	摘要
0	データがアプリケーションデータバッファにフェッチされた場合、成功
1	エラーが発生しました。エラーコードとエラーメッセージは、 mysql_stmt_errno() および mysql_stmt_error() を呼び出すことによって取得することができます。
MYSQL_NO_DATA	もう列/データは存在しない
MYSQL_DATA_TRUNCATED	データの切り捨てが発生

切り捨て報告が有効なとき、`MYSQL_DATA_TRUNCATED`は戻されます。(報告はデフォルトによって有効化されますが、`mysql_options()`を使って制御することができます。)この値が返されるとき、いずれのパラメータが取り除かれたか査定するため、`MYSQL_BIND`パラメータ構造`error`メンバーをチェックしてください。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

- `CR_UNSUPPORTED_PARAM_TYPE`

バッファータイプは、`MYSQL_TYPE_DATE`、`MYSQL_TYPE_TIME`、`MYSQL_TYPE_DATETIME`、`MYSQL_TYPE_TIMESTAMP`ですが、データタイプは`DATE`、`TIME`、`DATETIME`、`TIMESTAMP`ではありません。

- サポートされていない変換に基づくエラーはすべて、`mysql_stmt_bind_result()`から返されます。

例

次の例は、`mysql_stmt_result_metadata()`、`mysql_stmt_bind_result()`、and `mysql_stmt_fetch()`を使って、テーブルからデータをフェッチする方法を明らかにするものです。(この例は`mysql_stmt_execute()`の中に示された例によって挿入された2つの横列を復元することを目指すものです。)mysql変数は有効な接続ハンドルと見なされません。

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
    FROM test_table"

MYSQL_STMT *stmt;
MYSQL_BIND bind[4];
MYSQL_RES *prepare_meta_result;
MYSQL_TIME ts;
unsigned long length[4];
int param_count, column_count, row_count;
short small_data;
int int_data;
char str_data[STRING_SIZE];
my_bool is_null[4];
my_bool error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, "mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
    fprintf(stderr, "mysql_stmt_prepare(), SELECT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
}
```



```
exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
    fprintf(stderr,
        " mysql_stmt_result_metadata(), \
        returned no meta information\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout,
    " total columns in SELECT statement: %d\n",
    column_count);

if (column_count != 4) /* validate column count */
{
    fprintf(stderr, " invalid column count returned by MySQL\n");
    exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)&str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
```

```
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
    fprintf(stderr, "mysql_stmt_bind_result() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Now buffer all results to client */
if (mysql_stmt_store_result(stmt))
{
    fprintf(stderr, "mysql_stmt_store_result() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
    row_count++;
    fprintf(stdout, " row %d\n", row_count);

    /* column 1 */
    fprintf(stdout, " column1 (integer) : ");
    if (is_null[0])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

    /* column 2 */
    fprintf(stdout, " column2 (string) : ");
    if (is_null[1])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

    /* column 3 */
    fprintf(stdout, " column3 (smallint) : ");
    if (is_null[2])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

    /* column 4 */
    fprintf(stdout, " column4 (timestamp) : ");
    if (is_null[3])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %04d-%02d-%02d %02d:%02d (%ld)\n",
            ts.year, ts.month, ts.day,
            ts.hour, ts.minute, ts.second,
            length[3]);
    fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
    fprintf(stderr, " MySQL failed to return all rows\n");
    exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);

/* Close the statement */
if (mysql_stmt_close(stmt))
{

```

```
fprintf(stderr, " failed while closing the statement\n");
fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
exit(0);
}
```

mysql_stmt_fetch_column()

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int column, unsigned long offset)
```

説明

現在の結果セット列から 1 個のカラムをフェッチしてください。bind はデータを置くべきバッファを提供します。それを `mysql_stmt_bind_result()` に対すると同じ方法でセットすべきです。column はどのカラムにフェッチすべきかを示します。最初のカラムには 0 のナンバーが付いています。offset はデータの復元を始めるべきデータ値の中のオフセットです。これはピースの中にデータ値をフェッチするのに使われます。値の始まりはオフセット 0 です。

戻り値

値がうまくフェッチされた場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_INVALID_PARAMETER_NO`

無効なカラムナンバー。

- `CR_NO_DATA`

結果セットはすでに終わりに達しています。

mysql_stmt_field_count()

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

説明

ステートメントハンドラーに対する最近のステートメントのためのカラムのナンバーを戻します。この値は、結果セットを生成しない `INSERT` または `DELETE` のようなステートメントのためのゼロです。

`mysql_stmt_prepare()` を取り出すことによって、ステートメントを準備した後、`mysql_stmt_field_count()` を呼び出すことができます。

戻り値

結果セット中にあるカラムの数を表す無署名の整数。

エラー

なし。

mysql_stmt_free_result()

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

説明

準備されたステートメントを実行することによって生成された結果セットと関係のあるメモリーを解放してください。ステートメントのために開いているカーソルがある場合、`mysql_stmt_free_result()` がそれを閉じます。

戻り値

結果セットが解放された場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

mysql_stmt_init()

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

説明

Create a `MYSQL_STMT` handle. ハンドルを `mysql_stmt_close(MYSQL_STMT *)` を使って解放すべきです。

戻り値

成功の場合 `MYSQL_STMT` 構造をさすポインター。 `NULL` メモリー不足の場合

エラー

- `CR_OUT_OF_MEMORY`

メモリ不足。

mysql_stmt_insert_id()

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

説明

`AUTO_INCREMENT` カラムのために、準備された `INSERT` or `UPDATE` ステートメントによって生成された値を返します。 `AUTO_INCREMENT` フィールドを含むテーブルの上で、準備された `INSERT` ステートメントを実行した後、この機能を使ってください。

明細な情報については、 `mysql_insert_id()` をご参照ください。

戻り値

準備されたステートメントの実行中に自動的に生成されたか明確にセットされた `AUTO_INCREMENT` カラムのための値または `LAST_INSERT_ID(expr)` 機能によって生成された値。 `AUTO_INCREMENT` 値をセットしない場合、戻り値は定義されません。

エラー

なし。

mysql_stmt_num_rows()

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

説明

それは、結果セット中の行数を返します。

`mysql_stmt_num_rows()` の使用は、全結果セットをステートメントハンドルの中に一時的に記憶する `mysql_stmt_store_result()` を使用するが否かに依存します。

`mysql_stmt_store_result()` を使うと、 `mysql_stmt_num_rows()` を直ちに呼び出すことができます。

`mysql_stmt_num_rows()` は `SELECT` を含む結果セットを返すステートメントと一緒に使用するよう意図されています。 `INSERT`、`UPDATE` または `DELETE` のようなステートメントのために、影響された列のナンバーを `mysql_stmt_affected_rows()` を使って取得することができます。

戻り値

結果セット中の行の数。

エラー

なし。

mysql_stmt_param_count()

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

説明

準備されたステートメントに存在しているパラメータマーカのナンバーを返します。

戻り値

ステートメント中のパラメータのナンバーを表す無署名の長い整数。

エラー

なし。

例

[mysql_stmt_affected_rows\(\)](#)の使用について、[mysql_stmt_execute\(\)](#)から例を参照してください。

mysql_stmt_param_metadata()

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

この機能は現在何もしません。

説明

戻り値

エラー

mysql_stmt_prepare()

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long length)
```

説明

[mysql_stmt_init\(\)](#)によって戻されたステートメントハンドルを附与すると、ストリング`stmt_str`をさすSQLステートメントを用意し、ステータスの値を返します。ストリングの長さは`length`引数によって与えられるべきです。ストリングは一つのSQLステートメントで構成しなければなりません。ステートメントの終にセミコロン(';') or `\g`を加えるべきではありません。

アプリケーションには、疑問詞('?')文字をSQLストリングの適当な位置に埋め込むことによって、複数のパラメータマーカを含めることができます。

マーカはSQLステートメント中の特定の場所でだけ合法的です。例えば、これらは、(列にカラム値を規定する) [VALUES\(\)](#) ステートメントの [INSERT](#) リストの中またはカラムと比較して比較値を規定する [WHERE](#) 条項中で容認されます。しかし、それらは、(テーブル名あるいはコラム名称のような)識別子に対しては容認されず、`=`等号のようなバイナリーオペレーターの両オペランドを規定することも許されません。パラメータのタイプを決めることは多分不可能なので、後者の規制は必要です。パラメータは一般に、Data Manipulation Language (DML)ステートメントの中でのみ法的に有効で、Data Definition Language (DDL)ステートメント中では無効です。

パラメータマーカは、ステートメントを実行する前に[mysql_stmt_bind_param\(\)](#)を使ってアプリケーション変数に縛り付けなければなりません。

戻り値

ステートメントがうまく準備された場合、ゼロ。エラーが起きた場合、ゼロ以外。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_OUT_OF_MEMORY](#)

メモリ不足。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

準備作業が不成功であった場合(即ち、[mysql_stmt_prepare\(\)](#)がゼロを戻す場合)、[mysql_stmt_error\(\)](#)を呼び出すことによって、エラーメッセージを取得することができます。

例

[mysql_stmt_prepare\(\)](#)の使用については、[mysql_stmt_execute\(\)](#)から、例を参照してください。

[mysql_stmt_reset\(\)](#)

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

説明

準備されたステートメントをクライアントとサーバ上にリセットして、準備後、陳述してください。これは主に[mysql_stmt_send_long_data\(\)](#)を使って送ったデータをリセットするのに使われます。ステートメントに対して開かれているカーソルが全て閉じられます。

他のクエリーを使ってステートメントを再び準備するには、[mysql_stmt_prepare\(\)](#)を使います。

戻り値

ステートメントがリセットされた場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- [CR_COMMANDS_OUT_OF_SYNC](#)

コマンドが妥当でないオーダーで実行されました。

- [CR_SERVER_GONE_ERROR](#)

MySQLサーバが立ち去りました。

- [CR_SERVER_LOST](#)

サーバへの接続がクエリー中に失われました。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

[mysql_stmt_result_metadata\(\)](#)

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```


説明

`mysql_stmt_prepare()`に渡したステートメントが結果セットを生成するものである場合、`mysql_stmt_result_metadata()`は、ポインタの形の結果セットメタデータを、フィールドの合計ナンバーと個別フィールド情報を処理するのに利用できる**MYSQL_RES**構造に戻します。この結果セットポインタは引数として、フィールドベースの結果セットメタデータを処理するAPI機能のいずれにも渡すことができます。

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

結果セット構造はそれが必要なとき、解放されるべきです。解放は、それを**mysql_free_result()**に渡すことによって実行されます。これは、**mysql_store_result()**に対する呼び出しから取得した結果セットを解放すると同じ方法です。

`mysql_stmt_result_metadata()`によって戻された結果セットにはメタデータだけが含まれています。それは列の結果を含んでいません。列は、`mysql_stmt_fetch()`を含むステートメントハンドルを使うことによって取得することができます。

戻り値

MYSQL_RES結果構造。**NULL**メタ情報が準備されたクエリーのために存在しない場合。

エラー

- **CR_OUT_OF_MEMORY**
メモリ不足。
- **CR_UNKNOWN_ERROR**
未知のエラーが起きました。

例

`mysql_stmt_result_metadata()`の用途については、`mysql_stmt_fetch()`から例をご参照ください。

`mysql_stmt_row_seek()`

MYSQL_ROW_OFFSET `mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)`

説明

ステートメント結果セット中の任意の横列に列カーソルをセットしてください。**offset**値は**mysql_stmt_row_tell()**または**mysql_stmt_row_seek()**から戻された値であるべきオフセットです。この値は列ナンバーではありません。結果セットの中で、列を番号別に探索する場合、代わりに**mysql_stmt_data_seek()**を使ってください。

この機能は、結果セット構造にクエリーの全結果を含めることを求めるので、**mysql_stmt_data_seek()**は、**mysql_stmt_store_result()**と併用する場合に限り使用することができます。

戻り値

列カーソルの前の値。この値を**mysql_row_seek()**に対する次の呼び出しに渡すことができます。

エラー

なし。

mysql_stmt_row_tell()

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

説明

それは、最後のmysql_stmt_fetch()に使用した列カーソルの現位置を戻します。この値をmysql_stmt_row_seek()に対する引数として使うことができます。

mysql_stmt_row_tell()を、mysql_stmt_store_result()の後にだけ使うべきです。

戻り値

列カーソルの現在のオフセット。

エラー

なし。

mysql_stmt_data_seek()

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number, const char *data, unsigned long length)
```

説明

アプリケーションがパラメータデータを一個ずつサーバ(または「chunks」)に送ることを可能にします。この機能をmysql_stmt_bind_param()の後およびmysql_stmt_execute()の前にこの機能呼び出して下さい。それを、文字の一部またはカラムのためのバイナリーデータを送るため、複数回呼び出すことができます。それはTEXT or BLOBデータタイプの一種でなければなりません。

parameter_number>はデータに添付すべきパラメータを示します。パラメータには、ゼロで始まる番号が付いています。dataは送るべきデータを含むバッファをさすポインターで、lengthはバッファ中のバイトのナンバーを示します。

注:次のmysql_stmt_execute()コールは、最後のmysql_stmt_send_long_data()以来、mysql_stmt_execute() or mysql_stmt_reset()と一緒に使われてきたパラメータに対するバインドバッファを無視します。

送ったデータをリセットし忘れた場合、それをmysql_stmt_reset()を使って実行することができます。mysql_stmt_reset()を参照してください。

戻り値

データがうまくサーバに送られる場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- CR_INVALID_BUFFER_USE

パラメータにはストリングあるいはバイナリータイプは含まれていません。

- CR_COMMANDS_OUT_OF_SYNC

コマンドが妥当でないオーダーで実行されました。

- CR_SERVER_GONE_ERROR

MySQLサーバが立ち去りました。

- CR_OUT_OF_MEMORY

メモリ不足。

- [CR_UNKNOWN_ERROR](#)

未知のエラーが起きました。

例

次の例はどのようにチャンクでTEXTコラムのデータを送るべきが明示します。それはデータ値'[MySQL - The most popular Open Source database](#)'をtext_columnコラムに挿入します。mysql変数は有効な接続ハンドルと見なされません。

```
#define INSERT_QUERY "INSERT INTO \
    test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long    length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}
memset(bind, 0, sizeof(bind));
bind[0].buffer_type= MYSQL_TYPE_STRING;
bind[0].length= &length;
bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
    fprintf(stderr, "\n param bind failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Supply data in chunks to server */
if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Supply the next piece of data */
if (mysql_stmt_send_long_data(stmt,0,
    " - The most popular Open Source database",40))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Now, execute the query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, "\n mysql_stmt_execute failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}
```

mysql_stmt_sqlstate()

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

説明

`stmt`によって規定された接続に対して、`mysql_stmt_sqlstate()`は、成功する場合もあり失敗する場合もあるAPI 機能に対するエラーコードから最近使用したものを返します。エラーコードは5つの文字から成り立っています。`'00000'`は「no error」を意味します。値はANSI SQLとODBCによって規定されています。可能な値のリストについては、[Errors, Error Codes, and Common Problems](#)をご参照ください。

すべてのMySQLエラーはもうSQLSTATEコードにマップされないことにご注目ください。値`'HY000'`(一般エラー)がマップされていないエラーナンバー用に使われます。

戻り値

SQLSTATEエラーコードを含むゼロで終わる文字ストリング。

`mysql_stmt_store_result()`

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

説明

結果セット(`SELECT``SHOW`, `DESCRIBE`, `EXPLAIN`)をうまく生成するすべてのステートメントのため および、クライアントに全結果セットを一時保存して、その後の`mysql_stmt_fetch()`コールが一時保存データを戻すようにして欲しい場合、`mysql_stmt_fetch()` を呼び出さなければなりません。

他のステートメントのため`mysql_stmt_store_result()`を呼び出すことは不要です。しかし、そうしても、性能を害するか傷つけることはありません。`mysql_stmt_result_metadata()`が`NULL`を戻すかチェックすることによって、ステートメントが結果セットを生成したか否かを検知することができます。明細については、`mysql_stmt_result_metadata()`をご参照ください。

注:MySQLは、`ysql_stmt_store_result()`をかなりスローダウンさせ、殆どのアプリケーションが`max_length`を必要としなくなるので、デフォルトによって、`YSQL_FIELD->max_length`を`max_length`中のすべてのカラムに対して計算しません。`max_length`を更新したい場合、`mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)`を呼び出してこれを有効にすることができます。`mysql_stmt_attr_set()`を参照してください。

戻り値

結果がうまく一時保存された場合、ゼロ。エラーが起こった場合、ゼロ以外。

エラー

- `CR_COMMANDS_OUT_OF_SYNC`

コマンドが妥当でないオーダーで実行されました。

- `CR_OUT_OF_MEMORY`

メモリ不足。

- `CR_SERVER_GONE_ERROR`

MySQLサーバが立ち去りました。

- `CR_SERVER_LOST`

サーバへの接続がクエリー中に失われました。

- `CR_UNKNOWN_ERROR`

未知のエラーが起きました。

準備されたC API ステートメントの問題

ここでは、準備されたステートメントに対して現在周知の問題のリストが後に続きます：

- `TIME`、`TIMESTAMP`、そして `DATETIME` 秒の部位をサポートしません。(たとえば`DATE_FORMAT()`から。
- 整数をストリングに変換するとき、MySQLサーバが先頭のゼロをプリントしない若干のケースでは、`ZEROFILL`は準備されたステートメントを使って支持されます。(例えば、`MIN(number-with-zerofill)`).
- クライアント中の浮動小数点数をストリングに変換するとき、変換された値の最右桁はオリジナルの値のそれらとは少し違うかもしれません。
- 準備されたステートメントは、クエリーがプレースホルダーを含んでいないケースでも、クエリーキャッシュを使いません。[クエリ キャッシュの動作](#)を参照してください。
- 準備されたステートメントはマルチステートメント ((即ち、`'`文字によって隔離されたシングルスートメントの中にあるマルチステートメント)をサポートしません。これは、準備されたステートメントは、マルチ結果セットをサポートしないので、結果セットを戻す記憶された手順を取り出すことができないことも意味します。

マルチプルステートメントを実行する C APIハンドリング

初期設定によって、`mysql_query()`及び`mysql_real_query()`は自分のステートメントストリング引数を、実行すべきシングルスートメントとして解釈し、あなたは結果を、ステートメントが結果セット(`SELECT`については一組の列または`INSERT`、`UPDATE`等に関する影響された列カウント)を生成するか否かに基づいて処理します。

MySQL 5.1はセミコロン(`;`)文字によって隔離されたマルチプルステートメントを含むストリングの実行をもサポートしています。この能力は、あなたが`mysql_real_connect()`を使ってサーバに接続するとき、もしくは`mysql_set_server_option()`を呼び出すことによる接続後、規定される特別オプションによって有効化されます。

マルチプルステートメント・ストリングを実行するとことによって、マルチプル結果セットまたは列カウント・インジケータを生成させることができます。これらの結果処理には、シングルスートメントの処理の場合と異なる方法が関与します。最初のステートメントから結果を処理した後、もっと多くの結果が存在するかどうか調べて、もしあった場合、それらを順番に処理することが必要です。マルチ結果の処理をサポートするため、C API には、`mysql_more_results()`機能と`mysql_next_result()`機能が含まれています。これらの機能は一般に、もっと多くの結果が得られる場合に限り、反復するループの終わりに使われます。結果の処理に失敗すると、この方法は、サーバへの接続をドロップさせる結果を生む恐れがあります。

`CALL`ステートメントを記憶された手順に対して実行する場合、マルチ結果処理も必要です。記憶された手順は、終わるとき状態結果を戻しますが、運転(例えば、`SELECT`ステートメントの実行)するとき結果セットを生成します。最終ステータスに加え、結果セットを生成する記憶された手順のために、マルチ結果を復元するよう準備しなければなりません。

マルチステートメント機能とマルチ結果機能は`mysql_query()`または`mysql_real_query()`と一緒にだけ使うことができます。それらは準備されたステートメント・インタフェースと一緒に使うことができません。準備されたステートメントハンドルは一つのステートメントを含むストリングだけを使って働くハンドルと定義されます。[準備されたC API ステートメント](#)。を参照してください。

マルチステートメントの実行と結果の処理を有効にするには、次のオプションを使うことができます：

- `mysql_real_connect()`機能には、2つのオプション値が関連する `flags` 引数が含まれています。
 - `CLIENT_MULTI_RESULTS`はクライアントプログラムがマルチ結果を処理することを可能にします。このオプションmustは、結果セットを生成する記憶された手順のために`CALL`ステートメントを実行する場合有効化されます。さもなければ、このような手順はエラー `Error 1312 (0A000): PROCEDURE proc_name can't return a result set in the given context`.
 - `CLIENT_MULTI_STATEMENTS`は`mysql_query()`および`mysql_real_query()`にセミコロンで仕切られたマルチステートメントを含むステートメントストリングを実行することを可能にします。このオプションは、暗黙に`CLIENT_MULTI_RESULTS`を有効にするので、`mysql_real_connect()`に対する`CLIENT_MULTI_STATEMENTS`の`flags`引数は、`CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS`の引数と同等になります。即ち、`CLIENT_MULTI_STATEMENTS`はマルチステートメントの実行とマルチ結果の処理を十分に有効化します。
- サーバに対する接続が確立された後、それを`MYSQL_OPTION_MULTI_STATEMENTS_ON`または`MYSQL_OPTION_MULTI_STATEMENTS_OFF`の引数に渡すことによって、マルチステートメントの実行を可能もしくは不能にするため、`mysql_set_server_option()`機能を使用することができます。この機能を使ってマルチス

ステートメントの実行を有効にすると、各ステートメントが1個の結果を生成するマルチステートメントストリングに対する「simple」結果の処理も有効化されませんが、notは結果セットを生成する記憶された手順を許すに十分です。

次の手順はマルチステートメントの取り扱いに対して示唆された戦略を概説したものです。

1. `CLIENT_MULTI_STATEMENTS`を`mysql_real_connect()`に渡して、マルチステートメントの実行とマルチ結果の処理を有効化してください。
2. `mysql_query()`または`mysql_real_query()`を呼び出し、それが成功したことを確認した後、ステートメントの結果を処理するループを入力してください。
3. ループを繰り返して入力するたびに、現在のステートメントの結果を処理して、結果セット又は影響された列カウントを復元してください。エラーが起こったら、ループを終了してください。
4. ループの終わりに、他の結果が存在するかどうか調べ、もしある場合、修正を先導するため、`mysql_next_result()`を呼び出してください。それ以上結果が得られなくなったら、ループを終了してください。

戦略に関して、実行可能な1つの例を以下で紹介します。

```
/* connect to server with option CLIENT_MULTI_STATEMENTS */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
    printf("mysql_real_connect() failed\n");
    mysql_close(mysql);
    exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
    "DROP TABLE IF EXISTS test_table;\n"
    "CREATE TABLE test_table(id INT);\n"
    "INSERT INTO test_table VALUES(10);\n"
    "UPDATE test_table SET id=20 WHERE id=10;\n"
    "SELECT * FROM test_table;\n"
    "DROP TABLE test_table");

if (status)
{
    printf("Could not execute statement(s)");
    mysql_close(mysql);
    exit(0);
}

/* process each statement result */
do {
    /* did current statement return data? */
    result = mysql_store_result(mysql);
    if (result)
    {
        /* yes; process rows and free the result set */
        process_result_set(mysql, result);
        mysql_free_result(result);
    }
    else /* no result set or error */
    {
        if (mysql_field_count(mysql) == 0)
        {
            printf("%ld rows affected\n",
                mysql_affected_rows(mysql));
        }
        else /* some error occurred */
        {
            printf("Could not retrieve result set\n");
            break;
        }
    }
} while ((status = mysql_next_result(mysql)) > 0);
/* more results? -1 = no, >0 = error, 0 = yes (keep looping) */
if ((status = mysql_next_result(mysql)) > 0)
    printf("Could not execute statement\n");
} while (status == 0);
```



```
mysql_close(mysql);
```

ループの最後の部分を、`mysql_next_result()`が非ゼロを戻すかどうかの単純なテストに縮小することができます。書き込まれたコードは、更なる結果とエラーを区別し、これによって、メッセージに後者の発生をプリントすることを許します。

日付とタイム値のC API式取り扱い

バイナリー(準備されたステートメント)プロトコールはあなたに、日付値とタイム値(`DATE`、`TIME`、`DATETIME`および`TIMESTAMP`)を、`MYSQL_TIME`構造を使って、送り且つ受け取ることを許します。この構造のメンバーは準備されたC APIステートメントデータタイプで記述されます。

時間のデータ値を送るため、`mysql_stmt_prepare()`を使って準備されたステートメントを生成してください。その後、`mysql_stmt_execute()`を呼び出して、ステートメントを実行する前に、以下の手順を実行して各時間パラメータをセットしてください。

1. データ値に関連する`MYSQL_BIND`構造の中に、`buffer_type`メンバーを、送ろうとしている時間値の種類を示すタイプにセットしてください。`DATE`値、`TIME`値、`DATETIME`値または`TIMESTAMP`値に対して、`buffer_type`を`MYSQL_TYPE_DATE`、`MYSQL_TYPE_TIME`、`MYSQL_TYPE_DATETIME`または`MYSQL_TYPE_TIMESTAMP`にそれぞれセットしてください。
2. `MYSQL_BIND`構想の`buffer`メンバーを、時間値を渡す`MYSQL_TIME`構造のアドレスにセットしてください。
3. 渡すべき時間値のタイプに適した`MYSQL_TIME`構造のメンバーを書き込んでください。

`mysql_stmt_bind_param()`を使ってパラメータデータをステートメントに固定(バインド)してください。こうすると、`mysql_stmt_execute()`を呼び出すことが可能になります。

時間値を復元する手順は、`buffer_type`メンバーを受け取りを期待する値のタイプにセットし更に、`buffer`メンバーを戻り値を置くべき`MYSQL_TIME`構造のアドレスにセットする以外、同じです。`mysql_stmt_execute()`を呼び出した後、結果をフェッチする前に、バッファをステートメントに固定(バインド)する`mysql_bind_results()`を使ってください。

`DATE`データ、`TIME`データ並びに`TIMESTAMP`データを挿入する簡単な例がここにあります。`mysql`変数は有効な接続ハンドルと見なされます。

```
MYSQL_TIME ts;
MYSQL_BIND bind[3];
MYSQL_STMT *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
            timestamp_field) VALUES(?,?,?);");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, "mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);
```

```

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..

```

C APIスレッド機能の説明

スレッドを付けたクライアントを生成させたいとき、あなたは次の機能を使う必要があります。[スレッド付きクライアントを作る方法](#)を参照してください。

my_init()

```
void my_init(void)
```

説明

この機能はMySQL機能呼び出す前に一回プログラム中に呼び出される必要があります。[my_init\(\)](#)はMySQLが必要とする幾つかのグローバル変数を初期化します。スレッドに安全なクライアントライブラリを使っている場合、それはこのスレッドのために、[mysql_thread_init\(\)](#)も呼び出します。

[my_init\(\)](#) は[mysql_init\(\)](#)、[mysql_library_init\(\)](#)、[mysql_server_init\(\)](#)、そして[mysql_connect\(\)](#)に自動的に呼び出されます。

[my_init\(\)](#)にアクセスするため、プログラムには、[my_sys.h](#)ヘッダーファイルを含めなければなりません:

```
#include <my_sys.h>
```

戻り値

なし。

mysql_thread_init()

```
my_bool mysql_thread_init(void)
```

説明

スレッドに固有名変数を初期化するため、生成された各スレッドごとに、この機能が呼び出される必要があります。

[mysql_thread_init\(\)](#)は、[my_init\(\)](#) と [mysql_connect\(\)](#)に自動的に呼び出されます。

戻り値

成功している場合ゼロ。エラーが起こった場合、ゼロ以外。

mysql_thread_end()

```
void mysql_thread_end(void)
```

説明

[pthread_exit\(\)](#)を呼び出す前に、[mysql_thread_init\(\)](#)によって割り当てられたメモリーを解放するため、この機能を呼び出す必要があります。

この[mysql_thread_end\(\)](#) クライアントライブラリによって自動的に取り出されないことにご注目ください。それは記憶漏れを避けるために明示的に呼び出されなくてはなりません。

戻り値

なし。

mysql_thread_safe()

```
unsigned int mysql_thread_safe(void)
```

説明

この機能は、スレッドにとって安全であるようにクライアントが編集されるか否かを示します。

戻り値

クライアントがスレッドにとって安全であるなら 1、さもなければ 0。

埋め込まれたC API機能の説明

MySQLアプリケーションには、埋め込みサーバーを使うように書き込むことができます。[埋め込まれたMySQLサーバライブラリ](#)、`libmysqld`を参照してください。それをこのようなアプリケーションを書き込むため、`libmysqld`フラグを使うことによって、`ibmysqlclient`クライアントライブラリに対してリンクしないで、`-lmysqld`ライブラリに対して、`ibmysqld`フラグを使ってリンクしなければなりません。しかしながら、ライブラリを初期化し且つ完成させる呼び出しは、クライアントアプリケーションをあなたが書き込むか、埋込サーバーを使うものが書き込むかに関係なく、同じです。ライブラリを初期化する`mysql_library_init()`を呼び出し、必要な場合、`mysql_library_end()`も呼び出してください。[C API機能の概要](#)。を参照してください。

mysql_server_init()

```
int mysql_server_init(int argc, char **argv, char **groups)
```

説明

この機能はMySQLライブラリを初期化します。この場合、当該初期化は他のMySQL機能が呼び出される前に実行しなければなりません。しかし、`mysql_server_init()`は忌避されるので、代わりに、`mysql_library_init()`を呼び出すべきです。`mysql_library_init()`を参照してください。

戻り値

Okなら 0、エラーが起これば 1。

mysql_server_end()

```
void mysql_server_end(void)
```

説明

この機能はMySQLライブラリを完成化させます。この場合、それをライブラリの使用が必要になったとき行うことができます。しかし、`mysql_server_init()`は忌避されるので、代わりに、`mysql_library_end()`を呼び出すべきです。`mysql_library_end()`を参照してください。

戻り値

なし。

自動再接続挙動の管理

実行されるべきステートメントをサーバーに送ろうと試みるとき、接続がダウンしているとわかると、MySQLクライアントライブラリはサーバーのため、自動再接続を実施します。この場合、ライブラリはサーバーに再接続しようとして一回試み、ステートメントを再び送ります。

自動再接続は自分の再接続コートを搭載する必要がないので、便利であるが、再接続が起これば、接続状態の幾つかの局面がリセットされ、それがアプリケーションに知らされません。接続が関係する状態は、次のように影響を与えます：

- クティブな取引がすべてロールバックされ、オートコミットモードがリセットされます。
- すべてのテーブルロックが解放されます。

- すべての**TEMPORARY** テーブルが閉じ (且つドロップ)されます。
- セッション変数が対応する変数値に再初期設定されます。これは**SET NAMES**のようなステートメントによって暗黙にセットされる変数に影響を与えます。
- ユーザー可変の設定が失われます。
- 準備されたステートメントがリリースされます。
- **HANDLER** 変数が閉じられます。
- **LAST_INSERT_ID()** の値が 0 にリセットされます。
- **GET_LOCK()** を使って取得されたロックが解放されます。

接続のドロップを知り、(閉じるか状態情報の喪失に対して調整アクションを起こせるようにする)ことがアプリケーションにとって重要である場合、自動再接続が無効になっていることを確認してください。**MYSQL_OPT_RECONNECT** オプションを使って、**mysql_options()** を呼び出すことによって、これを暗黙に実行することができます。

```
my_bool reconnect = 0;
mysql_options(&mysql, MYSQL_OPT_RECONNECT, &reconnect);
```

MySQLでは5.1、自動再接続はデフォルトによって無効化されています。

C APIを使うときよく尋ねられる質問と問題

なぜ**mysql_store_result()**時々返す**NULL**の後**mysql_query()** 成功を返す

mysql_query()の呼び出しが成功した直ぐ後、**mysql_store_result()**が**NULL**を返すことは可能です。これが起きるとき、それは次の状態の1つが起こったことを意味します：

- **malloc()**機能停止(例えば、結果セットが大きすぎた場合)が起こりました。
- データが読み込めませんでした。(接続の上に起こったエラー)
- クエリーがデータを戻さなかった(例えば、**INSERT**、**UPDATE**または**DELETE**)。

mysql_field_count()を呼び出すことによって、ステートメントが空でない結果を返すべきか否かをいつでもチェックすることができます。**mysql_field_count()**はゼロを返し、結果は空で、最後のクエリーは結果値を戻さないステートメントでした。(例えば、**INSERT**または**DELETE**)。 **mysql_field_count()**がゼロでない値を戻す場合、ステートメントは空でない結果を返していなければなりません。については、**mysql_field_count()**機能の説明をご参照ください。

mysql_error()または**mysql_errno()**を呼び出すことによって、エラーがないかテストすることができます。

クエリーからどんな結果を得ることができるか

クエリーによって返される結果セットほかに、以下の情報も受けとることができます：

- **mysql_affected_rows()**は、**INSERT**、**UPDATE**または**DELETE**を実行中に、最後のクエリーによって影響を受けた列のナンバーを返します。

高速再生成のために、**TRUNCATE TABLE**を使ってください。

- **mysql_num_rows()** は結果セット中に列のナンバーを返します。**mysql_store_result()**を使って、**mysql_num_rows()**を、**mysql_store_result()**が返るとすぐ、戻すことができます。**mysql_use_result()**を使って、**mysql_num_rows()**を、すべての列を**mysql_fetch_row()**を使ってフェッチした後、戻すことができます。
- **mysql_insert_id()**は、**AUTO_INCREMENT**インデックスを使ってテーブル中に列を挿入した最後のクエリーによって生成されたIDを返します。**mysql_insert_id()**を参照してください。
- 幾つかのクエリー(**LOAD DATA INFILE ...**、**INSERT INTO ... SELECT ...**、**UPDATE**) は追加情報を返します。結果は**mysql_info()**によって戻されます。それが戻すストリングのフォーマットについては、**mysql_info()**に対する説明をご参照ください。追加情報がない場合、**mysql_info()**は**NULL**ポインターを返します。

最後に挿入された列に対するユニークIDを取得する方法

AUTO_INCREMENTカラムを含むテーブルの中に記録を挿入する場合、`mysql_insert_id()`機能呼び出すことによって、そのカラムの中に記憶された値を取得することができます。

C アプリケーションから、或る値が(あなたがステートメントが成功したと判定したとみなす)以下のコードを実行することによって、**AUTO_INCREMENT**カラム中に記憶されたか否かをチェックすることができます。それは、クエリーが**AUTO_INCREMENT**インデックスを含む**INSERT**であったか否かを査定します：

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

新しい**AUTO_INCREMENT**値が生成されたとき、`mysql_query()`を使って、**SELECT LAST_INSERT_ID()**を実行し、ステートメントによって、結果セットからその値を複製することによって、それを取得することができます。

複数の値を差し込むとき、自動的に増加した最後の値が戻されます。

LAST_INSERT_ID()のために、最近生成されたIDが接続ごとにサーバ中に維持されます。それは他のクライアントによって変更されません。それは、**AUTO_INCREMENT**カラムを非マジック値(即ち、**NULL**でも**0**でもない値)を使って更新する場合でも変更されません。**LAST_INSERT_ID()**カラムと**AUTO_INCREMENT**カラムを複数のクライアントから同時に使うことは完全に有効です。各クライアントは、thatクライアントが実行した最後のステートメントに対するIDを受け取ります。

1つのテーブルのために生成されたIDを使って、それを2番目のテーブルに挿入したい場合、このようなSQLステートメントを使うことができます：

```
INSERT INTO foo (auto,text)
VALUES(NULL,'text'); # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
VALUES(LAST_INSERT_ID(),'text'); # use ID in second table
```

その値が**NULL**または**0**を記憶することによって、自動的に生成された値または明確な値として規定された値であっても、`mysql_insert_id()`が**AUTO_INCREMENT**カラムの中に記憶された値を戻すことにご注目ください。**LAST_INSERT_ID()**は自動的に生成された**AUTO_INCREMENT**値だけを戻します。**NULL**または**0**ではなく、明示された値を記憶させる場合、それは、**LAST_INSERT_ID()**によって戻された値に影響を与えません。

AUTO_INCREMENTカラム中に最後のIDの取得する方法に関する明細情報:

- SQLステートメントの中で使われる**LAST_INSERT_ID()**に関する情報については、[??? \[637\]](#)をご参照ください。
- `mysql_insert_id()`に関する情報、C APIの中から使用する機能については、[mysql_insert_id\(\)](#)をご参照ください。
- Connector/Jを使用するとき、自動的に増加した値を取得する方法に関する情報については、[Connector/J に関する注記とヒント](#)をご参照ください。
- Connector/ODBCを使用するとき、自動的に増加した値を取得する方法に関する情報については、[オートインクリメント値の獲得](#)をご参照ください。

C APIにリンクする問題

C APIとリンクするとき、次のエラーが幾つかのシステム上で起こる恐れがあります。

```
gcc -g -o client test.o -L/usr/local/lib/mysql \
    -lmysqlclient -lsocket -lnsl

Undefined      first referenced
symbol         in file
floor          /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

これがシステム上に起こった場合、**-lm**をcompile/linkラインの終わりに追加することによって、計算ライブラリを含めなければなりません。

クライアントプログラムの構築

自分で書いたか、第三者から取得するMySQLクライアントを編集する場合、これらはリンクコマンド中に `-lmysqlclient -lz` オプションを使ってリンクされなければなりません。 `-L` オプションを規定して、リンカーにどこでライブラリを見つけるか告げる必要があるかもしれません。例えば、ライブラリを `/usr/local/mysql/lib` の中にインストールする場合、 `-L/usr/local/mysql/lib -lmysqlclient -lz` をリンクコマンド中で使ってください。

MySQLヘッダーファイルを使うクライアントのために、これら (例えば、 `-I/usr/local/mysql/include`) を編集するとき、編集者がヘッダーファイルを見つけることができるように、 `-I` オプションを規定する必要があるかもしれません。

Unix上でMySQLプログラムをコンパイルすることを単純にするため、我々はあなたに `mysql_config` スクリプトを提供しました。 [mysql_config — コンパイルオプションをコンパイルクライアントのために \(用\) 取得してください](#)。を参照してください。

MySQLクライアントを編集するために、それを次のように使うことができます：

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname '$CFG --cflags' progname.c '$CFG --libs'"
```

`sh -c` は、アウトプットを1つのワードとして `mysql_config` から処理しないシェルを取得するすることを必要とします。

スレッド付きクライアントを作る方法

クライアントライブラリはスレッドに対して殆ど安全です。最も大きい問題は、ソケットから読み込んだ `net.c` 中のサブルーチンが割り込みに対して安全でないということです。これは、長い読み取りをブレイクすることができる自分自身の警報装置を持ちたいという考えで実施されました。 `SIGPIPE` 中断のために、中断ハンドラーをインストールする場合、ソケットの取り扱いをスレッドに対して安全にすべきです。

接続が終わるとき、プログラムが中断されるのを避けるため、MySQLは、 `mysql_library_init()`、 `mysql_init()` または `mysql_connect()` 上の `SIGPIPE` の最初の呼び出しをブロックします。自分自身の `SIGPIPE` ハンドラーを使いたい場合、先ず `mysql_library_init()` を呼び出して、それからハンドラーをインストールすべきです。

我々のウェブサイト (<http://www.mysql.com/>) で配布している古いバイナリー中では、Windowsのためのそれら以外のクライアントライブラリはスレッドに対して安全なオプションを使って通常にコンパイルされません。新しいバイナリーには、通常のライブラリとスレッドに対して安全なライブラリが両方含まれています。

他のスレッドからクライアントを遮断でき、MySQLサーバーと語るとき、タイムアウトをセットできる、スレッド付きクライアントを取得するため、サーバーが使っている `net_serv.o` コードおよび `mysys` ライブラリ、 `mystrings` ライブラリおよび `ldbug` ライブラリを使うべきです。

中断やタイムアウトが必要でない場合、スレッドに安全なクライアントライブラリ (`mysqlclient_r`) を編集して使うことができます。 [MySQL C API](#) を参照してください。この場合、 `net_serv.o` オブジェクトファイルあるいは他のMySQLライブラリについて心配する必要はありません。

スレッド付きクライアントを使い、タイムアウトと中断を使いたい時、あなたは `thr_alarm.c` ファイル中にそのルーチンの大いに使用をすることができます。ルーチンを `mysys` ライブラリから使っている場合、忘れてはならない唯一のことは、 `my_init()` を真っ先にび出すことです。 [C APIスレッド機能の説明](#) を参照してください。

全ての場合において、他のMySQL機能呼び出す前に、 `mysql_library_init()` を呼び出すことによって、クライアントライブラリを初期化したか確認してください。ライブラリが必要となったとき、 `mysql_library_end()` を呼び出してください。

`mysql_real_connect()` を除くすべての機能はデフォルトによって、スレッドに対して安全です。次のノートはどのようにスレッドにとって安全なクライアントライブラリをコンパイルして、スレッドにとって安全な方法でそれを使うべきか述べています。 (`mysql_real_connect()` のための次のノートは、 `mysql_connect()` が忌避されるので、実際に `mysql_connect()` に適用します。)

`mysql_real_connect()` をスレッドに対して安全にするため、MySQL ディストリビューションにこのコマンドを設定しなければなりません。

```
shell> ./configure --enable-thread-safe-client
```


その後、配分を編集して、スレッドに安全なクライアントライブラリ、`libmysqlclient_r`を生成させてください。(オペレーティング・システムにスレッドに安全な`gethostbyname_r()`機能が含まれていると想定します。)このライブラリは接続毎に、スレッドに対して安全になっています。2つのスレッドに、次の警告と使って同じ接続を共有させることができます：

- 2つのスレッドは同じ接続上で同時にMySQLサーバにクエリーを送ることはできません。`mysql_query()`に対する呼び出しと`mysql_store_result()`に対する呼び出しの間に、他のスレッドが同じ接続に使われていないか特に確認してください。
- 多くのスレッドは`mysql_store_result()`を使って復元される異なった結果セットにアクセスすることができます。
- `mysql_use_result`を使う場合、結果セットが閉じられるまで、他のスレッドが同じ接続を使っていないことを確認してください。しかしながら、スレッドクライアントが`mysql_store_result()`を使っている同じ接続を共有することは本当に最も良いことです。
- 同じ接続に複数のスレッドを使いたい場合、`mysql_query()`と`mysql_store_result()`の呼び出しペアの回にあるりにmutexをロックさせなければなりません。`mysql_store_result()`の準備が終わった途端に、ロックを解放することが出来、他のスレッドが同じ接続を尋ねることが許されます。
- POSIXスレッドを使う場合、`pthread_mutex_lock()`および`pthread_mutex_unlock()`をmutexロックを確立し且つ解放するのに使うことができます。

MySQLデータベースに接続を生成しなかったMySQL機能呼び出ししているスレッドを持っている場合、以下を知る必要があります。

`mysql_init()`または`mysql_connect()`を呼び出すとき、MySQLは、(他のものの中から選んだ)デバグライブラリによって使われているスレッドのために、スレッドに固有な変数を生成します。

スレッドが`mysql_init()`または`mysql_connect()`呼び出した後、MySQL機能呼び出す場合、スレッドには必要なスレッドに固有な変数を定位置に含んでいないので、遅かれ早かれコアダンプになる可能性があります。

順調に働かせるものを手に入れるため、あなたは次のことをしなければなりません：

1. それが他のMySQL機能でも呼び出す場合、`mysql_real_connect()`を呼び出す前のプログラムの立ち上げで、`my_init()`を呼び出してください。
2. MySQL機能呼び出す前に、スレッドハンドラーの中に、`mysql_thread_init()`を呼び出してください。
3. `mysql_thread_end()`を呼び出す前に、スレッドの中に`pthread_exit()`を呼び出してください。これはMySQLスレッドに固有な変数によって使われたメモリを解放します。

クライアントを「undefined symbol」とリンクするとき、`libmysqlclient_r`エラーが発生した場合、殆どの場合これは、リンク/コンパイルコマンド上にスレッドライブラリを含めなかったことに起因して発生します。

MySQL PHP API

PHPはダイナミックなWeb ページを作るために使うことが出来るサーバ サイドのHTML用埋め込み式スクリプト言語です。それは殆どのオペレーティング・システムやWebサーバの要求を満たし、MySQLを含む共通データベースの殆どにアクセスすることができます。PHPは独立したプログラムとして運転するか、もしくはアパッチWebサーバと一緒に使用するためのモジュールとして翻訳することができます。

PHPは実際に 2 つの異なったMySQL APIエクステンションを提供します。

- `mysql`: PHPのバージョン4と5の要件を満たすこのエクステンションは、MySQL 4.1より前のバージョンのMySQLと一緒に使用することを目的としたものです。この拡張子はMySQL5.1に使われている改良認証プロトコルも、準備されたステートメントあるいは複数のステートメントもサポートしません。このエクステンションをMySQL5.1と一緒に使いたい場合、`--old-passwords`オプションを使うために、MySQLサーバを設定したくなるでしょう。(Client does not support authentication protocolをご参照ください) この拡張子は <http://php.net/mysql>のPHPウェブサイトに記載されています。
- `mysqli` - は「改良されたMySQL」をサポートし、;それはMySQL. 1. 1 およびそのその後バージョンで使用するよう意図されています。このエクステンションはMySQLとその現シリーズの中で使用される検証プロトコル5.1並

びに準備されたステートメント用APIとマルチステートメント用APIを完全にサポートしています。これに加え、このエクステンションは進歩したオブジェクト指向のプログラミングインタフェースをも提供します。`mysqli`エクステンションのために書かれた文書を<http://php.net/mysqli>で読むことができます。更に明細な情報は<http://www.zend.com/php5/articles/php5-mysqli.php>に掲載してありますのでご覧ください。

Linux上にPHPを構築する時、`mysql`エクステンションと`mysqli`エクステンションの両方が有効化される問題を経験した場合には、`mysql` と `mysqli` の両方を PHP内で可能にするをご参照ください。

PHPとその資料は、[PHP ウェブサイト](#)から入手することができます。MySQLはWindowsオペレーティング・システムのための`mysql`エクステンションと`mysqli`エクステンションをに提供します。MySQLが提供するエクステンションの好ましい使用法については、同ウェブページをご覧ください。

MySQLとPHPに対する共通問題

- **Error: Maximum Execution Time Exceeded**これは`php.ini`ファイルに入るPHPの限界で、ここに、必要とに応じて、30秒からそれより若干長い最高実行時間を設定します。スクリプト毎に許されるRAMを8MBにする代わりに、2倍の16MBにすることは悪い考えではありません。
- **Fatal error: Call to unsupported or undefined function mysql_connect() in ...:**これは、PHPバージョンがMySQLをサポートするように編集されていないことを意味します。ダイナミックMySQLモジュールを編集して、それをPHPに装着するか、あるいは組み込みのMySQLサポートを使って編集することができます。このプロセスはPHPマニュアルに詳述されています。
- **Error: Undefined reference to 'uncompress':**これは、クライアント・ライブラリが圧縮されたクライアント/サーバ・プロトコルに対するサポートを使って編集されていることを意味します。その解決策は、`-lmysqlclient`と結合するとき、`-lz`を最後に加えることです。
- **Error: Client does not support authentication protocol:**これは、MySQL 4.4.1がそれより新しいバージョンのMySQLを使って古い`mysql`エクステンションを使おうとする時、最もしばしば起こります。可能な解決は：MySQL 4.0にグレードを下げ、PHP5およびもっと新しい`mysqli`エクステンションに切り替えるか、あるいは`--old-passwords`の付いたMySQLサーバに構成を設定することです。明細な情報については、[Client does not support authentication protocol](#)をご参照ください。

PHP4の旧式なコードを持つそれらには、このような、古いMySQLライブラリと新しいMySQLライブラリに対する互換性レイヤを利用することができます：<http://www.coggeshall.org/oss/mysq2i>。

`mysql` と `mysqli` の両方を PHP内で可能にする

Linux上にPHPを構築する時、`mysql`エクステンションと`mysqli`エクステンションの両方が有効化される問題を経験した場合には、以下の手順を試してみるべきです。

1. PHPをこのように設定してください。

```
./configure --with-mysql=/usr/bin/mysql_config --with-mysql=/usr
```

2. `Makefile`を編集し、`EXTRA_LIBS`を編集し、探してください。それは(すべてが1行の上にある)これのように見えるかもしれません。

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
             -lxml2 -lz -lm -lxml2 -lz -lm -lmysqlclient -lz -lcrypt -lnsl -lm
             -lxml2 -lz -lm -lcrypt -lxml2 -lz -lm -lcrypt
```

ラインが(すべてが1行の上にある)これのように見えるように、すべての重複部分を除去してください：

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
             -lxml2
```

3. PHPを構築して搭載してください。

```
make
make install
```

MySQL Perl API

PerlDBIモジュールはデータベースにアクセスする一般的なインタフェースを提供します。そのまま、多くの異なったデータベースエンジンで作動するDBIスクリプトを書くことができます。DBIを使うためにあなたは、DBIモジュール並びにアクセスしたい各タイプのサーバごとに、データベースドライバ (DBD) モジュールをインストールしなくてはなりません。MySQLの場合、このドライバはDBD::mysqlモジュールです。

Perl DBIは推薦されたPerlインタフェースです。それは、時代遅れとみなされるべき古い(タイプの)インタフェースであるmysqlperlを置き換えます。

Perl DBIサポートをインストールするためのインストラクションはPerl のインストールに関する注釈に掲載されていますのでご覧ください。

DBIに関する情報はコマンドライン、オンラインで入手するか、あるいは印刷物として取得することができます。

- DBIとDBD::mysqlモジュールをインストールさせるとすぐ、コマンドラインにおいてperldocコマンドを使って、それらに関する情報を得ることができます。

```
shell> perldoc DBI
shell> perldoc DBI::FAQ
shell> perldoc DBD::mysql
```

この情報を、pod2man、pod2html等を他のフォーマットに翻訳するためにも使うことができます。

- Perl DBIに関するオンライン情報を得るために、DBI Webサイト<http://dbi.perl.org/>をご訪問ください。そのサイトは一般的DBIメーリングリストのホストとしての機能を具備しています。MySQL ABは DBD::mysqlに特に関するリストを主催しています。MySQL メーリング リストをご参照ください。
- 印刷された情報を掲載した公式DBIブックは、Programming the Perl DBI (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000)です。このブックに関する情報はWebサイト、<http://dbi.perl.org/>から入手することができます。

MySQLを使ったDBIの使用に特に焦点をあてた情報を見たい場合には、MySQL and Perl for the Web (Paul DuBois, New Riders, 2001)をご参照ください。このブックのWebサイトは<http://www.kitebird.com/mysql-perl/>です。

MySQL C++ API

MySQL++はC++のためのMySQL APIです。Warren Youngがこのプロジェクトを引き継ぎました。更に明細な情報は以下に掲載してありますのでご覧ください。<http://www.mysql.com/products/mysql++/>。

MySQL Python API

MySQLdbは Python DB API バージョン2.0に準拠した Python用MySQLサポートを提供します。それは<http://sourceforge.net/projects/mysql-python/>に掲載してありますのでご覧ください。

MySQL Tcl API

MySQLtclはTclプログラミング言語からMySQLデータベースサーバーにアクセスするためのシンプルなAPIです。それは<http://www.xdobry.de/mysqltcl/>に掲載してありますのでご覧ください。

MySQLエッフェルラッパー

Eiffel MySQLは、Michael Ravitsが書いたEiffelプログラミング言語を使ったMySQLデータベースサーバに対するインタフェースです。それは<http://efsa.sourceforge.net/archive/ravits/mysql.htm>に掲載してありますのでご覧ください。

MySQLプログラム開発ユーティリティー

このセクションで、MySQLプログラムを開発する時、あなたにとって有用であると思われる幾つかのユーティリティーについて説明します。

- [mysql2mysql](#)

mSQLプログラムをMySQLに変換するシェルスクリプト。それはすべてのケースを扱うわけではありませんが、変換するとき良いスタートを与えます。

- `mysql_config`

MySQLプログラムを翻訳するとき必要なオプション値を作り出すシェルスクリプト。

mysql2mysql — MySQLと一緒に使うため、mSQLプログラムを変換してください。

MySQL APIは最初、mSQLデータベースシステムのためのそれに非常に類似するように開発されました。このため、mSQLプログラムはMySQLと一緒に使用するため、C API機能の名前を変えることによって、比較的容易にしばしば変換することができます。

mysql2mysqlユーティリティプログラムは、mSQL C API機能コールのそのMySQL同等言語への変換を実施し、mysql2mysqlはインプットファイルを定位置に変換して、それを変換する前に、オリジナルのコピーを作ります。例えば、次のコマンドをmysql2mysqlのように使用してください。

```
shell> cp client-prog.c client-prog.c.orig
shell> mysql2mysql client-prog.c
client-prog.c converted
```

その後、`client-prog.c`を調べて、必要に応じて変換後の修正を行ってください。

mysql2mysqlは機能名を取り替えるため、`replace`ユーティリティを使います。詳しくは[replace — 文字列置き換えユーティリティ](#)を参照してください。

mysql_config — コンパイルオプションをコンパイルクライアントのために (用に) 取得してください。

mysql_configはMySQLクライアントを編集して、それをMySQLに接続するのに有用な情報をあなたに提供します。

mysql_configは次のオプションをサポートします。

- `--cflags`

検知すべきコンパイラフラグは、ファイルとクリティカルなコンパイラフラグ並びに `libmysqlclient`ライブラリを編集するとき使う定義等です。

- `--include`

MySQLを見いだすコンパイラオプションにはファイルが含まれています。(通常、このオプションの代わりに`--cflags`を使うことにご注意ください。)

- `--libmysql-lib, --embedded`

MySQLが埋め込まれているサーバとリンクするのに必要なライブラリとオプション。

- `--libs`

MySQLクライアントライブラリとリンクするのに必要なライブラリとオプション。

- `--libs_r`

スレッドに対して安全なMySQLクライアントライブラリとリンクするのに必要なライブラリとオプション。

- `--port`

MySQLを配置するとき規定されたデフォルトTCP/IPポート番号。

- `--socket`

MySQLを配置するとき規定されたデフォルトUnixソケットファイル。

- `--version`

配付されたMySQLのバージョン番号。

オプションの付いていない`mysql_config`を呼び出すと、それがサポートするすべてのオプションとそれらの値のリストが表示されます。

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
--cflags      [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
--include     [-I/usr/local/mysql/include/mysql]
--libs        [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
              -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
--libs_r      [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
              -lpthread -lz -lcrypt -lnsl -lm -lpthread]
--socket      [/tmp/mysql.sock]
--port        [3306]
--version     [4.0.16]
--libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                -lcrypt -lnsl -lm -lpthread -lrt]
```

`mysql_config`をコンマンドラインの中で使って、それが特定のオプションに対して表示する値を含めることができます。例えば、MySQLクライアントプログラムをコンパイルするために、`mysql_config`を次のように使います。

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname ` $CFG --cflags ` progname.c ` $CFG --libs `"
```

このような方法で`mysql_config`を使う時、バックチック(`)文字の中にそれが呼び出されているか確認してください。それは、シェルに、それを実行し、周囲のコマンドの中にそのアウトプットを代替えすることを告げます。

第24章 MySQL コネクタ

目次

MySQL Connector/ODBC	1236
Connector/ODBC の概要	1236
Connector/ODBC のインストール	1240
Connector/ODBC の構成	1259
Connector/ODBC 例	1275
Connector/ODBC 参考資料	1301
Connector/ODBC に関する注釈とヒント	1306
Connector/ODBC サポート	1315
MySQL Connector/NET	1316
Connector/NET のバージョン	1317
Connector/NET のインストール	1317
Connector/NET の例と使用ガイド	1323
Connector/NET に関する注記とヒント	1367
Connector/NET のサポート	1375
MySQL Visual Studio プラグイン	1376
MySQL Visual Studio プラグインのインストール	1376
MySQL サーバとの接続の作成	1378
MySQL Visual Studio プラグインの使用	1379
Visual Studio プラグインのサポート	1387
MySQL Connector/J	1387
Connector/J バージョン	1388
Connector/J のインストール	1388
Connector/J 例	1392
Connector/J (JDBC) の参考	1392
Connector/J に関する注記とヒント	1408
Connector/J のサポート	1425
Connector/PHP	1427

この章では、クライアントプログラムから MySQL サーバへ接続するドライバ、MySQL コネクタについて説明します。現在使用されている MySQL コネクタは 5 種類あります。

- Connector/ODBC は、Open Database Connectivity (ODBC) API を使用して MySQL サーバへ接続する際にドライバをサポートします。サポートは Windows、Unix、および Mac OS X プラットフォームで利用できます。
- Connector/NET は、MySQL データベースに保存されたデータを使用する .NET アプリケーションの作成を可能にします。Connector/NET は完全な ADO.NET インターフェースを実装し、ADO.NET Aware ツールとの併用を支援します。開発者はサポートされた .NET 言語で、Connector/NET を使用するアプリケーションを作成することができます。
- MySQL Visual Studio プラグインは、Connector/NET と Visual Studio 2005 にインストールすることができます。このプラグインは MySQL DDEX プロバイダで、Visual Studio 内でスキーマやデータ操作ツールを使用して MySQL データベース内のオブジェクトを作成および編集することができます。
- Connector/J は、Java アプリケーション から標準 Java Database Connectivity (JDBC) API を使用して MySQL に接続する際にドライバをサポートします。
- Connector/MXJ は、Java アプリケーションを通して MySQL サーバとデータベースの開発と管理を容易にするツールです。
- Connector/PHP は、MySQL 5.0.18 以降での使用を目的に、[mysql](#) および [mysqli](#) エクステンションを備えた PHP のための Windows 用コネクタです。

Perl や Python、他のプラットフォームや環境での PHP など、上記にある以外の言語やインターフェースを使用している MySQL サーバへの接続に関しては、[23章APIとライブラリー](#) を参照してください。

MySQL Connector/ODBC

MySQL Connector/ODBC は、業界基準の Open Database Connectivity (ODBC) API を使用して、MySQL データベースへアクセスする MySQL ODBC (前称は MyODBC ドライバ) の一種です。この資料では、ODBC 3.5x に対応して MySQL データベース へのアクセスを可能にするバージョン、Connector/ODBC 3.51 について説明します。

Connector/ODBC 3.51 以前のバージョンのマニュアルは、該当バイナリもしくはソース配布物に含まれている場合があります。

ODBC API 標準と使い方についての詳細は、<http://www.microsoft.com/data/> を参照してください。

この資料のアプリケーション開発に関する情報は、読み手が C 言語の実務知識、DBMS に関する一般知識を持ち、なにより MySQL に精通していることを前提に記載されています。MySQL の機能とそのシンタックスの詳しい情報は、<http://dev.mysql.com/doc/> をご覧ください。

通常、Connector/ODBC は Windows マシンのみにインストールされます。Unix および OS X では、ネイティブ MySQL ネットワークか、名前付きパイプを使用して、MySQL データベースと通信することができます。Unix か Mac OS X を使用するならば、データベースとの通信に ODBC インターフェイスが必要なアプリケーションには、Connector/ODBC が必要になることもあります。MySQL との通信に ODBC が必要なアプリケーションには、ColdFusion、Microsoft Office、Filemaker Pro などがあります。

Unix ホストに Connector/ODBC コネクタをインストールする場合は、ODBC マネージャーも忘れずにインストールしてください。

重要事項：

- Connector/ODBC のインストールのヘルプは [Connector/ODBC のインストール](#) を参照。
- Windows ホストから、Connector/ODBC を使用して MySQL データベースへ接続する方法の詳細は [Connector/ODBC を介する MySQL データベースとの接続のステップガイド](#) を参照。
- Connector/ODBC を使い、Microsoft Access を MySQL データベースへのインターフェイスとして利用する場合は [Microsoft Access で Connector/ODBC を使用する](#) を参照。
- 最後の自動インクリメント ID の取得方法等、Connector/ODBC の使い方の一般的なヒントは [Connector/ODBC 一般機能](#) を参照。
- Connector/ODBC を特定のアプリケーションと使用する際のヒントやよくある質問は、[Connector/ODBC アプリケーション別情報](#) を参照。
- よくある質問に対する回答のリストは [Connector/ODBC エラーコードと 解決法](#) 。
- Connector/ODBC の利用に関して、さらにサポートが必要な場合は、[Connector/ODBC サポート](#) を参照。

Connector/ODBC の概要

ODBC (Open Database Connectivity) はクライアント プログラムに、多様なデータベースやデータソースへのアクセスを提供します。ODBC は SQL データベースへの接続を可能にする、標準化された API です。SQL Access Group の規格に準じて開発され、関数呼び出しやエラー コード、データベース非依存アプリケーションの開発に使用されるデータタイプのセットを定義します。ODBC は通常、複数のデータソースへのデータベース非依存性、または同時アクセスが必要な場合に使用されます。

ODBC に関しての詳細は、<http://www.microsoft.com/data/> を参照してください。

Connector/ODBC バージョン

現在利用できる Connector/ODBC には 2 バージョンあります。

- 現在テスト段階にある Connector/ODBC 5.0 は、Connector/ODBC 3.51 ドライバの機能を拡大し、ストアド プロシージャやビューを含む、MySQL 5.0 server releas の機能の完全サポートを組み込むべくデザインされていま

す。Connector/ODBC 3.51 を使用しているアプリケーションであれば Connector/ODBC 5.0 にも対応し、同時に Connector/ODBC 5.0 の新機能も利用できます。Connector/ODBC 5.0 ドライバの特性や機能は現在のところ、本ガイドには記載されていません。

- Connector/ODBC 3.51 は 32-bit ODBC ドライバの現行リリースで、MySQL ODBC 3.51 ドライバとしても知られています。このバージョンは、旧 Connector/ODBC 2.50 よりも拡大された機能を備えています。ODBC の MySQL へのアクセス機能を全て引き続き提供するため、ODBC 3.5x 規格レベル 1 (完全なコア API + レベル 2 機能) に対応しています。
- MyODBC 2.50 は、ODBC 2.50 規格レベル 0 (レベル 1 および 2 の機能を装備) を基にした MySQL AB の 32-bit ODBC ドライバの前バージョンです。MyODBC 2.50 ドライバの情報は、比較目的のみのために本ガイドに記載されています。

注記

このセクション以降、このガイドの主要焦点は Connector/ODBC 3.51 ドライバについてになります。MyODBC 2.50 ドライバの詳細資料は、該当バージョンのインストール パッケージに含まれています。2.50 バージョンのみに影響する個別問題 (エラーまたは既知の問題) があれば、参考として本資料に記載されている場合もあります。

注記

MySQL 製品のバージョン ナンバーは X.X.X というように構成されています。しかし、Windows ツール (コントロールパネル、プロパティ表示) では、バージョン ナンバーが XX.XX.XX. と表示されることがあります。例えば、MySQL の正式なバージョン ナンバーである 5.0.9 が、Windows ツールでは 5.00.09 と表示される場合があります。これは単にナンバーが異なって表示されるだけであり、バージョンそのものに違いはありません。

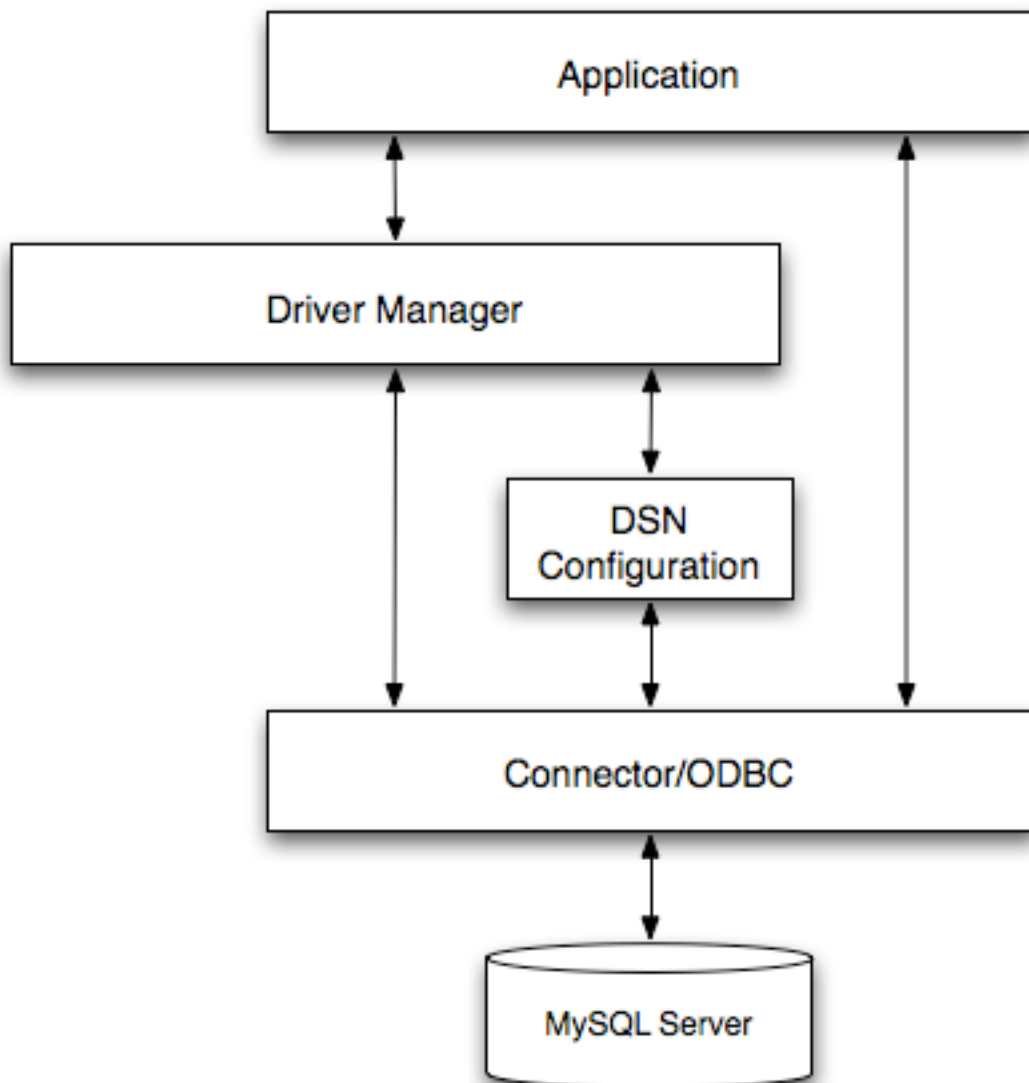
ODBC と Connector/ODBC の一般情報

Open Database Connectivity (ODBC) は、データベース アクセスのためのアプリケーション プログラム インターフェース (API) として広く受け入れられています。データベース API には、X/Open および ISO/IEC の Call-Level Interface (CLI) 規格が基になっており、データベース アクセス言語には Structured Query Language (SQL) が使われています。

Connector/ODBC にサポートされた ODBC 関数の概括は [Connector/ODBC API 参考資料](#) に記載されています。ODBC の一般情報は、<http://www.microsoft.com/data/> を参照してください。

Connector/ODBC の構造

Connector/ODBC の構造は、次の図のように 5 つのコンポーネントから成っています：



- アプリケーション :

アプリケーションは ODBC API を使用して、MySQL サーバのデータにアクセスします。次に、ODBC API がドライバ マネージャーと接続します。アプリケーションは 標準 ODBC セルを使って、ドライバ マネージャーとコミュニケーションを取ります。データが保存される場所、保存の方法、またはデータにアクセスするためのシステムがどのように構成されているかなども、アプリケーションは一切問いません。アプリケーションに必要な情報は Data Source Name (DSN) のみです。

ODBC をどのように使用している場合でも、アプリケーションは通常的に複数の作業をこなします。作業には以下のようなものがあります:

- MySQL サーバを選択して接続する
- SQL 文で実行を要求
- (もし結果があれば) 結果を検索
- エラーを処理

- SQL 文を含んだトランザクションを完遂もしくはロールバック
- MySQL サーバの接続解除

ほとんどのデータアクセス作業は SQL で行うため、ODBC を使用するアプリケーションの主な役割は、SQL 文の発行とその結果の検索になります。

- ドライバ マネージャー :

ドライバ マネージャーは、アプリケーションとドライバ間のコミュニケーションを管理するライブラリです。その役割には次のようなものがあります :

- Data Source Names (DSN) の解釈 DSN は、所定のデータベース ドライバ、データベース、データベース ホスト、またオプションとして、標準化されたリファレンスを使ってデータベースへ接続する ODBC アプリケーションを有効にする承認情報、などを識別する構成ストリングです。

データベース接続情報は DSN で区別されるため、どの ODBC 準拠アプリケーションでも、同じ DSN リファレンスを使ってデータソースに接続することができます。これによって、所定のデータベースにアクセスする各アプリケーションを別々に構成する必要がなくなり、事前に設定された DSN を使用するようアプリケーションに指示するだけで済みます。

- ドライバをロードおよびアンロードするには、DSN 内で定められた特定のデータベースにアクセスする必要があります。例えば、MySQL データベースへ接続する DSN を設定した場合、ドライバ マネージャーが Connector/ODBC ドライバをロードして ODBC API を有効にし、MySQL ホストと接続します。
 - ODBC 関数呼び出しを処理し、または呼び出しをドライバに送って処理させます。
- Connector/ODBC ドライバ :

Connector/ODBC ドライバは、ODBC API にサポートされた機能を実現するライブラリです。ODBC 関数呼び出しを処理し、SQL リクエストを MySQL サーバにサブミットし、結果をアプリケーションに戻します。ドライバは必要に応じて、アプリケーションの要求を、MySQL にサポートされたシンタックスに適合するよう改修します。

- DSN 設定 :

ODBC 設定ファイルは、サーバへの接続が必須のドライバとデータベースの情報を保管します。そのファイルは、ドライバ マネージャーが、DSN の設定によって、どのドライバをロードするかを決定するのに使われます。ドライバはこれを使用し、DSN の指定に沿って接続パラメータを読み取ります。詳細は [Connector/ODBC の構成](#) をご覧ください。

- MySQL サーバ :

情報を保管する場所が、MySQL データベースです。データベースは、データの情報源 (問い合わせの間)、そしてデータの宛先 (挿入または更新の間) として利用されます。

ODBC ドライバ マネージャー

ODBC ドライバ マネージャーは、ODBC 認識アプリケーションと各種ドライバ間の通信を管理するライブラリです。以下がその主な機能です :

- Data Source Names (DSN) の解決
- ドライバのロードおよびアンロード
- ODBC 関数呼び出しの処理、またはドライバへの呼び出し送信

Windows および Mac OS X のオペレーション システムには、ODBC ドライバ マネージャーが含まれています。また、ほとんどの ODBC ドライバ マネージャーには、DSN とドライバの設定を容易にする運用管理アプリケーションが実装されています。Unix ODBC ドライバ マネージャーを含む、これらのマネージャーの例と詳細については、以下の記述を参考にしてください :

- Microsoft Windows ODBC ドライバ マネージャー ([odbc32.dll](#)) <http://www.microsoft.com/data/>

- Mac OS X には、よりシンプルな構成メカニズムを Unix iODBC ドライバ マネージャーに提供する GUI アプリケーション、[ODBC Administrator](#) が組み込まれています。ODBC Administrator もしくは iODBC 設定ファイルを介して、DSN とドライバ情報を構成することができます。これはまた、[iodbctest](#) コマンドを使用して、ODBC Administrator の設定をテストできるということでもあります。<http://www.apple.com>。
- Unix 用 [unixODBC](#) ドライバ マネージャー ([libodbc.so](#))。詳細は <http://www.unixodbc.org> をご覧ください。[unixODBC](#) 2.1.2. 以降のバージョンでは、[unixODBC](#) ドライバ マネージャーのインストール パッケージに Connector/ODBC ドライバ 3.51 が含まれています。
- Unix 用 [iODBC](#) ODBC ドライバ マネージャー ([libiodbc.so](#))。詳細は <http://www.iodbc.org> を参照。

Connector/ODBC のインストール

Connector/ODBC ドライバのインストールには、バイナリ インストールとソース インストールという 2 種類の方法があります。バイナリ インストールは最も容易で簡潔なインストール法です。ソース インストール法は、プラットフォームにバイナリ インストールのパッケージがない場合か、インストールのプロセスや、Connector/ODBC ドライバをインストール前に変更・修正したい場合にのみ使用します。

Connector/ODBC の入手方法

MySQL AB では、General Public License (GPL) に基づくすべての製品を配布しています。Connector/ODBC バイナリやソースの最新バージョンを、MySQL AB ウェブサイト、<http://dev.mysql.com/downloads/> で入手することができます。

Connector/ODBC の詳細は、<http://www.mysql.com/products/myodbc/> を参照してください。

ライセンスに関する情報は、<http://www.mysql.com/company/legal/licensing/> でご確認ください。

サポート対象のプラットフォーム

Connector/ODBC は、MySQL にサポートされた、すべての代表的なプラットフォームで使用が可能です。インストールできるプラットフォームは以下：

- Windows 95、98、Me、NT、2000、XP、2003
- 次を含むすべての Unix 互換のオペレーション システム :AIX、Amiga、BSDI、DEC、FreeBSD、HP-UX 10/11、Linux、NetBSD、OpenBSD、OS/2、SGI Irix、Solaris、SunOS、SCO OpenServer、SCO UnixWare、Tru64 Unix
- Mac OS X および Mac OS X サーバ

バイナリ配布物が利用できないプラットフォームの場合は、[ソース配布物から Connector/ODBC をインストールする](#)を参照して、オリジナルのソースコードからドライバを構築してください。独自に作成したバイナリを MySQL に提供し、他のユーザにも公開していただける場合は、[<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) までメールをお送りください。

バイナリ配布物から Connector/ODBC をインストールする

バイナリ配布物を利用すると、簡潔に Connector/ODBC をインストールすることができます。ドライバやインストール ロケーションをもっと細かに調節したい、または、ドライバの要素をカスタマイズしたいという場合は、ソースから構築してインストールする必要があります。詳細は [ソース配布物から Connector/ODBC をインストールする](#) をご覧ください。

Connector/ODBC をバイナリ配布物から Windows に インストールする

Windows に Connector/ODBC ドライバをインストールする前に、Microsoft Data Access Components (MDAC) がアップデートされているか確認してください。最新バージョンは [Microsoft Data Access and Storage](#) ウェブサイトから入手することができます。

Windows にインストールする場合、3 つのタイプの配布物が利用できます。それぞれのコンテンツはまったく同じもので、インストールの方法だけが異なります。

- Zip インストーラは、スタンドアロン インストール アプリケーションを含む Zip 形式のパッケージ から成ります。このパッケージからインストールするには、インストーラを解凍してから、インストール アプリケーションを実行

してください。インストールの完全な方法については、[インストーラを使用して Windows Connector/ODBC ドライバをインストールする](#) を参照してください。

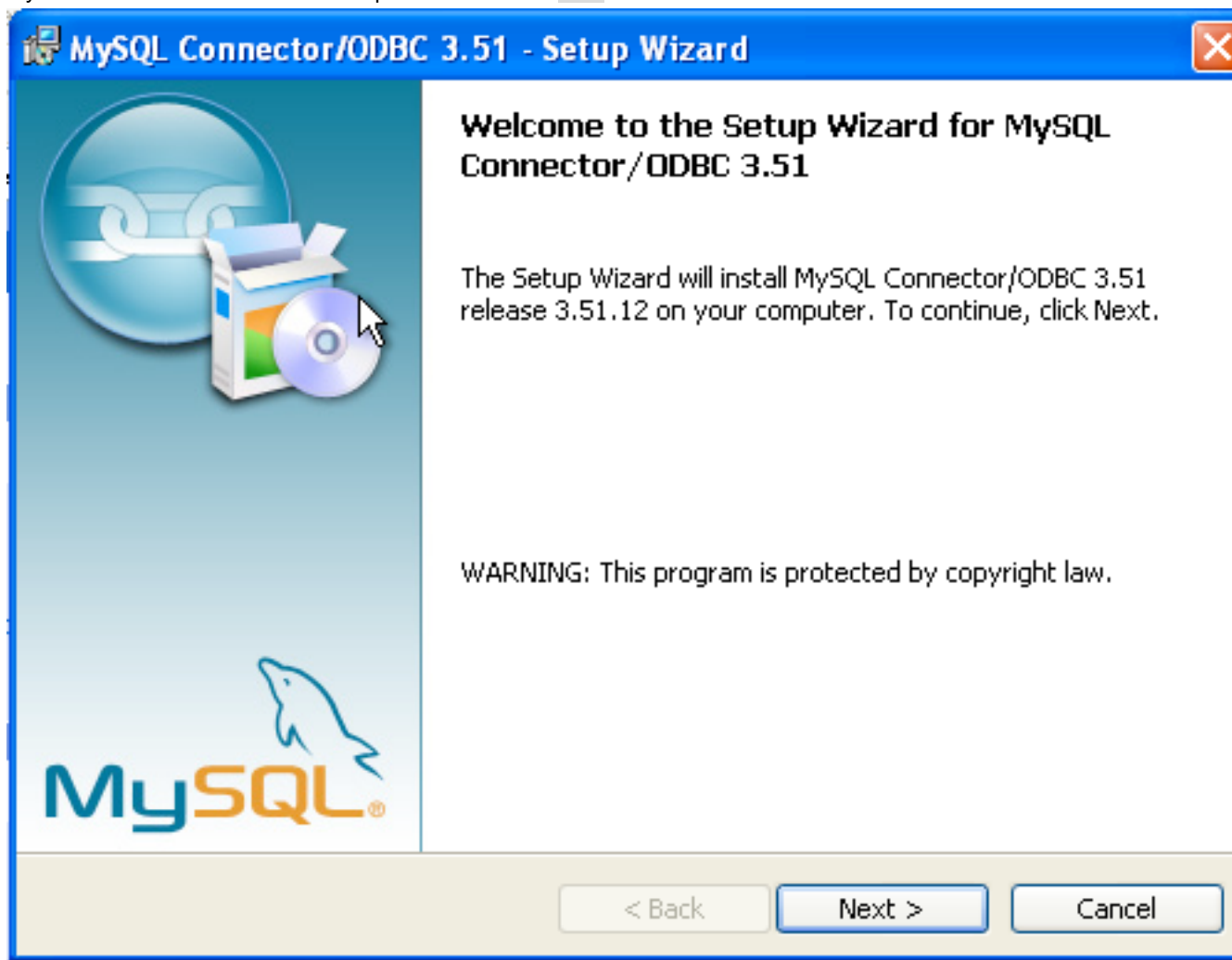
- MSI インストーラは、Windows 2000、Windows XP、Windows Server 2003 に含まれるインストーラとの併用が可能なインストール ファイルです。インストールの完全詳細は [インストーラを使用して Windows Connector/ODBC ドライバをインストールする](#) をご覧ください。
- DDL ファイルを含む、Zip形式の DLL パッケージは、手動でインストールを行います。インストールの完全な方法については、[Zip 形式 DLL パッケージを使用して Windows Connector/ODBC ドライバをインストールする](#) を参照してください。

インストーラを使用して Windows Connector/ODBC ドライバをインストールする

インストール パッケージを使用すると、非常に簡単に Connector/ODBC ドライバをインストールすることができます。Zip 形式のインストーラをダウンロードした場合は、まずインストール アプリケーションを抽出してください。基本的なインストールの手順は、どちらのインストーラも同じです。

インストールを実行するには、次の手順に従ってください。

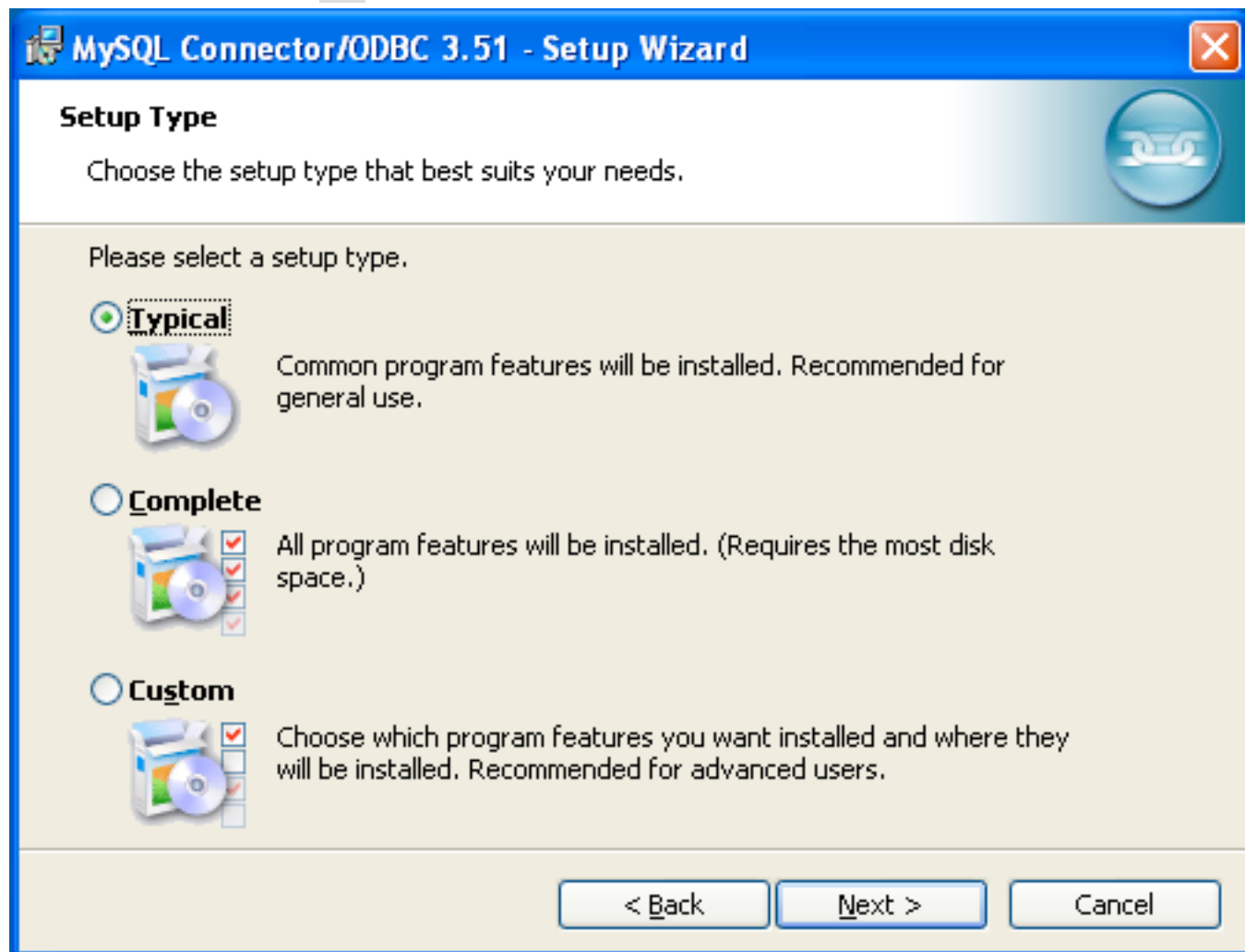
1. 抽出したスタンドアロン インストーラか、ダウンロードした MSI ファイルをダブルクリック。
2. MySQL Connector/ODBC 3.51 - Setup Wizard が起動。Next ボタンをクリックして、インストールを開始。



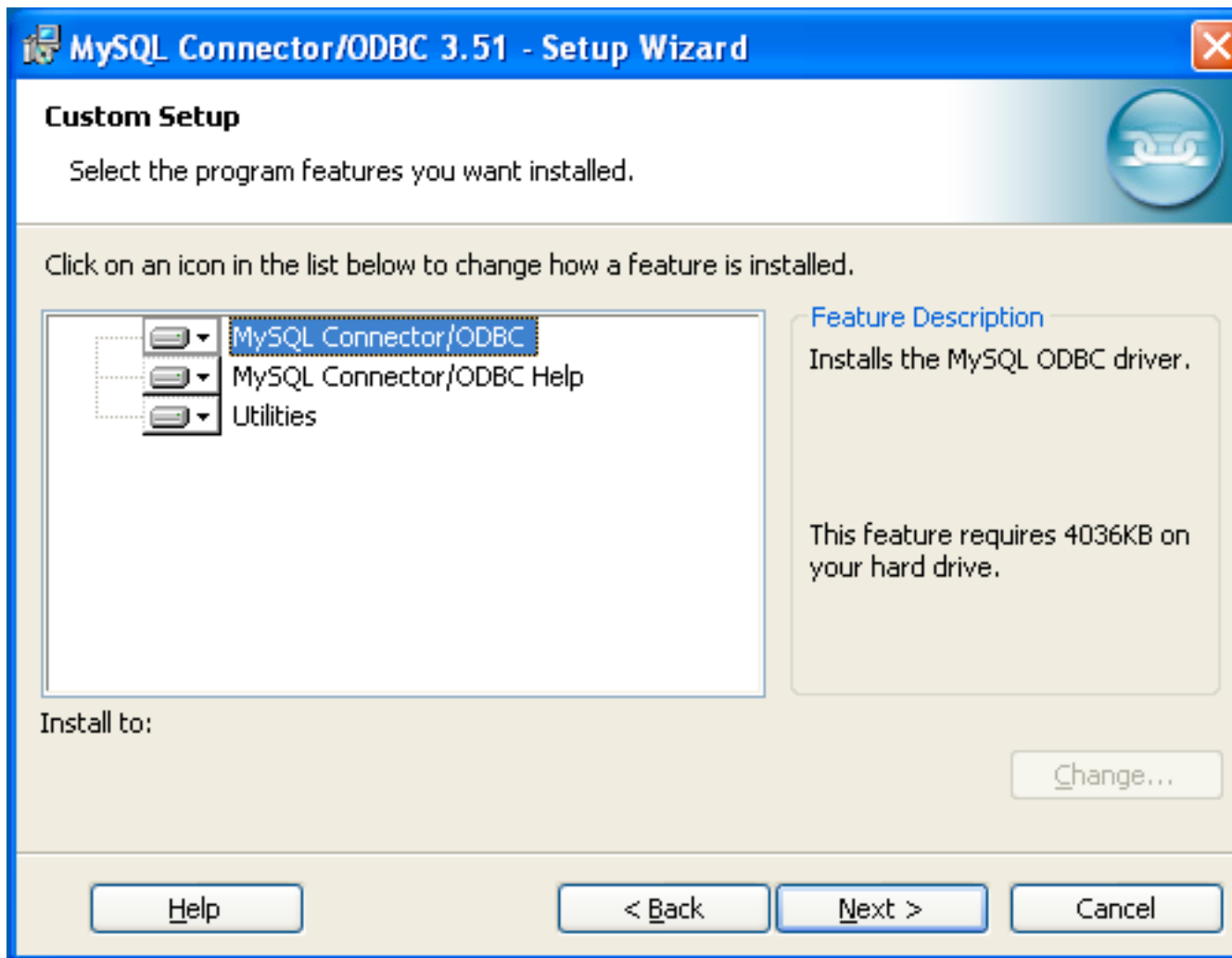
3. インストールのタイプを選択。Typical インストールは標準ファイルを提供し、これを選択すると、ODBC を使用して MySQL データベースに接続することになります。Complete オプションは、デバッグとユーティリティのコンポーネントを含む、すべての使用可能なファイルをインストールします。インストールを実行するには、この

ふたつのオプションから選択することをお勧めします。これらの方法を使用する場合は、**Next** をクリックして、ステップ 5 へ進んでください。

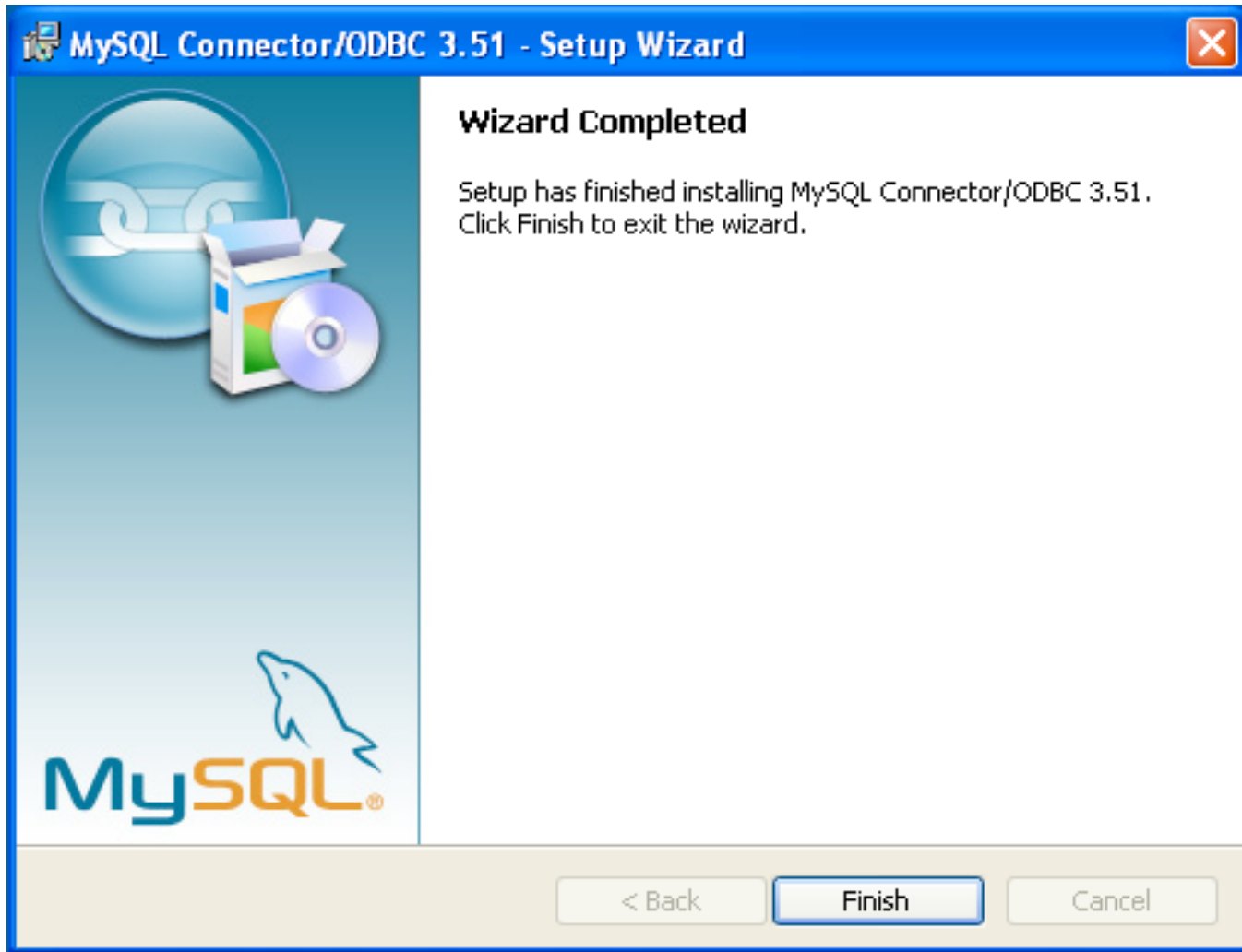
Custom インストールを選択すると、インストールしたいコンポーネントをそれぞれ自分で選択することができます。この方法を選んだ場合は、**Next** をクリックしてステップ 4 へ進んでください。



4. Custom インストールを選択した場合、ポップアップ画面からインストールしたいコンポーネントを選び、**Next** をクリックして、必要なファイルをインストールします。



5. ファイルがマシンにコピーされたら、インストールは完了です。Finish をクリックして、インストーラを終了します。



以上でインストールは完了です。[Connector/ODBC の構成](#) を参照して、引き続き ODBC 接続を作成することができます。

Zip 形式 DLL パッケージを使用して Windows Connector/ODBC ドライバをインストールする

Zip 形式の DLL パッケージをダウンロードしたら、Connector/ODBC オペレーションに必要な各種ファイルを手動でインストールします。インストール ファイルを解凍したあと、手動でこの操作を行うか、各ステートメントをひとつずつ実行するか、もしくは組み込まれている Batch ファイルを使用して、デフォルトのロケーションにインストールすることもできます。

Batch ファイルを使用したインストールの手順：

1. Connector/ODBC Zipped DLL パッケージを解凍。
2. Command Prompt を開く。
3. Connector/ODBC Zipped DLL パッケージを解凍した時に作成されるディレクトリに移動。
4. [Install.bat](#) を実行：

```
C:\> Install.bat
```

以上で必要なファイルがデフォルトのロケーションにコピーされ、Windows ODBC マネージャーに Connector/ODBC ドライバが登録されます。

他のロケーションにファイルをコピーしたい場合、例えば同じマシン上でバージョンの異なる Connector/ODBC ドライバを実行、またはテストしたい場合などには、手動でファイルをコピーしなければなりません。しかし、これらのファイルの非標準ロケーションへのインストールはお勧めできません。ファイルを手動でデフォルト ロケーションへコピーするには、以下の手順に従ってください。

1. Connector/ODBC Zipped DLL パッケージを解凍。
2. Command Prompt を開く。
3. Connector/ODBC Zipped DLL パッケージを解凍した時に作成されたディレクトリに移動。
4. ライブラリ ファイルを適切なディレクトリへコピー。初期設定では、ファイルのコピー先はデフォルトの Windows システム ディレクトリ、`\Windows\System32` :

```
C:\> copy lib\myodbc3S.dll \Windows\System32
C:\> copy lib\myodbc3S.lib \Windows\System32
C:\> copy lib\myodbc3.dll \Windows\System32
C:\> copy lib\myodbc3.lib \Windows\System32
```

5. Connector/ODBC ツールをコピー。システム `PATH` のディレクトリにコピーすること。初期設定でのツールのインストール先は、Windows システム ディレクトリ `\Windows\System32` :

```
C:\> copy bin\myodbc3i.exe \Windows\System32
C:\> copy bin\myodbc3m.exe \Windows\System32
C:\> copy bin\myodbc3c.exe \Windows\System32
```

6. オプションとして、ヘルプ ファイルをコピー。ヘルプ システムからファイルへのアクセスを有効にするには、ファイルを Windows システム ディレクトリにインストールする :

```
C:\> copy doc\*.hlp \Windows\System32
```

7. 最後に、ODBC マネージャーに Connector/ODBC ドライバを登録 :

```
C:\> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;\
DRIVER=myodbc3.dll;SETUP=myodbc3S.dll"
```

ファイルをデフォルト ロケーション外にインストールした場合は、必ずリフェレンスを DLL ファイルに変更し、上記のステートメントでロケーションを指定する。

インストール エラーの処理

Windows では、旧バージョンの MyODBC 2.50 ドライバをインストールしようとする、次のエラーが発生することがあります :

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL.
Restart Windows and try installing again (before running any
applications which use ODBC)
```

エラーの原因には、他のアプリケーションが ODBC システムを使用中ということが考えられます。その場合、Windows がインストールを差し止めることがあります。大抵は、`Ignore` を押すと Connector/ODBC ファイルのコピーが続行され、最終的なインストールも実行されます。それでもインストールが行われない時は、「safe mode.」でコンピュータを再起動すると問題は解消されます。コンピュータが Windows を起動する前に F8 を押してセーフモードを選び、Connector/ODBC ドライバをインストールしてから、もう一度通常モードでコンピュータを再起動してください。

Connector/ODBC をバイナリ配布物から Unix に インストールする

バイナリ配布物から Unix に Connector/ODBC をインストールするには、2 種類の方法があります。ほとんどの Unix 環境では、tar アーカイブ配布物を使うことになります。Linux システムでは、RPM 配布物も利用可能です。

バイナリ tar アーカイブ配布物から Connector/ODBC をインストールする

tar アーカイブ配布物 (`.tar.gz` ファイル) からドライバをインストールするには、オペレーション システムに最新バージョンのドライバをダウンロードし、tar アーカイブの Linux バージョンを使用する次の手順に従ってください :

```
shell> su root
```

```
shell> gunzip mysql-connector-odbc-3.51.11-i686-pc-linux.tar.gz
shell> tar xvf mysql-connector-odbc-3.51.11-i686-pc-linux.tar
shell> cd mysql-connector-odbc-3.51.11-i686-pc-linux
```

INSTALL-BINARY ファイルにあるインストール説明を読んで、これらのコマンドを実行してください。

```
shell> cp libmyodbc* /usr/local/lib
shell> cp odbc.ini /usr/local/etc
shell> export ODBCINI=/usr/local/etc/odbc.ini
```

次に、[Unix での Connector/ODBC DSN の構成](#) に進み、Connector/ODBC の DSN を設定します。詳細は、配布物に含まれている [INSTALL-BINARY](#) ファイルを参照してください。

RPM 配布物から Connector/ODBC をインストールする

Linux で、RPM 配布物から Connector/ODBC をインストールまたはアップグレードするには、Connector/ODBC 最新バージョンの RPM 配布物をダウンロードし、以下の手順に従って下さい。su root を使って root に変更してから、RPM ファイルをインストールします。

初めてインストールする場合：

```
shell> su root
shell> rpm -ivh mysql-connector-odbc-3.51.12.i386.rpm
```

すでにドライバがあり、アップグレードする場合：

```
shell> su root
shell> rpm -Uvh mysql-connector-odbc-3.51.12.i386.rpm
```

MySQL クライアント ライブラリ、libmysqlclient に依存エラーが生じる場合は、--nodeps オプションを与えてエラーを無視し、その後に MySQL クライアント共有ライブラリがパスになっているか確認、または LD_LIBRARY_PATH を通よう設定してください。

以上でドライバ ライブラリと関連ドキュメントが、/usr/local/lib と /usr/share/doc/MyODBC にそれぞれインストールされます。[Unix での Connector/ODBC DSN の構成](#) に進んでください。

ドライバを uninstall するには、root に変更して rpm コマンドを実行します：

```
shell> su root
shell> rpm -e mysql-connector-odbc
```

Mac OS X に Connector/ODBC をインストールする

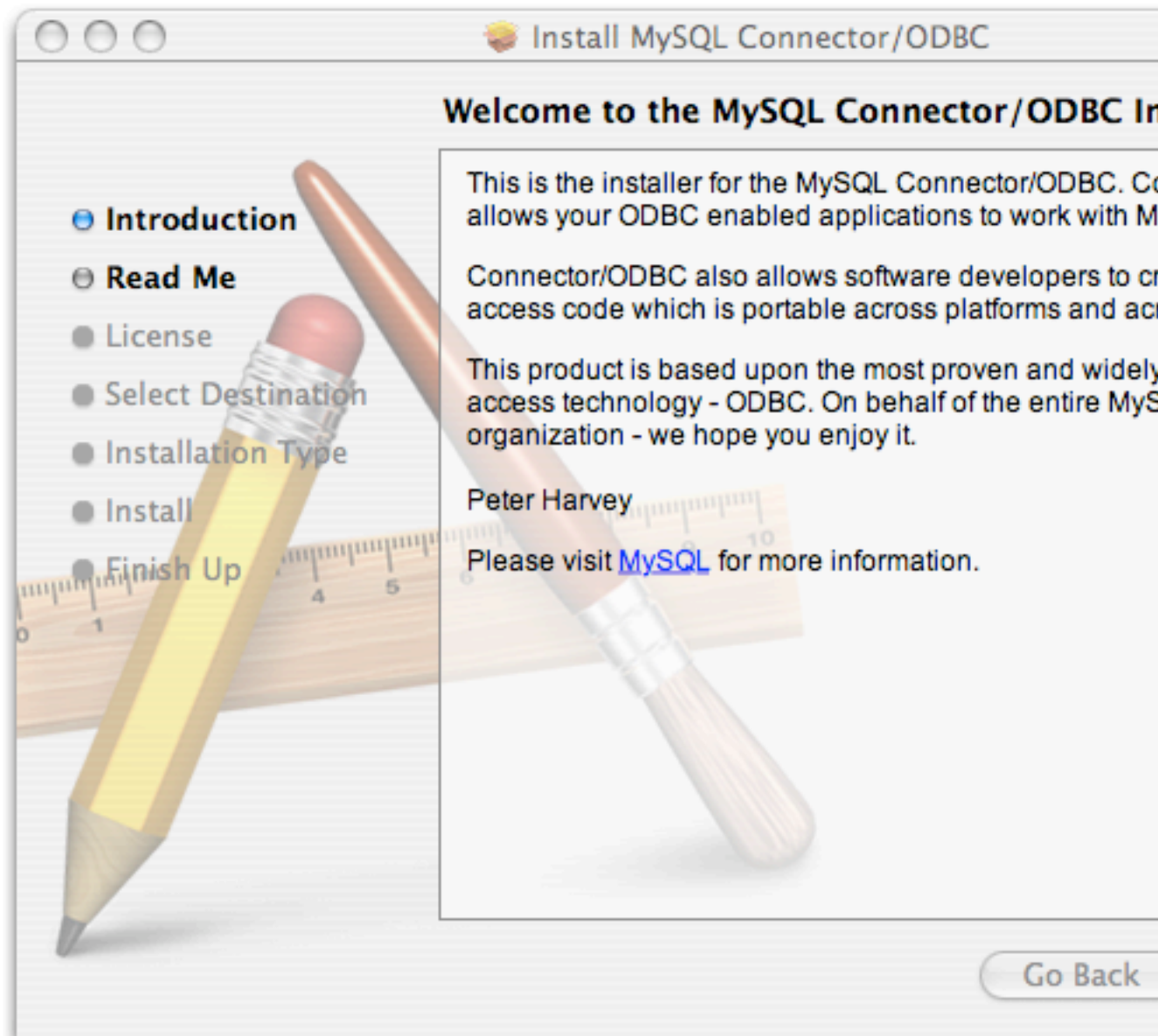
Mac OS X は FreeBSD オペレーション システムがベースになっており、通常は MySQL ネットワーク ポートを使って、他のホストの MySQL サーバに接続することができます。Connector/ODBC ドライバをインストールすることで、ODBC インターフェイスを介して、どんなプラットフォームにある MySQL データベースにも接続が可能になります。ODBC インターフェイスを必須とするアプリケーションでは、Connector/ODBC ドライバのみをインストールすることになります。ODBC (ひいては Connector/ODBC ドライバ) を必要とする、または利用できるアプリケーションは、ColdFusion、Filemaker Pro、4th Dimension 他多数あります。

Mac OS X には、iODBC を基にした独自の ODBC マネージャーが含まれています。Mac OS X は、ODBC ドライバの管理を容易にする 管理ツールと、基礎になる iODBC 構成ファイルを更新する設定を所有しています。

Connector/ODBC ドライバのインストール

バイナリ配布物を利用して、Connector/ODBC を Mac OS X もしくは Mac OS X サーバ コンピュータにインストールすることができます。パッケージは圧縮ディスクイメージ (.dmg) として配布されています。この方法でコンピュータに Connector/ODBC をインストールするには、次の手順に従ってください：

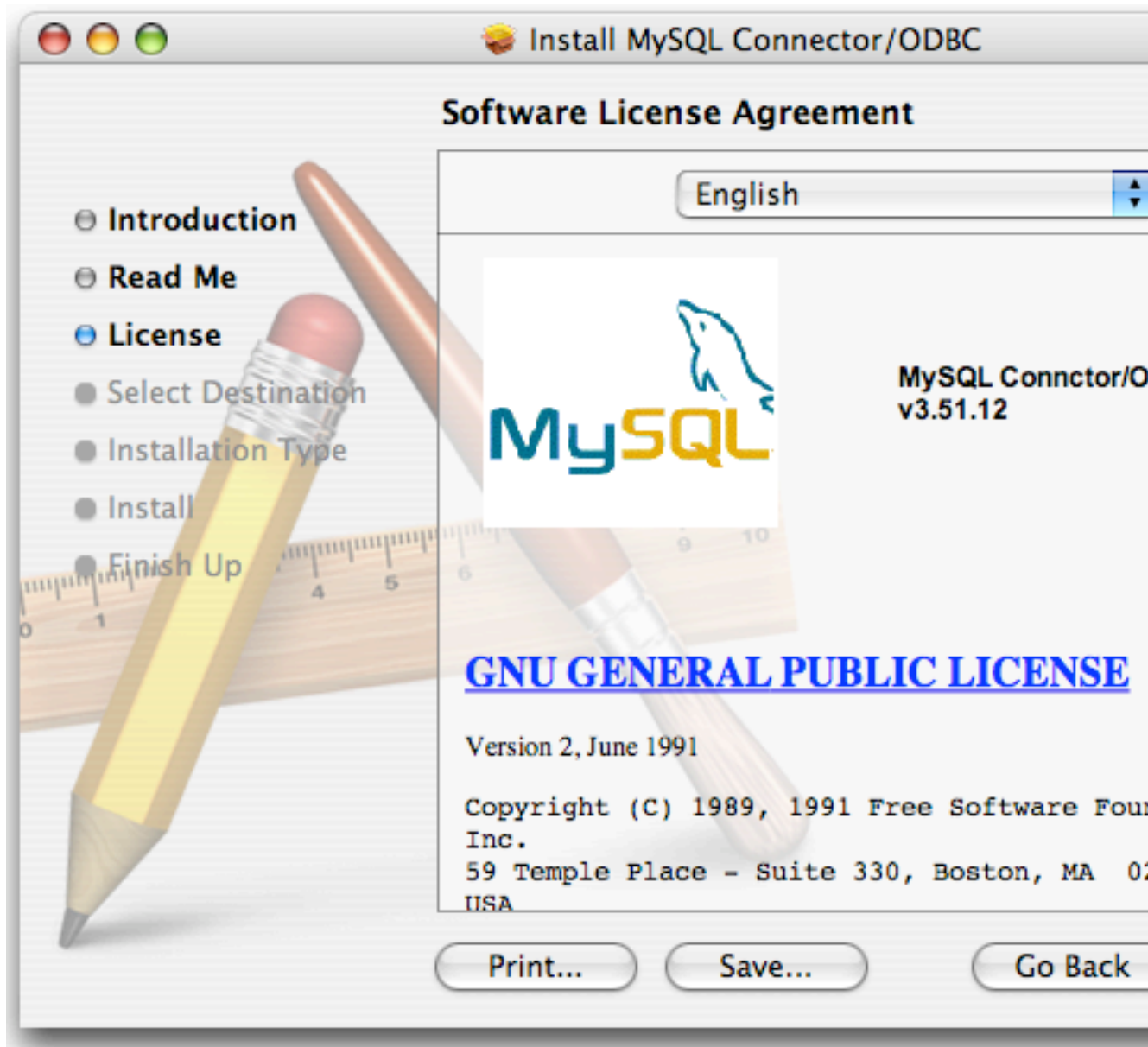
1. ファイルをコンピュータにダウンロードし、そのイメージファイルをダブルクリック。
2. ディスクイメージの中から、インストール パッケージ (拡張子は .pkg) を選択。そのファイルをダブルクリックして、Mac OS X インストーラを起動。
3. インストーラの welcome メッセージが表示されます。Continue ボタンをクリックして、インストールを開始。



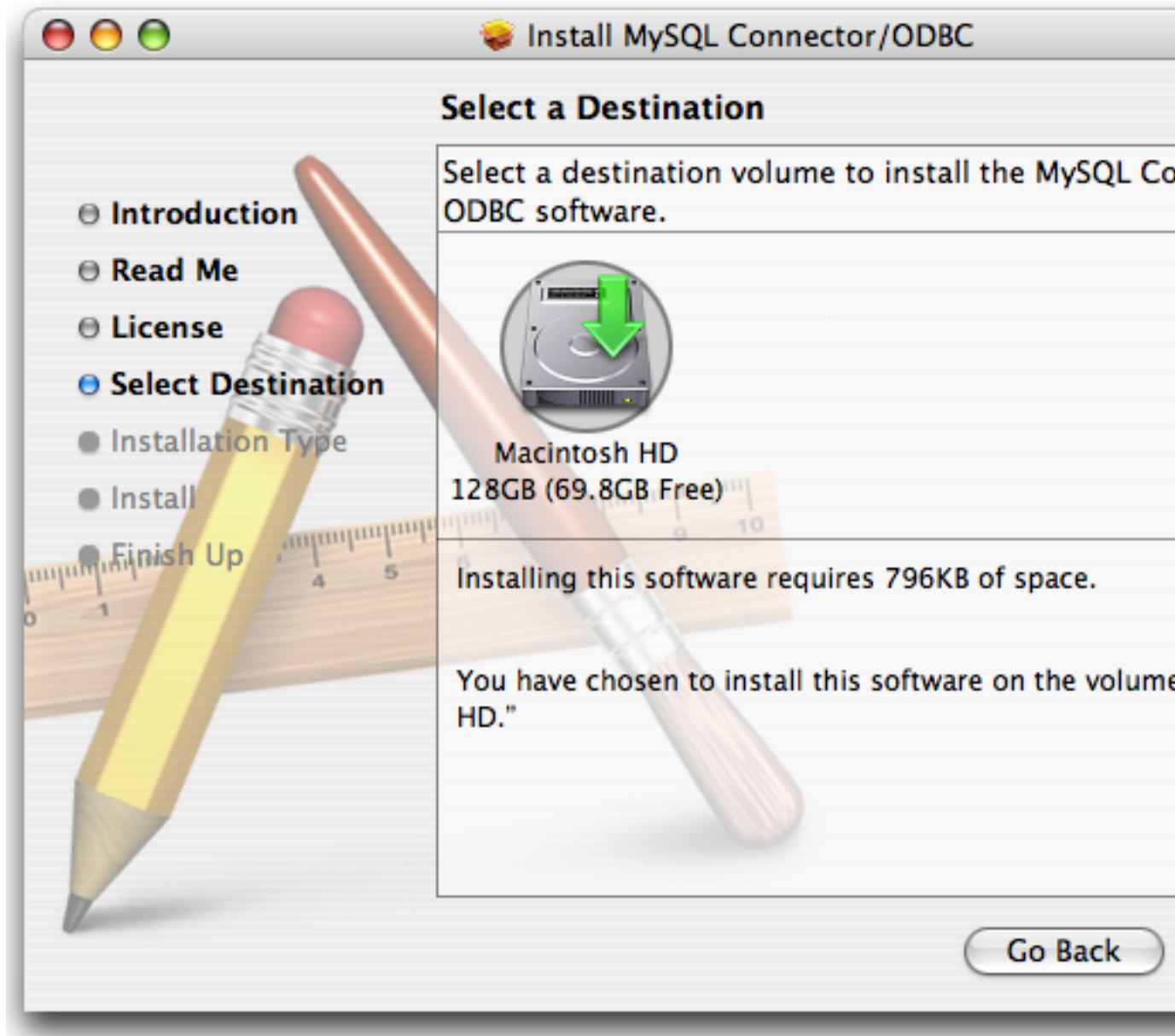
4. インストール実行手順のガイドが記載された Important Information を注意して読んでください。告示を読み、必要な情報を得たら、Continue をクリック。



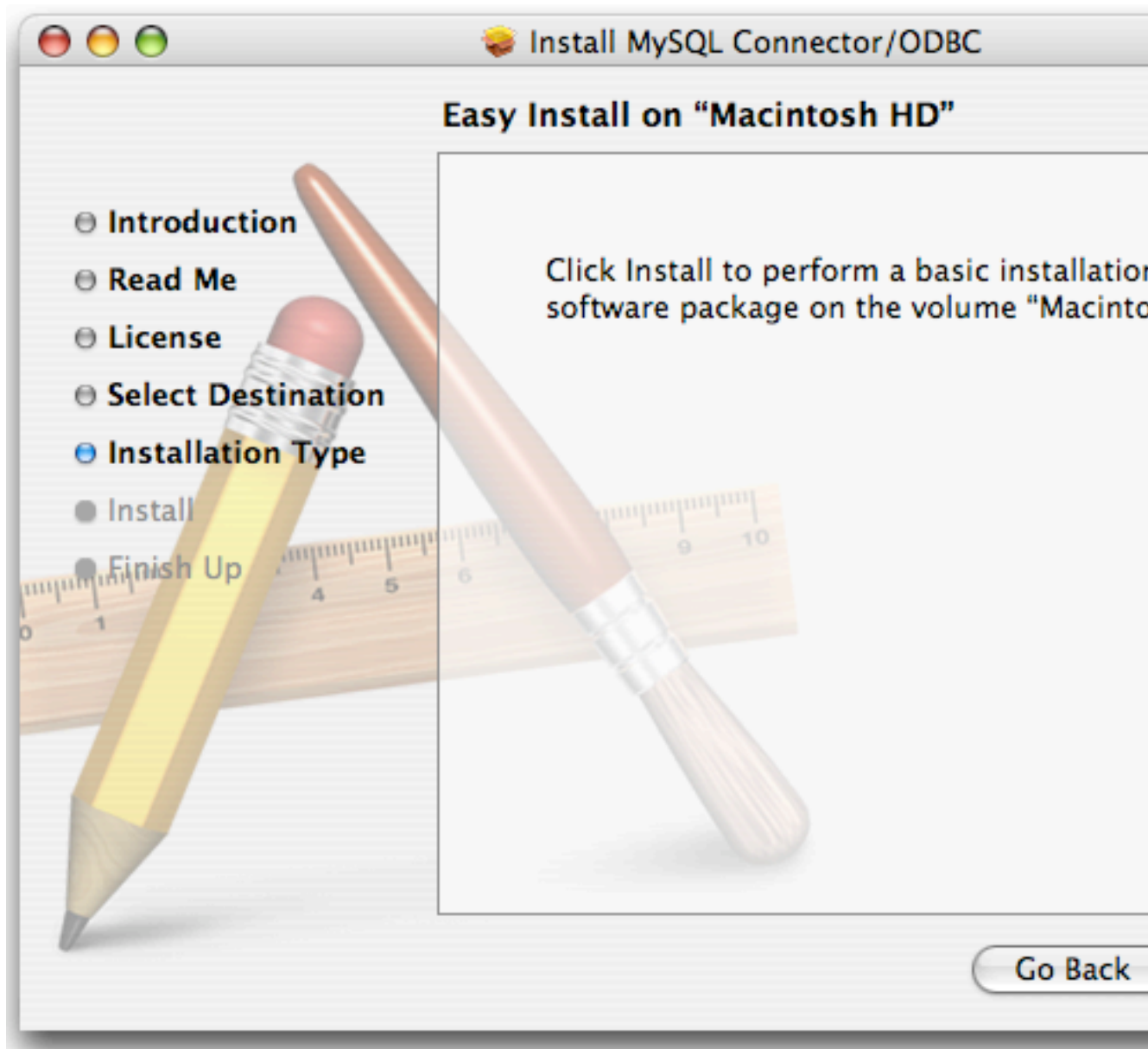
5. Connector/ODBC ドライバは、GNU General Public License に準じて提供されています。ライセンス情報をご存知でない場合は、インストールを進める前にご一読ください。Continue をクリックしてライセンス承認に同意 (同意の確認がされます) し、インストールを続けます。



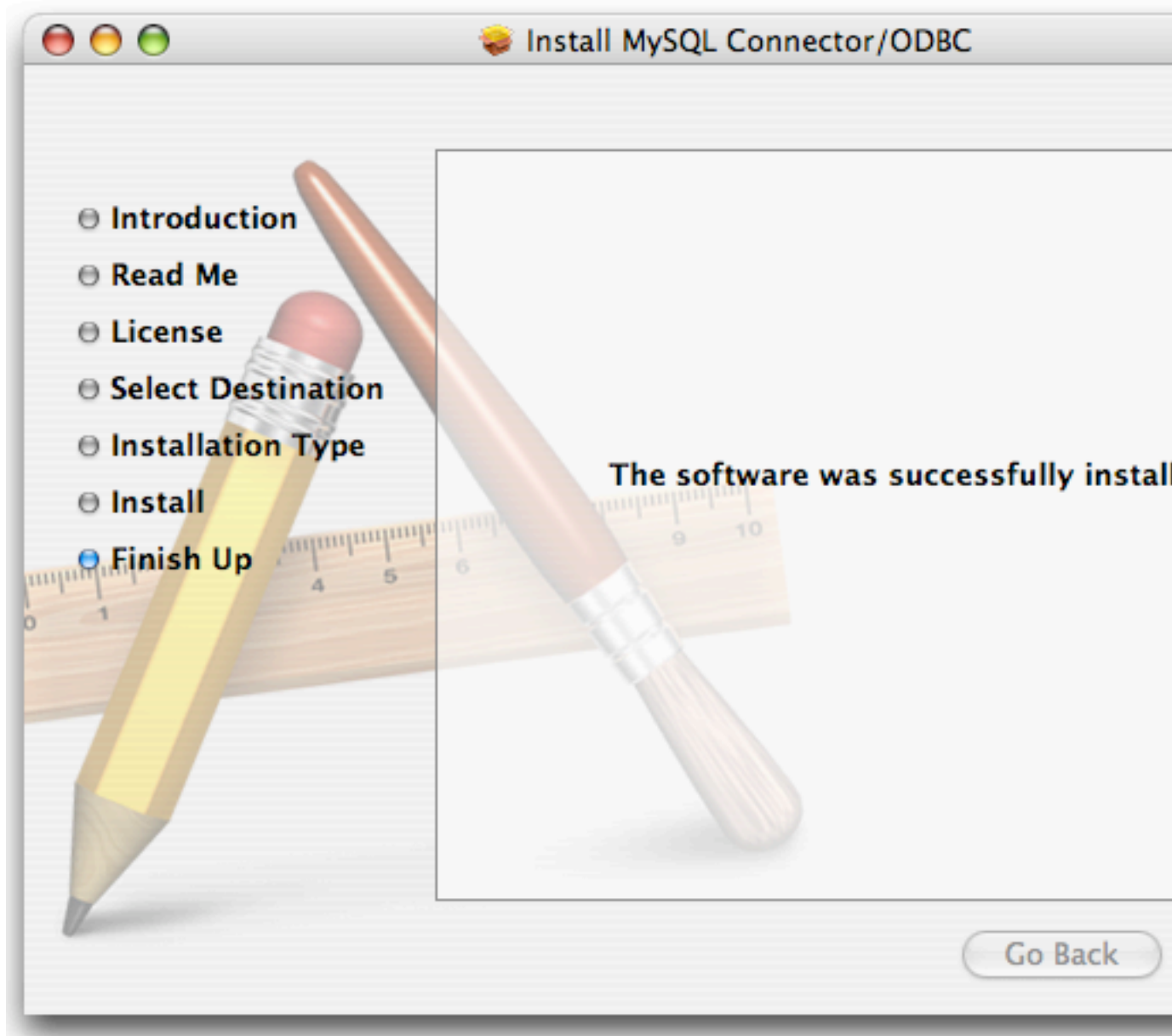
6. Connector/ODBC ドライバと ODBC Administrator アプリケーションのインストール先を選択。ファイルはオペレーティング システムと共にドライブにインストールしなければならず、選択の余地が限られる場合があります。使用したいドライブを選び、**Continue** をクリック。



7. インストーラが、コンピュータに必要なファイルを自動的に選んでインストールします。Install をクリックして、インストールを続けます。インストーラが必要なファイルをマシンにコピーします。インストールの経過を知らせる進行状況バーが表示されます。



8. インストールが完了すると、次のようなウィンドウが表示されます。Close をクリックし、インストーラを終了します。



ソース配布物から Connector/ODBC をインストールする

ソース配布物から Connector/ODBC をインストールすることで、Connector/ODBC コンポーネントの内容やインストール ロケーションをより自由に設定することができます。また、コンパイル済みバイナリが利用できないプラットフォームでも、Connector/ODBC の構築およびインストールが可能になります。

Connector/ODBC ソースは、パッケージのダウンロード、または Connector/ODBC 開発者が使う修正管理システムからも入手することができます。

Connector/ODBC をソース配布物から Windows にインストールする

Windows では、ソースやインストールを変更したい場合にのみ、ソースから Connector/ODBC をインストールすることになります。ソースからインストールするべきか定かでない場合は、[Connector/ODBC をバイナリ配布物から Windows にインストールする](#) で説明されている バイナリ インストールを行ってください。

ソースから Windows に Connector/ODBC をインストールするには、様々なツールやパッケージが必要です：

- MDAC (Microsoft Data Access SDK <http://www.microsoft.com/data/>)
- Microsoft Visual C++ または、Microsoft Visual Studio に組み込まれた C コンパイラ といった、適切な C コンパイラ。
- 互換性のある `make` ツール。Microsoft の `nmake` は、このセクションで例として取り上げられています。
- MySQL クライアント ライブラリと、MySQL 4.0.0 以降の取り込みファイル (MySQL 4.0.16 以降を推奨)。ここには Connector/ODBC が、このバージョンのライブラリから適用された新しい呼び出しと構成を使用するため、必要になります。指定のクライアント ライブラリと取り込みファイルは、<http://dev.mysql.com/downloads/> から入手できます。

Connector/ODBC 3.51 の構築

Connector/ODBC ソース配布物には、`nmake` または他の `make` ユーティリティを必要とする `Makefiles` が含まれています。配布物のうち、`Makefile` はリリース バージョンの構築、`Makefile_debug` はドライバ ライブラリと DLL のデバッグ バージョンの構築に使われます。

ドライバを構築は、次の手順で行って下さい：

1. ソースをダウンロードし、フォルダへ抽出してから、ディレクトリをそのフォルダに変更。次のコマンドでは、フォルダ名を `myodbc3-src` としています：

```
C:\> cd myodbc3-src
```

2. `Makefile` を変更して、MySQL クライアント ライブラリとヘッダ ファイルの正確なパスを指定。その後、次のコマンドを使用して、リリース バージョンを構築・インストールします：

```
C:\> nmake -f Makefile
C:\> nmake -f Makefile install
```

`nmake -f Makefile` が、ドライバのリリース バージョンを構築し、`Release` と呼ばれるサブディレクトリにバイナリを配置します。

`nmake -f Makefile install` が、ドライバ DLL とライブラリ (`myodbc3.dll`、`myodbc3.lib`) をシステム ディレクトリにインストール (コピー) します。

3. デバッグ バージョンを構築するには、`Makefile` よりも、次の `Makefile_Debug` を使用してください：

```
C:\> nmake -f Makefile_debug
C:\> nmake -f Makefile_debug install
```

4. 次に使用して、ドライバの削除およびインストールを行うことができます：

```
C:\> nmake -f Makefile clean
C:\> nmake -f Makefile install
```

注意：

- `Makefiles` では正しい MySQL クライアント ライブラリとヘッダ ファイルのパスを指定してください (`MYSQL_LIB_PATH` と `MYSQL_INCLUDE_PATH` 変数を設定)。デフォルトのヘッダ ファイル パスは、仮に `C:\mysql\include` とします。デフォルトのライブラリ パスは、リリース バージョンの DLL には `C:\mysql\lib\opt`、デバッグ バージョンのものは `C:\mysql\lib\debug` と仮定します。
- `nmake` の完全な使用方法については、http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfrRunningNMAKE.asp を参照してください。
- コンパイルにサブバージョン ツリーを使用している場合、Windows におけるすべての `Makefiles` は、`Win_Makefile*` と名づけられています。

テスト

ドライバ ライブラリがシステム ディレクトリにコピー、もしくはインストールされたら、`samples` サブディレクトリにあるサンプルを使用して、ライブラリが適切に構築されているかテストすることができます：

```
C:\> cd samples
```

```
C:\> nmake -f Makefile all
```

MyODBC 2.50 の構築

MyODBC 2.50 ソース配布物には、VC workspace ファイルが含まれています。そのファイル (`.dsp` および `.dsw`) を Microsoft Visual Studio 6.0 以降からロードし、それらを使ってドライバを構築することができます。

Connector/ODBC をソース配布物から Unix にインストールする

Unix でソースから MySQL を構築する場合、次のツールが必要になります：

- ANSI C++ 実用コンパイラ。gcc 2.95.2 以降、egcs 1.0.2 以降、もしくは egcs 2.91.66、SGI C++、および SunPro C++ などが、使用できるコンパイラの例です。
- 優良な make プログラム。GNU make はいかなる場合にも推奨され、必須になっていることもあります。
- MySQL クライアント ライブラリと、MySQL 4.0.0 以降の取り込みファイル。(MySQL 4.0.16 以降を推奨)。これは Connector/ODBC が、このバージョンのライブラリから適用された新しい呼び出しと構成を使用するため、必要になります。指定のクライアント ライブラリと取り込みファイルは、<http://dev.mysql.com/downloads/> から入手できます。

ソースから独自に MySQL サーバ、および / またはクライアント ライブラリを構築した場合、ライブラリの構築時に、`configure` の実行で `--enable-thread-safe-client` が使われていなければなりません。

また、`libmysqlclient` ライブラリが、共用ライブラリとして構築され、インストールされているか確認してください。

- 互換性のある ODBC マネージャーのインストールは必須です。Connector/ODBC は、`iODBC` および `unixODBC` マネージャとの適応が知られています。詳細は [ODBC ドライバ マネージャー](#) をご覧ください。
- MySQL クライアント ライブラリにコンパイルされていないキャラクタ セットを使用している場合は、MySQL キャラクタ定義を `charsets` ディレクトリから `SHAREDIR` (デフォルトでは `/usr/local/mysql/share/mysql/charsets`) にインストールしてください。MySQL サーバを同じコンピュータにインストールしていれば、これらは所定の位置にあるはずです。キャラクタ セット サポートの詳細は、[9章 キャラクタセットサポート](#) を参照してください。

必要なファイルが揃ったら、ソース ファイルを別々のディレクトリに解凍し、`configure` を実行、そして `make` を使ってライブラリを構築します。

典型的な configure オプション

`configure` スクリプトは、Connector/ODBC 構成をどのようにコンフィグするかにおいて、多大な制御力を提供します。通常は、`configure` コマンドラインのオプションを使用してこれを行います。また、特定の環境変数を使って、`configure` に影響を与えることもできます。`configure` がサポートするオプションと環境変数のリストを見るには、次のコマンドを実行してください：

```
shell> ./configure --help
```

よく使われる `configure` オプションのいくつかを以下で説明します：

1. Connector/ODBC をコンパイルするには、`--with-mysql-path=DIR` を使用して、ライブラリ ファイルを含む MySQL クライアントを供給する必要があります。`DIR` は、MySQL がインストールされるディレクトリです。
MySQL コンパイル オプションは、`DIR/bin/mysql_config` を実行することで判別が可能です。
2. 標準ヘッダとライブラリ ファイル パスを、ODBC ドライバ マネージャー (`iODBC` または `unixODBC`) に供給します。
 - `iODBC` を使用しており、同時に `iODBC` がデフォルト ロケーション (`/usr/local`) にインストールされていない場合は、`--with-iodbc=DIR` オプションを使用しなければならないことがあります。`DIR` は `iODBC` がインストールされるディレクトリです。

もし `iODBC` ヘッダが `DIR/include` に入っていない場合、`--with-iodbc-includes=INCDIR` オプションで、その所在を特定することができます。

ライブラリへのアプライ。DIR/lib に入っていないければ、`--with-iodbc-libs=LIBDIR` オプションを使うことができます。

- `unixODBC` を使用している場合、`--with-unixODBC=DIR` オプション (大文字と小文字を区別) を使って `configure` を作成します。`iODBC` ではなく、`unixODBC` を探してください。デフォルトでは、`DIR` が `unixODBC` のインストールされるディレクトリです。

`unixODBC` ヘッダとライブラリが `DIR/include` と `DIR/lib` がない場合は、`--with-unixODBC-includes=INCDIR` と `--with-unixODBC-libs=LIBDIR` オプションを使用してください。

3. インストールの接頭子に、`/usr/local` 以外を指定したほうがよい場合もあります。例えば、`/usr/local/odbc/lib` に Connector/ODBC をインストールする場合、`--prefix=/usr/local/odbc` オプションを使用します。

最終構成コマンドは次のようなものです：

```
shell> ./configure --prefix=/usr/local \
--with-iodbc=/usr/local \
--with-mysql-path=/usr/local/mysql
```

追加の configure オプション

Connector/ODBC を作成する前の構成での設定に、必要または望ましいオプションは他にも多数あります。

- ドライバを、`libmysqlclient_r.so` または `libmysqlclient_r.a` 等の スレッドセーフ MySQL クライアント ライブラリにリンクするには、次の `configure` オプションを指定してください：

```
--enable-thread-safe
```

また、次を使って無効 (デフォルト) にすることもできます。

```
--disable-thread-safe
```

このオプションを使用すると、スレッドセーフ MySQL クライアント ライブラリ `libmysqlclient_r.so` (拡張子は OS 依存性) にリンクすることで、ドライバ スレッドセーフ ライブラリ `libmyodbc3_r.so` を構築することができます。

スレッドセーフ オプションとのコンパイルが失敗した場合、その原因として、システム上にある正しいスレッド ライブラリが特定できないことが挙げられます。正しいスレッド ライブラリが特定できるよう、`LIBS` の値をセットしてください。

```
LIBS="-lpthread" ./configure ..
```

- 以下のオプションを使用して、Connector/ODBC の共用およびスタティック バージョンを、有効または無効にすることができます：

```
--enable-shared[=yes/no]
--disable-shared
--enable-static[=yes/no]
--disable-static
```

- デフォルトでは、すべてのバイナリ配布物は非デバッグ バージョンとして構築されています (`--without-debug` で構成)。

デバッグ情報を有効にするには、ソース配布物からドライバを構築し、`configure` を実行する際に `-with-debug` オプションを使用します。

- このオプションは、Subversion レポジトリから得たソース ツリーにのみ利用が可能です。パッケージ済みのソース配布物には適用できません。

デフォルトでは、ドライバは `--without-docs` オプションで構築されます。ドキュメンテーションを作成したい場合は、次で `configure` を実行してください：

```
--with-docs
```

構築とコンパイル

ドライバ ライブラリを構築するには、`make` を実行してください。

```
shell> make
```

なんらかのエラーが発生すれば、修正して構築を続けてください。構築が行えない場合は、詳細を明記のうえ、[<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com)までメールでお問い合わせ下さい。

共用ライブラリの構築

ほとんどのプラットフォームでは、MySQL が `.so` (共有) を建築、またはサポートするデフォルト設定にはなっていません。これは共用ライブラリ構築の経験に基づいた情報です。

そのような場合は、MySQL 配布物をダウンロードし、次のオプションで構成してください：

```
--without-server --enable-shared
```

共用ドライバ ライブラリを構築するには、`--enable-shared` オプションを `configure` で指定する必要があります。`configure` が、デフォルト設定でこのオプションを有効にすることはありません。

`--disable-shared` オプションを使用して構成した場合、次のコマンドで、スタティック ライブラリから `.so` ファイルを作成することができます：

```
shell> cd mysql-connector-odbc-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error \
-o .libs/libmyodbc3-3.51.01.so \
catalog.o connect.o cursor.o dll.o error.o execute.o \
handle.o info.o misc.o myodbc3.o options.o prepare.o \
results.o transact.o utility.o \
-L/usr/local/mysql/lib/mysql/ \
-L/usr/local/iodbc/lib/ \
-lz -lc -lmysqlclient -liodbcinst
```

iODBC でなく、unixODBC を使用している場合は、`-liodbcinst` を必ず `-lodbcinst` に変更し、ライブラリ パスを適切に構成してください。

これで `libmyodbc3-3.51.01.so` ファイルが構築され、`.libs` ディレクトリに収容されます。このファイルを Connector/ODBC ライブラリ インストール ディレクトリ (`/usr/local/lib`、もしくは `--prefix` と共に供給したインストール ディレクトリ下の `lib` ディレクトリ) にコピーしてください。

```
shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

スレッドセーフ ドライバ ライブラリを構築するには：

```
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error \
-o .libs/libmyodbc3_r-3.51.01.so \
catalog.o connect.o cursor.o dll.o error.o execute.o \
handle.o info.o misc.o myodbc3.o options.o prepare.o \
results.o transact.o utility.o \
-L/usr/local/mysql/lib/mysql/ \
-L/usr/local/iodbc/lib/ \
-lz -lc -lmysqlclient_r -liodbcinst
```

ドライバ ライブラリのインストール

ドライバ ライブラリをインストールするには、次のコマンドを実行してください：

```
shell> make install
```

このコマンドは、次のライブラリ セットのうちのどれかをインストールします：

Connector/ODBC 3.51 の場合：

- [libmyodbc3.so](#)
- [libmyodbc3-3.51.01.so](#) ドライバのバージョンは 3.51.01
- [libmyodbc3.a](#)

スレッドセーフ Connector/ODBC 3.51 の場合：

- [libmyodbc3_r.so](#)
- [libmyodbc3-3_r.51.01.so](#)
- [libmyodbc3_r.a](#)

MyODBC 2.5.0 の場合：

- [libmyodbc.so](#)
- [libmyodbc-2.50.39.so](#) ドライバのバージョンは 2.50.39
- [libmyodbc.a](#)

構築手順の詳しい情報は、ソース配布物に添付されている [INSTALL](#) ファイルを参照してください。Sun から [make](#) を使用すると、エラーが発生する恐れがありますので注意してください。一方、GNU [gmake](#) は、すべてのプラットフォームで問題なく使用できるでしょう。

Unix で Connector/ODBC をテストする

配布物に含まれている簡単なサンプルを、自作のライブラリで実行するには、次のコマンドを使用してください：

```
shell> make test
```

テストを実行する前に、[odbc.ini](#) 内に DSN 'myodbc3' を作成し、環境変数 [ODBCINI](#) を、適切な [odbc.ini](#) ファイルに設定してください。また、MySQL サーバは作動しています。サンプル [odbc.ini](#) は、ドライバ配布物に添付されています。

[samples/run-samples](#) スクリプトを改変し、各サンプルへのコマンドライン引数として、所望の DSN、UID、PASSWORD 値に受け渡すことも可能です。

Mac OS X でソースから Connector/ODBC を構築する

Mac OS X (Darwin) でドライバを構築するには、次の [configure](#) 例を利用してください：

```
shell> ./configure --prefix=/usr/local
--with-unixODBC=/usr/local
--with-mysql-path=/usr/local/mysql
--disable-shared
--enable-gui=no
--host=powerpc-apple
```

このコマンドでは、[unixODBC](#) と MySQL がデフォルト ロケーションにインストールされたものと仮定します。そうでない場合は、適切に構成してください。

Mac OS X では、デフォルト設定により [--enable-shared](#) が [.dylib](#) ファイルを構築するようになっています。[.so](#) を以下のように構築することができます：

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient -lz -lc
```

スレッドセーフ ドライバ ライブラリを構築するには：

```
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient_r -lz -lc -lpthread
```

iODBC でなく、unixODBC を使用する場合は、`-liodbcinst` を必ず `-lodbcinst` に変更し、ライブラリ パスを適切に構成してください。

Apple 版の GCC では、`cc` と `gcc` の双方が `gcc3` へのシンボリック リンクになっています。

このライブラリを `$prefix/lib` ディレクトリにコピーし、`libmyodbc3.so` へ symlink してください。

このコマンドを使用して、アウトプット共用ライブラリのプロパティをクロスチェックすることができます：

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

HP-UX でソースから Connector/ODBC を構築する

HP-UX 10.x もしくは 11.x でドライバを構築するには、次の `configure` 例を利用してください：

`cc` を使用する場合：

```
shell> CC="cc" \
CFLAGS="+z" \
LDFLAGS="-Wl,+b:-Wl,+s" \
./configure --prefix=/usr/local
--with-unixodbc=/usr/local
--with-mysql-path=/usr/local/mysql/lib/mysql
--enable-shared
--enable-thread-safe
```

`gcc` を使用する場合：

```
shell> CC="gcc" \
LDFLAGS="-Wl,+b:-Wl,+s" \
./configure --prefix=/usr/local
--with-unixodbc=/usr/local
--with-mysql-path=/usr/local/mysql
--enable-shared
--enable-thread-safe
```

ドライバが構築されたら、`chatr .libs/libmyodbc3.sl` を使ってその属性をクロスチェックし、`SHLIB_PATH` 環境変数を使用して MySQL クライアント ライブラリ パスを設定する必要があるか確認してください。スタティック バージョンでは、すべての共用ライブラリ オプションを無視し、`--disable-shared` オプションで `configure` を実行してください。

AIX でソースから Connector/ODBC を構築する

AIX でドライバを構築するには、次の `configure` 例を利用してください：

```
shell> ./configure --prefix=/usr/local
--with-unixodbc=/usr/local
--with-mysql-path=/usr/local/mysql
--disable-shared
--enable-thread-safe
```

注意：異なるプラットフォームに跨るスタティックおよび共用ライブラリの構築と設定に関しては、「[Using static and shared libraries across platforms](#)」で詳細をご覧ください。

開発ソースツリーから Connector/ODBC をインストールする

注意：このセクションは、当社の新しいコードのテストに協力していただける場合にのみお読みください。単に MySQL Connector/ODBC をシステム上で稼働させるだけであれば、標準のリリース配布物を使用してください。

Connector/ODBC ソースツリーにアクセスするには、Subversion をインストールする必要があります。Subversion は <http://subversion.tigris.org/> から無料で入手することができます。

ソースツリーから構築するには、次のツールが必要です：

- autoconf 2.52 (またはそれ以降)
- automake 1.4 (またはそれ以降)
- libtool 1.4 (またはそれ以降)
- m4

最新の開発ソースツリーは、<http://dev.mysql.com/tech-resources/sources.html> の公開 Subversion ツリーから得られます。

Connector/ODBC ソースを点検するには、Connector/ODBC ツリーのコピーを保存したいディレクトリに移動し、次のコマンドを使用します：

```
shell> svn co http://svn.mysql.com/svnpublic/connector-odbc3
```

これで、全 Connector/ODBC ソースツリーが、ディレクトリ `connector-odbc3` にコピーされました。Unix または Linux で、このソースツリーから構築する場合は、次の手順に従ってください：

```
shell> cd connector-odbc3
shell> aclocal
shell> autoheader
shell> autoconf
shell> automake;
shell> ./configure # Add your favorite options here
shell> make
```

構築に関する詳細は、同じディレクトリにある `INSTALL` ファイルを参照してください。`configure` の他のオプションについては、[典型的な configure オプション](#) をご覧ください。

構築が完了したら、`make install` を実行して、システムに Connector/ODBC 3.51 ドライバをインストールしてください。

`make` ステージで配布物がコンパイルしない場合は、[<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) までお知らせください。

Windows でのドライバの構築に、Windows Makefiles `WIN-Makefile` および `WIN-Makefile_debug` をご利用ください。詳細は [Connector/ODBC をソース配布物から Windows にインストールする](#) をご覧ください。

ソースツリーを得るための初回のチェックアウト操作のあと、ソースを定期的に最新バージョンにアップデートする `svn update` を実行してください。

Connector/ODBC の構成

Connector/ODBC ドライバを使用して MySQL データベースに接続する前に、ODBC Data Source Name を構成する必要があります。DSN は、データベースとの接続に必要な種々の構成パラメータを、特定の名称に関連づけます。アプリケーション自体の中で各パラメータを指定するより、アプリケーション内で DSN を使用し、データベースとの接続を行うことになります。DSN 情報は、ユーザー固有、システム特定、または特殊ファイルで提供されることもあります。ODBC データソース名は、プラットフォームと ODBC ドライバによって、異なる方法で構成されます。

データソース名

Data Source Name は、特定のデータベースとの接続のための構成パラメータと関連しています。通常、DSN は次のパラメータで構成されています：

- Name
- Hostname

- Database Name
- Login
- Password

In addition, different ODBC drivers, including Connector/ODBC, may accept additional driver-specific options and parameters.

DSN には 3 種類あります :

- System DSN は、全ユーザと特定のシステムのアプリケーションが利用できる DSN のグローバル定義です。通常 System DSN は、システム アドミニストレータ、または System DSN を作成する特定の許可を得たユーザのみが構成することができます。
- User DSN はそれぞれのユーザ特有のもので、各ユーザが常用するデータベース接続情報の保管にも使用できます。
- File DSN は、シンプルなファイルを用いて DSN 設定を定義しますファイル DSN はユーザとマシン間で共用することができ、そのため、DSN 情報をアプリケーションの一部として複数のマシンにインストール、もしくはデプロイする際、より実用的です。

DSN 情報は、プラットフォームや環境によって、別のロケーションに保管されます。

Windows での Connector/ODBC DSN の構成

Windows 内の [ODBC Data Source Administrator](#) は、DSN の作成、ドライバのインストールのチェック、そしてトレース (デバッグに使用) や接続のプールなどの ODBC システムの構成を可能にします。

使用している Windows のエディションやバージョンによって、[ODBC Data Source Administrator](#) が保存されているロケーションは異なります。

Windows Server 2003 で [ODBC Data Source Administrator](#) を開くには :

1. [Start](#) メニューで [Administrative Tools](#) を選択、そして [Data Sources \(ODBC\)](#) をクリックする。

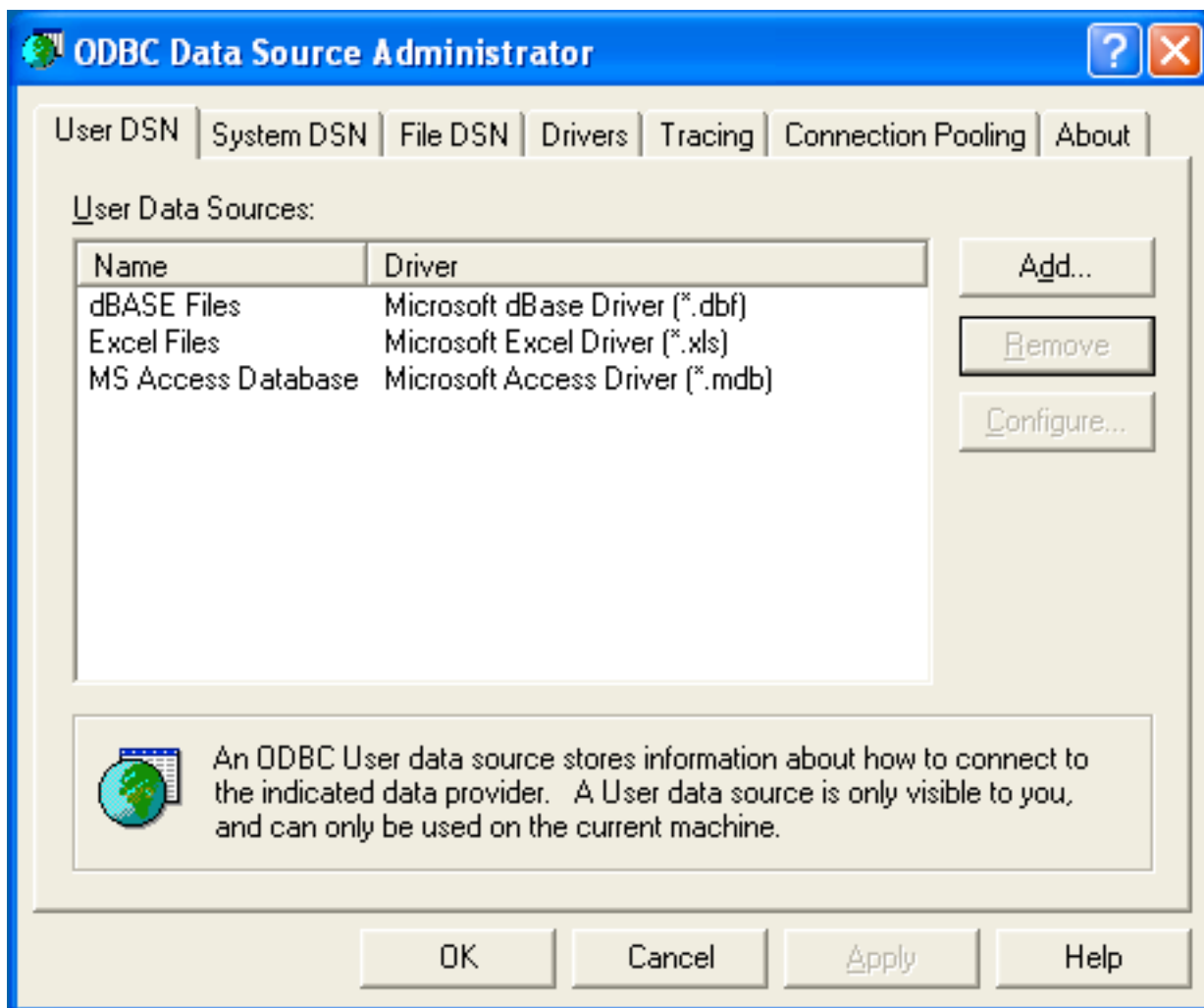
Windows 2000 Server または Windows 2000 Professional で [ODBC Data Source Administrator](#) を開くには :

1. [Start](#) メニューで [Settings](#) を選択、そして [Control Panel](#) をクリックする。
2. [Control Panel](#) で [Administrative Tools](#) をクリック。
3. [Administrative Tools](#) で、[Data Sources \(ODBC\)](#) をクリック。

Windows XP で [ODBC Data Source Administrator](#) を開くには :

1. [Start](#) メニューで [Control Panel](#) をクリック。
2. [Control Panel](#) の [Category View](#) で [Performance and Maintenance](#) をクリックし、そして [Administrative Tools](#) を押す。Classic View で [Control Panel](#) を見ている場合は、[Administrative Tools](#) をクリックする。
3. [Administrative Tools](#) で、[Data Sources \(ODBC\)](#) をクリック。

Windows のバージョンに関係なく、[ODBC Data Source Administrator](#) 画面が開きます :



Windows XP 内において、Start メニューに [Administrative Tools](#) フォルダを加え、ODBC Data Source Administrator の所在の特定をより容易にすることができます。実行の手順は次になります：

1. Start メニューを右クリックする。
2. [Properties](#) を選択。
3. [Customize...](#) をクリック。
4. [Advanced](#) タブを選択。
5. [System Administrative Tools](#) セクション内の [Start menu items](#) で、[Display on the All Programs menu](#) を選択。

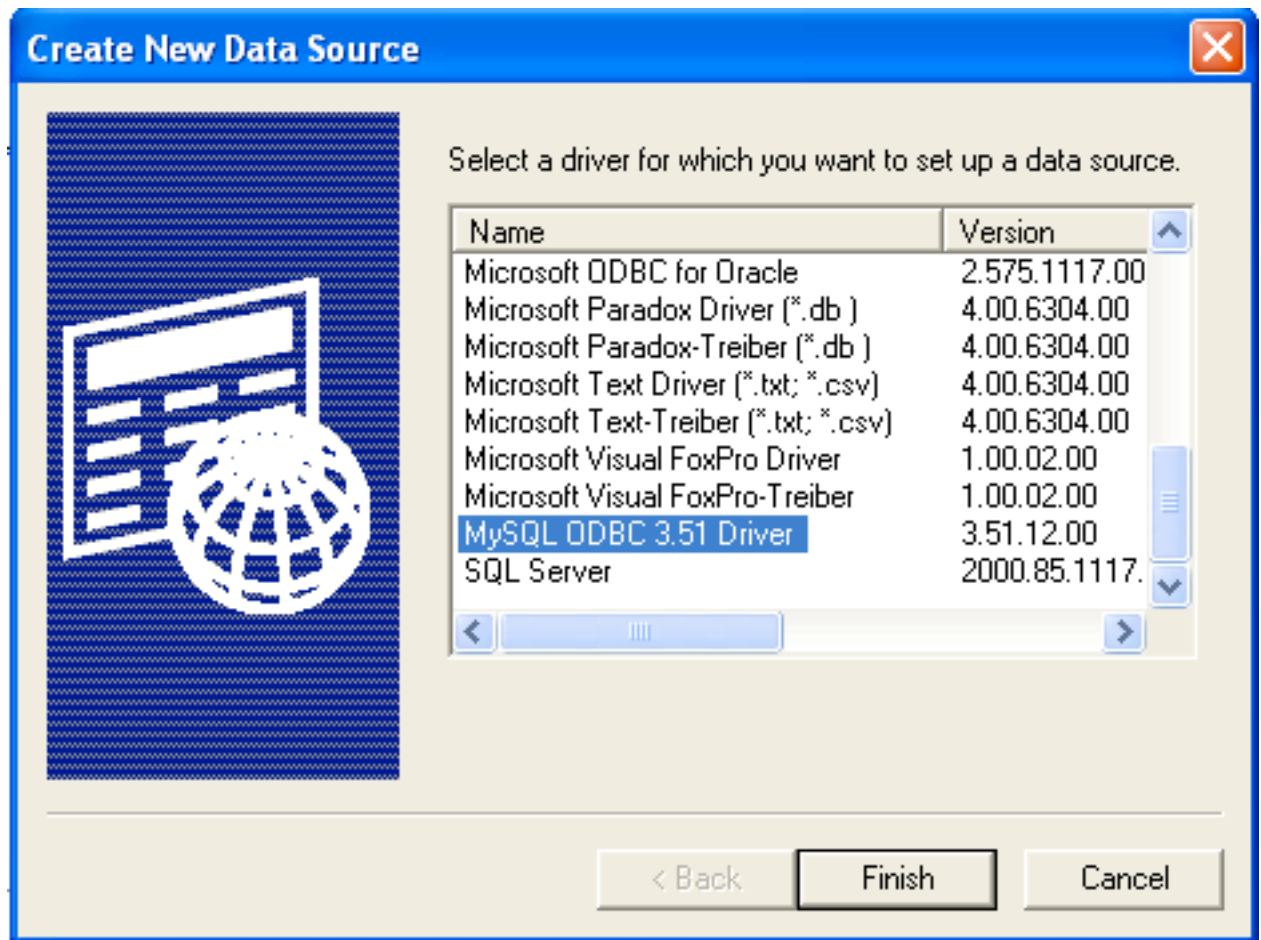
Windows Server 2003 と Windows XP、両方において、Start メニューに永久的に [ODBC Data Source Administrator](#) を加えることをお勧めします。それには、示された方法を使って [Data Sources \(ODBC\)](#) アイコンの所在を特定し、そのアイコンを右クリック、そして [Pin to Start Menu](#) を選択します。

Windows での Connector/ODBC DSN の追加

新たな Connector/ODBC データソースを Windows に追加および構成するには、[ODBC Data Source Administrator](#) を使用します：

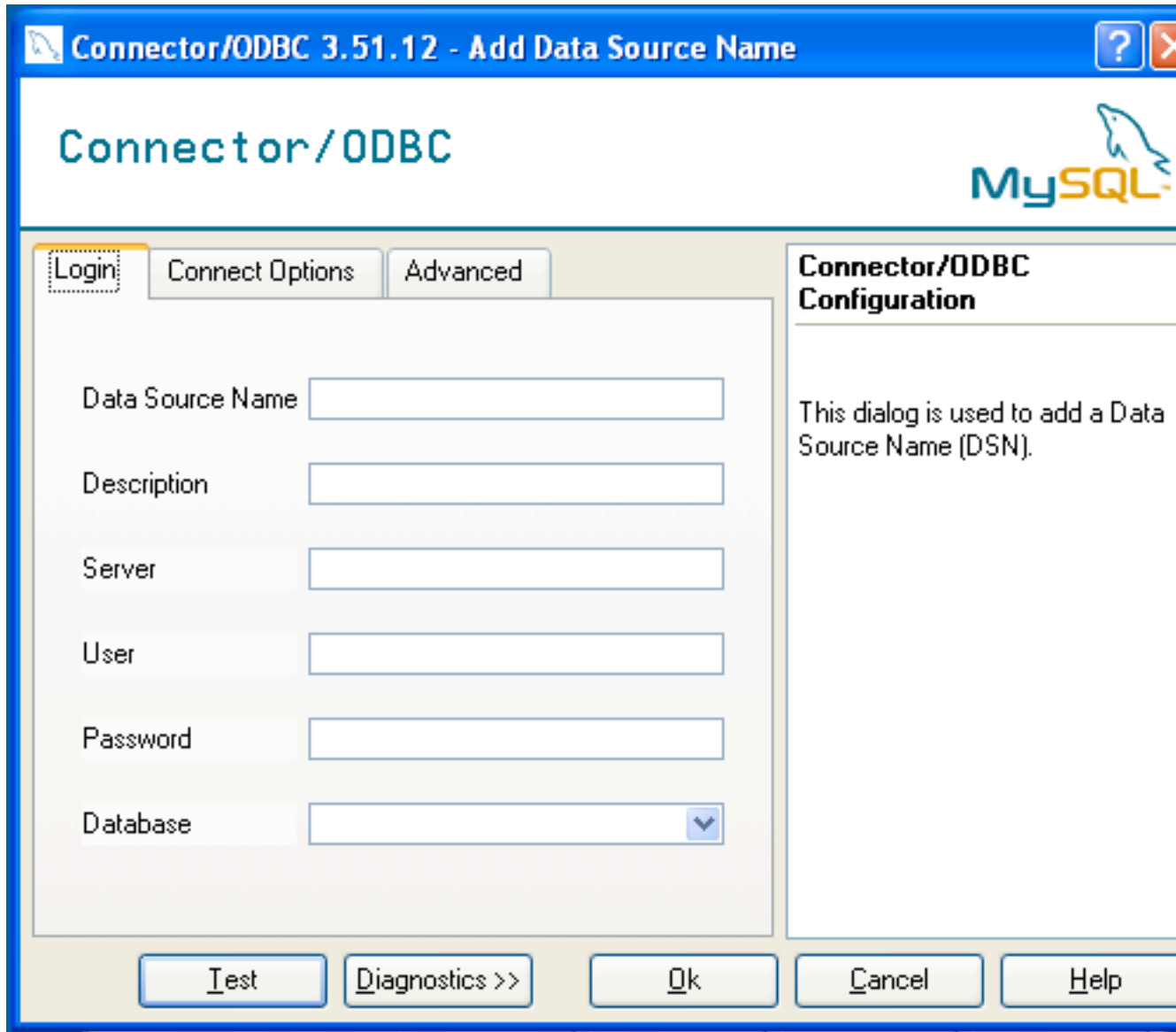
1. [ODBC Data Source Administrator](#) を開く。
2. TSystem DSN (すべてのユーザが使用可能) を作成するため、[System DSN](#) タブを選択。使用中のユーザ特定の User DSN を作成するため、[Add...](#) ボタンを押す。

3. その DSN のために、ODBC ドライバを選択。



MySQL ODBC 3.51 Driver を選択し、Finish を押す。

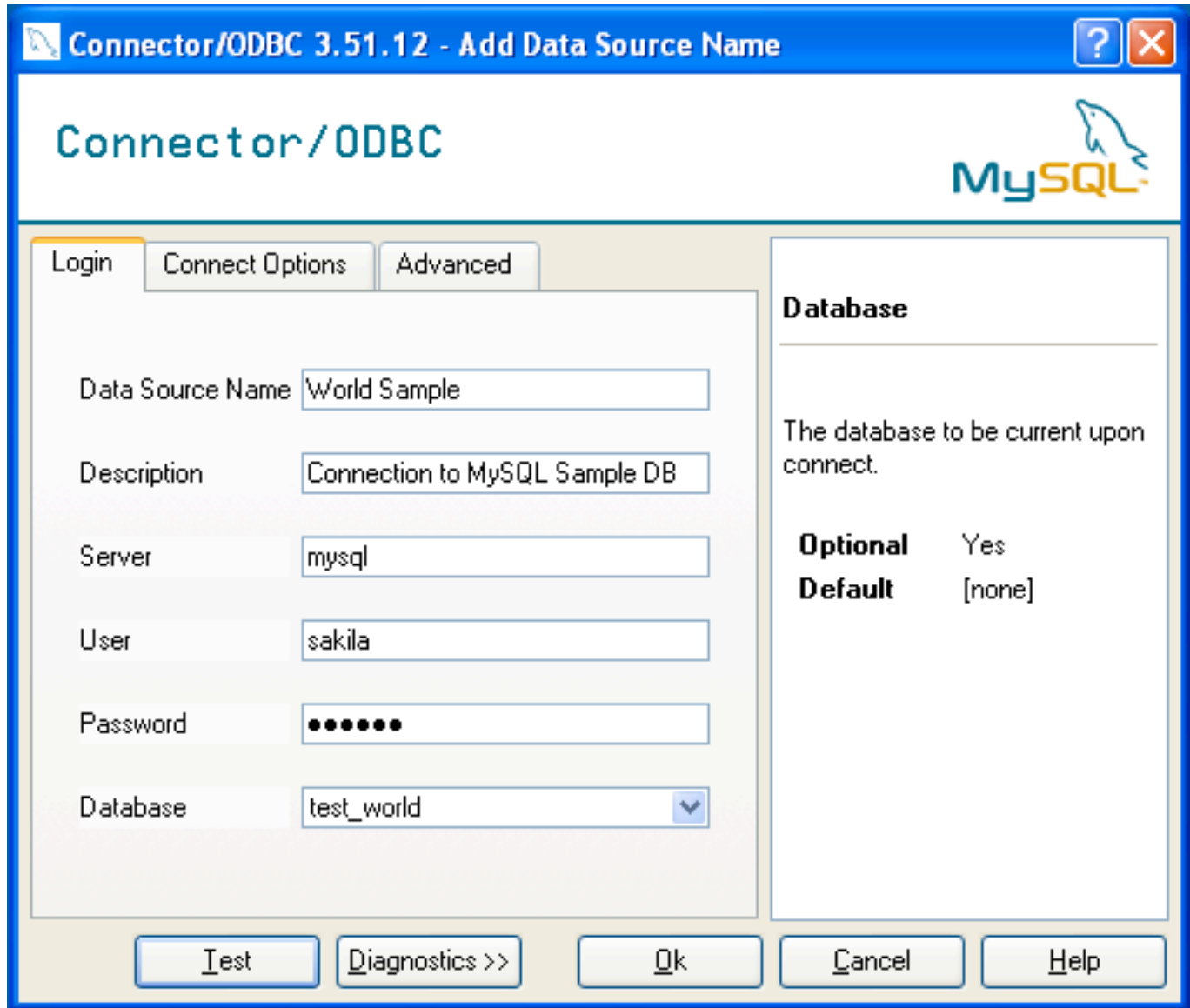
4. ここで、作成している DSN の特定のフィールドを、Add Data Source Name ダイアログを介して構成する。



Data Source Name ボックスに、アクセスしたいデータソースの名称を入力。有効な名前であれば、どんなものでもよい。

5. **Description** ボックスに何らかのテキストを入力し、コネクションを特定できるようにする。
6. **Server** フィールドに、アクセスしたい MySQL サーバ ホストの名称を入力。デフォルト設定では、**localhost**。
7. **User** フィールドに、この接続に使用するユーザー名を入力。
8. **Password** フィールドに、この接続へのパスワードを入力。
9. **Database** ポップアップが自動的に、アクセスが許可されたデータベースのリストを移植。
10. **OK** をクリックして、DSN を保存。

DSN 構成が完了すると、このような状態になります：



Windows で Connector/ODBC DSN 構成を検査

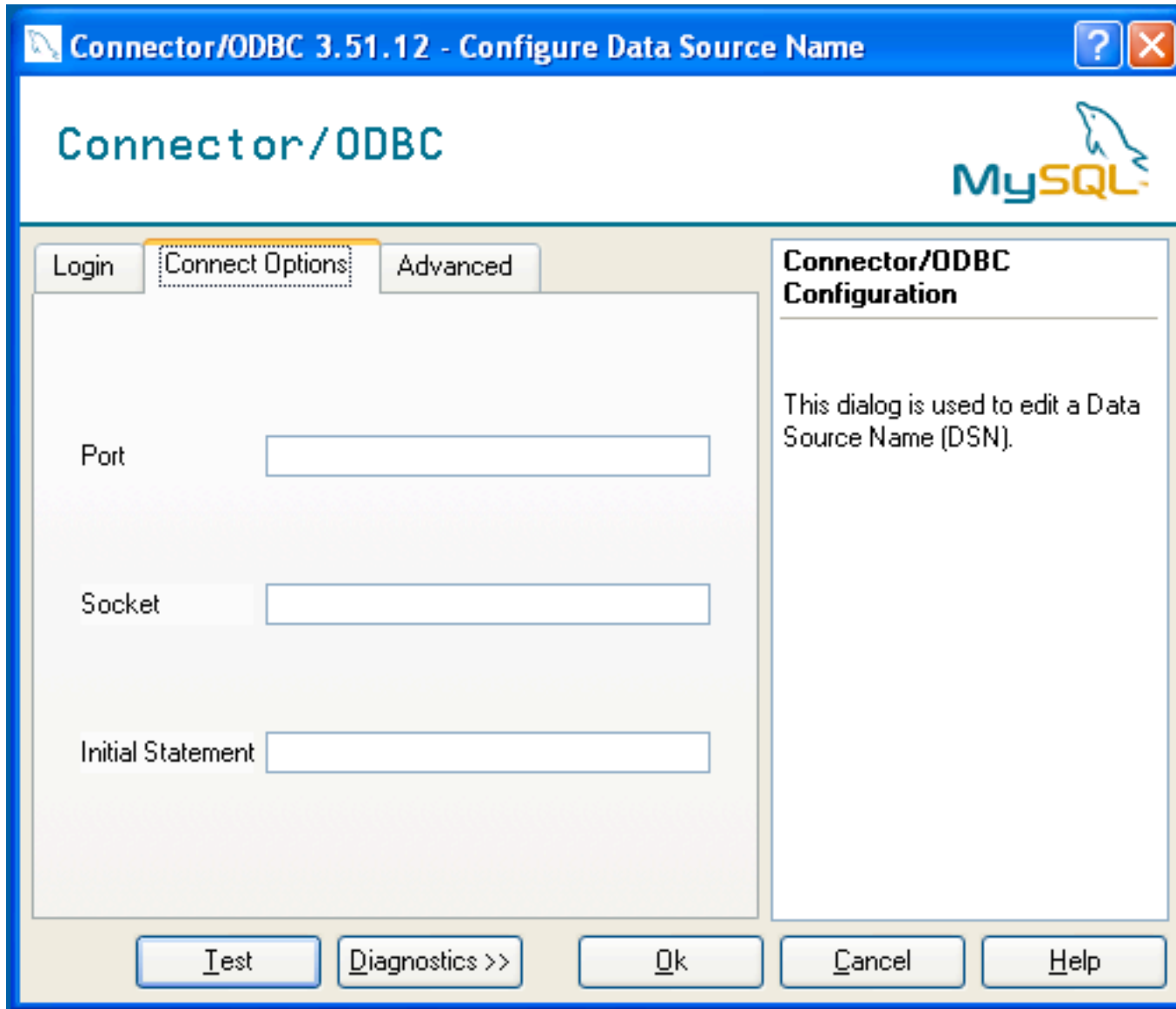
Test ボタンをクリックして入力したパラメータを使用して、接続を確認することができます。接続が成功した場合は、**Success; connection was made!** ダイアログが表示されます。

接続に失敗した場合は、**Diagnostics...** ボタンをクリックすると、テスト結果と失敗の原因が追加エラーメッセージで提示されます。

Connector/ODBC DSN 構成オプション

DSN 構成ダイアログの **Connect Options** もしくは **Advanced** タブで、特定の DSN に様々なオプションを構成することができます。

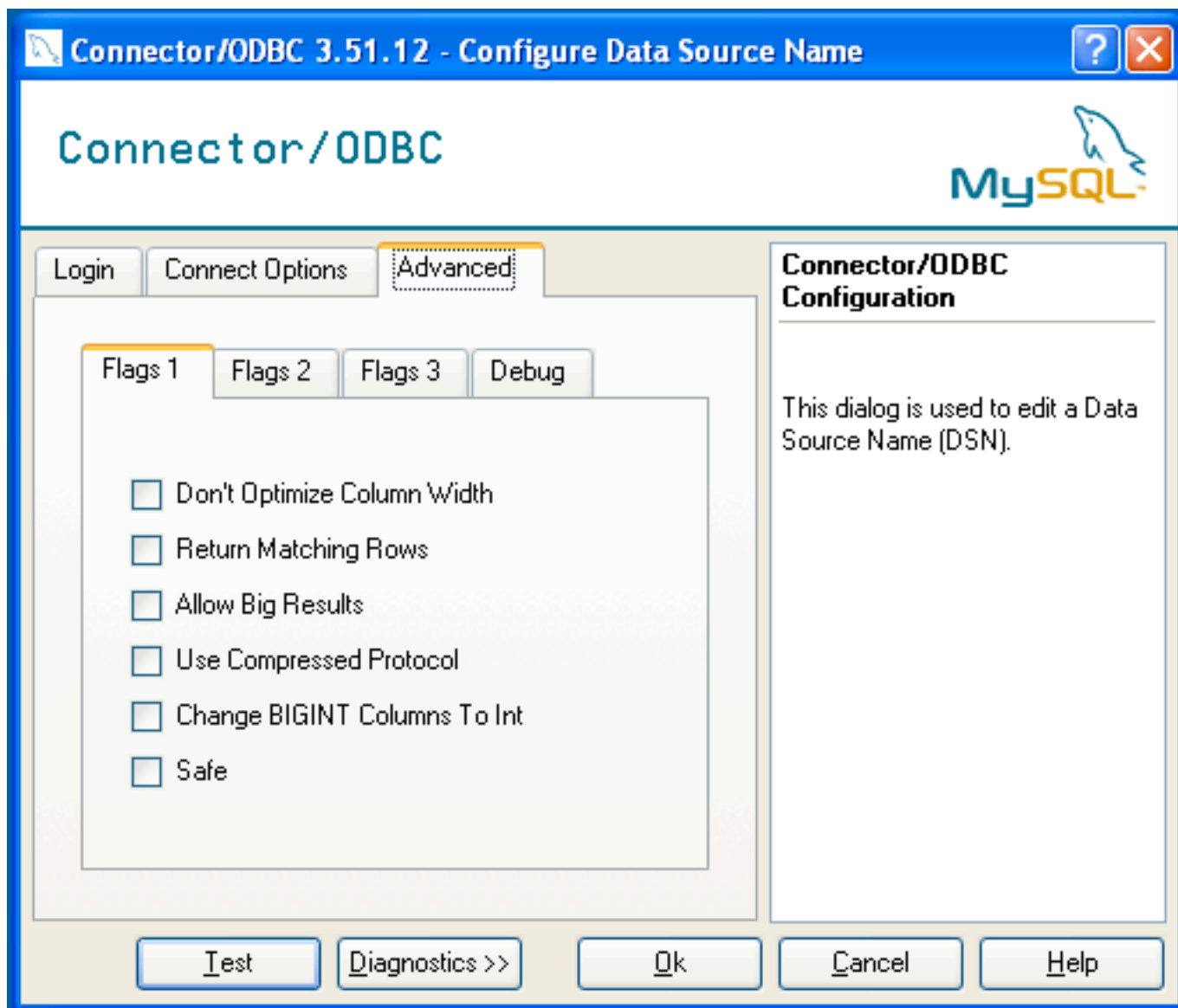
Connection Options ダイアログは次のようなものです。



構成が可能な 3 種類のオプション :

- **Port** は、MySQL との接続に使用する TCP/IP ポートナンバーを設定します。デフォルト設定では、MySQL との接続にはポート 3306 が使用されます。使用するサーバが、別の TCP/IP ポートを使うように構成されている場合は、ここでポートナンバーを特定する必要があります。
- **Socket** は、MySQL との接続に使用する特定のソケットか Windows パイプの名称、またはそのロケーションを設定します。
- **Initial Statement** は、MySQL への接続が開通した時に実行される SQL 文を定義します。これによって、デフォルトのキャラクタ セットや、接続中に使用するデータベースの設定など、接続の MySQL オプションをセットすることができます。

Advanced タブでは、Connector/ODBC 接続パラメータの構成をすることができます。これらのオプションの効果については、[Connector/ODBC Connection Parameters](#) を参照してください。



エラーとデバッグ

このセクションでは、Connector/ODBC の接続に関する質問に答えます。

- Connector/ODBC DSN の構成中、[Could Not Load Translator or Setup Library](#) エラーが発生します。

詳細は、[MS KnowledgeBase Article\(Q260558\)](#) をご覧ください。また、最新の有効な [cti3d32.dll](#) を、システム ディレクトリに忘れずに設置してください。

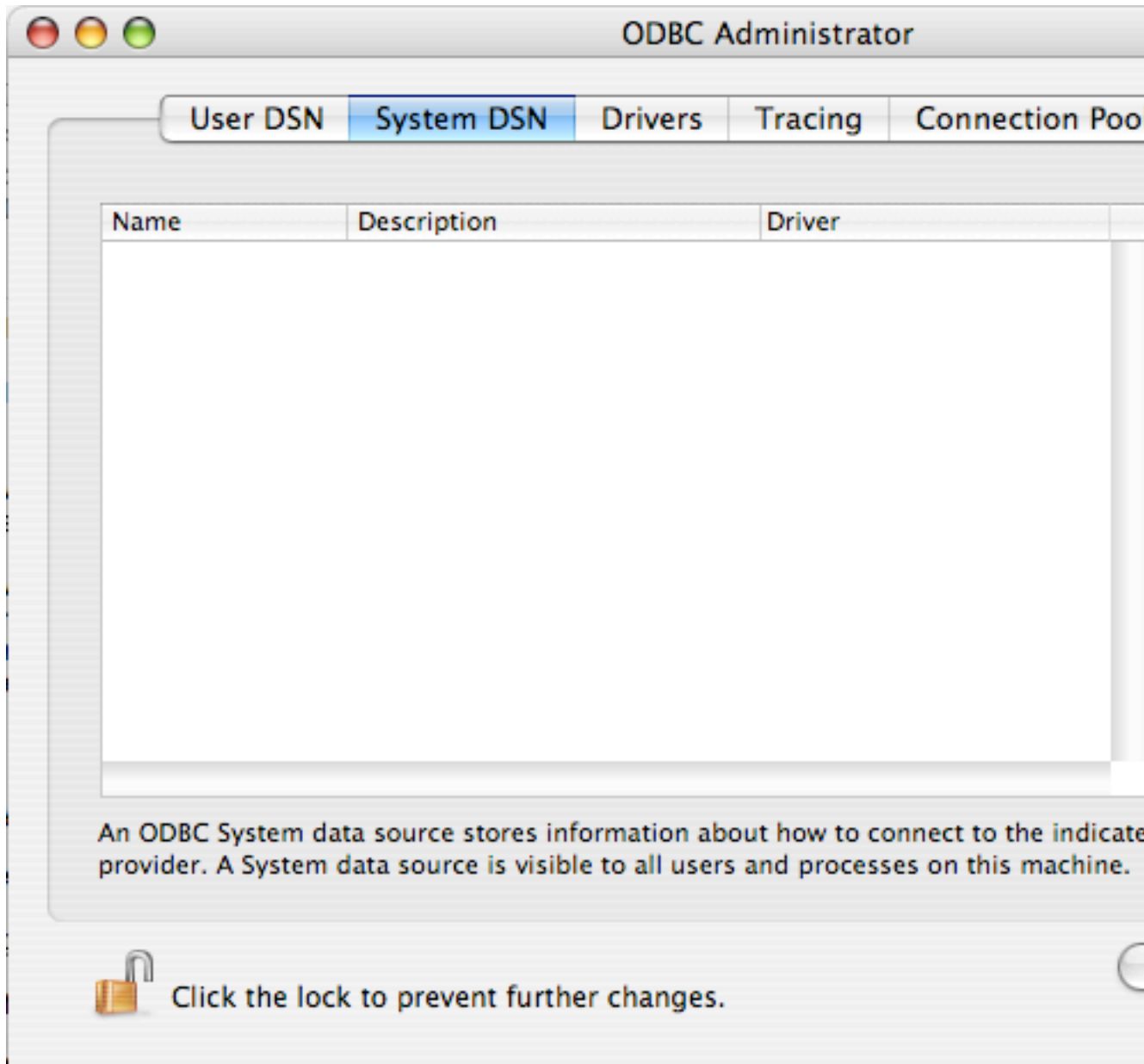
- Windows では、デフォルトの [myodbc3.dll](#) は最適な性能を得られるようコンパイルされています。Connector/ODBC 3.51 をデバッグしたい場合（例えば、トレースを有効にしたいなど）には、[myodbc3d.dll](#) の使用をお勧めします。このファイルをインストールするには、[myodbc3d.dll](#) を、インストールされた [myodbc3.dll](#) の上にコピーします。デバッグバージョンは機能に支障をきたす原因になる場合がありますので、デバッグが終了した時点で、ドライバ DLL のリリースバージョンに戻すようにしてください。[myodbc3d.dll](#) は、Connector/ODBC 3.51.07 から 3.51.11. には含まれておりませんので注意してください。それらのバージョンを使用している場合は、それ以前のバージョン（例：3.51.06）からその DLL をコピーしてください。

MyODBC 2.50 では、[myodbc.dll](#) および [myodbcd.dll](#) がかわりに使用されています。

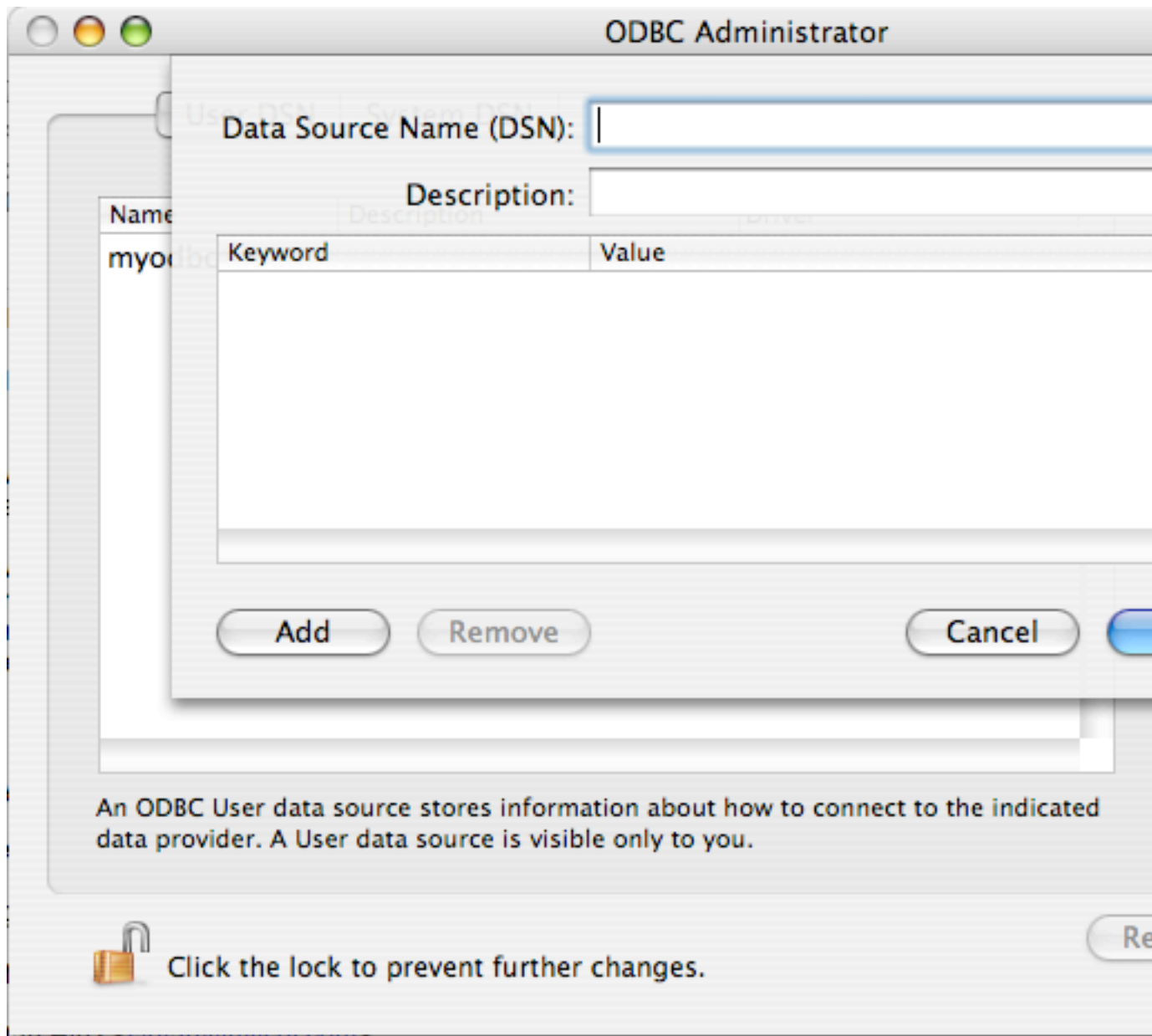
Mac OS X での Connector/ODBC DSN の構成

Mac OS X で DSN を構成するには、ODBC Administrator の使用をお勧めします。Mac OS X 10.2 以前を使用している場合は、[Unix での Connector/ODBC DSN の構成](#) をご覧ください。User DSN が System DSN で、作成したい方を選びます。System DSN を加える場合は、システムの認証が必要になる場合があります。鍵アイコンをクリックし、管理者権限でユーザー名とパスワードを入力してください。

1. [Applications](#) フォルダ内の [Utilities](#) フォルダから、ODBC Administrator を開く。



2. User DSN もしくは System DSN パネルで、Add をクリック。
3. Connector/ODBC ドライバを選択し、OK をクリック。
4. [Data Source Name](#) ダイアログが表示される。DSN に [Data Source Name](#) と、オプションの [Description](#) を入力。



5. Add をクリックして、新しいキーワードと値のペアをパネルに加える。少なくとも、[server](#)、[username](#)、[password](#)、そして [database](#) の接続パラメータを指定する 4 つのペアを構成する。詳細は [Connector/ODBC Connection Parameters](#) を参照。
6. OK をクリックして、DSN に構成されたデータのソースネームを加える。

DSN 構成が完了すると、このような状態になります：

Data Source Name (DSN): **WorldSample**

Description: **Connection to sample World database**

Keyword	Value
server	mysql
user	sakila
password	Sample
database	test_world

Buttons: Add, Remove, Cancel, OK

さらにキーワードと値のペアを加え、適切な値を設定することで、DSN に追加の ODBC オプションを構成することができます。詳細は [Connector/ODBC Connection Parameters](#) を参照。

Unix での Connector/ODBC DSN の構成

Unix では、DSN の入力を、`odbc.ini` ファイル内で直接構成します。下記は、MyODBC 2.50 と Connector/ODBC 3.51 に、DSN 名として `myodbc` および `myodbc3` をそれぞれ構成する典型的な `odbc.ini` です：

```

;
; odbc.ini configuration for Connector/ODBC and Connector/ODBC 3.51 drivers
;

```

```

[ODBC Data Sources]
myodbc      = MyODBC 2.50 Driver DSN
myodbc3     = MyODBC 3.51 Driver DSN

```

```

[myodbc]
Driver      = /usr/local/lib/libmyodbc.so
Description = MyODBC 2.50 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =

```

```

[myodbc3]
Driver      = /usr/local/lib/libmyodbc3.so
Description = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =

```

```
[Default]
Driver      = /usr/local/lib/libmyodbc3.so
Description = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =
```

提供されている接続パラメータのリストは、[Connector/ODBC Connection Parameters](#) を参照してください。

注記 :[unixODBC](#) を使用している場合は、次のツールを使って DSN を設定することができます：

- ODBCConfig GUI ツール ([HOWTO:ODBCConfig](#))
- odbcinst

[unixODBC](#) を使用していると、次のエラーが発生する場合があります：

```
Data source name not found and no default driver specified
```

このエラーが発生した時は、[ODBCINI](#) および [ODBCSYSINI](#) 環境変数が、正しい [odbc.ini](#) ファイルを示しているか確認してください。例えば、[odbc.ini](#) ファイルが [/usr/local/etc](#) にある場合、環境変数はこのように設定します：

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

Connector/ODBC Connection Parameters

You can specify the parameters in the following tables for Connector/ODBC when configuring a DSN. Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see the table for information on which options relate to which fields and checkboxes. Unix と Mac OS X ではパラメータ名と値を、DSN 構成のキーワードおよび値のペアとして使用してください。あるいは、[SQLDriverConnect\(\)](#) 呼び出しにある [InConnectionString](#) 引数内で、これらのパラメータを設定することもできます。

パラメータ	デフォルト値	コメント
user	ODBC (Windows)	MySQL の接続に使用されるユーザ名
server	localhost	MySQL サーバのホスト名
database		デフォルトのデータベース
option	0	Connector/ODBC の動作を指定するオプション。下記参照。
port	3306	server が localhost 出ない場合に、TCP/IP ポートが使用
stmt		MySQL への接続時に実行されるステートメント
password		server 上の user アカウントのパスワード
socket		server が localhost の場合に接続される Unix ソケット ファイルもしくは Windows の名前付きパイプ

[option](#) 引数は、クライアントの ODBC への適合が 100% でない場合に、Connector/ODBC にその旨を告知します。Windows では通常、接続スクリーンのチェックボックスをトグルしてオプションを選択しますが、[option](#) 引数でオプションを選択することもできます。次のオプションは、Connector/ODBC 接続スクリーンに表示される順序でリストされています：

値	Windows チェックボックス	概要
1	Don't Optimized Column Width	Connector/ODBC が実際のカラムの幅をリターンするのを、クライアントは許容できない。
2	Return Matching Rows	MySQL が、影響下にある行の真の値をリターンするのを、クライアントは許容できない。このフラグがセットされる場合、MySQL は代わりに「found rows」をリターンする。これには MySQL 3.21.14 以降が必要になる。

4	Trace Driver Calls To myodbc.log	デバッグ ログを、Windows では <code>C:\myodbc.log</code> 、Unix 形式では <code>/tmp/myodbc.log</code> に作成する。
8	Allow Big Results	結果とパラメータにパケット制限を設定しない。
16	Don't Prompt Upon Connect	ドライバが希望しても、質問をプロンプトしない。
32	Enable Dynamic Cursor	動的カーソルのサポートを有効または無効にする (Connector/ODBC 2.50. では利用不可)。
64	Ignore # in Table Name	<code>db_name.tbl_name.col_name</code> でのデータベース名の使用を無視。
128	User Manager Cursors	ODBC マネージャー カーソルの使用を強制実行する (実験用)。
256	Don't Use Set Locale	拡張されたフェッチの使用を無効にする (実験用)。
512	Pad Char To Full Length	<code>CHAR</code> カラムを最長のカラム幅にパッドする。
1024	Return Table Names for SQLDescribeCol	<code>SQLDescribeCol()</code> が完全に修飾されたカラム名をリターンする。
2048	Use Compressed Protocol	圧縮されたクライアント / サーバ プロトコルを使用。
4096	Ignore Space After Function Names	ファンクション名の後と、 <code>'</code> の前にあるスペースを無視するようサーバに通達 (PowerBuilder に必要)。これですべてのファンクション名がキーワードになる。
8192	Force Use of Named Pipes	名前つきパイプで、NT で起動している <code>mysqld</code> に接続する。
16384	Change BIGINT Columns to Int	<code>BIGINT</code> カラムを <code>INT</code> カラムに変更 (ある種のアプリケーションは <code>BIGINT</code> を処理できない)。
32768	No Catalog (exp)	<code>SQLTables</code> から、 <code>'user'</code> を <code>Table_qualifier</code> および <code>Table_owner</code> ととしてリターンする (実験用)。
65536	Read Options From my.cnf	<code>my.cnf</code> の <code>[client]</code> および <code>[odbc]</code> グループからパラメータを読み取る。
131072	Safe	追加のセーフティー チェックを加える (本来なら必要ない)。
262144	Disable transaction	トランザクションを無効にする。
524288	Save queries to myodbc.sql	<code>c:\myodbc.sql(/tmp/myodbc.sql)</code> ファイルへのクエリのロギングを有効にする (デバッグ モードでのみ有効)。
1048576	Don't Cache Result (forward only cursors)	結果をドライバ内でローカルにキャッシュせず、サーバ (<code>mysql_use_result()</code>) から読み取る。これは forward-only カーソルのみで利用できる。このオプションは、大きなテーブルを扱っていて、ドライバが全結果セットをキャッシュするのを防ぎたい時に重要になる。
2097152	Force Use Of Forward Only Cursors	<code>Forward-only</code> カーソルの使用を強制行使する。アプリケーションがデフォルトのスタティック / 動的カーソルの種類をセットし、それが非キャッシュの結果セットを使おうとする場合は、このオプションが forward-only カーソルの挙動を保証する。
4194304	Enable auto-reconnect.	自動再接続の機能を有効にする。不完全なトランザクションの間の自動再接続は破損の原因となるため、トランザクションとこのオプションを併用しないこと。自動再接続での接続は、元来の設定や環境を受け継いでいないので注意。このオプションは、Connector/ODBC 3.5.13. で有効。
8388608	Flag Auto Is Null	このオプションはセットの際に、接続が <code>SQL_AUTO_IS_NULL</code> オプションを 1 にセットする起因なる。標準の挙動が無効になるが、古いアプリケーションを有効にして、 <code>AUTO_INCREMENT</code> 値を正確に指定することもある。詳細は、を参照。このオプションは Connector/ODBC 3.5.13. で有効。

複数のオプションを選択するには、それらの値を加算します。例えば、option を 12 (4+8) にセットすると、パケット制限なしでデバッグすることができます。

次のテーブルは、様々な構成で推奨する option 値の例のリストです：

構成	オプション値
Microsoft Access、Visual Basic	3
ドライバトレース生成 (デバッグ モード)	4
Microsoft Access (DELETE クエリが改善されたもの)	35
行数過多の大規模なテーブル	2049
Sybase PowerBuilder	135168
クエリ ログ生成 (デバッグ モード)	524288
クエリ ログと同時にドライバトレースを生成 (デバッグ モード)	524292
非キャッシュ結果を伴う大規模なテーブル	3145731

定義済み DSN なしでの接続

DRIVER 名フィールドを指定することで、SQLDriverConnect を使用して MySQL サーバに接続することができます。以下は DSN-Less コネクションを使用した Connector/ODBC 用の接続ストリングです：

MyODBC 2.5.0 の場合：

```
ConnectionString = "DRIVER={MySQL};\
SERVER=localhost;\
DATABASE=test;\
USER=venu;\
PASSWORD=venu;\
OPTION=3;"
```

Connector/ODBC 3.51 の場合：

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
SERVER=localhost;\
DATABASE=test;\
USER=venu;\
PASSWORD=venu;\
OPTION=3;"
```

プログラム言語が、バックスラッシュに空白が続くとスペースに変換してしまう場合は、接続ストリングをひとつの長いストリングとして指定するか、複数のストリングを間にスペースを入れずにつなげて使用することをお勧めします。例：

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
\"SERVER=localhost;\
\"DATABASE=test;\
\"USER=venu;\
\"PASSWORD=venu;\
\"OPTION=3;"
```

注意： Mac OS X では、Connector/ODBC ドライバ ライブラリへの完全パスの指定が必要になる場合があります。

提供されている接続パラメータのリストは、[Connector/ODBC Connection Parameters](#) を参照してください。

ODBC 接続プーリング

接続プーリングは、データベースがアクセスを受けるたびに新しい接続を開くかわりに、ODBC ドライバが接続プールから、与えられたデータベースへの既存の接続を再利用することを可能にします。接続プーリングを有効することによって、データベースへの接続を開く手間が縮小され、アプリケーションの全体的な能力を高めることができます。

接続プーリングに関する情報は：<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470>。

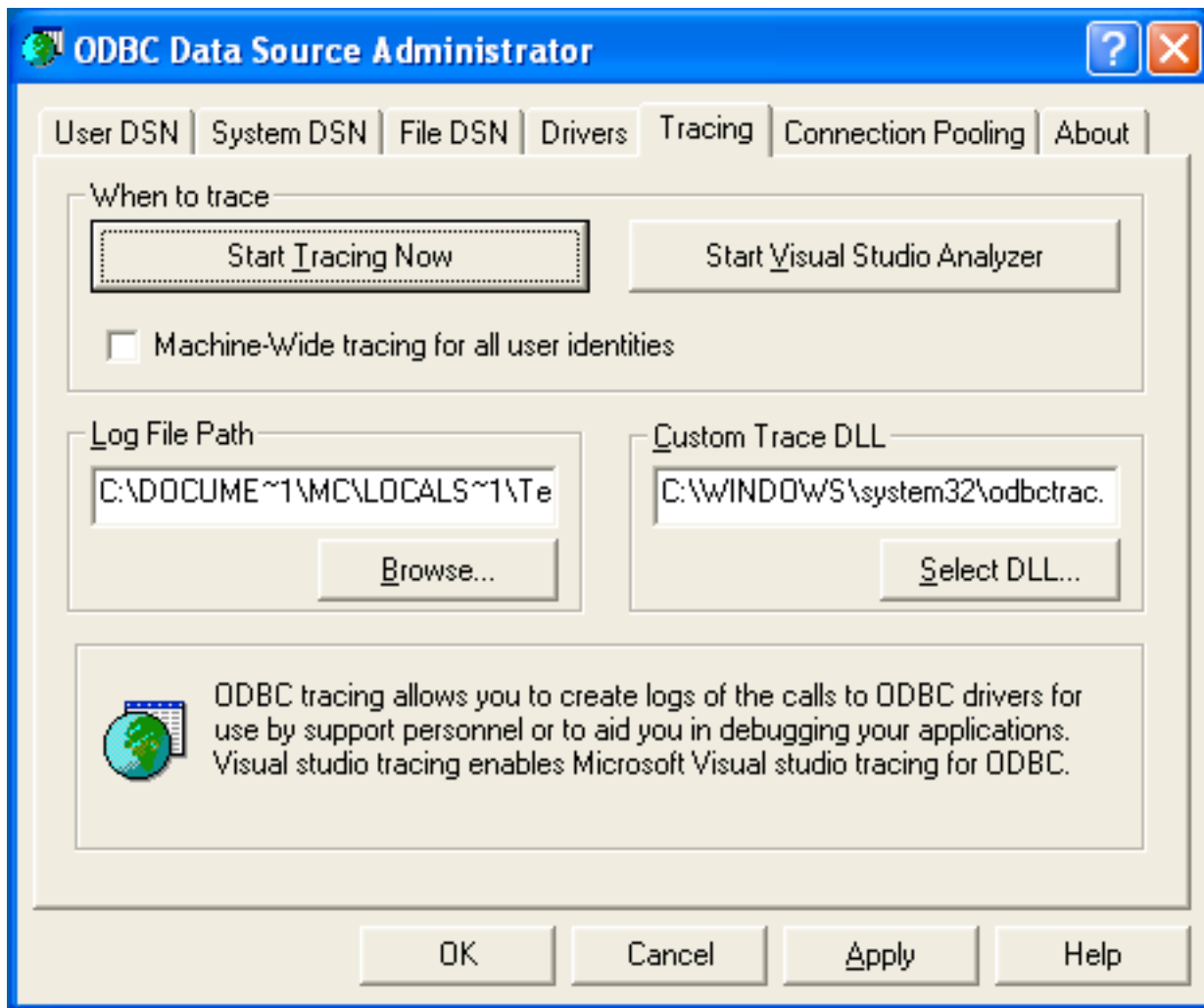
ODBC トレース ファイルの獲得

Connector/ODBC で不具合や問題があった場合は、[ODBC Manager](#) と Connector/ODBC からログ ファイルを作り始めましょう。これは tracing と呼ばれ、ODBC Manager を通して利用できます。そのプロシージャは、Windows、Mac OS X、および Unix 間で異なります。

Windows で ODBC トレーシングを有効にする

Windows でトレース オプションを有効にするには：

1. Data Source Administrator ダイアログ ボックスの **Tracing** タブで、ODBC 関数呼び出しのトレースの方法を構成することができます。

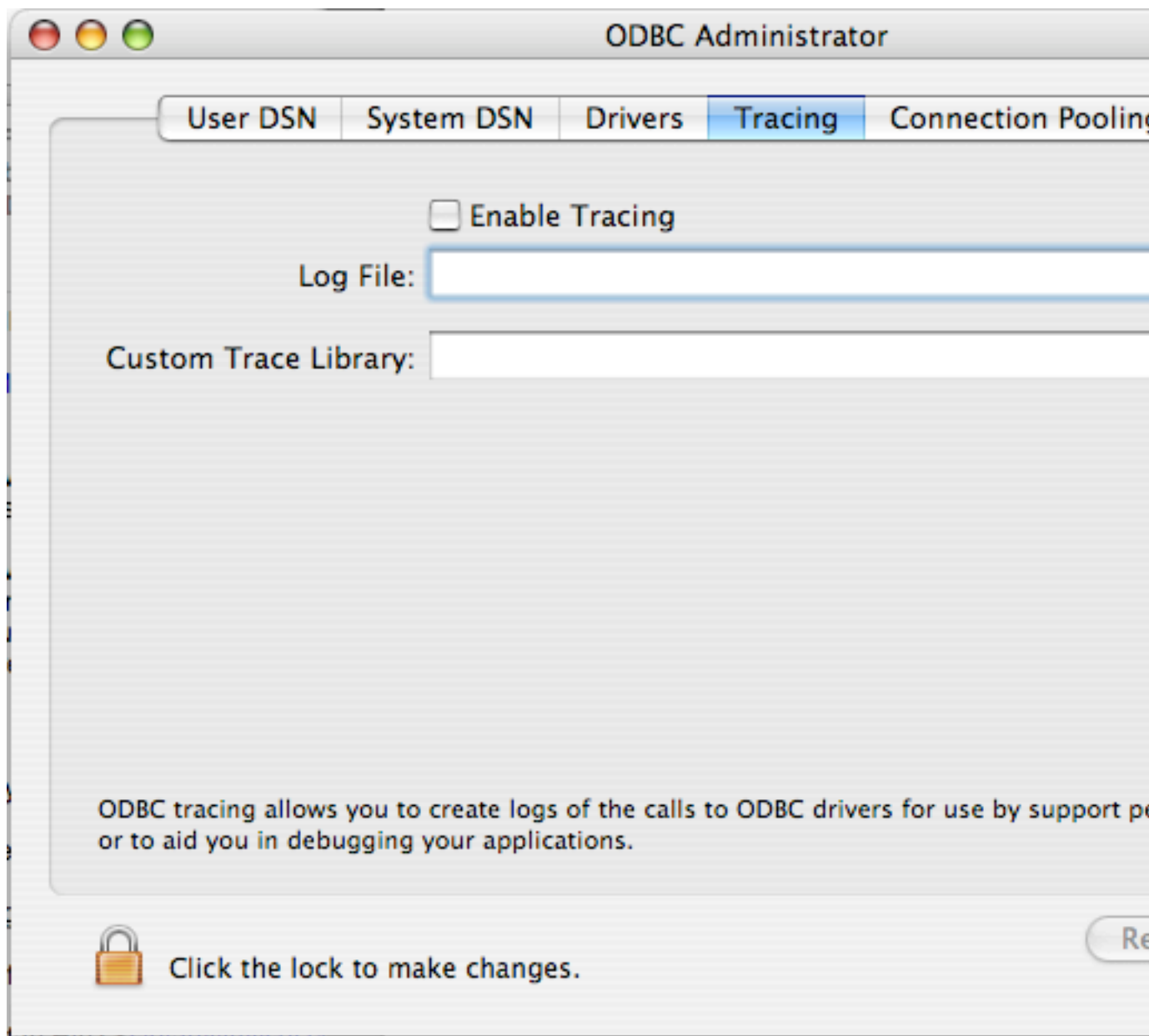


2. **Tracing** タブからトレーシングを発動すると、**Driver Manager** が後に起動するすべてのアプリケーションの、すべての ODBC 関数呼び出しを記録。
3. トレーシングが有効になる前に起動していたアプリケーションからの ODBC 関数呼び出しは記録されません。ODBC 関数呼び出しは、指定したログ ファイルに記録されます。
4. **Stop Tracing Now** をクリックするまで、トレーシングは続きます。トレーシングをしている間は、ログ ファイルのサイズは増量し続け、そのトレーシングはすべての ODBC アプリケーションのパフォーマンスに影響を及ぼします。

Mac OS X で ODBC トレーシングを有効にする

Mac OS X 10.3 以降でトレース オプションを有効にするには、ODBC Administrator 内の **Tracing** タブを使用してください。

1. ODBC Administrator を開く。
2. **Tracing** タブを選択。



3. **Enable Tracing** チェックボックスを選択。
4. Tracing ログを保存したいロケーションを入力。既存のログ ファイルに情報を追加したい場合は、**Choose...** ボタンをクリックする。

Unix で ODBC トレーシングを有効にする

Mac OS X 10.2 (またはそれ以降) か Unix でトレース オプションを有効にしたい場合は、ODBC 構成に **trace** オプションを加える必要があります：

1. Unix では、**ODBC.INI** ファイルの **Trace** オプションを明示的に設定する必要があります。

下のように、**odbc.ini** の **TraceFile** および **Trace** を使用して、トレーシングの **ON** または **OFF** を設定してください：

```
TraceFile = /tmp/odbc.trace
Trace     = 1
```

[TraceFile](#) がトレース ファイルの名前や完全パスを指定し、[Trace](#) は [ON](#) または [OFF](#) に設定されます。[ON](#) には、[1](#) または [YES](#)、[OFF](#) には [0](#) または [NO](#) を使用することもできます。[unixODBC](#) の [ODBCConfig](#) を使用する場合は、[HOWTO-ODBCConfig](#) にある [unixODBC](#) 呼び出しのトレース方法の手順に従ってください。

Connector/ODBC ログの有効化

Connector/ODBC ログを生成するには、次の手順に従ってください：

1. Windows 内で、Connector/ODBC connect/configure スクリーンの [Trace Connector/ODBC](#) オプション フラグを有効にする。[C:\myodbc.log](#) にログが書き込まれる。上のスクリーンに戻る時、トレース オプションが記憶されていなければ、[myodbcd.dll](#) が使用されていないということになる。[エラーとデバッグ](#) 参照。

Mac OS X、Unix では、またもしくは DSN-Less コネクションを使用している場合は、接続ストリングに [OPTION=4](#) を供給するか、該当するキーワードおよび値のペアを DSN に設定する必要がある。

2. アプリケーションを開始し、起動を失敗させてみる。それから Connector/ODBC トレースをチェックし、問題を確認する。

問題の特定にヘルプが必要な場合は、[Connector/ODBC のコミュニティー支援](#) をご覧ください。

Connector/ODBC 例

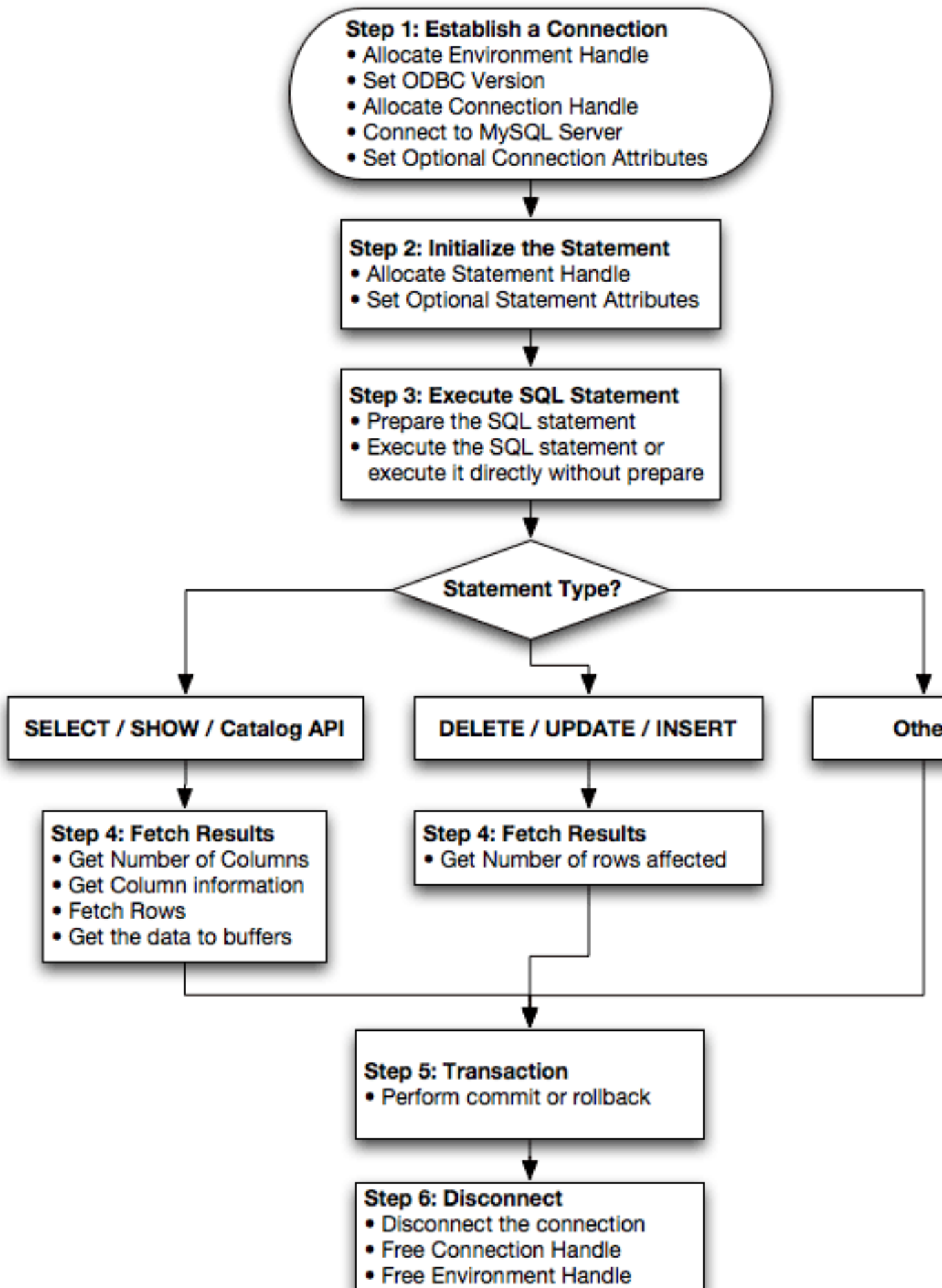
DSN を構成し、データベースへのアクセスを供給し終われば、実際の接続のアクセス方法や使用法は、アプリケーションまたはプログラム言語次第になります。ODBC は標準化されたインターフェイスなので、ODBC に対応するすべてのアプリケーションもしくは言語は DSN を使用することができ、構成されたデータベースに接続することが可能です。

Connector/ODBC アプリケーションの基本ステップ

通常、Connector/ODBC を使用したアプリケーションからの MySQL サーバとの干渉には、次のオペレーションが伴います：

- Connector/ODBC DSN の構成
- MySQL サーバへの接続
- 初期化動作
- SQL 文の実行
- 結果の検索
- トランザクションの実行
- サーバの接続解除

ほとんどのアプリケーションは、これらのステップのバリエーションを使用します。基本的なアプリケーションのステップは、次の図のようになります：



Connector/ODBC を介する MySQL データベースとの接続のステップガイド

Connector/ODBC を導入する動機としては、Windows のマシンから、Linux または Unix ホストのデータベースにアクセスしたいというのが一般的です。

2 台のマシン間のアクセス設定が必要なプロセスの例として、下記のステップで基本手順を説明します。この説明は、[myuser](#) と [mypassword](#) のユーザ名とパスワードで、システム BETA からシステム ALPHA へ接続するという前提でなされています。

システム ALPHA (MySQL サーバ) では、次のステップに従ってください：

1. MySQL サーバを起動。
2. [GRANT](#) を使用し、システム BETA から [myuser](#) のパスワードでデータベース [test](#) に接続できる、[myuser](#) ユーザ名でのアカウントを設定する：

```
GRANT ALL ON test.* to 'myuser'@'BETA' IDENTIFIED BY 'mypassword';
```

MySQL 特権に関する詳細は、[MySQL ユーザアカウント管理](#) を参照してください。

システム BETA (Connector/ODBC クライアント) では、次のステップに従ってください：

1. システム ALPHA で構成したサーバ、データベースおよび認証情報にマッチするパラメータを使用して、Connector/ODBC DSN を構成する。

パラメータ	値	コメント
DSN	remote_test	接続を特定する名前
SERVER	ALPHA	リモート サーバのアドレス
DATABASE	test	デフォルト データベースの名前
USER	myuser	このデータベースへのアクセスのために作成されたユーザ名
PASSWORD	mypassword	myuser へのパスワード

2. Microsoft Office など、ODBC 対応アプリケーションを使用すると、作成したばかりの DSN を使って MySQL に接続することができる。接続に失敗した場合は、接続プロセスを検査するためにトレーシングを使う。詳細は [ODBC トレース ファイルの獲得](#) を参照。

Connector/ODBC と サード パーティ ODBC ツール

独自の Connector/ODBC DSN を構成し終えたら、プログラム言語やサード パーティ アプリケーションを含む、ODBC インターフェースに対応するすべてのアプリケーションを通して、MySQL データベースにアクセスすることができます。このセクションは、様々な ODBC 対応のツールとおよび Microsoft Word、Microsoft Excel、Adobe/Macromedia ColdFusion を含むアプリケーションの使用に関するガイドとヘルプを含みます。

Connector/ODBC は次のアプリケーションでテストされています：

発行元	アプリケーション	注記
Adobe	ColdFusion	以前は Macromedia ColdFusion
Borland	C++ Builder	
	Builder 4	
	Delphi	
Business Objects	Crystal Reports	
Claris	Filemaker Pro	
Corel	Paradox	
Computer Associates	Visual Objects	別称 CAVO

	AllFusion ERwin Data Modeler	
Gupta	Team Developer	以前の名称は Centura Team Developer; Gupta SQL/Windows
Gensym	G2-ODBC Bridge	
Inline	iHTML	
Lotus	Notes	バージョン 4.5 および 4.6
Microsoft	Access	
	Excel	
	Visio Enterprise	
	Visual C++	
	Visual Basic	
	ODBC.NET	C#、Visual Basic、C++ を使用
	FoxPro	
	Visual Interdev	
OpenOffice.org	OpenOffice.org	
Perl	DBD::ODBC	
Pervasive Software	DataJunction	
Sambar Technologies	Sambar Server	
SPSS	SPSS	
SoftVelocity	Clarion	
SQLExpress	SQLExpress for Xbase++	
Sun	StarOffice	
SunSystems	Vision	
Sybase	PowerBuilder	
	PowerDesigner	
theKompany.com	Data Architect	

他にも Connector/ODBC に対応するアプリケーションがあれば、<myodbc@lists.mysql.com> までメールでお知らせください。

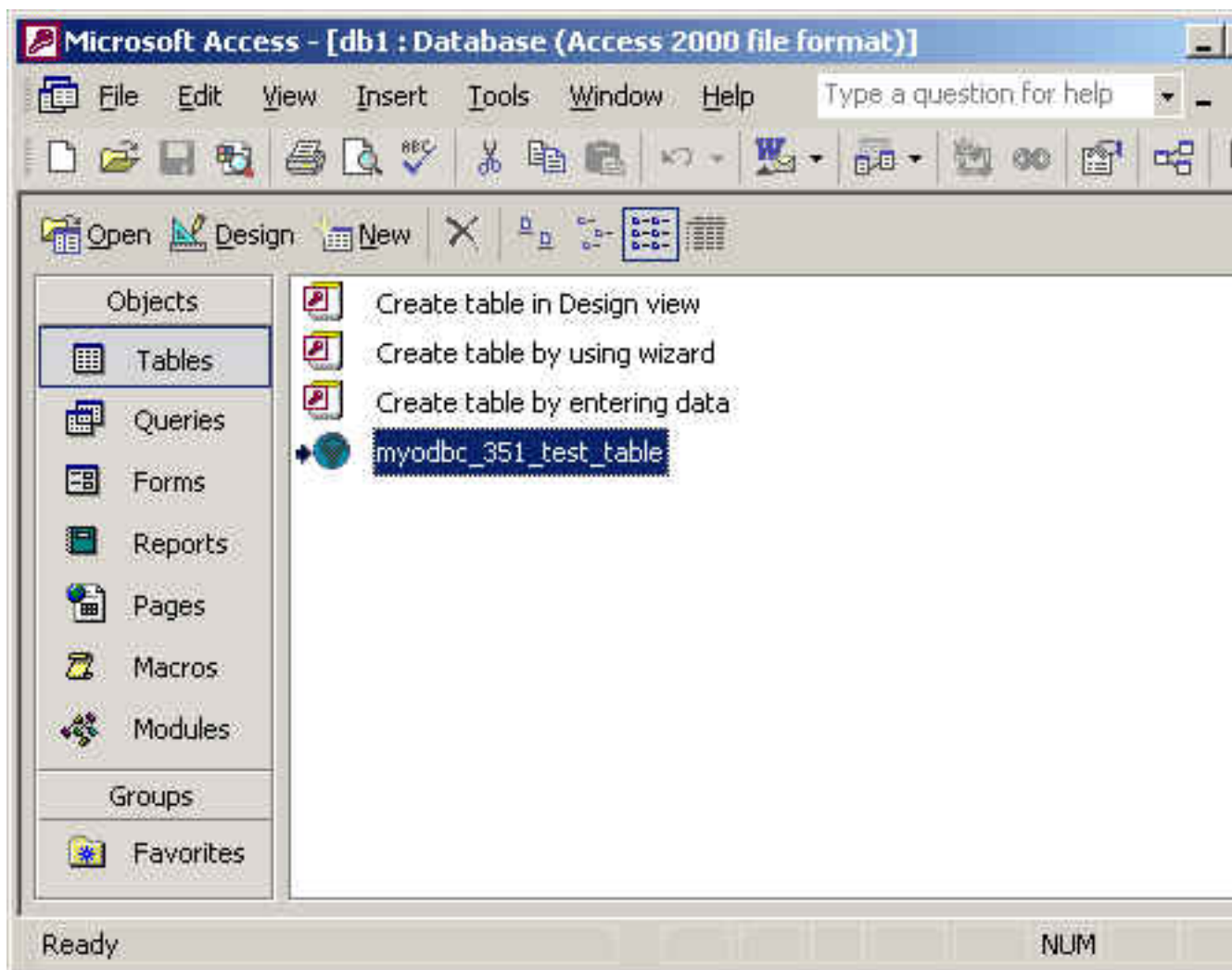
Microsoft Access で Connector/ODBC を使用する

Connector/ODBC を使用して、Microsoft Access で MySQL データベースを利用することができます。MySQL データベースは Access アプリケーション内で、インポート元、エクスポート元、または直接使用のためにリンクされたテーブルとして利用できるため、Access を MySQL データベースへのフロントエンド インターフェイスとして使うことができます。

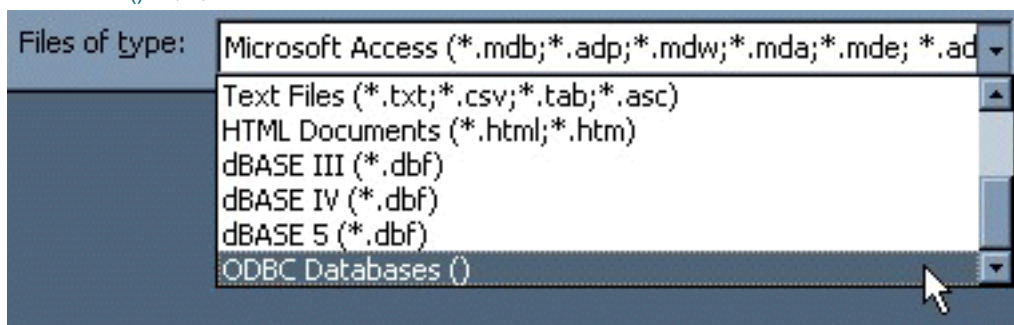
Access データを MySQL へエクスポートする

データのテーブルを Access データベースから MySQL にエクスポートするには、次の手順に従ってください：

1. Access データベース、もしくは Access プロジェクトを開くと、Database ウィンドウが出現。新しいデータベース オブジェクトの作成、そして既存のオブジェクトのオープンのショートカットが表示される。



2. エクスポートしたい **table** もしくは **query** の名前をクリックし、**File** で **Export** を選択。
3. **Export Object Type Object name To** ダイアログ ボックスの **Save As Type** ボックスで、下記のように **ODBC Databases ()** を選択：



4. **Export** ダイアログ ボックスにファイル名 (もしくは提示される名前を使用) を入力し、**OK** を選択。
5. **Select Data Source** ダイアログボックスが出現。コンピュータにインストールされたすべての ODBC ドライバの、定義されたデータソースがリストアップされる。File Data Source もしくは Machine Data Source タブをクリックし、エクスポートしたい Connector/ODBC が Connector/ODBC 3.51 のデータソースをダブルクリックする。Connector/ODBC の新たなデータソースを定義するには、[Windows での Connector/ODBC DSN の構成](#) を参照。

Microsoft Access がこのデータソースを介して MySQL サーバに接続し、新しいテーブル、そしてまたはデータをエクスポートします。

Access に MySQL データをインポートする

テーブルを MySQL から Access にインポートするには、次の手順に従ってください：

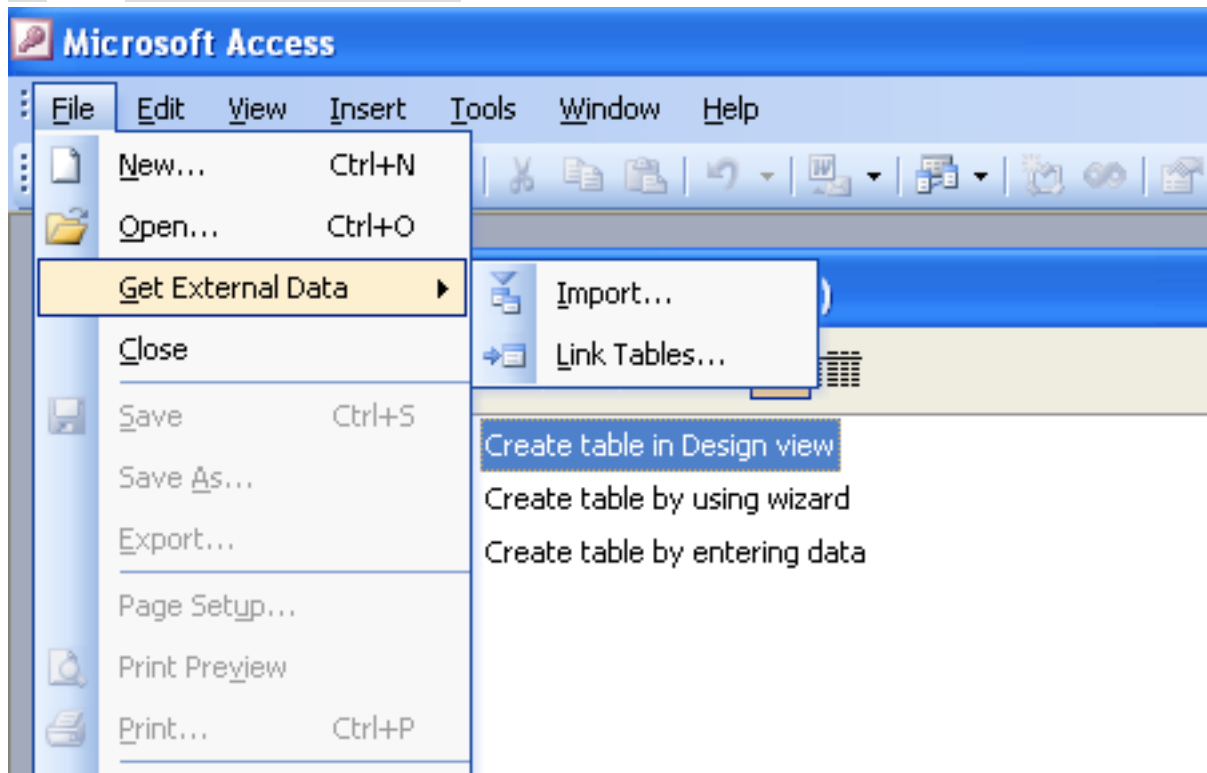
1. データベースを開くか、Database ウィンドウに切り替えてデータベースを開く。
2. テーブルをインポートするには、**File** メニューで **Get External Data** を選び、**Import** をクリック。
3. **Import** ダイアログ ボックスの **Files Of Type** ボックスで、**ODBC Databases ()** を選択。定義されたデータソースをリストアップした、**The Select Data Source** ダイアログ ボックスが表示される。
4. 選択した ODBC データソースにはログインが必要な場合は、ログイン ID とパスワード (場合によってはその他の情報も) 入力し、**OK** をクリック。
5. Microsoft Access が **ODBC data source** を介して MySQL サーバに接続し、**import** が可能なテーブルのリストを表示する。
6. **import** したい各テーブルをクリックし、**OK** を押す。

Microsoft Access を MySQL のフロントエンドとして使用する

Microsoft Access データベース内のテーブルと、MySQL データベース内のテーブルをリンクすることで、Microsoft Access を MySQL データベースのフロントエンドとして使うことができます。Access のテーブルでクエリが要求された場合、ODBC を使って MySQL でクエリが実行されます。

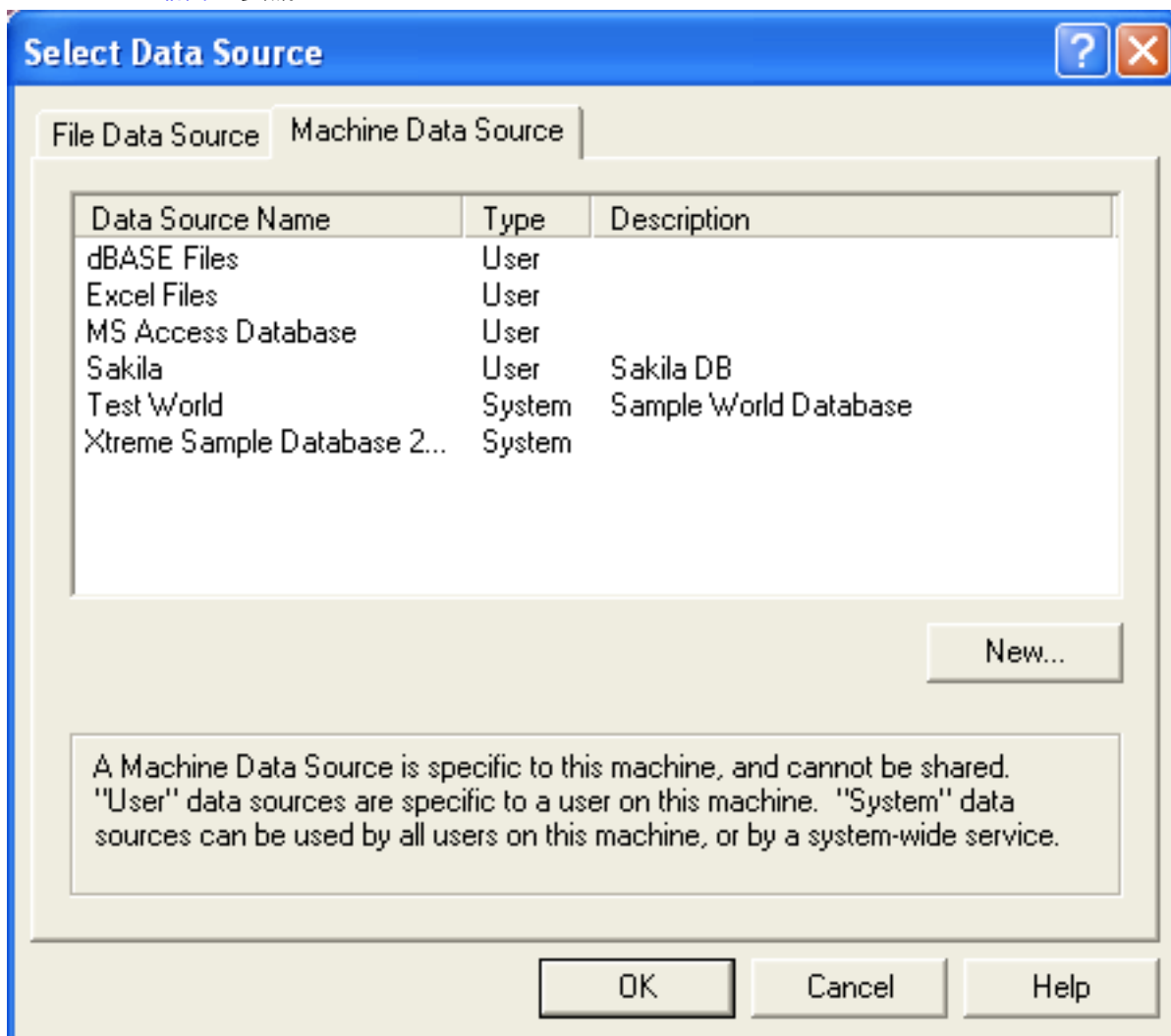
テーブルのリンクを作成：

1. MySQL にリンクしたい Access Database を開く。
2. **File** から、**Get External Data->Link Tables** を選択。

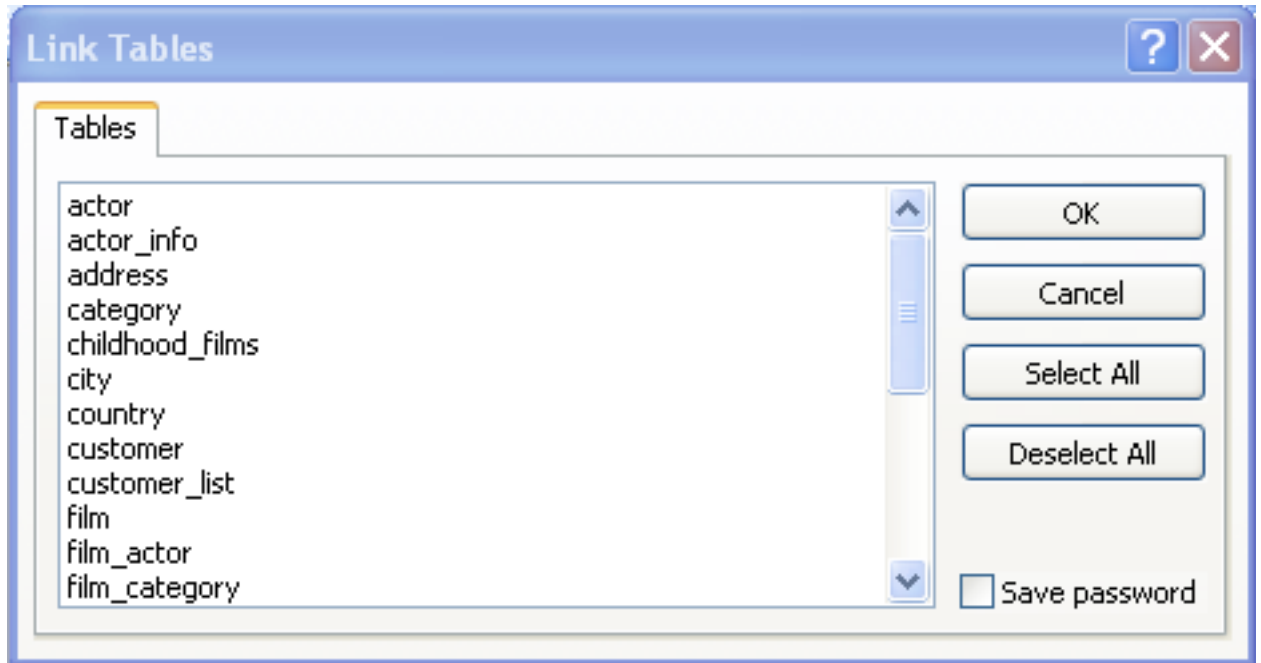


3. ブラウザで、**Files of type** ポップアップから **ODBC Databases ()** を選択。

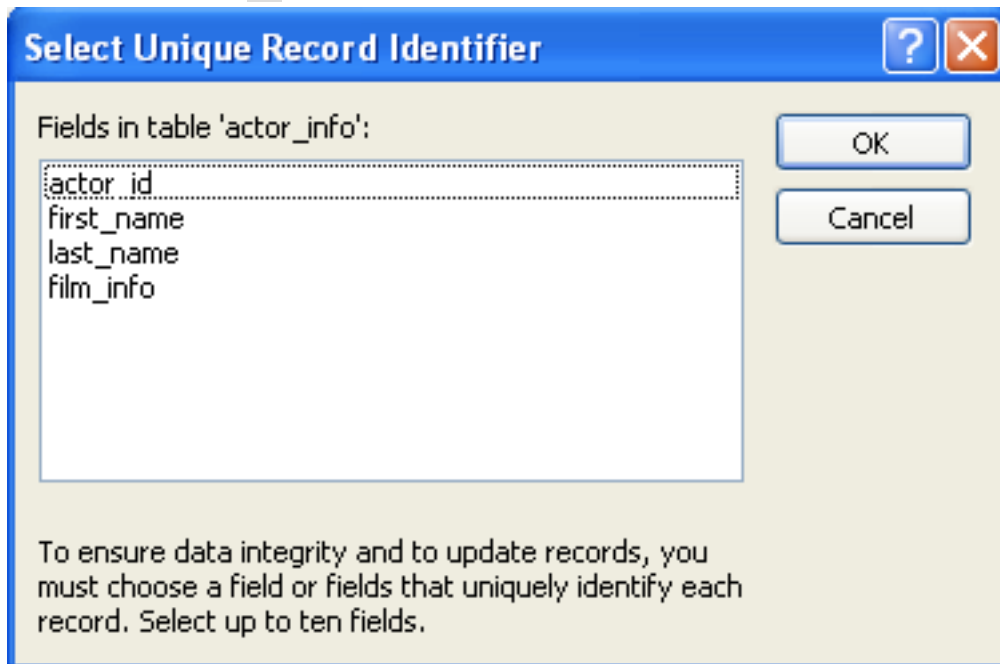
4. Select Data Source ウィンドウで既存の DSN を、File Data Source か Machine Data Source から選ぶ。または、New... ボタンで新たな DSN を作成することもできる。DSN の作成については、[Windows での Connector/ODBC DSN の構成](#) を参照。



5. Link Tables ダイアログで、MySQL データベースからひとつ以上のテーブルを選択。リストから選択した各テーブルに対してリンクが作成される。



6. Microsoft Access が自動的にテーブルからユニークレコード識別子を決定しない場合、ソーステーブルの各横列をユニークに特定するため、各カラム、もしくはカラムのコンビネーションの確認が要求されることがある。使用したいカラムを選び、OK を選択する。



プロセスが完了すれば、Access データベースできるように、リンクされたテーブルにインターフェイスやクエリを作成することができます。

リンクしたテーブルのストラクチャやロケーションが変わった場合は、次のプロシージャを使用して閲覧するか、リンクをリフレッシュします。Linked Table Manager が、現在リンクされているすべてのテーブルへのパスをリストアップします。

リンクの閲覧もしくはリフレッシュ：

1. MySQL テーブルへのリンクを含むデータベースを開く。
2. **Tools** メニューで、**Add-ins** (Access 2000 以降では **Database Utilities**) を指定し、**Linked Table Manager** をクリック。
3. リフレッシュしたいリンクがあるテーブルのチェックボックスをクリック。
4. **OK** をクリックし、リンクをリフレッシュ。

Microsoft Access がリフレッシュの成功を確認し、テーブルが見つからなかった場合は、**Select New Location of < テーブル名 >** ダイアログ ボックスが表示されて、そのテーブルの新たなロケーションを特定することができます。複数の選択されたテーブルが、特定した新たなロケーションに移された場合、Linked Table Manager が、選択されたテーブルのすべてのロケーションを検索し、すべてのリンクを一度にアップデートします。

リンクされたテーブル グループのパス変更：

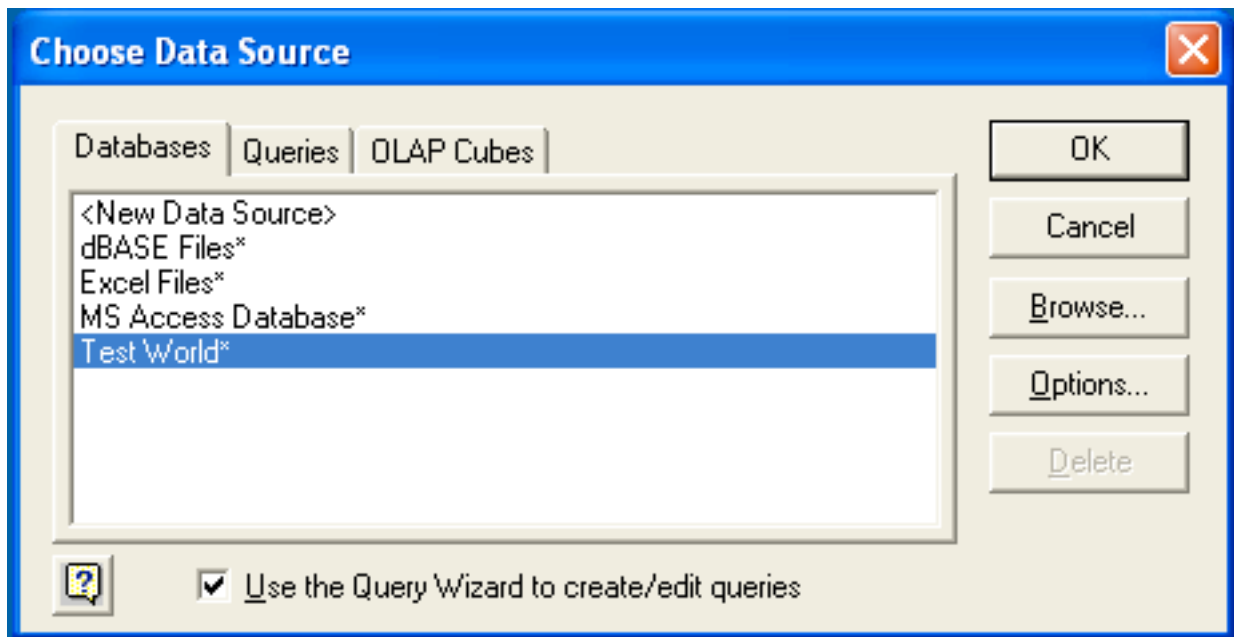
1. テーブルへのリンクを含むデータベースを開く。
2. **Tools** メニューで、**Add-ins** (Access 2000 以降では **Database Utilities**) を指定し、**Linked Table Manager** をクリック。
3. **Always Prompt For A New Location** チェックボックスを選択。
4. リフレッシュしたいリンクがあるテーブルのチェックボックスをクリックし、**OK** を押す。
5. **Select New Location of < テーブル名 >** ダイアログ ボックスで、新しいロケーションを指定して **Open** をクリック、そして **OK** を押す。

Microsoft Word または Excel で Connector/ODBC を使用する

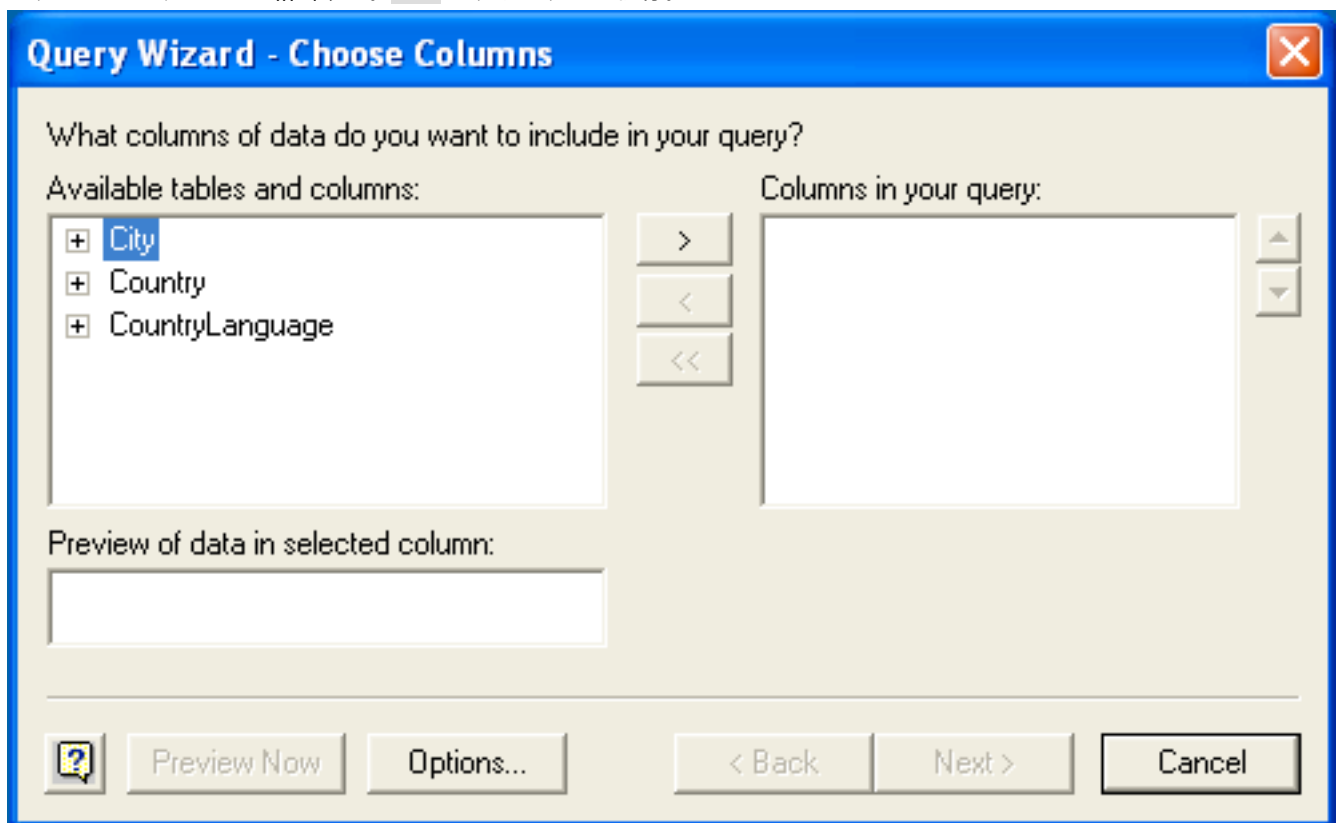
Microsoft Word と Microsoft Excel で、Connector/ODBC を使って MySQL データベースの情報にアクセスすることができます。Microsoft Word 内で、これはメール合成や、テーブルやデータがレポートに含まれるようにするのに、最も役立つ機能です。Microsoft Excel 内で、MySQL サーバ上でクエリを実行したり、データを Excel Worksheet に直接インポートして、データをrowやカラムの形式で表示することができます。

双方のアプリケーションで、データのアクセスとアプリケーションへのインポートには、ODBC ソースを介してクエリを実行することのできる Microsoft Query が使用されます。Microsoft Query を使用して、実行する SQL 文、テーブルの選択、フィールド、選択基準、ソートの順序を作成します。例えば、**Connector/ODBC の構成** にある DSN サンプルを使用して、World テスト データベースにあるテーブルの情報を Excel のスプレッドシートにインポートするには：

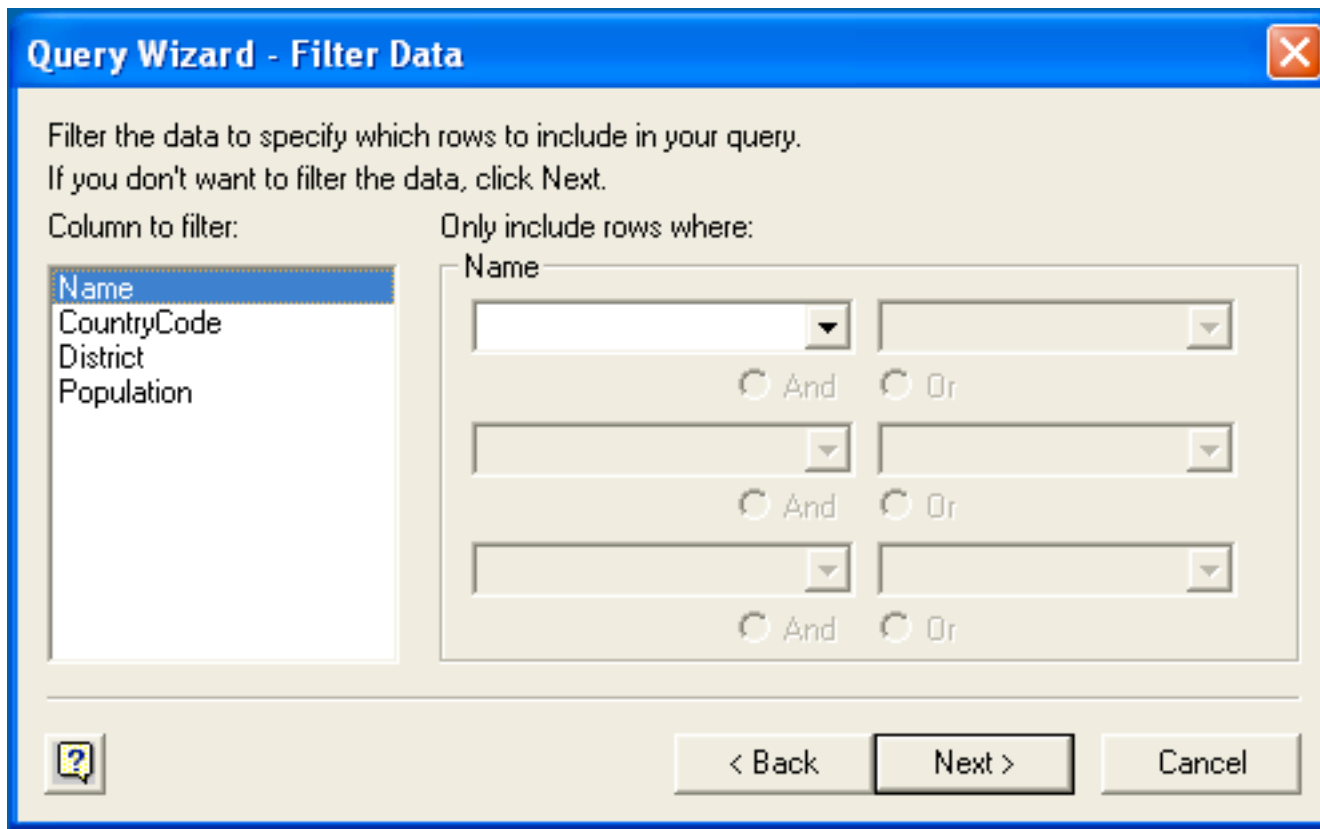
1. 新規の Worksheet を作成。
2. **Data** メニューから、**Import External Data** を選択し、さらに **New Database Query** を選択。
3. Microsoft Query が起動。まず、既存の Data Source Name を選択してデータソースを選ぶ。



4. **Query Wizard** 内で、インポートしたいカラムを選択する。DSN を介して構成され、ユーザが利用できるテーブルのリストが左側に、クエリに追加されるカラムが右側に表示される。選んだカラムは、**SELECT** クエリの最初のセクションにあるものに相当する。Next をクリックして続行。

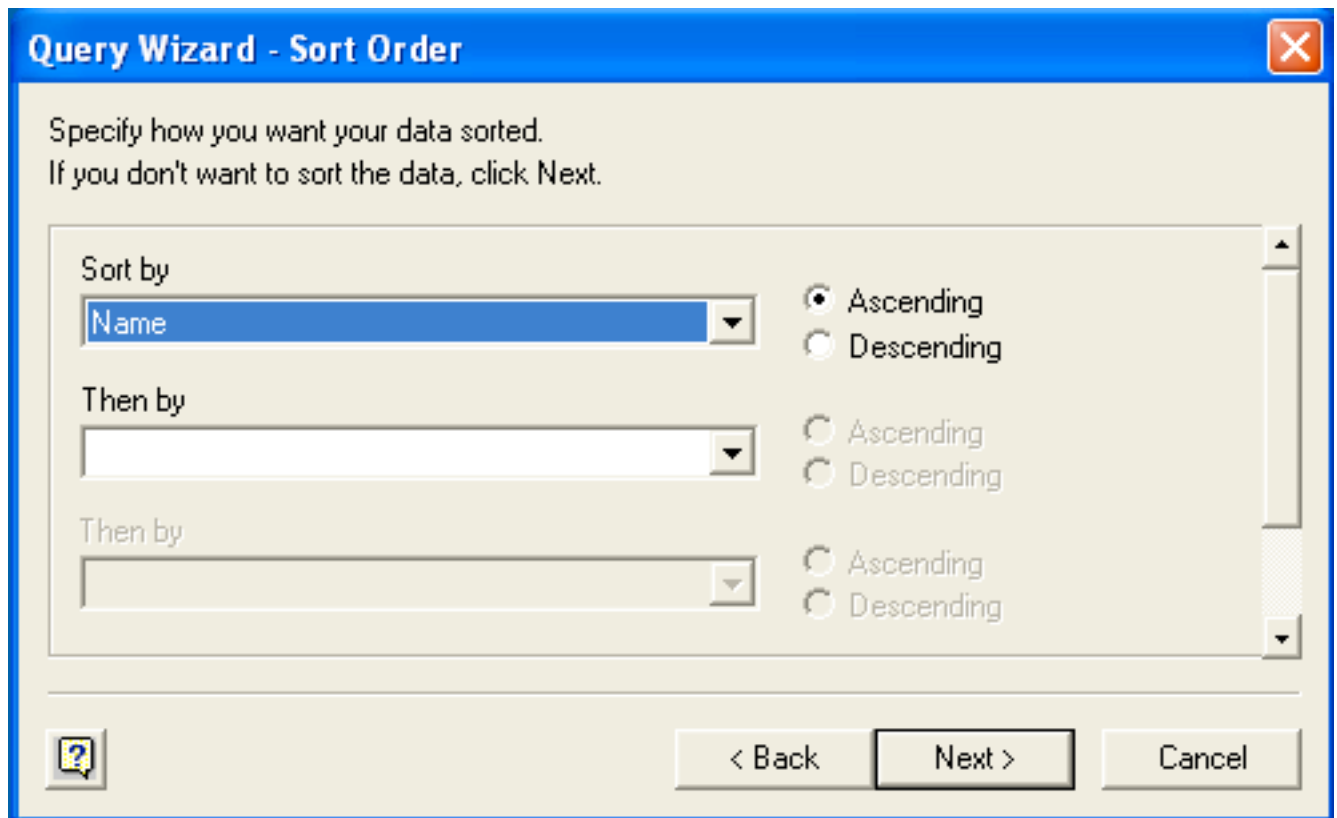


5. **Filter Data** ダイアログを使用して、クエリ (**WHERE** 条項に相当) から行をフィルターにかけることができる。Next をクリックして続行。



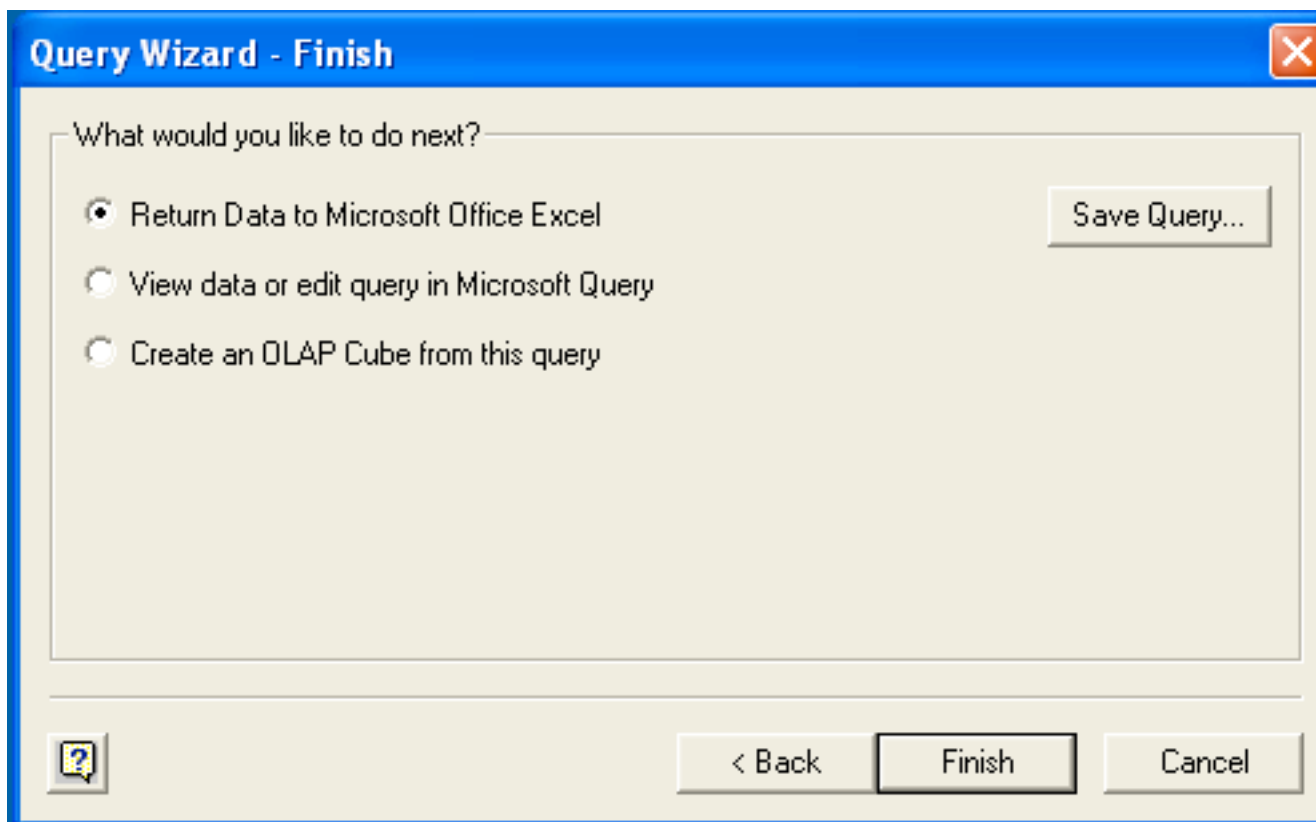
The image shows a Windows-style dialog box titled "Query Wizard - Filter Data". It has a blue title bar with a close button (X) in the top right corner. The main area is light beige. At the top, it says "Filter the data to specify which rows to include in your query. If you don't want to filter the data, click Next." Below this, there are two sections. On the left, under "Column to filter:", there is a list box containing "Name", "CountryCode", "District", and "Population". "Name" is selected and highlighted in blue. On the right, under "Only include rows where:", there is a section titled "Name" followed by three rows of filter criteria. Each row consists of two dropdown menus, a radio button labeled "And", and another radio button labeled "Or". The first row has empty dropdown menus. At the bottom left is a help icon (question mark in a square). At the bottom right are three buttons: "< Back", "Next >", and "Cancel".

6. データのソート順序を選択 (任意)。これは SQL クエリでの **ORDER BY** 条項の使用に相当する。最大 3 つのフィールドを選び、クエリから戻された情報をソートすることができる。Next をクリックして続行。



The image shows a Windows-style dialog box titled "Query Wizard - Sort Order". The title bar is blue with a close button (X) in the top right corner. The main area has a light beige background. At the top, there is instructional text: "Specify how you want your data sorted." followed by "If you don't want to sort the data, click Next." Below this, there are three sorting criteria sections. The first section is labeled "Sort by" and has a dropdown menu showing "Name". To its right are two radio buttons: "Ascending" (which is selected) and "Descending". The second section is labeled "Then by" and has an empty dropdown menu. To its right are two radio buttons: "Ascending" and "Descending", both of which are unselected. The third section is also labeled "Then by" and has an empty dropdown menu. To its right are two radio buttons: "Ascending" and "Descending", both of which are unselected. At the bottom of the dialog, there is a help icon (a question mark in a square) on the left, and three buttons: "< Back", "Next >", and "Cancel".

7. クエリの移動先を選択。データの返送先に Microsoft Excel を選び、データをインサートしたい部分のワークシートやセルを選択することができ、Microsoft Query 内でクエリと結果の閲覧を続け、SQL クエリとさらなるフィルタを編集、戻された情報をソートすることができる。または、クエリから OLAP Cube を作成し、それを Microsoft Excel 内で直接使用することも可能。Finish をクリックする。



同じプロセスで、Word ドキュメントにデータをインポートすることができ、その場合データはテーブル形式でインサートされます。これは、メール合成に（その場合は Word テーブルからフィールド データが読み取られる）に使用したり、もしくはレポートや他のドキュメントにデータおよび報告を含めたい場合に使用することができます。

Crystal Reports で Connector/ODBC を使用する

Crystal Reports は ODBC DSN を使って、レポート用にデータや情報が抽出できるデータベースに接続することができます。

注記

Crystal Reports の特定のバージョンでは、アプリケーションが開けない、ODBC コネクションを通してテーブルやフィールドをブラウズできない、という問題が報告されています。Crystal Reports で MySQL を使用する前に、不具合に対するサービス パックやホットフィックスを含む、最新バージョンへのアップデートをするようにしてください。この問題に関する情報は、[Business\) Objects Knowledgebase](#) でご覧ください。

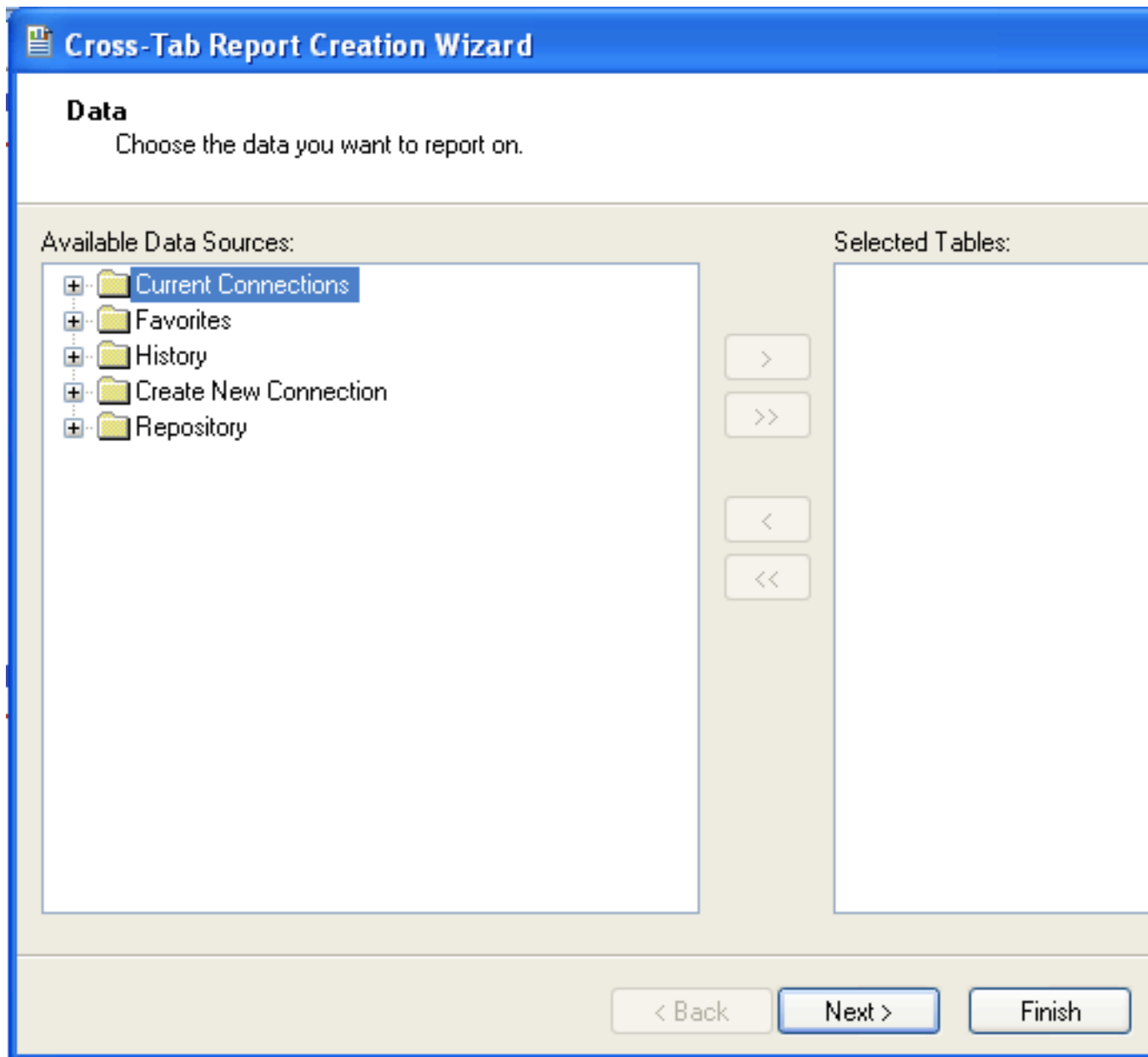
例として、Crystal Reports XI 内で簡単なクロス集計レポートを作成するには、次のステップに従ってください：

1. [Data Sources \(ODBC\)](#) ツールを使用して DSN を作成。ユーザ名とパスワードのある完全なデータベースを指定するか、基本的な DSN を作成し、Crystal Reports を使ってユーザ名とパスワードを設定する。

この例の目的のために、MySQL Sakila サンプル データベースの実例に接続を提供する DSN が作成されている。

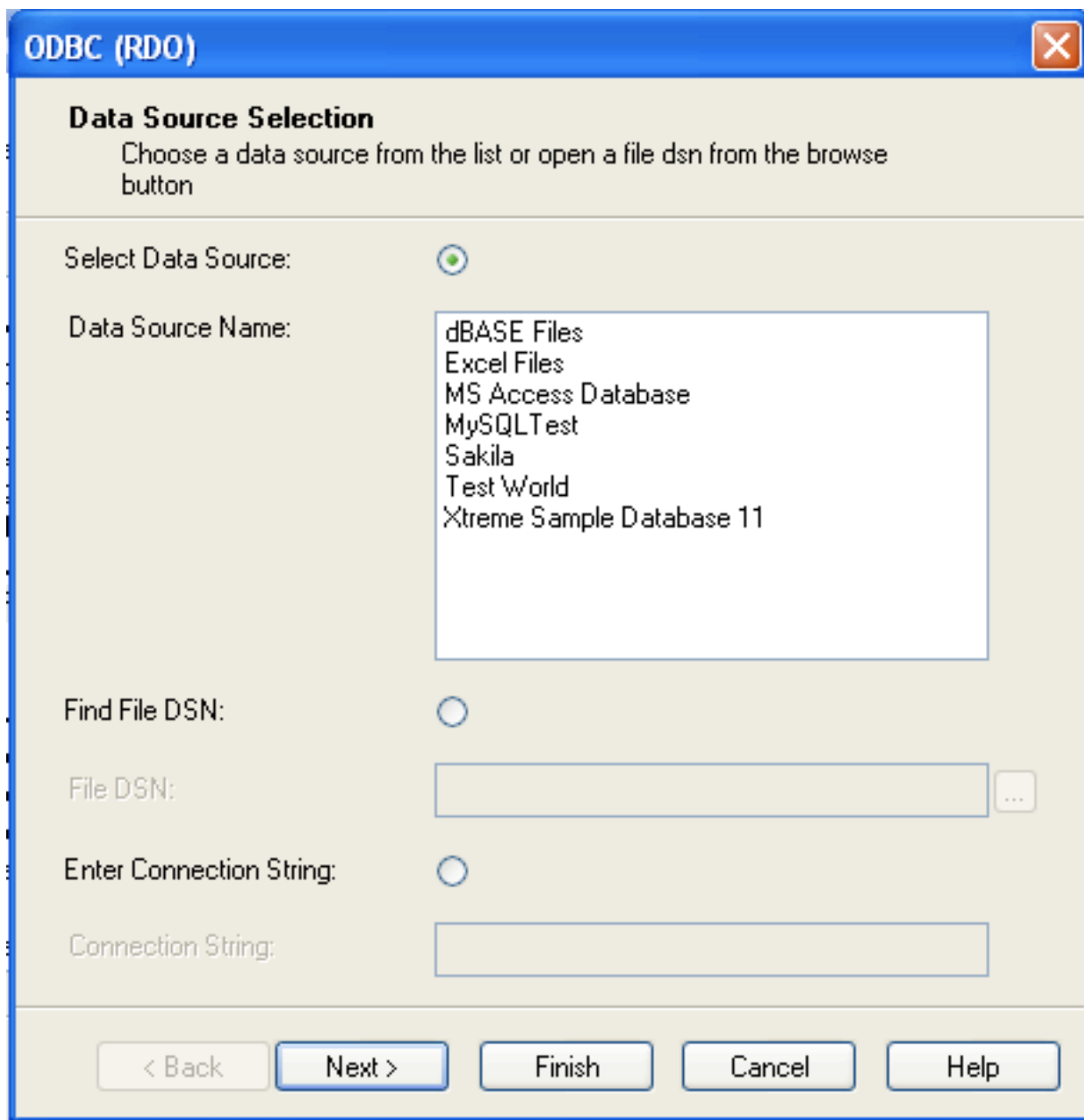
2. Crystal Reports を開き、新しいプロジェクトを作成するか、MySQL データソースからのデータをインサートしたい既存のレポート プロジェクトを開く。
3. Start Page のオプションをクリックするなどして、Cross-Tab Report Wizard を起動。Create New Connection フォルダを展開し、それから ODBC (RDO) フォルダを広げて ODBC データソースのリストを獲得する。

データソースの選択が要請される。



4. ODBC (RDO) フォルダを最初に展開する際、Data Source Selection スクリーンが表示される。ここからは、事前に構成された DSN を選択する、ファイルベース DSN を開く、もしくは手動接続ストリングを入力する、からどれかを選ぶことができる。この例では、Sakila DSN を使用。

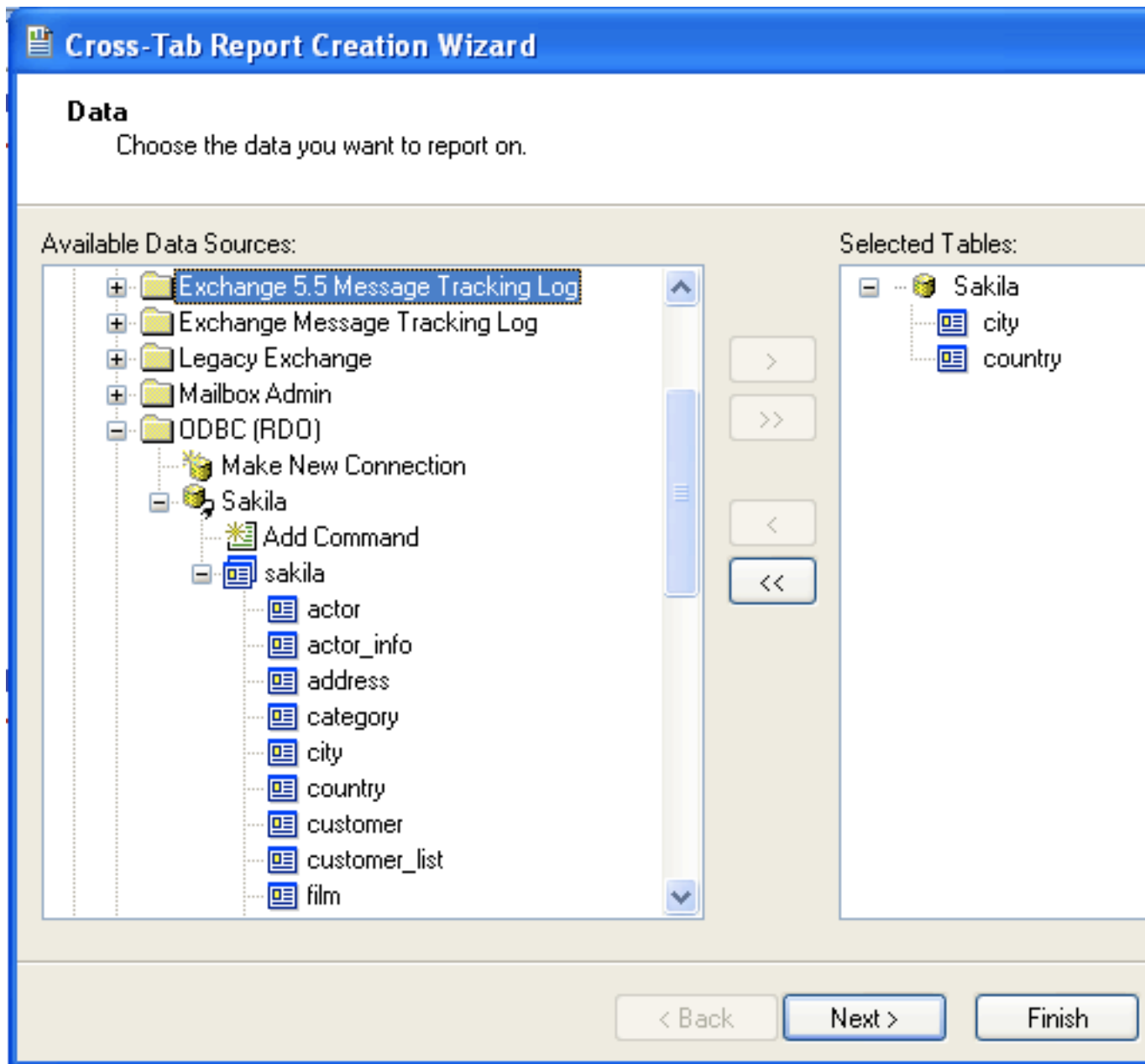
DSN がユーザ名とパスワードのコンビネーションを持つ場合、または別の承認信任状を使用する場合は、Next をクリックして、使用するユーザ名とパスワードを入力する。それ以外では、Finish をクリックして、データソース選択ウィザードを続ける。



5. Cross-Tab Report Creation Wizard に戻る。レポートに含めたいデータベースとテーブルを選択。ここでは例として、選ばれた Sakila データベースを展開する。city テーブルをクリックし、➤ ボタンでテーブルをレポートに加える。そして、country テーブルで同じ動作を繰り返す。あるいは、複数のテーブルを選択し、それらをレポートに加えることもできる。

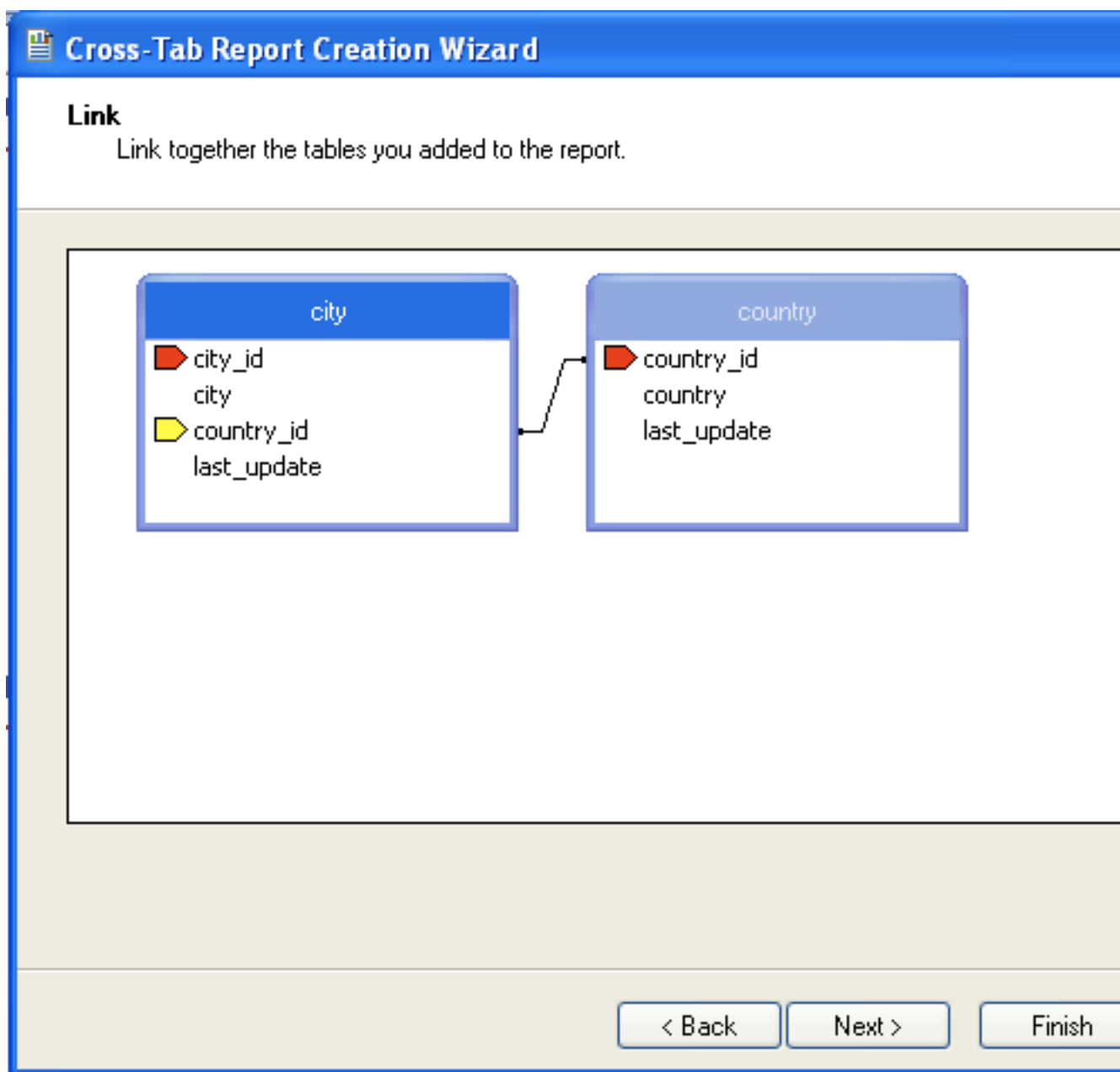
最後に、Sakila の親リソースを選択し、そのテーブルをレポートに加えることができる。

取り入れたいテーブルを選択したら、Next を押して次に進む。



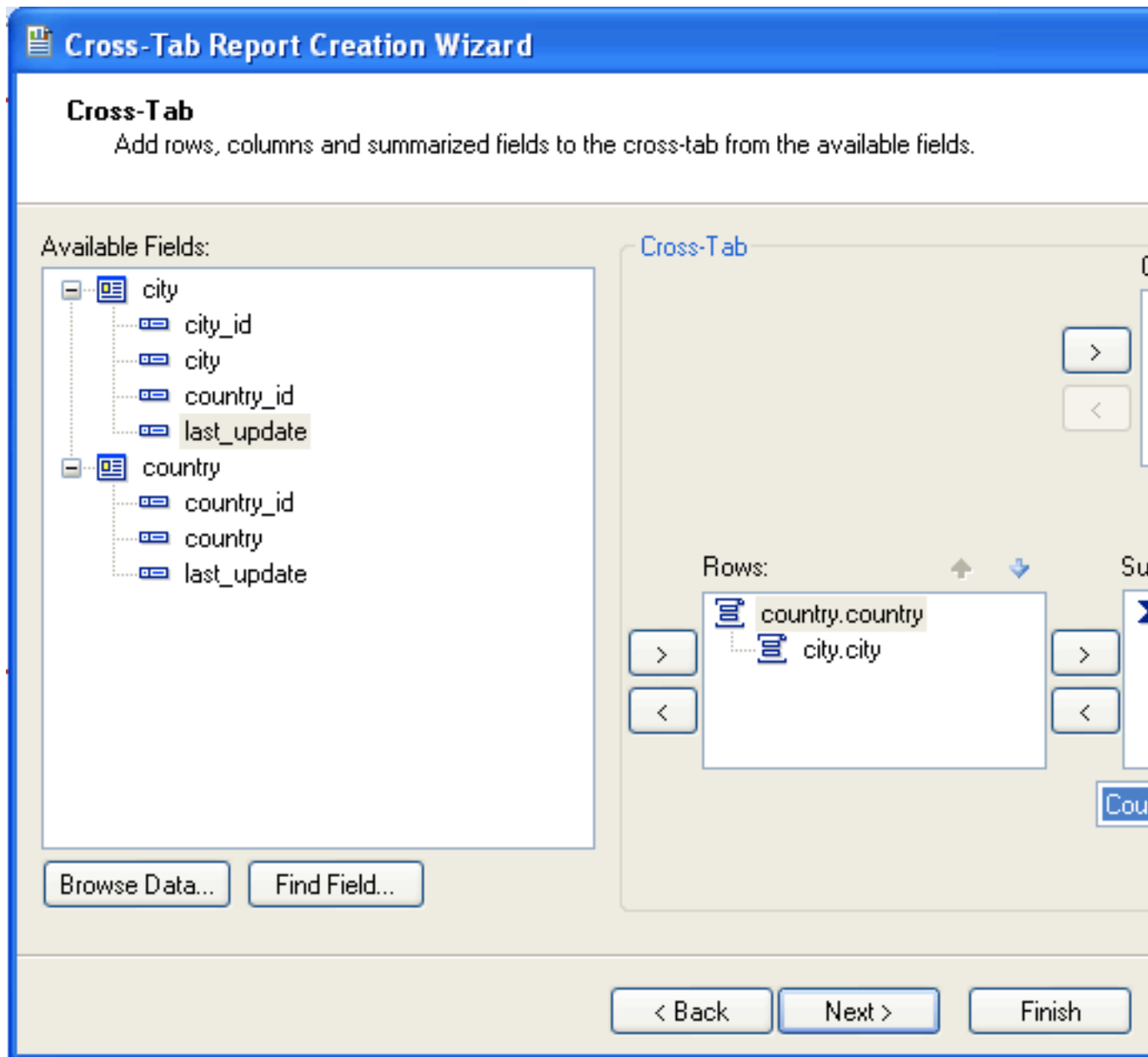
6. これで、Crystal Reports はテーブルの定義を読み取り、テーブル間のリンクを自動的に特定するようになる。テーブル間のリンクの特定により、Crystal Reports はクエリに従って、データベース内の全テーブルに基づく情報を参照そして要約することができる。Crystal Reports がリンクそのものを実行できない場合、選択したテーブルのフィールド間に手動でリンクを作成することができる。

Next をクリックしてプロセスを続行。



7. 次に、クロス集計レポートに取り込みたいカラムと行を選択。ドラッグ アンド ドロップか、> ボタンを使って、レポートの各部にフィールドを加える。この例の画面では、city をレポートし、country で整理、各 country の city の数を取り込むことにする。データをブラウスしたい場合は、フィールドを選択して **Browse Data...** ボタンをクリックする。

Next をクリックして結果のグラフを作成。このデータからはグラフを作成していないので、**Finish** をクリックしてレポートを発行する。



8. 完成したレポートが表示される。以下はSakila サンプル データベースからの出力サンプル。

		Total
Total		600
Afghanistan	Total	1
	Kabul	1
Algeria	Total	3
	Batna	1
	Bchar	1
	Skikda	1
American Samoa	Total	1
	Tafuna	1
Angola	Total	2
	Benguela	1
	Namibe	1
Anguilla	Total	1
	South Hill	1
Argentina	Total	13
	Almirante Brow	1

Clystal Reports 内で ODBC 接続が開いたら、使用可能なテーブルのどのフィールドでも、レポートに加えることができます。

Connector/ODBC プログラミング

適切な ODBC Manager と Connector/ODBC ドライバがインストールされていれば、ODBC に対応するすべてのプログラム言語または環境で、Connector/ODBC を介して MySQL データベースに接続することができます。

該当するプログラム言語や環境には、Microsoft 対応言語 (Visual Basic や C# 、そして ODBC.NET のようなインターフェイスも含む) 、Perl (DBI モジュール、DBD::ODBC ドライバ を仲介) をはじめ、様々な種類があります。

Visual Basic での、ADO、DAO および RDO を使った Connector/ODBC の使用

このセクションには、ADO、DAO および RDO を使った MySQL ODBC 3.51 ドライバの簡単な使用例が記載されています。

ADO :rs.addNew、rs.delete、および rs.update

次の ADO (ActiveX Data Objects) 例は、テーブル `my_ado` を作成し、`rs.addNew`、`rs.delete`、および `rs.update` の使用を実証します。

```
Private Sub myodbc_ado_Click()

Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fld As ADODB.Field
Dim sql As String

'connect to MySQL server using MySQL ODBC 3.51 Driver
Set conn = New ADODB.Connection
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};" _
& "SERVER=localhost;" _
& "DATABASE=test;" _
& "UID=venu;PWD=venu; OPTION=3"

conn.Open

'create table
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20)," _
& "txt text, dt date, tm time, ts timestamp)"

'direct insert
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"

Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer

'fetch the initial table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close

'rs insert
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close

'rs update
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close
```



```

'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close

'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
conn.Close
End Sub

```

DAO :rs.addNew、rs.update、およびスクローリング

次の DAO (Data Access Objects) 例では、テーブル `my_ado` を作成し、`rs.addNew`、`rs.update`、および結果セットのスクローリングの使用を実証します。

```

Private Sub myodbc_dao_Click()

Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String

'connect to MySQL using MySQL ODBC 3.51 Driver
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc:DRIVER={MySQL ODBC 3.51 Driver};_"
& "SERVER=localhost;_"
& "DATABASE=test;_"
& "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

'Create table my_dao
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute

Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMENT PRIMARY KEY, " _
& "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")
queryDef.Execute

'Insert new records using rs.addNew
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer

For i = 10 To 15
rs.AddNew
rs!Name = "insert record" & i
rs!Id2 = i
rs.Update
Next i
rs.Close

```

```

'rs update..
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close

'fetch the table back...
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend

'rs Scrolling
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

rs.MoveLast
str = " LAST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str

'free all resources
rs.Close
queryDef.Close
conn.Close
ws.Close

End Sub

```

RDO :rs.addNew と rs.delete

次の RDO (Remote Data Objects) 例は、テーブル `my_ado` を作成し、`rs.addNew` および `rs.update` の使用を実証します。

```

Dim rs As rdoResultset
Dim cn As New rdoConnection
Dim cl As rdoColumn
Dim SQL As String

'cn.Connect = "DSN=test;"
cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};" _
& "SERVER=localhost;" _
& " DATABASE=test;" _
& "UID=venu;PWD=venu; OPTION=3"

cn.CursorDriver = rdUseOdbc
cn.EstablishConnection rdDriverPrompt

'drop table my_rdo
SQL = "drop table if exists my_rdo"
cn.Execute SQL, rdExecDirect

'create table my_rdo
SQL = "create table my_rdo(id int, name varchar(20))"
cn.Execute SQL, rdExecDirect

'insert - direct
SQL = "insert into my_rdo values (100,'venu')"
cn.Execute SQL, rdExecDirect

SQL = "insert into my_rdo values (200,'MySQL')"
cn.Execute SQL, rdExecDirect

```

```

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close

'rs update
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close

'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
Do Until rs.EOF
For Each cl In rs.rdoColumns
Debug.Print cl.Value,
Next
rs.MoveNext
Debug.Print
Loop
Debug.Print "Row count="; rs.RowCount

'close
rs.Close
cn.Close

End Sub

```

.NET を使った Connector/ODBC の使用

このセクションでは、ODBC.NET を使った Connector/ODBC ドライバの簡単な使用実例が記載されています。

ODBC.NET と C# (C sharp) を使った Connector/ODBC の使用

次のサンプルは、テーブル `my_odbc_net` を作成し、C# でのその使用を実証します。

```

/**
 * @sample   : mycon.cs
 * @purpose  : Demo sample for ODBC.NET using Connector/ODBC
 * @author   : Venu, <myodbc@lists.mysql.com>
 *
 * (C) Copyright MySQL AB, 1995-2006
 */

/* build command
 *
 * csc /t:exe
 * /out:mycon.exe mycon.cs
 * /r:Microsoft.Data.Odbc.dll
 */

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3

```

```

{
class mycon
{
static void Main(string[] args)
{
try
{
//Connection string for MyODBC 2.50
/*string MyConString = "DRIVER={MySQL};" +
"SERVER=localhost;" +
"DATABASE=test;" +
"UID=venu;" +
"PASSWORD=venu;" +
"OPTION=3";
*/
//Connection string for Connector/ODBC 3.51
string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
"SERVER=localhost;" +
"DATABASE=test;" +
"UID=venu;" +
"PASSWORD=venu;" +
"OPTION=3";

//Connect to MySQL using Connector/ODBC
OdbcConnection MyConnection = new OdbcConnection(MyConString);
MyConnection.Open();

Console.WriteLine("\n !!! success, connected successfully !!!\n");

//Display connection information
Console.WriteLine("Connection Information:");
Console.WriteLine("\tConnection String:" +
MyConnection.ConnectionString);
Console.WriteLine("\tConnection Timeout:" +
MyConnection.ConnectionTimeout);
Console.WriteLine("\tDatabase:" +
MyConnection.Database);
Console.WriteLine("\tDataSource:" +
MyConnection.DataSource);
Console.WriteLine("\tDriver:" +
MyConnection.Driver);
Console.WriteLine("\tServerVersion:" +
MyConnection.ServerVersion);

//Create a sample table
OdbcCommand MyCommand =
new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",
MyConnection);
MyCommand.ExecuteNonQuery();
MyCommand.CommandText =
"CREATE TABLE my_odbc_net(id int, name varchar(20), idb bigint)";
MyCommand.ExecuteNonQuery();

//Insert
MyCommand.CommandText =
"INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
Console.WriteLine("INSERT, Total rows affected:" +
MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
"INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
Console.WriteLine("INSERT, Total rows affected:" +
MyCommand.ExecuteNonQuery());

//Insert
MyCommand.CommandText =
"INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
Console.WriteLine("INSERT, Total rows affected:" +
MyCommand.ExecuteNonQuery());

//Update
MyCommand.CommandText =
"UPDATE my_odbc_net SET id=999 WHERE id=20";
Console.WriteLine("Update, Total rows affected:" +

```

```

        MyCommand.ExecuteNonQuery();

//COUNT(*)
MyCommand.CommandText =
    "SELECT COUNT(*) as TRows FROM my_odbc_net";
Console.WriteLine("Total Rows:" +
    MyCommand.ExecuteScalar());

//Fetch
MyCommand.CommandText = "SELECT * FROM my_odbc_net";
OdbcDataReader MyDataReader;
MyDataReader = MyCommand.ExecuteReader();
while (MyDataReader.Read())
{
    if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
        //Supported only by Connector/ODBC 3.51
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt64(2));
    }
    else {
        //BIGINTs not supported by Connector/ODBC
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt32(2));
    }
}

//Close all resources
MyDataReader.Close();
MyConnection.Close();
}
catch (OdbcException MyOdbcException) //Catch any ODBC exception ..
{
    for (int i=0; i < MyOdbcException.Errors.Count; i++)
    {
        Console.WriteLine("ERROR #" + i + "\n" +
            "Message: " +
            MyOdbcException.Errors[i].Message + "\n" +
            "Native: " +
            MyOdbcException.Errors[i].NativeError.ToString() + "\n" +
            "Source: " +
            MyOdbcException.Errors[i].Source + "\n" +
            "SQL: " +
            MyOdbcException.Errors[i].SQLState + "\n");
    }
}
}
}
}
}

```

ODBC.NET と Visual Basic を使った Connector/ODBC の使用

次のサンプルは、テーブル `my_vb_net` を作成し、VB でのその使用を実証します。

```

' @sample : myvb.vb
' @purpose : Demo sample for ODBC.NET using Connector/ODBC
' @author : Venu, <myodbc@lists.mysql.com>
'
' (C) Copyright MySQL AB, 1995-2006
'
'
'
' build command
'
' vbc /target:exe
' /out:myvb.exe
' /r:Microsoft.Data.Odbc.dll
' /r:System.dll
' /r:System.Data.dll
'
Imports Microsoft.Data.Odbc

```

Imports System

Module myvb

Sub Main()

Try

'Connector/ODBC 3.51 connection string

Dim MyConString As String = "DRIVER={MySQL ODBC 3.51 Driver};" & _

"SERVER=localhost;" & _

"DATABASE=test;" & _

"UID=venu;" & _

"PASSWORD=venu;" & _

"OPTION=3;"

'Connection

Dim MyConnection As New OdbcConnection(MyConString)

MyConnection.Open()

Console.WriteLine("Connection State::" & MyConnection.State.ToString)

'Drop

Console.WriteLine("Dropping table")

Dim MyCommand As New OdbcCommand()

MyCommand.Connection = MyConnection

MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"

MyCommand.ExecuteNonQuery()

'Create

Console.WriteLine("Creating....")

MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name varchar(30))"

MyCommand.ExecuteNonQuery()

'Insert

MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"

Console.WriteLine("INSERT, Total rows affected:" & _

MyCommand.ExecuteNonQuery())

'Insert

MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"

Console.WriteLine("INSERT, Total rows affected:" & _

MyCommand.ExecuteNonQuery())

'Insert

MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"

Console.WriteLine("INSERT, Total rows affected:" & _

MyCommand.ExecuteNonQuery())

'Insert

MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"

Console.WriteLine("INSERT, Total rows affected:" & _

MyCommand.ExecuteNonQuery())

'Update

MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"

Console.WriteLine("Update, Total rows affected:" & _

MyCommand.ExecuteNonQuery())

'COUNT(*)

MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"

Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

'Select

Console.WriteLine("Select * FROM my_vb_net")

MyCommand.CommandText = "SELECT * FROM my_vb_net"

Dim MyDataReader As OdbcDataReader

MyDataReader = MyCommand.ExecuteReader

While MyDataReader.Read

If MyDataReader("name") Is DBNull.Value Then

Console.WriteLine("id = " & _

CStr(MyDataReader("id")) & " name = " & _

"NULL")

Else

Console.WriteLine("id = " & _

CStr(MyDataReader("id")) & " name = " & _

CStr(MyDataReader("name")))


```

End If
End While

'Catch ODBC Exception
Catch MyOdbcException As OdbcException
Dim i As Integer
Console.WriteLine(MyOdbcException.ToString)

'Catch program exception
Catch MyException As Exception
Console.WriteLine(MyException.ToString)
End Try
End Sub

```

Connector/ODBC 参考資料

このセクションでは、Connector/ODBC API の対応関数、方法、Connector/ODBC でサポートされている MySQL カラム タイプと適合ネイティブ タイプ、失敗があった時に Connector/ODBC が報告するエラーコード等を記載した参考資料を提供します。

Connector/ODBC API 参考資料

このセクションでは、関数によって分類された ODBC ルーチンの要約を記載します。

完全な ODBC API 参考資料は、http://msdn.microsoft.com/library/en-us/odbc/hlm/odbcabout_this_manual.asp の ODBC Programmer's Reference をご覧ください。

アプリケーションは [SQLGetInfo](#) 関数を呼び出し、Connector/ODBC の適合性情報を得ることができます。ドライバの特定の機能へのサポートに関する情報を得るには、アプリケーションで [SQLGetFunctions](#) を呼び出すことができます。

注記

後方互換性に関しては、Connector/ODBC 3.51 はすべての推奨されない関数をサポートしています。

以下のテーブルは、作業別に分類された Connector/ODBC API 呼び出しをリストアップしています。

データソースへの接続：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLAllocHandle	No	Yes	ISO 92	環境、接続、ステートメント、または記述子ハンドルの獲得
SQLConnect	Yes	Yes	ISO 92	データソース名、ユーザ ID とパスワードで特定のドライバに接続
SQLDriverConnect	Yes	Yes	ODBC	接続ストリングもしくは、Driver Manager とドライバがユーザのために接続ダイアログ ボックスを表示する要求によって特定のドライバに接続
SQLAllocEnv	Yes	Yes	Deprecated	ドライバから割り振られた環境ハンドルの獲得
SQLAllocConnect	Yes	Yes	Deprecated	接続ハンドルの獲得

ドライバとデータソースの情報獲得：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLDataSources	No	No	ISO 92	利用可能なデータソースのリストを戻す、Driver Manager で扱う

SQLDrivers	No	No	ODBC	インストールされたドライバとその属性情報のリストを 戻す、Driver Manager で扱う
SQLGetInfo	Yes	Yes	ISO 92	特定のドライバとデータソースの情報を戻す
SQLGetFunctions	Yes	Yes	ISO 92	サポートされているドライバ関数を戻す
SQLGetTypeInfo	Yes	Yes	ISO 92	サポートされているドライバ関数を戻す

ドライバ属性の設定と検索

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLSetConnectAttr	No	Yes	ISO 92	接続属性の設定
SQLGetConnectAttr	No	Yes	ISO 92	接続属性の値を戻す
SQLSetConnectOption	Yes	Yes	Deprecated	接続オプションの設定
SQLGetConnectOption	Yes	Yes	Deprecated	接続オプションの値を戻す
SQLSetEnvAttr	No	Yes	ISO 92	環境属性を設定
SQLGetEnvAttr	No	Yes	ISO 92	環境属性の値を戻す
SQLSetStmtAttr	No	Yes	ISO 92	ステートメント属性を設定
SQLGetStmtAttr	No	Yes	ISO 92	ステートメント属性の値を戻す
SQLSetStmtOption	Yes	Yes	Deprecated	ステートメント オプションの設定
SQLGetStmtOption	Yes	Yes	Deprecated	ステートメント オプションの値を戻す

SQL リクエストの準備 :

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLAllocStmt	Yes	Yes	Deprecated	ステートメント ハンドルの割り振り
SQLPrepare	Yes	Yes	ISO 92	後の実行のために SQL 文を準備
SQLBindParameter	Yes	Yes	ODBC	SQL 文中のパラメータのストレージを割り当て
SQLGetCursorName	Yes	Yes	ISO 92	ステートメント ハンドルに関連するカーソル名を戻す
SQLSetCursorName	Yes	Yes	ISO 92	カーソル名の指定
SQLSetScrollOptions	Yes	Yes	ODBC	カーソルの挙動をコントロールするオプションの設定

リクエストのサブミット :

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLExecute	Yes	Yes	ISO 92	準備されたステートメントの実行
SQLExecDirect	Yes	Yes	ISO 92	ステートメントの実行
SQLNativeSql	Yes	Yes	ODBC	ドライバに変換された SQL 文のテキストを戻す
SQLDescribeParam	Yes	Yes	ODBC	ステートメント中の特定のパラメータの概要を戻す
SQLNumParams	Yes	Yes	ISO 92	ステートメント中のパラメータの数を戻す
SQLParamData	Yes	Yes	ISO 92	実行時にパラメータ データを供給するために SQLPutData と併せて使用 (長いデータ値に最適)
SQLPutData	Yes	Yes	ISO 92	パラメータのデータ値の一部または全てを送信 (長い データ値に最適)

結果と結果の情報の検索：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLRowCount	Yes	Yes	ISO 92	INSERTに影響を受ける行の数、アップデート、または削除要求を戻す
SQLNumResultCols	Yes	Yes	ISO 92	結果セット内のカラムの数を戻す
SQLDescribeCol	Yes	Yes	ISO 92	結果セット内のカラムを説明
SQLColAttribute	No	Yes	ISO 92	結果セット内のカラムの属性を説明
SQLColAttributes	Yes	Yes	Deprecated	結果セット内のカラムの属性を説明
SQLFetch	Yes	Yes	ISO 92	複数の結果行を戻す
SQLFetchScroll	No	Yes	ISO 92	スクロール可能な結果行を戻す
SQLExtendedFetch	Yes	Yes	Deprecated	スクロール可能な結果行を戻す
SQLSetPos	Yes	Yes	ODBC	カーソルをデータの抽出されたブロックに配置し、行セットでアプリケーションがデータをリフレッシュできるようにする、または結果セットのデータをアップデートもしくは削除する
SQLBulkOperations	No	Yes	ODBC	アップデート、削除、ブックマークによる抽出を含むバルク INSERTおよびバルク ブックマーク操作を実施

エラーまたは診断情報の検索：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLError	Yes	Yes	Deprecated	追加のエラーまたはステータス情報を戻す
SQLGetDiagField	Yes	Yes	ISO 92	追加の診断情報 (診断データ構造のひとつのフィールド) を戻す
SQLGetDiagRec	Yes	Yes	ISO 92	追加の診断情報 (診断データ構造の複数の) を戻す

データソースのシステム テーブル (カタログ機能) アイテム情報の獲得：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLColumnPrivileges	Yes	Yes	ODBC	ひとつまたは複数のテーブルにカラムと関連権限のリストを戻す
SQLColumns	Yes	Yes	X/Open	特定のテーブルにカラム名のリストを戻す
SQLForeignKeys	Yes	Yes	ODBC	外部キーになるカラム名が特定のテーブルに既存していれば、リストを戻す
SQLPrimaryKeys	Yes	Yes	ODBC	テーブルの基本キーになるカラム名のリストを戻す
SQLSpecialColumns	Yes	Yes	X/Open	指定のテーブルで行をユニークに特定するカラムの最適なセット、またはトランザクションで行の値のどれかがアップデートされた際、自動的にアップデートされるカラムを戻す
SQLStatistics	Yes	Yes	ISO 92	単一のテーブルのスタティック、およびテーブルに関連するインデックスのリストを戻す
SQLTablePrivileges	Yes	Yes	ODBC	テーブルのリストと、各テーブルに関連する権限を戻す
SQLTables	Yes	Yes	X/Open	特定のデータソースに格納されたテーブル名のリストを戻す

トランザクションの実行：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLTransact	Yes	Yes	Deprecated	トランザクションのコミットまたはロールバック
SQLEndTran	No	Yes	ISO 92	トランザクションのコミットまたはロールバック

ステートメントの終了：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLFreeStmt	Yes	Yes	ISO 92	ステートメントの処理の終了、保留中の結果の破棄、またオプションとしてステートメント ハンドルに関連するすべてのソースの解放
SQLCloseCursor	Yes	Yes	ISO 92	ステートメント ハンドルで開かれていたカーソルを閉じる
SQLCancel	Yes	Yes	ISO 92	SQL 文のキャンセル

接続の終了：

関数名	Connector/ ODBC		標準	目的
	2.50	3.51		
SQLDisconnect	Yes	Yes	ISO 92	接続の終了
SQLFreeHandle	No	Yes	ISO 92	環境、接続、ステートメント、または記述子ハンドルの解放
SQLFreeConnect	Yes	Yes	Deprecated	接続ハンドルの解放
SQLFreeEnv	Yes	Yes	Deprecated	環境ハンドルの解放

Connector/ODBC データタイプ

次のテーブルは、ドライバがサーバのデータタイプをデフォルト SQL と C データタイプに位置づける方法を図式化したものです。

ネイティブ値	SQL タイプ	C タイプ
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR

mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT
double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
text	SQL_VARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

Connector/ODBC エラーコード：

次のテーブルは、サーバエラーとは別にドライバから戻されるエラーコードのリストです。

ネイティブ コード	SQLSTATE 2	SQLSTATE 3	エラーメッセージ
500	01000	01000	一般警告
501	01004	01004	ストリング データ、右打ち切り
502	01S02	01S02	オプション値が変更
503	01S03	01S03	行のアップデート / 削除なし
504	01S04	01S04	ひとつ以上の行がアップデート / 削除
505	01S06	01S06	結果セットが最初の行セットを戻す前にフェッチを試行
506	07001	07002	SQLBindParameter がすべてのパラメータでは使用されていない
507	07005	07005	カーソル規定ではないプリペアド ステートメント
508	07009	07009	無効な記述子インデックス
509	08002	08002	接続名が使用中

510	08003	08003	接続が存在していない
511	24000	24000	無効なカーソル状態
512	25000	25000	無効なトランザクション状態
513	25S01	25S01	トランザクション状態が不明
514	34000	34000	無効なカーソル名
515	S1000	HY000	一般ドライバ定義エラー
516	S1001	HY001	メモリ割り振りエラー
517	S1002	HY002	無効なカラム番号
518	S1003	HY003	無効なアプリケーション バッファ タイプ
519	S1004	HY004	無効な SQL データタイプ
520	S1009	HY009	ヌル ポインタの無効な使用
521	S1010	HY010	関数シーケンス エラー
522	S1011	HY011	現在は属性を設定できない
523	S1012	HY012	無効なトランザクション操作コード
524	S1013	HY013	メモリ管理エラー
525	S1015	HY015	カーソル名の利用不可
526	S1024	HY024	無効な属性値
527	S1090	HY090	無効なストリングまたはバッファ長
528	S1091	HY091	無効な記述子フィールド識別子
529	S1092	HY092	無効な属性 / オプション識別子
530	S1093	HY093	無効なパラメータ番号
531	S1095	HY095	範囲外の関数タイプ
532	S1106	HY106	範囲外のフェッチ タイプ
533	S1117	HY117	範囲外の行値
534	S1109	HY109	無効なカーソル位置
535	S1C00	HYC00	オプション機能が実装されていない
0	21S01	21S01	カラム カウントと値カウントの不一致
0	23000	23000	整合性制約違反
0	42000	42000	シンタックス エラーまたはアクセス違反
0	42S02	42S02	基本テーブルまたはビューの不検出
0	42S12	42S12	インデックスの不検出
0	42S21	42S21	カラムが既存
0	42S22	42S22	カラムの不検出
0	08S01	08S01	通信リンク故障

Connector/ODBC に関する注釈とヒント

ここでは、Connector/ODBC を異なる環境、アプリケーション、ツールで使用する際の一般的な注釈とアドバイスを記載しています。これらの注釈は、Connector/ODBC 開発者とユーザの経験に基づいたものです。

Connector/ODBC 一般機能

このセクションでは、MySQL の一般的なクエリや機能の範囲、そして Connector/ODBC とのそれらの使い方について記載します。

オートインクリメント値の獲得

INSERT 文の後で、**AUTO_INCREMENT** を使うカラムの値を獲得する方法は様々あります。INSERT の直後に獲得するには、**SELECT** クエリを **LAST_INSERT_ID()** 関数と使用します。

例えば、Connector/ODBC の使用中に、ふたつの別々のステートメント、**INSERT** 文と **SELECT** クエリを実行してオートインクリメント値を獲得します。

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

アプリケーション内では値は必要でないものの、他の **INSERT** の一部として値が必要である場合は、次のステートメントを実行するとすべてのプロセスが処理されます：

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

特定の ODBC アプリケーション (Delphi および Access を含む) では、上の例でオートインクリメント値を獲得するのは困難な場合があります。その場合は、代わりに次のステートメントで試みてください：

```
SELECT * FROM tbl WHERE auto IS NULL;
```

最後に挿入された列に対するユニークIDを取得する方法 参照。

動的カーソルのサポート

dynamic cursor のサポートは Connector/ODBC 3.51 で提供されていますが、デフォルト設定では動的カーソルが有効になっていません。Windows では、ODBC Data Source Administrator で **Enable Dynamic Cursor** チェックボックスを選択して、この機能を有効にしてください。

他のプラットフォームでは、DSN を作成する際、**OPTION** 値に **32** を加えることで動的カーソルを有効にすることができます。

Connector/ODBC の性能

Connector/ODBC ドライバは高いパフォーマンス性を想定しています。Connector/ODBC の性能に問題を感じた場合、または簡単な要求に多量のディスク アクティビティが費やされる場合は、いくつかのことを確認してください：

- **ODBC Tracing** を無効にしてください。トレースが有効になっていると、ODBC Manager が多くの情報をトレース ファイルに記録します。Windows では、ODBC Data Source Administrator の Tracing パネルでトレースの状況を確認し、無効にすることができます。Mac OS X では、ODBC Administrator の Tracing パネルで確認してください。[ODBC トレース ファイルの獲得](#) 参照。
- ドライバはデバッグ バージョンでなく、標準バージョンを使用していることを確認してください。デバッグ バージョンには、追加の査閲や報告指標が含まれています。
- Connector/ODBC ドライバのトレースとクエリ ログを無効にしてください。これらのオプションは各 DSN に対して有効になっていますので、アプリケーションに使用している DSN だけを検査するようにしてください。Windows では、DSN 構成を改修することで Connector/ODBC とクエリ ログを無効にすることができます。Mac OS X と Unix では、ドライバトレース (オプション値 4) とクエリ ログ (オプション値 524288) が有効になっていないか確かめてください。

Windows で ODBC クエリ タイムアウトを設定

Microsoft Windows で、ODBC 接続を通してクエリを実行する際にクエリ タイムアウトを設定する方法に関する情報は、<http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756> にある Microsoft ナレッジベース資料をお読みください。

Connector/ODBC アプリケーション別情報

ほとんどのプログラムは Connector/ODBC に対応しますが、以下にリストされているものにはそれぞれに、Connector/ODBC とそのアプリケーションとの作業を向上拡大するための注釈とアドバイスがあります。

すべてのアプリケーションで、Connector/ODBC ドライバ、ODBC Manager、およびアプリケーションで使われているライブラリとインターフェイス全部が最新バージョンになっているか確認してください。例えば Windows では、最新バージョンの Microsoft Data Access Components (MDAC) を使うことによって、ODBC との全体的な互換性、そして Connector/ODBC ドライバとの互換性が向上します。

Connector/ODBC の Microsoft アプリケーションとの使用

Microsoft Office、Microsoft Access、そして ASP と Microsoft Visual Studio でサポートされている様々なプログラム言語を含む Microsoft アプリケーションの大多数は、Connector/ODBC との使用がテストで確認済みです。

Connector/ODBC に不具合が発生し、またそのプログラムが OLEDB にも対応する場合は、OLEDB ドライバの使用をお勧めします。

Microsoft Access

Connector/ODBC を通して Microsoft Access と MySQL の融合性を高めたい場合：

- すべてのバージョンの Access で、Connector/ODBC [Return matching rows](#) オプションを有効にしてください。Access 2.0 では、[Simulate ODBC 1.0](#) オプションも加えて有効にしてください。
- アップデートしたいすべてのテーブルに、[TIMESTAMP](#) カラムを設けてください。ポータビリティを最大にするには、カラム宣言 (MySQL 4.1 より前のバージョンではサポートされていません) で長さ指定を使わないでください。
- Access で使いたい MySQL テーブル個々に基本キーを設置してください。それを行わない場合、新規またはアップデートされた行が [#DELETED#](#) と表示される場合があります。
- [DOUBLE](#) 浮動小数点フィールドのみを使用。単精度浮動小数点値と比較する際、Access に不具合が起きるためです。この徴候では通常、新規またはアップデートされた行が [#DELETED#](#) と表示される、または行の検出もしくはアップデートができなくなるということがあります。
- Connector/ODBC を使って [BIGINT](#) カラムのあるテーブルへリンクしている場合、結果が [#DELETED#](#) と表示されます。この問題を避けるには：
 - データタイプとして、[TIMESTAMP](#) にもうひとつダミー カラムを設ける。
 - ODBC DSN Administrator の接続ダイアログで、[Change BIGINT columns to INT](#) オプションを選択。
 - Access からテーブル リンクを削除し、また再作成する。

古い記録がまだ [#DELETED#](#) と表示されるが、新しく加えられた、またはアップデートされた記録は正常に表示される。

- [TIMESTAMP](#) カラムを設けた後でも、エラー [Another user has changed your data](#) が発生する場合は、次の手段で解決することもあります：

[table](#) データシート ビューを使用しない。代わりに、希望のフィールドでフォームを作成し、その [form](#) データシート ビューを使用する。[TIMESTAMP](#) カラムの [DefaultValue](#) プロパティを、[NOW\(\)](#) に設定。後のユーザの混乱を避けるため、[TIMESTAMP](#) カラムをビューから隠すとよい。

- あるケースでは、Access が MySQL には理解不能な SQL 文を発行することがあります。Access のメニューから ["Query|SQLSpecific|Pass-Through"](#) を選択すると、これを解決できます。
- Windows NT では、Access は [BLOB](#) カラムを [OLE OBJECTS](#) として報告します。代わりに [MEMO](#) カラムを希望する場合は、[ALTER TABLE](#) で [BLOB](#) カラムを [TEXT](#) に変更してください。
- Access は MySQL [DATE](#) カラムを、常に正常に処理できるというわけではありません。もし問題がある場合は、カラムを [DATETIME](#) に変更してください。
- Access に [BYTE](#) と定義されたカラムがある場合、Access はこれを [TINYINT UNSIGNED](#) ではなく [TINYINT](#) としてエクスポートしようとします。そのカラムに 127 より大きな値がある場合、これが問題となります。
- Access に極めて大規模な (長い) テーブルがある場合、それらを開くのに長い時間がかかる場合があります。または、仮想メモリが不足していると、最終的には [ODBC Query Failed](#) エラーが発生し、テーブルを開くことができません。この問題に対処するには、次のオプションを選んでください：
 - Return Matching Rows (2)
 - Allow BIG Results (8).

これらを足した値は 10 ([OPTION=10](#)) になります。

Access、ODBC および Connector/ODBC の使用に役立つ外部からの記事や情報：

- [How to Trap ODBC Login Error Messages in Access](#) 参照。
- Access ODBC アプリケーションの最適化
 - [Optimizing for Client/Server Performance](#)
 - [Tips for Converting Applications to Using ODBCDirect](#)
 - [Tips for Optimizing Queries on Attached SQL Tables](#)
- Access や ODBC データソースで利用できるツールのリストに関しては、<http://www.mysql.com/portal/software/convertors/> セクションを参照してください。

Microsoft Excel と カラム タイプ

Microsoft Excel へのデータのインポート、特に数値やデータ、タイム値のインポートで問題がある場合は、ワークシート内のセルにデータがインサートされる時、ソース データのカラム タイプでデータ タイプが定義される Excel のバグが原因だと思われます。その結果、Excel がコンテンツを誤って識別し、それが計算に使用される時、表示フォーマットとそのデータの双方に影響を及ぼします。

この問題を解消するには、[CONCAT\(\)](#) 関数をクエリに使用してください。[CONCAT\(\)](#) を使うことで、Excel が値をストリングとして扱うよう仕向け、それによって Excel が埋め込まれた情報を解析し、ほぼ正確に識別できるようになります。

しかし、このオプションをもってしてもデータによっては、ソースデータに変化はないにもかかわらず、不正確にフォーマットされることがあります。Excel の [Format Cells](#) オプションを使用して、表示された情報のフォーマットを変えてください。

Microsoft Visual Basic

テーブルをアップデートするには、テーブルに基本キーを定義する必要があります。

ADO を使用した Visual Basic は、大きな整数を扱うことができません。つまり、[SHOW PROCESSLIST](#) のようなクエリは正常に作動しません。これを解決するには、ODBC 接続ストリングで [OPTION=16384](#) を使用するか、Connector/ODBC 接続スクリーンで [Change BIGINT columns to INT](#) オプションを選択します。また、[Return matching rows](#) オプションを選択するのもよいでしょう。

Microsoft Visual InterDev

結果に [BIGINT](#) がある場合、[\[Microsoft\]\[ODBC Driver Manager\] Driver does not support this parameter](#) エラーが発生する恐れがあります。Connector/ODBC 接続スクリーンで、[Change BIGINT columns to INT](#) オプションを選択してみましょう。

Visual Objects

[Don't optimize column widths](#) オプションを選択してください。

Microsoft ADO

ADO API と Connector/ODBC でコード作成をしている場合、MySQL サーバでサポートされていないデフォルトのプロパティに注意してください。例えば、[CursorLocation Property](#) を [adUseServer](#) として使用すると、-1 という結果が [RecordCount Property](#) に戻されます。正しい値を得るには、このプロパティを下記の VB コードのように、[adUseClient](#) に設定する必要があります。

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long

myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
```

```
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close
```

もうひとつの回避法としては、類似するクエリに `SELECT COUNT(*)` 文を使用し、正しい行カウントを獲得します。

ADO の特定の SQL 文に影響を受けている行の数を検索するには、ADO execute メソッドに `RecordsAffected` プロパティを使用します。execute メソッドの使用に関する詳しい情報は、<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/html/mdmthcnexecute.asp> をご覧ください。

詳細は、[ActiveX Data Objects\(ADO\) Frequently Asked Questions](#) を参照。

Active Server Pages (ASP) との Connector/ODBC の使用

DSN で `Return matching rows` オプションを選択してください。

Connector/ODBC を使用し、ASP を介して MySQL にアクセスする方法は、次の資料を参考にしてください：

- [Using MyODBC To Access Your MySQL Database Via ASP](#)
- [ASP and MySQL at DWAM.NT](#)

ASP に関するよくある質問のリストは、<http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp> で閲覧できます。

Visual Basic (ADO, DAO and RDO) および ASP を使った Connector/ODBC の使用

Visual Basic および ASP に関する参考資料：

- [MySQL BLOB columns and Visual Basic 6](#) by Mike Hillyer (<mike@openwin.org>).
- [How to map Visual basic data type to MySQL types](#) by Mike Hillyer (<mike@openwin.org>).

Connector/ODBC の Borland アプリケーションとの使用

Borland Database Engine (BDE) が使用されるすべての Borland アプリケーションでは、互換性を高めるため次の手順に従ってください：

- BDE 3.2 以降にアップデートする。
- DSN で `Don't optimize column widths` オプションを有効にする。
- DSN で `Return matching rows` オプションを有効にする。

Connector/ODBC の Borland Builder 4 との使用

クエリを開始する際、`Active` プロパティが、`Open` メソッドを使用することができます。`Active` は、`SELECT * FROM ...` クエリを自動的に発行することで開始するので注意してください。大規模なテーブルを扱っている場合、これは欠点になりえます。

Delphi を使った Connector/ODBC の使用

また以下は、Connector/ODBC の ODBC エントリと BDE エントリの両方をセットアップする、役に立ちうる Delphi コードの一例です。BDE エントリには、身近な Delphi Super Page から無料で入手できる BDE Alias Editor が必要です。(Bryan Brunton <bryan@flesherfab.com> に提供を感謝します)：

```
fReg:= TRegistry.Create;
fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
fReg.WriteString('Database', 'Documents');
fReg.WriteString('Description', '');
fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', '');
```

```
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('Software\\ODBC\\ODBC.INI\\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;

Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');

AliasEditor.Add('DocumentsFab','MySQL',Memo1.Lines);
```

C++ Builder を使った Connector/ODBC の使用

BDE 3.0. でテスト済み。判明している問題は、テーブルのスキーマに変更がある時、クエリ フィールドがアップデイトされないことだけです。しかし、これまでのところ問題にはなっていませんが、BDE は基本キーを認識せず、**PRIMARY** と名付けられたインデックスのみを認めるようです。

ColdFusion を使った Connector/ODBC の使用

次の情報は ColdFusion の資料より引用したものです。

次の情報を使用して Linux 用の ColdFusion サーバを構成し、Connector/ODBC を使った **unixODBC** ドライバを MySQL データソースに使用します。Allaire は、MyODBC 2.50.26 が MySQL 3.22.27 および Linux 用の ColdFusion に対応することを立証しています (最近のバージョンであれば問題なく作動するはずです)。 <http://dev.mysql.com/downloads/connector/odbc/> で Connector/ODBC をダウンロードすることができます。

ColdFusion version 4.5.1 があれば、ColdFusion Administrator を使って MySQL を加えることができます。しかし、ColdFusion version 4.5.1 にドライバは含まれていません。MySQL ドライバが ODBC データソースのドロップダウン リストに現れる前に、Connector/ODBC ドライバを構築し、[/opt/coldfusion/lib/libmyodbc.so](http://opt/coldfusion/lib/libmyodbc.so) にコピーする必要があります。

Contrib ディレクトリは [mydsn-xxx.zip](#) プログラムを含んでおり、これにより ColdFusion アプリケーション上の Connector/ODBC ドライバ用に、DSN レジストリ ファイルを構築または除去することができます。

ColdFusion と Connector/ODBC の使用に関する情報とガイドは、次の外部サイトをご覧ください：

- [Troubleshooting Data Sources and Database Connectivity for Unix Platforms.](#)

OpenOffice を使った Connector/ODBC の使用

Open Office (<http://www.openoffice.org>) [How-to: MySQL + OpenOffice. How-to: OpenOffice + MyODBC + unixODBC.](#)

Sambar Server を使った Connector/ODBC の使用

Sambar Server (<http://www.sambarserver.info>) [How-to: MyODBC + SambarServer + MySQL.](#)

Pervasive Software DataJunction を使った Connector/ODBC の使用

Data junction が **ENUM** をアウトプットすると MySQL に不具合が起きるので、**VARCHAR** をアウトプットするよう変更してください。

SunSystems Vision を使った Connector/ODBC の使用

[Return matching rows](#) オプションを選択してください。

Connector/ODBC エラーコード と 解決法：

以下のセクションでは、一般的なエラーとその解決法、または代替案を説明します。問題が引き続き起こる場合は、Connector/ODBC メーリングリストを利用してください。[Connector/ODBC のコミュニティー支援](#) 参照。

多くの問題は、Connector/ODBC ドライバを最新版に更新することで解決できます。Windows では、最新バージョンの Microsoft Data Access Components (MDAC) をインストールするようにしてください。

Questions

- [24.1.6.3.1: \[1312\]](#) 私の MyODBC 2.50 アプリケーションは Connector/ODBC 3.51 に対応しますか？
- [24.1.6.3.2: \[1312\]](#) Windows XP x64 版、または Windows Server 2003 R2 x64 に Connector/ODBC をインストールしました。インストールは成功したのですが、Connector/ODBC ドライバが [ODBC Data Source Administrator](#) に現われません。
- [24.1.6.3.3: \[1313\]](#) [ODBC Data Source Administrator](#) にある **Test** ボタンを接続、または使用すると、エラー 10061 (サーバー接続不可) が発生します。
- [24.1.6.3.4: \[1313\]](#) トランザクションを使用すると、次のエラーが報告されます :[Transactions are not enabled](#)
- [24.1.6.3.5: \[1313\]](#) クエリをサブミットすると、次のエラーが報告されます :[Cursor not found](#)
- [24.1.6.3.6: \[1313\]](#) リンクしたテーブルに記録をインサートまたはアップデートするとき、Access が記録を [#DELETED#](#) として報告してきます。
- [24.1.6.3.7: \[1314\]](#) Write Conflicts および Row Location エラーにはどう対処すべきですか？
- [24.1.6.3.8: \[1314\]](#) データを Access 97 から MySQL にエクスポートすると [Syntax Error](#) が報告されます。
- [24.1.6.3.9: \[1314\]](#) データを Microsoft DTS から MySQL にエクスポートすると [Syntax Error](#) が報告されます。
- [24.1.6.3.10: \[1314\]](#) ODBC.NET を Connector/ODBC と使用すると、空のストリング (長さ 0) を抽出している間、SQL_NO_DATA 例外を出し始めます。
- [24.1.6.3.11: \[1314\]](#) Visual Basic と ASP で [SELECT COUNT\(*\) FROMtbl_name](#) を使用すると、エラーが戻ってきます。
- [24.1.6.3.12: \[1314\]](#) [AppendChunk\(\)](#) または [GetChunk\(\)](#) ADO メソッドを使用すると、[Multiple-step operation generated errors.Check each status value](#) エラーが戻ってきます。
- [24.1.6.3.13: \[1314\]](#) Linked Table で記録を編集していると、Access が [Another user had modified the record that you have modified](#) を戻してきます。
- [24.1.6.3.14: \[1315\]](#) Unix/Linux 下にある Connector/ODBC ライブラリにアプリケーションを直接リンクしようとすると、アプリケーションがクラッシュします。
- [24.1.6.3.15: \[1315\]](#) Microsoft Office スイートのアプリケーションでは、[DATE](#) または [TIMESTAMP](#) カラムのあるテーブルをアップデートできません。
- [24.1.6.3.16: \[1315\]](#) Connector/ODBC 5.x (Beta) を MySQL 4.x サーバに接続しているとき、[1044 Access denied for user 'xxx'@'%' to database 'information_schema'](#) エラーが戻ってきます。

Questions and Answers

24.1.6.3.1: 私の MyODBC 2.50 アプリケーションは Connector/ODBC 3.51 に対応しますか？

MyODBC 2.50 を基にしたアプリケーションならば、Connector/ODBC 3.51 以降であれば問題なく作動するはずで
す。以前のバージョン下で使用していたことのある最新版 Connector/ODBC に不具合がある場合は、バグ レポートを
提出してください。[Connector/ODBC の不具合またはバグのレポート](#) 参照。

24.1.6.3.2: Windows XP x64 版、または Windows Server 2003 R2 x64 に Connector/ODBC をインストールしまし
た。インストールは成功したのですが、Connector/ODBC ドライバが [ODBC Data Source Administrator](#) に現われませ
ん。

これはバグではありませんが、Windows x64 版の ODBC ドライバとの作動傾向に関連しています。Windows x64 版では、Connector/ODBC ドライバが `%SystemRoot%\SysWOW64` フォルダにインストールされています。しかし、[Administrative Tools](#) または Windows x64 Editions の [Control Panel](#) で入手できる、デフォルトの [ODBC Data Source Administrator](#) は `%SystemRoot%\system32` フォルダに入っており、このフォルダでしか ODBC ドライバを探しません。

Windows x64 版では、`%SystemRoot%\SysWOW64\odbcad32.exe` にある ODBC 管理ツールを使用すれば、インストールされた Connector/ODBC ドライバの所在を正確に特定し、それによって Connector/ODBC DSN を作成することができます。

この問題は、当初 Bug #20301 として報告されました。

24.1.6.3.3: [ODBC Data Source Administrator](#) にある [Test](#) ボタンを接続、または使用すると、エラー 10061 (サーバー接続不可) が発生します。

このエラーの原因には、サーバーの不具合、ネットワークの問題、ファイアウォールとポートブロックの問題等、多くの事柄が考えられます。詳細は [Can't connect to \[local\] MySQL server](#) をご覧ください。

24.1.6.3.4: トランザクションを使用すると、次のエラーが報告されます :[Transactions are not enabled](#)

このエラーは、トランザクションをサポートしない MySQL テーブルでトランザクションを使用していることを示します。トランザクションは [InnoDB](#) データベース エンジンを使っているときに、MySQL でサポートされます。また、MySQL MySQL 5.1 以前のバージョンでは、[BDB](#) エンジンも使用してください。

先に進む前に、次を確認してください：

- あなたの MySQL サーバが トランザクション データベース エンジンをサポートしているか確認。[SHOW ENGINES](#) を使って、利用可能なエンジン タイプのリストを得る。
- あなたがアップデートしているテーブルが、トランザクション データベース エンジンを使用するか確認。
- DSN で、[disable transactions](#) オプションを有効にしていないことを確認。

24.1.6.3.5: クエリをサブミットすると、次のエラーが報告されます :[Cursor not found](#)

これはアプリケーションが古い MyODBC 2.50 バージョンを使用していることが原因で、そのため `SQLSetCursorName` を介して明確にカーソル名を設定していません。これを解決するには、Connector/ODBC 3.51 バージョンへアップグレードしてください。

24.1.6.3.6: リンクしたテーブルに記録をインサートまたはアップデートするとき、Access が記録を [#DELETED#](#) として報告してきます。

インサートされた、またはアップデートされた記録が、アクセスで [#DELETED#](#) と表示される場合：

- Access 2000 を使用している場合は、最新の (バージョン 2.6 以降) Microsoft MDAC ([Microsoft Data Access Components](#)) を <http://www.microsoft.com/data/> から入手してインストールしてください。データを MySQL にエクスポートすると、テーブルおよびカラムの名前が特定されない、という Access のバグは、これで解消されます。このバグに対する他の代替案は、MyODBC を 2.50.33 以降、そして MySQL を 3.23.x 以降にアップグレードすることで、このふたつがそろえば、この問題を回避することができます。

また、Microsoft Jet 4.0 Service Pack 5 (SP5) を <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114> から入手して加えることもお勧めします。それによって、Access でカラムが [#DELETED#](#) と表示されるという問題の一部は解決します。

注記

MySQL 3.22 を使用している場合は、MDAC パッチを施し、MyODBC 2.50.32 または 2.50.34 以降を使用することで問題が回避できます。

- すべてのバージョンの Access で、Connector/ODBC [Return matching rows](#) オプションを有効にしてください。Access 2.0 では、[Simulate ODBC 1.0](#) オプションも加えて有効にしてください。
- アップデートを可能にしたいすべてのテーブルに、タイムスタンプを設けてください。
- テーブルに基本キーを設けてください。それを行わない場合、新規またはアップデートされた行が [#DELETED#](#) と表示される場合があります。

- **DOUBLE** 浮動小数点フィールドのみを使用。単精度浮動小数点値と比較する際、Access に不具合が起きるためです。この徴候では通常、新規またはアップデートされた行が **#DELETED#** と表示される、または行の検出もしくはアップデートができなくなることがあります。
- Connector/ODBC を使って **BIGINT** カラムのあるテーブルへリンクしている場合、結果が **#DELETED#** と表示されます。この問題を避けるには：
 - データタイプとして、**TIMESTAMP** にもうひとつダミー カラムを設ける。
 - ODBC DSN Administrator の接続ダイアログで、**Change BIGINT columns to INT** オプションを選択。
 - Access からテーブル リンクを削除し、また再作成する。

古い記録はまだ **#DELETED#** と表示されるが、新しく加えられた、またはアップデートされた記録は正常に表示される。

24.1.6.3.7: Write Conflicts および Row Location エラーにはどう対処すべきですか？

下記のエラーが表示されたら、DSN 構成ダイアログで **Return Matching Rows** オプションを選択するか、**OPTION=2** を接続パラメータとして指定してください：

Write Conflict. Another user has changed your data.

Row cannot be located for updating. Some values may have been changed since it was last read.

24.1.6.3.8: データを Access 97 から MySQL にエクスポートすると **Syntax Error** が報告されます。

このエラーは Access 97 と、Connector/ODBC 3.51.02 より前のバージョンに特有のものです。最新バージョンの Connector/ODBC にアップグレードすれば、この問題は解決できます。

24.1.6.3.9: データを Microsoft DTS から MySQL にエクスポートすると **Syntax Error** が報告されます。

このエラーは、**TEXT** または **VARCHAR** データタイプを使っている MySQL テーブルでのみ起こります。Connector/ODBC ドライバを バージョン 3.51.02 以降にアップグレードすれば、この問題は解決されます。

24.1.6.3.10: ODBC.NET を Connector/ODBC と使用すると、空のストリング (長さ 0) を抽出している間、SQL_NO_DATA 例外を出し始めます。

この問題を解決するパッチが、<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243> から入手できます。

24.1.6.3.11: Visual Basic と ASP で **SELECT COUNT(*) FROMtbl_name** を使用すると、エラーが戻ってきます。

このエラーは、**COUNT(*)** 発現が **BIGINT** を戻しており、これほど大きな数字になると、ADO が理解しきれないのが原因です。**Change BIGINT columns to INT** オプション (オプション値 16384) を選択してください。

24.1.6.3.12: **AppendChunk()** または **GetChunk()** ADO メソッドを使用すると、**Multiple-step operation generated errors.Check each status value** エラーが戻ってきます。

ADO の **GetChunk()** と **AppendChunk()** メソッドは、カーソル ロケーションが **adUseServer** として定義されていると予想通りに作動しません。一方、**adUseClient** を使用することで、この問題を乗り越えることができます。

簡潔な例は、次のリンクで見つけることができます。 http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

24.1.6.3.13: Linked Table で記録を編集していると、Access が **Another user had modified the record that you have modified** を戻ってきます。

ほとんどの場合、この問題は次のうちからひとつを行うと解決します：

- テーブルに基本キーがなければ追加する。
- タイムスタンプ カラムがなければ追加する。
- 倍精度浮動小数フィールドのみを使用。プログラムによっては、単精度浮動小数と比較する際に不具合が発生する。

これらの対策でも解決しない時は、ODBC マネージャーからのログ ファイル (ODBCADMIN からログを要求する際に得られるログ)、そして Connector/ODBC ログ の作成を開始し、問題点の追求に役立ててください。詳しい方法は、[ODBC トレース ファイルの獲得](#) をご覧ください。

24.1.6.3.14: Unix/Linux 下にある Connector/ODBC ライブラリにアプリケーションを直接リンクしようとすると、アプリケーションがクラッシュします。

Unix/Linux 下の Connector/ODBC 3.51 は、アプリケーションへの直接リンクに対応していません。iODBC もしくは unixODBC などのドライバ マネージャーを使用して、ODBC ソースに接続しなければなりません。

24.1.6.3.15: Microsoft Office スイートのアプリケーションでは、[DATE](#) または [TIMESTAMP](#) カラムのあるテーブルをアップデートできません。

これは、Connector/ODBC のよく知られた問題です。フィールドが確実にデフォルト値 ([NULL](#) ではなく、また、ゼロでないデフォルト値。つまり [0000-00-00 00:00:00](#) のデフォルト値は無効) を持っているか確認してください。

24.1.6.3.16: Connector/ODBC 5.x (Beta) を MySQL 4.x サーバに接続しているとき、[1044 Access denied for user 'xxx'@'%' to database 'information_schema'](#) エラーが戻ってきます。

Connector/ODBC 5.x は MySQL 5.0 以降に対応するよう設計されており、データ定義情報を特定する [INFORMATION_SCHEMA](#) データベースを有効に使用しています。MySQL 4.1 のサポートは最終的なリリースで予定されています。

Connector/ODBC サポート

Connector/ODBC の使用に関するサポートは、多くの場所で受けることができます。Connector/ODBC Mailing List か Connector/ODBC Forum への参加もお勧めします。バグや不具合を MySQL に報告する前に、まずは [Connector/ODBC のコミュニティ支援](#) でご確認ください。

Connector/ODBC のコミュニティ支援

MySQL AB はメーリング リストを用いて、ユーザ コミュニティを後援します。Connector/ODBC 関連の問題については、[<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) メーリングリストから、経験のあるユーザに援助を求めることができます。アーカイブはオンラインで、<http://lists.mysql.com/myodbc> にて閲覧できます。

MySQL メーリングリストへの参加、リスト アーカイブの参照については、<http://lists.mysql.com/>、および [MySQL メーリング リスト](#) を参照してください。

経験を積んだユーザからのコミュニティ支援は、[ODBC Forum](#) でも得ることができます。また、<http://forums.mysql.com> にあるもうひとつの MySQL Forum でも、他のユーザーとの情報交換が行えます。[MySQL フォーラムにおける MySQL コミュニティサポート](#) をご覧ください。

Connector/ODBC の不具合またはバグのレポート

Connector/ODBC で不具合や問題があった場合は、[ODBC Manager](#) ([ODBC ADMIN](#) からログを請求した時に受け取るログ) と Connector/ODBC からログ ファイルを作成することをお勧めします。そのプロシージャについては、[ODBC トレース ファイルの獲得](#) で説明されています。

Connector/ODBC トレース ファイルをチェックし、問題を確認してください。[myodbc.log](#) ファイルにあるストリング [>mysql_real_query](#) を検索することで、どの文が発行されたかを特定することができます。

また、[mysql](#) クライアント プログラムか、[admndemo](#) からの文の発行も試みてください。エラーが Connector/ODBC か MySQL で発生しているものが調べる手助けになります。

問題があると判明した場合は、該当の口ウのみ (最大 40 行) を [myodbc](#) メーリングリストまでお送りください。[MySQL メーリング リスト](#) をご覧ください。Connector/ODBC または ODBC ログの全体を送らないようお願いいたします !

メールには次の情報をご記入いただくと参考になります。

- オペレーション システムとバージョン
- Connector/ODBC バージョン
- ODBC Driver Manager のタイプとバージョン

- MySQL サーバのバージョン
- Driver Manager からの ODBC トレース
- Connector/ODBC ドライバからの Connector/ODBC ログ ファイル
- 簡単な再現可能サンプル

詳しい詳細を提供していただくのが問題解決には一番ですので、その旨をご理解ください。

また、バグを投稿する前に、<http://lists.mysql.com/myodbc> で MyODBC メーリングリス アーカイブを確認してください。

問題の原因が分からない場合の最終手段としては、[tar](#) にアーカイブを作成するか、Connector/ODBC トレース ファイルを含む Zip フォーマット、ODBC ログ ファイル、そして問題を説明した [README](#) ファイルを作成します。そしてそれを <ftp://ftp.mysql.com/pub/mysql/upload/> までお送りください。このファイルには MySQL の技術者のみがアクセスし、データは極秘に扱われます。

問題を実証するプログラムを作成できる場合は、それもアーカイブに加えてください。

他の SQL サーバでは問題なく作動する場合は、まったく同じ SQL 文を実行した ODBC ログ ファイルも加えていただければ、ふたつのシステムでの結果を当社で比較することができます。

詳しい詳細を提供していただくのが問題解決には一番ですので、その旨をご理解ください。

Connector/ODBC Patch の提供

既存のコードや不具合へのパッチ、または問題解決の提案を提供していただける場合は、<myodbc@lists.mysql.com> までメールでお寄せください。

Connector/ODBC 変更履歴

Connector/ODBC Change History (Changelog) は、MySQL のメイン Changelog に収められています。[MySQL Connector/ODBC \(MyODBC\) Change History](#) を参照してください。

開発者

以下はConnector/ODBC および Connector/ODBC 3.51 ドライバの開発に携わった MySQL AB の開発者です。

- Michael (Monty) Widenius
- Venu Anuganti
- Peter Harvey

MySQL Connector/NET

Connector/NET は、安全で性能の高い MySQL とのデータの接続性を必要とする .NET アプリケーションを、開発者が容易に作成できるようにするものです。必要な ADO.NET インターフェイスを実装し、ADO.NET 対応ツールを統合します。開発者は、好きな .NET 言語を使用してアプリケーションを構築することができます。Connector/NET は 100% 純粋な C# で書かれ、完全に管理された ADO.NET ドライバです。

Connector/NET は次の機能に完全対応しています：

- MySQL 5.0 の機能 (ストアド プロシージャ等)
- MySQL 4.1 の機能 (サーバ側のプリペアド ステートメント、Unicode 、共有メモリへのアクセス、等)
- 行や 最大 2 ギガバイトの BLOB の送信と受信を行うための大型パケットのサポート
- クライアントとサーバ間のデータ ストリームの圧縮を可能にするプロトコル圧縮
- TCP/IP ソケット、名前付きパイプ、または共有メモリを使用した、Windows での接続のサポート
- TCP/IP ソケット、または Unix ソケットを使用した、Unix での接続のサポート
- Novell によって開発された Open Source Mono フレームワークのサポート

- 完全に管理され、MySQL クライアントを利用しないライブラリ

この資料は Connector.NET のユーザ ガイドとして書かれており、完全なシンタックスの参照を含みます。シンタックスの情報は [Documentation.chm](#) ファイルにも、Connector/NET の配布と共に収められています。

MySQL 5.0 以降のバージョンを使用し、Visual Studio が開発環境となっている場合、MySQL Visual Studio Plugin の利用もご考慮ください。このプラグインは DDEX (Data Designer Extensibility) プロバイダとして作動し、Visual Studio でデータ設計ツールを使用して、MySQL データベース内のスキーマやオブジェクトを操作することを可能にします。詳細は [MySQL Visual Studio プラグイン](#) をご覧ください。

Connector/NET のバージョン

現在利用できる Connector/ODBC には 2 バージョンあります：

- Connector/NET 1.0 は MySQL 4.0 と MySQL 5.0 の機能をサポートし、ADO.NET ドライバ インターフェイスとの完全な相互性を持っています。

Connector/NET 5.0 は現在テスト段階ですが、MySQL 4.0、MySQL 4.1、MySQL 5.0 および MySQL 5.1 の機能に対応します。Connector/NET 5.0 はまた、使用アドバイザーやパフォーマンス モニタのサポートを含む、ADO.NET 2.0 インターフェイスとサブクラスへの完全サポートを備えています。

注記

MySQL 製品のバージョン ナンバーは X.X.X というように構成されています。しかし、Windows ツール (コントロールパネル、プロパティ表示) では、バージョン ナンバーが XX.XX.XX. と表示されることがあります。例えば、MySQL の正式なバージョン ナンバーである 5.0.9 が、Windows ツールでは 5.00.09 と表示される場合があります。これは単にナンバリングが異なって表示されるだけであり、バージョンそのものに違いはありません。

Connector/NET のインストール

Connector/NET は、.NET フレームワークをサポートするすべてのプラットフォームで作動します。.NET フレームワークは第一に最新バージョンの Microsoft Windows でサポートされていますが、Open Source Mono フレームワークを介すると Linux にも対応します (<http://www.mono-project.com> 参照)。

Connector/NET は <http://dev.mysql.com/downloads/connector/net/1.0.html> からダウンロードすることができます。

Connector/NET の Windows へのインストール

Windows では、インストールはバイナリ インストールか、Connector/NET コンポーネントと共に Zip ファイル形式でダウンロードすることによって行うことができます。

インストールの前に、最新バージョンの .NET Framework がインストールされているかなど、使用するシステムが更新されているか確認してください。

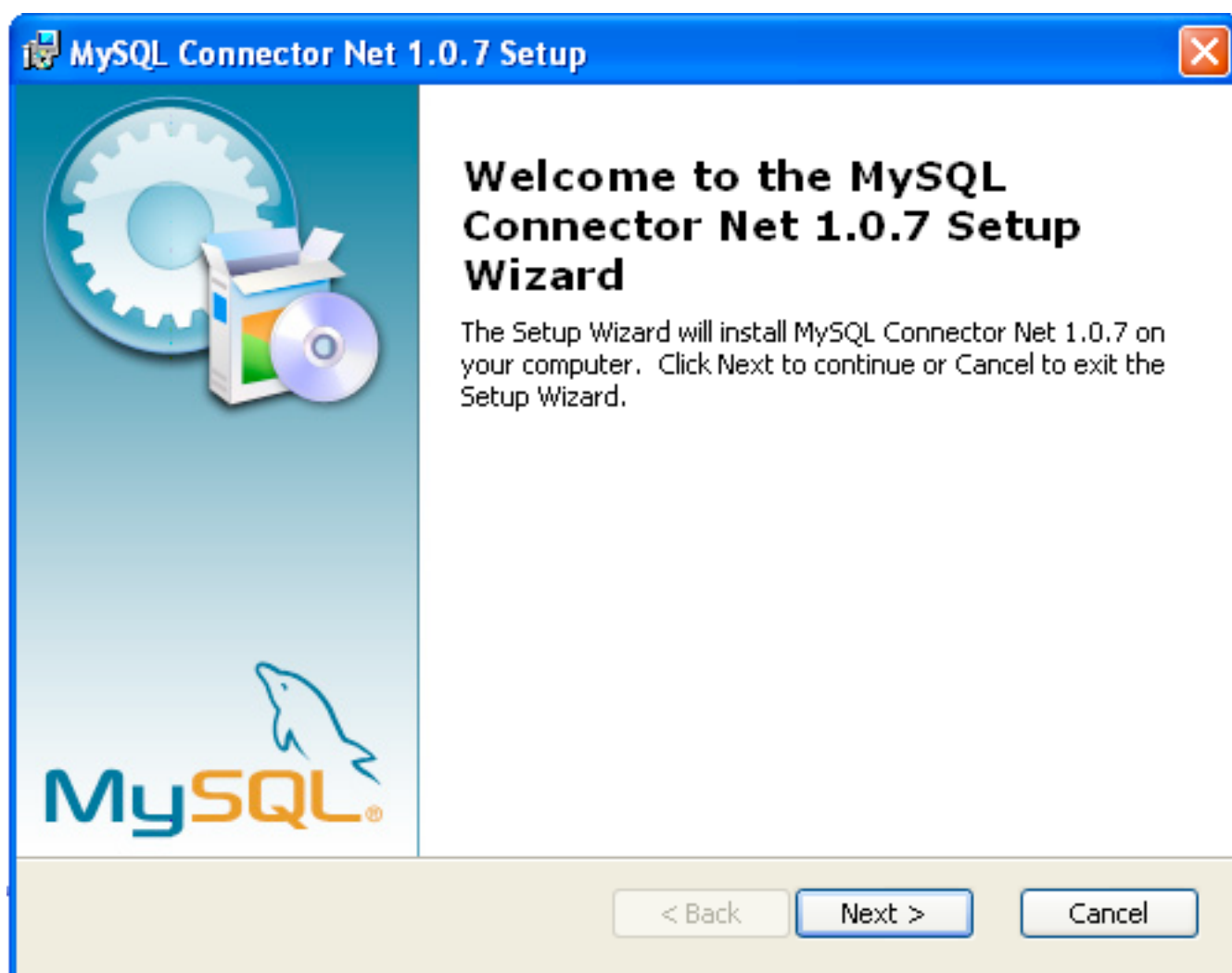
インストーラを使用した Connector/NET のインストール

Windows に Connector/NET をインストールする場合、インストーラを使用するのが最も容易な方法であり、インストールされたコンポーネントにはソースコード、テストコード、および完全な参考資料が含まれています。

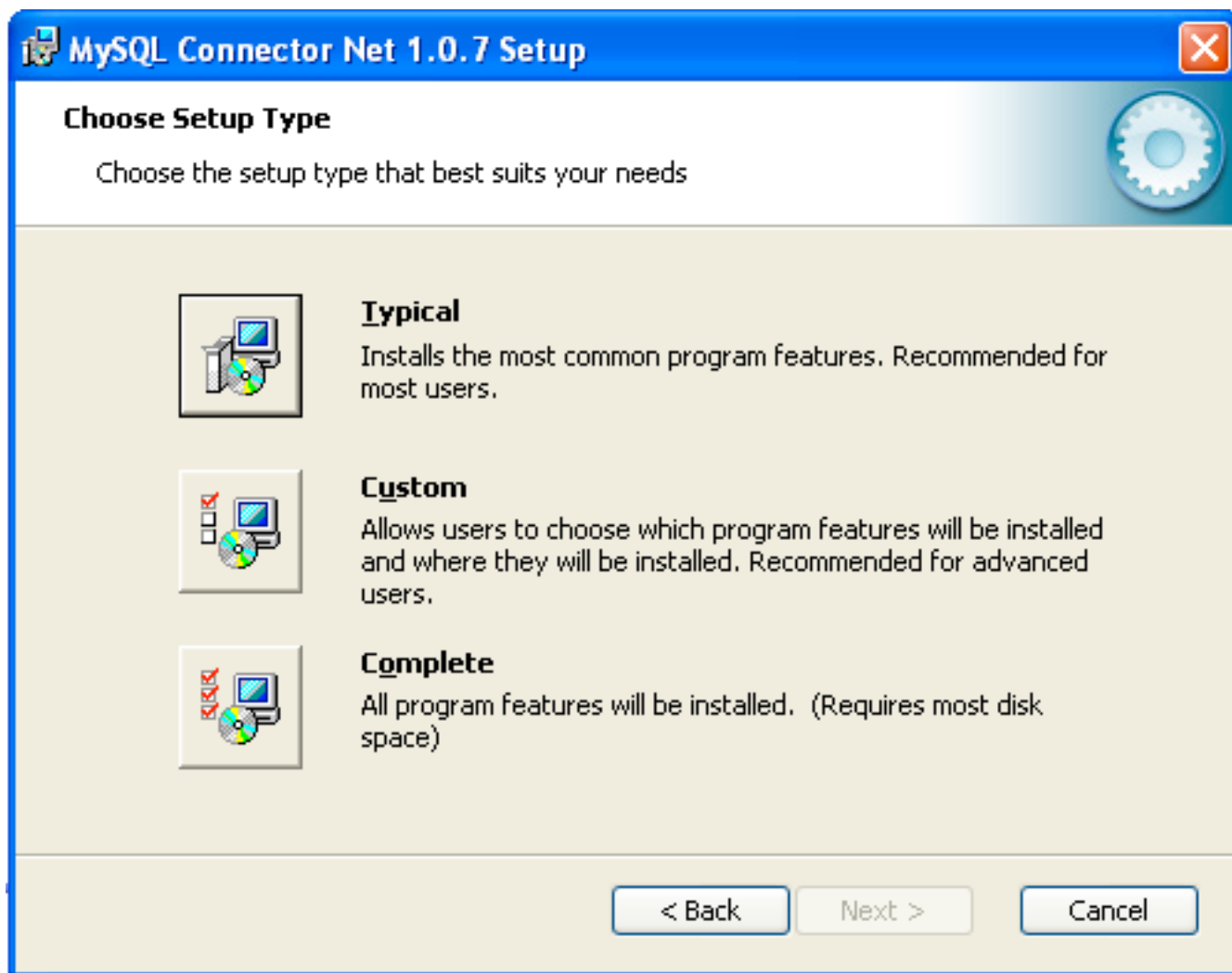
Connector/NET は、すべての Windows オペレーション システムへのインストールに使用できる、Windows Installer (.msi) インストール パッケージを使ってインストールされます。MSI パッケージは [mysql-connector-net-version.zip](#) と名付けられた ZIP アーカイブに含まれています。この [version](#) は Connector/NET のバージョンを指します。

Connector/NET のインストール方法：

1. ダウンロードした Zip から抽出した MSI インストーラ ファイルをダブルクリック。Next を押してインストールを開始します。



2. 実行したいインストールのタイプを選択。



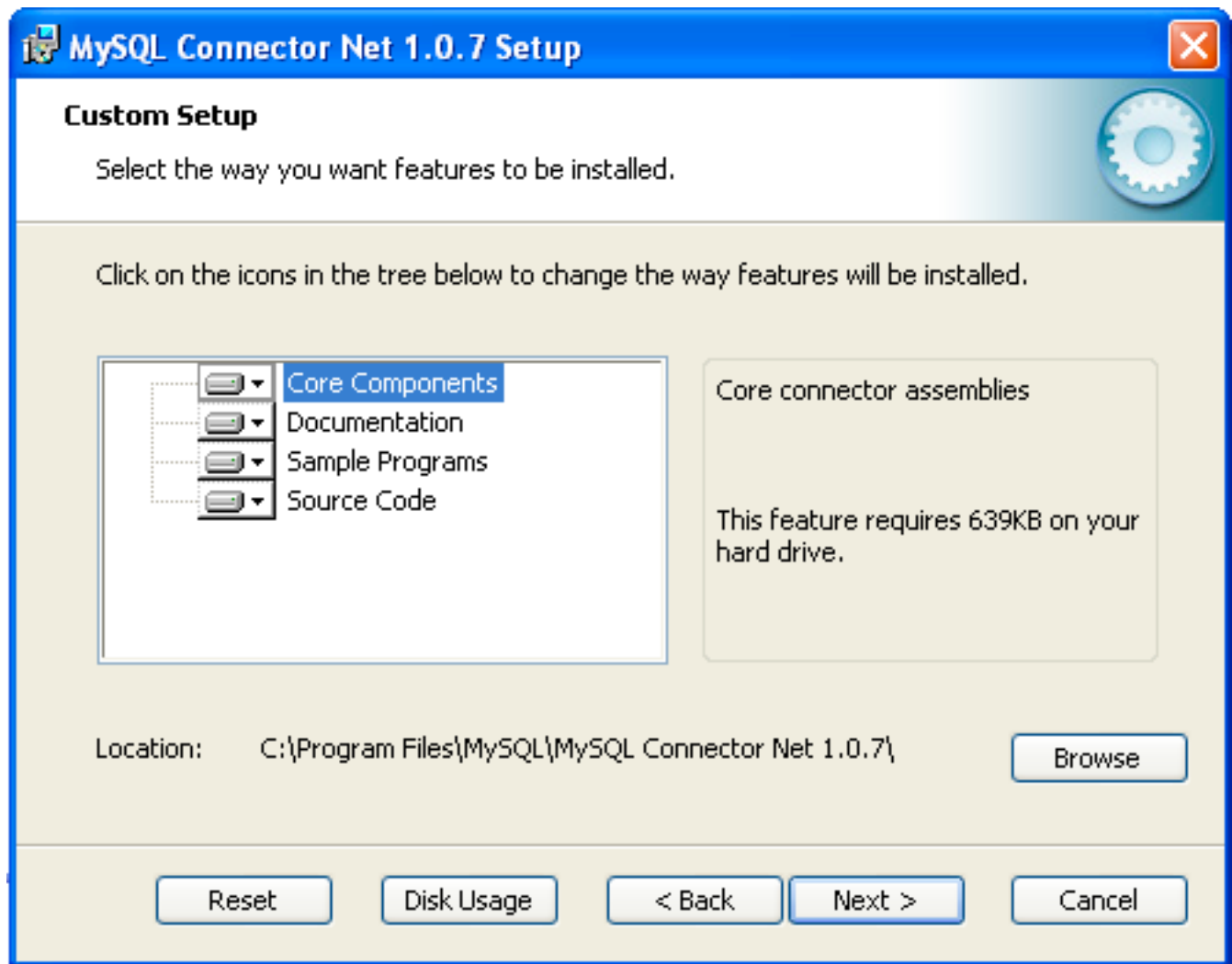
ほとんどの場合には、Typical インストールが適しています。Typical ボタンをクリックし、ステップ 5 に進んでください。Complete インストールは、利用可能なファイルをすべてインストールします。Complete インストールを行うには、Complete ボタンをクリックし、ステップ 5 に進みます。インストールするコンポーネントや、インストールのオプションを選ぶなど、インストールをカスタマイズしたい場合は、Custom ボタンを押してステップ 3 に進みます。

Connector/NET インストーラは、Global Assembly Cache (GAC) 内でコネクタを登録します。これにより、Connector/NET コンポーネントをはっきりと参照したものだけでなく、すべてのアプリケーションで Connector/NET コンポーネントを利用することができるようになります。このインストーラはまた、文章やリリース情報への必要なリンクを、Start メニューで作成します。

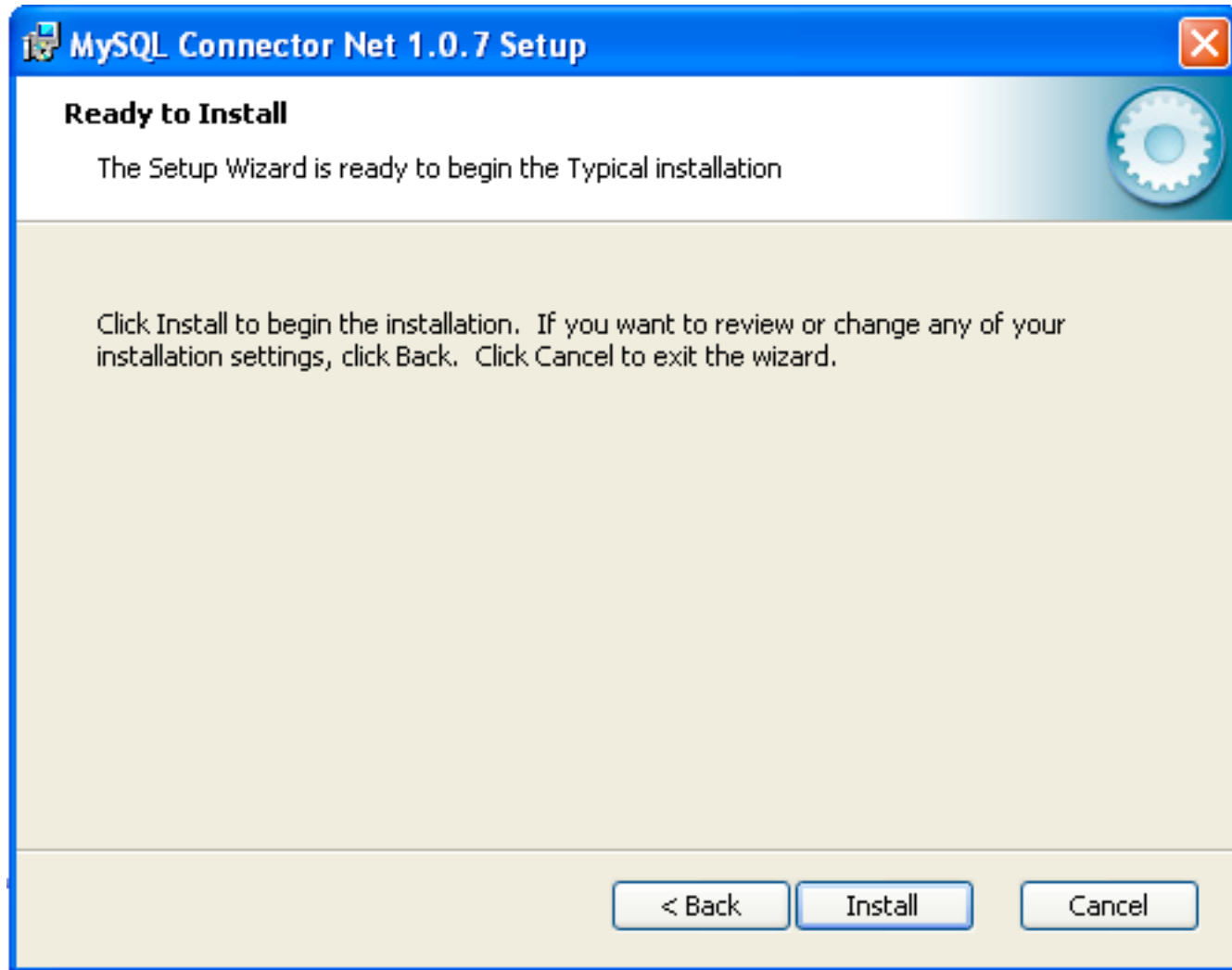
3. Custom インストールを選んだ場合、コア インターフェイス コンポーネント、解説文章 (CHM ファイル) サンプル、例、そしてソースコードを含む、インストールしたい各コンポーネントを選ぶことができます。アイテムとその設置レベルを選択し、Next をクリックしてインストールを続行します。

注記

Connector/NET 1.0.8 以下と Connector 5.0.4 以下では、インストーラは .NET Framework の 1.x と 2.x 両方のためにバイナリをインストールしようとします。フレームワークのバージョンをひとつだけインストールしている場合、コネクタのインストールが失敗する可能性があります。もしこれで失敗した場合は、Custom インストールの手順からインストールされるフレームワークのバージョンを選ぶことができます。



4. インストールの最終確認が行われます。**Install** をクリックして、ファイルをマシンへコピーおよびインストールします。



5. インストールが完了したら、**Finish** をクリックしてインストールを終了します。

特に指定をしない限り、Connector/NET は `C:\Program Files\MySQL\MySQL Connector Net X.X.X` にインストールされます。`X.X.X` はインストールした Connector/NET のバージョンの番号になります。新たなインストールが、既存の Connector/NET のバージョンを書き換えることはありません。

インストールのタイプによっては、インストールされたコンポーネントには、次のうちの一部分か、またはすべてが含まれることになります。

- **bin** - .NET 環境の各バージョンのための Connector/NET MySQL ライブラリ。
- **docs** - Connector/NET 資料の CHM を含む。
- **samples** - Connector/NET コンポーネントを使用するサンプル コードやアプリケーション。
- **src** - Connector/NET コンポーネントのためのソース コード。

`msiexec` ツールに `/quiet` or `/q` コマンドライン オプションを使用し、Connector/NET パッケージをユーザへの告知なしで自動的に (デフォルトのオプションを使用して) インストールすることもできます。このオプションを使用すると、オプションを選ぶことはできなくなり、プロンプトやメッセージ、またはダイアログ ボックスは表示されません。

```
C:\> msiexec /package conector-net.msi /quiet
```

オプションを選択するダイアログ ボックスは表示されないまま、自動インストール中に進行状況バーを表示するには、[/passive](#) オプションを使用します。

Zip パッケージを使用した Connector/NET のインストール

インストーラの起動に問題がある場合は、インストーラなしで .zip ファイルをダウンロードすることもできます。このファイルは [mysql-connector-net-version-noinstall.zip](#) と呼ばれるものです。ダウンロードを行ったら、ファイルを希望のロケーションへ抽出することができます。

この .zip ファイルは次のディレクトリを含みます：

- **bin** - .NET 環境の各バージョンのための Connector/NET MySQL ライブラリ。
- **doc** - Connector/NET 資料の CHM を含む。
- **Samples** - Connector/NET コンポーネントを使用するサンプル コードやアプリケーション。
- **mysqlclient** - Connector/NET コンポーネントのためのソース コード。
- **testsuite** - Connector/NET コンポーネントの動作を確認するために使用されるテスト スイート。

Mono を使用した Unix に Connector/NET をインストールする

Unix インストール システムに Connector/NET コンポーネントを設置ためのインストーラは入手不可能です。しかし、インストールは非常に簡単に行うことができます。インストールの前に、Mono プロジェクトのインストールが実行可能か確認してください。

Mono プロジェクトを通じて MySQL サーバに接続するようにしたい Unix 環境では、Connector/NET コンポーネントのみをインストールするよう注意してください。Java や Perl など、別の環境で展開または開発する場合には、より適切な接続コンポーネントを使用してください。詳細は [24章MySQL コネクタ](#)、または [23章APIとライブラリー](#) をご覧ください。

Unix/Mono に Connector/NET をインストールするには：

1. [mysql-connector-net-version-noinstall.zip](#) をダウンロードして、コンテンツを抽出。
2. [MySQL.Data.dll](#) ファイルを、Mono プロジェクトのインストール フォルダにコピーする。
3. Connector/NET コンポーネントを、[gacutil](#) コマンドで Global Assembly Cache に登録する：

```
shell> gacutil /i MySQL.Data.dll
```

インストールが完了すれば、Connector/NET コンポーネントとコンパイルされたアプリケーションは以後変更する必要はありません。しかし、アプリケーションをコンパイルする際、[-r:MySQLData.dll](#) コマンドライン オプションを使用して Connector/NET コンポーネントを確実に含むようにしてください。

ソースを使用した Connector/NET のインストール

注意

当社の新しいコードのテストにご興味がおありの場合は、このセクションをお読みください。単にシステムでの MySQL Connector/ODBC の利用をご希望の場合は、標準のリリース配布物をお使いください。

Connector/NET ソースツリーにアクセスするには、Subversion をインストールする必要があります。Subversion は <http://subversion.tigris.org/> から無料で入手することができます。

最新の開発ソースツリーは、<http://dev.mysql.com/tech-resources/sources.html> の公開 Subversion ツリーから得られます。

Connector/NET ソースを点検するには、Connector/NET ツリーのコピーを保存したいディレクトリに移動し、次のコマンドを使用します：

```
shell> svn co
http://svn.mysql.com/svnpublic/connector-net
```

Visual Studio プロジェクトは、Connector/NET の構築に使用できるソースに含まれています。

Connector/NET の例と使用ガイド

Connector/NET は、データベースへの接続、クエリとステートメントの実行、クエリの結果の管理に使用される複数のクラスで構成されています。

以下は Connector/NET の代表的なクラスです：

- [MySqlCommand](#):SQL 文を構成し、MySQL データベースに対して実行する。
- [MySqlCommandBuilder](#):MySQL データベースに関連付けられた DataSet の変更を調整するための単一テーブル コマンドを自動的に生成する。
- [MySqlConnection](#):MySQL サーバ データベースへのオープン接続を表す。
- [MySqlDataAdapter](#):データセットを満たし、MySQL データベースの更新に使用されるデータ コマンドのセットと データベース接続を表します。
- [MySqlDataReader](#):MySQL データベースの行の転送専用ストリーム読み取りの方法を提供。
- [MySqlException](#):MySQL がエラーを返す時に投入される例外。
- [MySqlHelper](#):プロバイダとの作業をより簡単にするヘルパー クラス。
- [MySqlTransaction](#):MySQL データベース内で作成される SQL トランザクションを表示。

このセクションでは、上記の各クラスの基本情報と例を記載します。さらに詳しいリファレンス ガイドは、[MySQL Connector/NET](#) をご覧ください。

MySqlCommand の使用

SQL 文を構成し、MySQL データベースに対して実行します。このクラスを継承することはできません。

[MySqlCommand](#) は MySQL データベースでコマンドを実行する次の手段を備えています：

アイテム	概要
ExecuteReader	行を返すコマンドを実行。
ExecuteNonQuery	SQL INSERT 、DELETE 、および UPDATE ステートメントなどのコマンドを実行。
ExecuteScalar	データベースから単一値 (例えば集約値) を取り出す。

[CommandText](#) プロパティをリセットし、[MySqlCommand](#) オブジェクトを再使用することができます。しかし、新しい、または以前のコマンドを実行する前に、[MySqlDataReader](#) を閉じる必要があります。

[MySqlException](#) が [MySqlCommand](#) を実行するメソッドで生成される場合、[MySqlConnection](#) は開いたままになります。この接続を閉じるのはプログラマの采配になります。

注記

プロバイダの以前のバージョンでは、SQL のパラメータをマークするのに '@' シンボルが使用されました。これは MySQL ユーザ変数に適合しないため、現在のプロバイダでは '?' シンボルを使用して SQL のパラメータを指定しています。以前のコードをサポートするには、'old syntax=yes' を接続ストリングで設定することができます。そうする場合、SQL で使用する予定のパラメータの定義に失敗しても、例外は投入されませんのでご注意ください。

例

以下は、[MySqlCommand](#) と [MySqlConnection](#) を作成する例です。[MySqlConnection](#) が開かれ、[MySqlCommand](#) の [Connection](#) として設定されます。この例はその後 [ExecuteNonQuery](#) を呼び出し、接続を閉じます。これを完遂するため、[ExecuteNonQuery](#) は SQL INSERT 文である接続ストリングとクエリ ストリングに渡されます。

Visual Basic 例：

```
Public Sub InsertRow(myConnectionString As String)
    " If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

C# 例 :

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionString);
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

クラス MySqlCommand コンストラクタ 形式 1

MySqlCommand のオーバーロード メソッド

MySqlCommand クラスの新規インスタンスを初期化します。

例

以下の例は MySqlCommand を作成し、そのプロパティの一部を設定します。

注記

この例は、[MySqlCommand](#) のオーバーロードされたコンストラクタのバージョンの使用法のひとつを挙げています。利用可能な他の例に関しては、各オーバーロードのトピックをご覧ください。

Visual Basic 例 :

```
Public Sub CreateMySqlCommand()
    Dim myConnection As New MySqlConnection _
        ("Persist Security Info=False;database=test;server=myServer")
    myConnection.Open()
    Dim myTrans As MySqlTransaction = myConnection.BeginTransaction()
    Dim mySelectQuery As String = "SELECT * FROM MyTable"
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection, myTrans)
    myCommand.CommandTimeout = 20
End Sub
```

C# 例 :

```
public void CreateMySqlCommand()
{
    MySqlConnection myConnection = new MySqlConnection("Persist Security Info=False;
        database=test;server=myServer");
    myConnection.Open();
    MySqlTransaction myTrans = myConnection.BeginTransaction();
    string mySelectQuery = "SELECT * FROM myTable";
```



```

MySQLCommand myCommand = new MySQLCommand(mySelectQuery, myConnection, myTrans);
myCommand.CommandTimeout = 20;
}

```

C++ 例：

```

public:
void CreateMySQLCommand()
{
    MySqlConnection* myConnection = new MySqlConnection(S"Persist Security Info=False;
    database=test;server=myServer");
    myConnection->Open();
    MySQLTransaction* myTrans = myConnection->BeginTransaction();
    String* mySelectQuery = S"SELECT * FROM myTable";
    MySQLCommand* myCommand = new MySQLCommand(mySelectQuery, myConnection, myTrans);
    myCommand->CommandTimeout = 20;
};

```

MySQLCommand クラスの新規インスタンスを初期化します。

ベース コンストラクタは、すべてのフィールドをそのデフォルト値に初期化します。愚痴のテーブルは、[MySQLCommand](#) のインスタンスの初期プロパティ値です。

プロパティ	初期値
CommandText	empty string ("")
CommandTimeout	0
CommandType	CommandType.Text
Connection	Null

これらのプロパティの値は、プロパティへの別の呼び出しを通して変更することができます。

例

以下の例は [MySQLCommand](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例：

```

Public Sub CreateMySQLCommand()
    Dim myCommand As New MySQLCommand()
    myCommand.CommandType = CommandType.Text
End Sub

```

C# 例：

```

public void CreateMySQLCommand()
{
    MySQLCommand myCommand = new MySQLCommand();
    myCommand.CommandType = CommandType.Text;
}

```

クラス MySQLCommand コンストラクタ 形式 2

クエリのテキストで [MySQLCommand](#) クラスの新規インスタンスを初期化します。

パラメータ：クエリのテキスト。

[MySQLCommand](#) のインスタンスが作成される際、次の赤/白のプロパティが初期値に設定されます。

プロパティ	初期値
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	Null

これらのプロパティの値は、プロパティへの別の呼び出しを通して変更することができます。

例

以下の例は `MySqlCommand` を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateMySqlCommand()
    Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySqlCommand(sql)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# 例 :

```
public void CreateMySqlCommand()
{
    string sql = "SELECT * FROM mytable";
    MySqlCommand myCommand = new MySqlCommand(sql);
    myCommand.CommandType = CommandType.Text;
}
```

クラス MySqlCommand コンストラクタ 形式 3

クエリのテキストと `MySqlConnection` で `MySqlCommand` クラスの新規インスタンスを初期化します。

パラメータ : クエリのテキスト。

パラメータ : SQL サーバのインターフェイスへの接続を表す `MySqlConnection`。

`MySqlCommand` のインスタンスが作成される際、次の赤/白のプロパティが初期値に設定されます。

プロパティ	初期値
<code>CommandText</code>	<code>cmdText</code>
<code>CommandTimeout</code>	0
<code>CommandType</code>	<code>CommandType.Text</code>
<code>Connection</code>	<code>connection</code>

これらのプロパティの値は、プロパティへの別の呼び出しを通して変更することができます。

例

以下の例は `MySqlCommand` を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateMySqlCommand()
    Dim conn as new MySqlConnection("server=myServer")
    Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySqlCommand(sql, conn)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# 例 :

```
public void CreateMySqlCommand()
{
    MySqlConnection conn = new MySqlConnection("server=myserver")
    string sql = "SELECT * FROM mytable";
    MySqlCommand myCommand = new MySqlCommand(sql, conn);
    myCommand.CommandType = CommandType.Text;
}
```

クラス MySqlCommand コンストラクタ 形式 4

クエリのテキスト、`MySqlConnection`、および `MySqlTransaction` で `MySqlCommand` クラスの新規インスタンスを初期化します。

パラメータ : クエリのテキスト。

パラメータ : SQL サーバのインターフェイスへの接続を表す [MySqlConnection](#)。

パラメータ : [MySqlCommand](#) が実行する [MySqlTransaction](#) 。

[MySqlCommand](#) のインスタンスが作成される際、次の赤/白のプロパティが初期値に設定されます。

プロパティ	初期値
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	connection

これらのプロパティの値は、プロパティへの別の呼び出しを通して変更することができます。

例

以下の例は [MySqlCommand](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateMySqlCommand()
    Dim conn as new MySqlConnection("server=myServer")
    conn.Open();
    Dim txn as MySqlTransaction = conn.BeginTransaction()
    Dim sql as String = "SELECT * FROM mytable"
    Dim myCommand As New MySqlCommand(sql, conn, txn)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# 例 :

```
public void CreateMySqlCommand()
{
    MySqlConnection conn = new MySqlConnection("server=myserver")
    conn.Open();
    MySqlTransaction txn = conn.BeginTransaction();
    string sql = "SELECT * FROM mytable";
    MySqlCommand myCommand = new MySqlCommand(sql, conn, txn);
    myCommand.CommandType = CommandType.Text;
}
```

ExecuteNonQuery

接続に対して SQL 文を実行し、影響される行の数を返します。

戻り値 : 影響を受ける行数

`ExecuteNonQuery` を使用して、すべてのタイプのデータベース操作を行うことができますが、返された結果セットは利用できません。ストアド プロシージャの呼び出しに使用された出力パラメータはデータを移植され、実行が完了した後で引き出すことができます。UPDATE、INSERT、および DELETE 文では、戻り値はそのコマンドに影響された行の数になります。その他のタイプのステートメントではすべて、戻り値は -1 です。

例

以下の例は [MySqlCommand](#) を作成し、`ExecuteNonQuery` を使用してそれを実行します。この例は SQL 文 (UPDATE、INSERT、または DELETE など) であるストリングや、データソースへの接続に使用するストリングに渡されます。

Visual Basic 例 :

```
Public Sub CreateMySqlCommand(myExecuteQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(myExecuteQuery, myConnection)
    myCommand.Connection.Open()
    myCommand.ExecuteNonQuery()
    myConnection.Close()
```

End Sub

C# 例 :

```
public void CreateMySqlCommand(string myExecuteQuery, MySqlConnection myConnection)
{
    MySqlCommand myCommand = new MySqlCommand(myExecuteQuery, myConnection);
    myCommand.Connection.Open();
    myCommand.ExecuteNonQuery();
    myConnection.Close();
}
```

ExecuteReader1

CommandText を **MySqlConnection** Connection に送り、**CommandBehavior** 値のひとつを使用して **MySqlDataReader** を構築します。

パラメータ : **CommandBehavior** 値のひとつ。

CommandType プロパティが **StoredProcedure** に設定される際、**CommandText** プロパティをストアドプロシージャ名に設定してください。 **ExecuteReader** を呼び出す時に、コマンドがこのストアドプロシージャを実行します。

MySqlDataReader は、大きなバイナリ値の効率的な読み取りを可能にする特殊なモードをサポートしています。詳細は、**CommandBehavior** の **SequentialAccess** 設定を参照してください。

MySqlDataReader が使用されている間、関連のある **MySqlConnection** は **MySqlDataReader** へ忙しく供給します。その状態では、**MySqlConnection** で閉じる以外の他の操作を行うことはできません。**MySqlDataReader** の **MySqlDataReader.Close** メソッドが呼び出されるまでは、その状態が続きます。**MySqlDataReader** が **CloseConnection** に設定される **CommandBehavior** で作成される場合、**MySqlDataReader** を閉じると、接続が自動的に切断されます。

注記

ExecuteReader を **SingleRow** 挙動で呼び出す時、SQL で **limit** 句を使用すると、すべての行 (提示された限界まで) がクライアントに引き出されることになりま
す。**MySqlDataReader.Read** メソッドは、最初の行の後でもまだ **false** を返しますが、データの全行をクライアントに入れると作業に影響が出ます。**limit** 句が必要でない場合は、避け
たほうがよいでしょう。

戻り値 : **MySqlDataReader** オブジェクト。

ExecuteReader の使用

CommandText を **MySqlConnection** Connection に送り、**MySqlDataReader** を構築します。

戻り値 : **MySqlDataReader** オブジェクト。

CommandType プロパティが **StoredProcedure** に設定される際、**CommandText** プロパティをストアドプロシージャ名に設定してください。 **ExecuteReader** を呼び出す時に、コマンドがこのストアドプロシージャを実行します。

MySqlDataReader が使用されている間、関連のある **MySqlConnection** は **MySqlDataReader** へ忙しく供給します。その状態では、**MySqlConnection** で閉じる以外の他の操作を行うことはできません。**MySqlDataReader** の **MySqlDataReader.Close** メソッドが呼び出されるまでは、その状態が続きます。

例

次の例は **MySqlCommand** を作成し、SQL **SELECT** 文であるストリングと、データソースへの接続に使用するストリングを渡すことによってそれを実行します。

Visual Basic 例 :

```
Public Sub CreateMySqlDataReader(mySelectQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
```

```

myReader = myCommand.ExecuteReader()
Try
While myReader.Read()
    Console.WriteLine(myReader.GetString(0))
End While
Finally
myReader.Close
myConnection.Close
End Try
End Sub

```

C# 例 :

```

public void CreateMySqlDataReader(string mySelectQuery, MySqlConnection myConnection)
{
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    try
    {
        while(myReader.Read())
        {
            Console.WriteLine(myReader.GetString(0));
        }
    }
    finally
    {
        myReader.Close();
        myConnection.Close();
    }
}

```

Prepare の使用

MySQL Server のインスタンスで、コマンドの準備されたバージョンを作成します。

プリペアド ステートメントは、MySQL バージョン 4.1 以降でのみサポートされています。それ以前のバージョンの MySQL に接続している間にプリペアド ステートメントを呼び出すことは可能ですが、そのステートメントは準備されていないステートメントと同様に実行されます。

例

次の例では、[Prepare](#) メソッドの使用法を示します。

Visual Basic 例 :

```

public sub PrepareExample()
    Dim cmd as New MySqlCommand("INSERT INTO mytable VALUES (?val)", myConnection)
    cmd.Parameters.Add( "?val", 10 )
    cmd.Prepare()
    cmd.ExecuteNonQuery()

    cmd.Parameters(0).Value = 20
    cmd.ExecuteNonQuery()
end sub

```

C# 例 :

```

private void PrepareExample()
{
    MySqlCommand cmd = new MySqlCommand("INSERT INTO mytable VALUES (?val)", myConnection);
    cmd.Parameters.Add( "?val", 10 );
    cmd.Prepare();
    cmd.ExecuteNonQuery();

    cmd.Parameters[0].Value = 20;
    cmd.ExecuteNonQuery();
}

```

ExecuteScalar

クエリを実行し、クエリが戻した結果セットの第一行の最初の列を返します。余分な列や行は無視されます。

戻り値：結果セットの第一行の最初の列、または結果セットが空な場合はヌルのリファレンス

`ExecuteScalar` メソッドを使用して、データベースから単一値（例えば集約値）を取り出します。これでは `ExecuteReader` メソッドよりも少ないコードを使用し、`MySqlDataReader` が戻したデータを使って、単一値の生成に必要な操作を行います。

典型的な `ExecuteScalar` クエリは、次の C# 例のようにフォーマットすることができます：

C# 例：

```
cmd.CommandText = "select count(*) from region";
Int32 count = (int32) cmd.ExecuteScalar();
```

例

以下の例は `MySqlCommand` を作成し、`ExecuteScalar` を使用してそれを実行します。この例は、集約結果を返す SQL 文、そしてデータソースへの接続に使用するストリングに渡されます。

Visual Basic 例：

```
Public Sub CreateMySqlCommand(myScalarQuery As String, myConnection As MySqlConnection)
    Dim myCommand As New MySqlCommand(myScalarQuery, myConnection)
    myCommand.Connection.Open()
    myCommand.ExecuteScalar()
    myConnection.Close()
End Sub
```

C# 例：

```
public void CreateMySqlCommand(string myScalarQuery, MySqlConnection myConnection)
{
    MySqlCommand myCommand = new MySqlCommand(myScalarQuery, myConnection);
    myCommand.Connection.Open();
    myCommand.ExecuteScalar();
    myConnection.Close();
}
```

C++ 例：

```
public:
    void CreateMySqlCommand(String* myScalarQuery, MySqlConnection* myConnection)
    {
        MySqlCommand* myCommand = new MySqlCommand(myScalarQuery, myConnection);
        myCommand->Connection->Open();
        myCommand->ExecuteScalar();
        myConnection->Close();
    }
```

CommandText

SQL 文を入手または設定し、データソースで実行します。

値：実行する SQL 文またはストアードプロシージャ。デフォルトは空のストリング。

`CommandType` プロパティが `StoredProcedure` に設定される際、`CommandText` プロパティをストアードプロシージャ名に設定してください。ストアードプロシージャ名に特殊な文字が含まれていると、ユーザはエスケープ文字構文を使用しなければならない場合があります。Execute メソッドのひとつを呼び出す時に、コマンドがこのストアードプロシージャを実行します。

例

以下の例は `MySqlCommand` を作成し、そのプロパティの一部を設定します。

Visual Basic 例：


```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandText = "SELECT * FROM Mytable ORDER BY id"
    myCommand.CommandType = CommandType.Text
End Sub
```

C# 例 :

```
public void CreateMySqlCommand()
{
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandText = "SELECT * FROM mytable ORDER BY id";
    myCommand.CommandType = CommandType.Text;
}
```

CommandTimeout

コマンドの実行を停止し、エラーを生成するまでの待ち時間を取得または設定します。

値 : コマンド実行までの待ち時間 (秒単位)。デフォルトは 0 秒。

MySQL は現在、保留または実行中の作業をキャンセルする方法をサポートしていません。MySQL サーバに対して発行するすべてのコマンドは、完了するか例外が発生するまで実行します。

CommandType

[CommandText](#) プロパティの解説方法を示す値を取得または設定します。

値 : [System.Data.CommandType](#) 値のひとつ。デフォルト設定では、[Text](#)。

[CommandType](#) プロパティを [StoredProcedure](#) に設定する際、[CommandText](#) プロパティをストアド プロシージャ名に設定してください。Execute メソッドのひとつを呼び出す時に、コマンドがこのストアド プロシージャを実行します。

例

以下の例は [MySqlCommand](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateMySqlCommand()
    Dim myCommand As New MySqlCommand()
    myCommand.CommandType = CommandType.Text
End Sub
```

C# 例 :

```
public void CreateMySqlCommand()
{
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandType = CommandType.Text;
}
```

Connection

[MySqlCommand](#) のこの具体例に使用される [MySQLConnection](#) を取得または設定します。

値 : データソースへの接続。デフォルト値はヌル リファレンス (Visual Basic の [Nothing](#))。

トランザクションが進行中に [Connection](#) を設定し、かつ [Transaction](#) プロパティがヌルでない場合、[InvalidOperationException](#) が生成されます。[Transaction](#) プロパティがヌルでなく、トランザクションがすでにコミットまたはロールバックされている場合は、[Transaction](#) はヌルに設定されます。

例

以下の例は [MySqlCommand](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateMySQLCommand()
    Dim mySelectQuery As String = "SELECT * FROM mytable ORDER BY id"
    Dim myConnectionString As String = "Persist Security Info=False;database=test;server=myServer"
    Dim myCommand As New MySqlCommand(mySelectQuery)
    myCommand.Connection = New MySqlConnection(myConnectionString)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# 例 :

```
public void CreateMySQLCommand()
{
    string mySelectQuery = "SELECT * FROM mytable ORDER BY id";
    string myConnectionString = "Persist Security Info=False;database=test;server=myServer";
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery);
    myCommand.Connection = new MySqlConnection(myConnectionString);
    myCommand.CommandType = CommandType.Text;
}
```

IsPrepared

ステートメントが準備されている場合に true を返します。

Parameters

[MySQLParameterCollection](#) を取得します。

値 :SQL 文もしくはストアードプロシージャのパラメータ。デフォルトは空のコレクション。

Connector/Net は名前なしのパラメータをサポートしていません。コレクションに加えられるパラメータにはすべて関連する名前が必要です。

例

次の例は [MySQLCommand](#) を作成し、そのパラメータを表示します。これを完遂するため、メソッドに [MySQLConnection](#)、SQL [SELECT](#) 文であるクエリ ストリング、そして [MySQLParameter](#) オブジェクトの配列が渡されます。

Visual Basic 例 :

```
Public Sub CreateMySQLCommand(myConnection As MySqlConnection, _
mySelectQuery As String, myParamArray() As MySQLParameter)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?age"
    myCommand.UpdatedRowSource = UpdateRowSource.Both
    myCommand.Parameters.Add(myParamArray)
    Dim j As Integer
    For j = 0 To myCommand.Parameters.Count - 1
        myCommand.Parameters.Add(myParamArray(j))
    Next j
    Dim myMessage As String = ""
    Dim i As Integer
    For i = 0 To myCommand.Parameters.Count - 1
        myMessage += myCommand.Parameters(i).ToString() & ControlChars.Cr
    Next i
    Console.WriteLine(myMessage)
End Sub
```

C# 例 :

```
public void CreateMySQLCommand(MySqlConnection myConnection, string mySelectQuery,
MySQLParameter[] myParamArray)
{
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?age";
    myCommand.Parameters.Add(myParamArray);
    for (int j=0; j<myParamArray.Length; j++)
    {
        myCommand.Parameters.Add(myParamArray[j]);
    }
    string myMessage = "";
```

```

for (int i = 0; i < myCommand.Parameters.Count; i++)
{
    myMessage += myCommand.Parameters[i].ToString() + "\n";
}
MessageBox.Show(myMessage);
}

```

Transaction

[MySQLCommand](#) が実行する中で [MySQLTransaction](#) を取得または設定します。

値 :[MySQLTransaction](#)。デフォルト値はヌル リファレンス (Visual Basic の [Nothing](#))。

すでに特定の値が設定されている場合、コマンドの実行が進行中の場合は、[Transaction](#) プロパティを設定することはできません。トランザクションのプロパティを [MySQLCommand](#) オブジェクトと同じ [MySQLConnection](#) に接続されていない [MySQLTransaction](#) に設定すると、次にステートメントの実行を試みる際に例外が投入されるでしょう。

UpdatedRowSource

[System.Data.Common.DbDataAdapter](#) の [System.Data.Common.DbDataAdapter.Update](#) メソッドで使用される時に、コマンドの結果がどのように [DataRow](#) に適用されるかを取得または設定します。

値 :[UpdateRowSource](#) 値のひとつ。

デフォルトの [System.Data.UpdateRowSource](#) 値は、コマンドが ([MySQLCommandBuilder](#) のケースのように) 自動的に生成されない限り [Both](#) になります。コマンドが自動的に生成される場合のデフォルト値は [None](#) になります。

MySQLCommandBuilder の使用

MySQL データベースに関連付けられた [DataSet](#) の変更を調整するための単一テーブル コマンドを自動的に生成します。このクラスを継承することはできません。

[MySQLDataAdapter](#) は、MySQL のインスタンスに関連付けられた [System.Data.DataSet](#) への変更を調整するのに必要な SQL 文を自動的に生成することはありません。しかし、[MySQLDataAdapter](#) の [MySQLDataAdapter.SelectCommand](#) を設定すれば、単一テーブルのアップデートのための SQL 文を自動的に生成する [MySQLCommandBuilder](#) オブジェクトを作成することができます。そうすると、設定していない追加の SQL 文は、[MySQLCommandBuilder](#) によって生成されます。

[MySQLCommandBuilder](#) は、[DataAdapter](#) プロパティを設定すると、自らを [MySQLDataAdapter.OnRowUpdating](#) [RowUpdating](#) イベントのリスナとして登録します。一度にはひとつの [MySQLDataAdapter](#) または [MySQLCommandBuilder](#) オブジェクトだけをお互いに関連付けることが可能です。

INSERT、UPDATE、もしくは DELETE 文を生成するため、[MySQLCommandBuilder](#) は [SelectCommand](#) プロパティを使用して必要なメタデータのセットを自動的に引き出します。メタデータが引き出された後に (例えば最初のアップデートの後) [SelectCommand](#) を変更する場合、[RefreshSchema](#) メソッドを呼び出してメタデータをアップデートしてください。

また、[SelectCommand](#) はプライマリ キーが一意列を少なくともひとつは戻さなければなりません。もしひとつもない場合は、[InvalidOperation](#) 例外が生成され、コマンドは生成されません。

[MySQLCommandBuilder](#) は、[SelectCommand](#) が参照する [MySQLCommand.Connection](#)、[MySQLCommand.CommandTimeout](#)、そして [MySQLCommand.Transaction](#) プロパティも使用します。これらのプロパティのどれかが改変されている場合、または [SelectCommand](#) そのものが取り替えられている場合、ユーザは [RefreshSchema](#) を呼び出してください。それを行わない場合、[MySQLDataAdapter.InsertCommand](#)、[MySQLDataAdapter.UpdateCommand](#)、そして [MySQLDataAdapter.DeleteCommand](#) プロパティはそれぞれ前値を保持します。

[Dispose](#) を呼び出すと、[MySQLCommandBuilder](#) は [MySQLDataAdapter](#) との関連を絶たれ、生成されたコマンドは使用されなくなります。

注記

[MySQLCommandBuilder](#) を MySQL 4.0 システムで使用する際は十分な注意が必要です。MySQL 4.0 での使用では、データベースとスキーマの情報はクエリ用にコネクタに提供

されません。つまり、ふたつ以上の異なるデータベースにあるふたつの同じ名前のテーブルからカラムを引き出すクエリが、例外の投入の原因になることなく正常に作動するということになります。より危険なのは、データベース X を参照しつつもデータベース Y で実行されるステートメントを選択し、そしてその両方のデータベースが似通ったレイアウトのテーブルを持つという状況です。このような状況は不要な変更や削除を引き起こす場合があります。この注意は MySQL バージョン 4.1 以降には当てはまりません。

例

次の例は [MySqlDataAdapter](#) および [MySqlConnection](#) に沿って [MySqlCommand](#) を使用し、データソースから行を選択します。この例は、初期化された [System.Data.DataSet](#)、接続ストリング、SQL [SELECT](#) 文であるクエリ ストリング、そしてデータベース テーブル名であるストリングに渡されます。そしてこの例は [MySqlCommandBuilder](#) を作成します。

Visual Basic 例：

```
Public Shared Function SelectRows(myConnection As String, mySelectQuery As String, myTableName As String) As DataSet
    Dim myConn As New MySqlConnection(myConnection)
    Dim myDataAdapter As New MySqlDataAdapter()
    myDataAdapter.SelectCommand = New MySqlCommand(mySelectQuery, myConn)
    Dim cb As SqlCommandBuilder = New MySqlCommandBuilder(myDataAdapter)
    myConn.Open()
    Dim ds As DataSet = New DataSet
    myDataAdapter.Fill(ds, myTableName)
    ' Code to modify data in DataSet here
    ' Without the MySqlCommandBuilder this line would fail.
    myDataAdapter.Update(ds, myTableName)
    myConn.Close()
End Function 'SelectRows
```

C# 例：

```
public static DataSet SelectRows(string myConnection, string mySelectQuery, string myTableName)
{
    MySqlConnection myConn = new MySqlConnection(myConnection);
    MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
    myDataAdapter.SelectCommand = new MySqlCommand(mySelectQuery, myConn);
    MySqlCommandBuilder cb = new MySqlCommandBuilder(myDataAdapter);
    myConn.Open();
    DataSet ds = new DataSet();
    myDataAdapter.Fill(ds, myTableName);
    //code to modify data in DataSet here
    //Without the MySqlCommandBuilder this line would fail
    myDataAdapter.Update(ds, myTableName);
    myConn.Close();
    return ds;
}
```

クラス MySqlCommandBuilder コンストラクタ

[MySqlCommandBuilder](#) クラスの新規インスタンスを初期化します。

クラス MySqlCommandBuilder コンストラクタ 形式 1

[MySqlCommandBuilder](#) クラスの新規インスタンスを初期化し、last one wins プロパティを設定します。

パラメータ :変更保護コードの生成には False 。それ以外は True 。

[lastOneWins](#) パラメータは、基礎となるデータの変更を確認する生成された DELETE および UPDATE コマンドに、SQL コードが含まれるべきかを指示します。[lastOneWins](#) が true である場合はこのコードは含まれず、データレコードはマルチユーザまたはマルチスレッドの環境で上書きされる場合があります。[lastOneWins](#) を false に設定するとこの確認が含まれ、知らない間に基礎になるデータレコードが変更される場合に同時実行例外が投入される原因になります。

クラス MySqlCommandBuilder コンストラクタ 形式 2

関連する [MySqlDataAdapter](#) オブジェクトで [MySqlCommandBuilder](#) クラスの新規インスタンスを初期化します。

パラメータ :使用する [MySqlDataAdapter](#) 。

[MySqlCommandBuilder](#) は自らを、このプロパティで指定された[MySqlDataAdapter](#) によって生成された [MySqlDataAdapter.RowUpdating](#) イベントのリスナとして登録します。

新規インスタンス [MySqlCommandBuilder](#) を作成する際、この [MySqlDataAdapter](#) に関連する既存の [MySqlCommandBuilder](#) がリリースされます。

クラス MySqlCommandBuilder コンストラクタ 形式 3

関連する [MySqlDataAdapter](#) オブジェクトで [MySqlCommandBuilder](#) クラスの新規インスタンスを初期化します。

パラメータ :使用する [MySqlDataAdapter](#) 。

パラメータ :変更保護コードの生成には False 。それ以外は True 。

[MySqlCommandBuilder](#) は自らを、このプロパティで指定された[MySqlDataAdapter](#) によって生成された [MySqlDataAdapter.RowUpdating](#) イベントのリスナとして登録します。

新規インスタンス [MySqlCommandBuilder](#) を作成する際、この [MySqlDataAdapter](#) に関連する既存の [MySqlCommandBuilder](#) がリリースされます。

[lastOneWins](#) パラメータは、基礎となるデータの変更を確認する生成された DELETE および UPDATE コマンドに、SQL コードが含まれるべきかを指示します。[lastOneWins](#) が true である場合はこのコードは含まれず、データレコードはマルチユーザまたはマルチスレッドの環境で上書きされる場合があります。[lastOneWins](#) を false に設定するとこの確認が含まれ、知らない間に基礎になるデータレコードが変更される場合に同時実行例外が投入される原因になります。

DataAdapter

SQL 文が自動的に生成される [MySqlDataAdapter](#) オブジェクトを取得または設定します。

値 :[MySqlDataAdapter](#)オブジェクト。

[MySqlCommandBuilder](#) は自らを、このプロパティで指定された[MySqlDataAdapter](#) によって生成された [MySqlDataAdapter.RowUpdating](#) イベントのリスナとして登録します。

新規インスタンス [MySqlCommandBuilder](#) を作成する際、この [MySqlDataAdapter](#) に関連する既存の [MySqlCommandBuilder](#) がリリースされます。

QuotePrefix

スペースや予約トークンなどの文字を名前に含む MySQL データベース オブジェクト (例えばテーブルやカラム) の指定に使用する文字や、開始文字を取得または設定します。

値 :使用する開始文字または文字。デフォルト値は ` 。

MySQL のデータベース オブジェクトは、普通の SQL ストリングなら正確な解析が不可能になるスペースなどの特殊な文字を含むことができます。[QuotePrefix](#) および [QuoteSuffix](#) プロパティを使用すると、この状況に対処する SQL コマンドを [MySqlCommandBuilder](#) が構築できるようになります。

QuoteSuffix

スペースや予約トークンなどの文字を名前に含む MySQL データベース オブジェクト (例えばテーブルやカラム) の指定に使用する文字や、開始文字を取得または設定します。

値 :使用する開始文字または文字。デフォルト値は ` 。

MySQL のデータベース オブジェクトは、普通の SQL ストリングなら正確な解析が不可能になるスペースなどの特殊な文字を含むことができます。[QuotePrefix](#) および [QuoteSuffix](#) プロパティを使用すると、この状況に対処する SQL コマンドを [MySqlCommandBuilder](#) が構築できるようになります。

DeriveParameters

GetDeleteCommand

データベースでの削除に必要な、自動的に生成された `MySqlCommand` オブジェクトを取得します。

戻り値：削除操作の処理のために生成された `MySqlCommand` オブジェクト。

`GetDeleteCommand` メソッドは実行される `MySqlCommand` を戻すので、アプリケーションはそれを情報提供やトラブルシューティングの目的に使用することができます。

また、 `GetDeleteCommand` を変更されたコマンドの基盤として使用することもできます。例えば、 `GetDeleteCommand` を呼び出して `MySqlCommand.CommandTimeout` 値を変更したとすると、それを `MySqlDataAdapter` に明示的に設定します。

最初に SQL 文が生成された後、ステートメントになんらかの変更が生じる場合はアプリケーションが `RefreshSchema` を明示的に呼び出す必要があります。さもなければ `GetDeleteCommand` は前のステートメントからの情報を使い続けることになり、それでは間違っている可能性があります。SQL 文が最初に生成されるのは、アプリケーションが `System.Data.Common.DataAdapter.Update` または `GetDeleteCommand` を呼び出す時です。

GetInsertCommand

データベースでの挿入に必要な、自動的に生成された `MySqlCommand` オブジェクトを取得します。

戻り値：挿入操作の処理のために生成された `MySqlCommand` オブジェクト。

`GetInsertCommand` メソッドは実行される `MySqlCommand` を戻すので、アプリケーションはそれを情報提供やトラブルシューティングの目的に使用することができます。

また、 `GetInsertCommand` を変更されたコマンドの基盤として使用することもできます。例えば、 `GetInsertCommand` を呼び出して `MySqlCommand.CommandTimeout` 値を変更したとすると、それを `MySqlDataAdapter` に明示的に設定します。

最初に SQL 文が生成された後、ステートメントになんらかの変更が生じる場合はアプリケーションが `RefreshSchema` を明示的に呼び出す必要があります。さもなければ `GetInsertCommand` は前のステートメントからの情報を使い続けることになり、それでは間違っている可能性があります。SQL 文が最初に生成されるのは、アプリケーションが `System.Data.Common.DataAdapter.Update` または `GetInsertCommand` を呼び出す時です。

GetUpdateCommand

データベースでの更新作業に必要な、自動的に生成された `MySqlCommand` オブジェクトを取得します。

戻り値：更新操作の処理のために生成された `MySqlCommand` オブジェクト。

`GetUpdateCommand` メソッドは実行される `MySqlCommand` を戻すので、アプリケーションはそれを情報提供やトラブルシューティングの目的に使用することができます。

また、 `GetUpdateCommand` を変更されたコマンドの基盤として使用することもできます。例えば、 `GetUpdateCommand` を呼び出して `MySqlCommand.CommandTimeout` 値を変更したとすると、それを `MySqlDataAdapter` に明示的に設定します。

最初に SQL 文が生成された後、ステートメントになんらかの変更が生じる場合はアプリケーションが `RefreshSchema` を明示的に呼び出す必要があります。さもなければ `GetUpdateCommand` は前のステートメントからの情報を使い続けることになり、それでは間違っている可能性があります。SQL 文が最初に生成されるのは、アプリケーションが `System.Data.Common.DataAdapter.Update` または `GetUpdateCommand` を呼び出す時です。

RefreshSchema

INSERT、UPDATE、または DELETE 文の生成に使用されるデータベーススキーマ情報をリフレッシュします。

`MySqlCommandBuilder` に関連する SELECT 文が変わるたびに、アプリケーションは `RefreshSchema` を呼び出します。

アプリケーションは、 `MySqlDataAdapter` の `MySqlCommandBuilder` 値が変わるたびに、 `RefreshSchema` を呼び出します。

MySqlConnection の使用

MySQL サーバ データベースへのオープン接続を表します。このクラスを継承することはできません。

MySqlConnection オブジェクトは、MySQL Server データソースへのセッションを示します。**MySqlConnection** のインスタンスを作成する際、すべてのプロパティはそれらの初期値に設定されます。それらの値のリストは、**MySqlConnection** コンストラクタをご覧ください。

MySqlConnection が範囲から出ると閉じません。そのため、**MySqlConnection.Close** または **MySqlConnection.Dispose** を呼び出すことによって、明示的に接続を閉じる必要があります。

例

以下は、**MySqlCommand** と **MySqlConnection** を作成する例です。**MySqlConnection** が開かれ、**MySqlCommand** の **MySqlCommand.Connection** として設定されます。この例はその後 **MySqlCommand.ExecuteNonQuery** を呼び出し、接続を閉じます。これを完遂するため、**ExecuteNonQuery** は SQL INSERT 文である接続ストリングとクエリ ストリングに渡されます。

Visual Basic 例：

```
Public Sub InsertRow(myConnectionString As String)
    ' If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

C# 例：

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionString);
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

クラス MySqlCommandBuilder コンストラクタ (デフォルト)

MySqlConnection クラスの新規インスタンスを初期化します。

MySqlConnection の新規インスタンスが作成される時、赤/白のプロパティは、**ConnectionString** の関連するキーワードを使って特別に設定されない限り、次の初期値に設定されます。

プロパティ	初期値
ConnectionString	empty string ("")
ConnectionTimeout	15
Database	empty string ("")
DataSource	empty string ("")

ServerVersion	empty string ("")

これらのプロパティの値は、[ConnectionString](#) プロパティを使うことによつてのみ変更できます。

例

MySqlConnection のオーバーロード メソッド

[MySqlConnection](#) クラスの新規インスタンスを初期化します。

クラス MySqlConnection コンストラクタ 形式 1

接続ストリングを含むストリングが与えられた時に、[MySqlConnection](#) クラスの新規インスタンスを初期化します。

[MySqlConnection](#) の新規インスタンスが作成される時、赤/白のプロパティは、[ConnectionString](#) の関連するキーワードを使って特別に設定されない限り、次の初期値に設定されます。

プロパティ	初期値
ConnectionString	empty string ("")
ConnectionTimeout	15
Database	empty string ("")
DataSource	empty string ("")
ServerVersion	empty string ("")

これらのプロパティの値は、[ConnectionString](#) プロパティを使うことによつてのみ変更できます。

例

パラメータ : MySQL データベースを開くのに使用される接続プロパティ。

Open

ConnectionString によって指定されたプロパティ設定で、データベース接続を開きます。

例外 : データソースかサーバの指定なしに接続を開くことができない。

例外 : 接続を開く途中で発生した接続レベル エラー。

[MySqlConnection](#) は、接続プールに利用可能なオープン接続があれば引き出します。もしくは、MySQL のインスタンスに新しい接続を確立します。

例

以下の例は [MySqlConnection](#) を作成して開き、そのプロパティの一部を表示、そして接続を閉じます。

Visual Basic 例 :

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# 例 :

```
public void CreateMySqlConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

```
}

```

Database

使用中のデータベース、または接続が開いた後に使用されるデータベースの名前を取得します。

戻り値 : 使用中のデータベース名、または接続が開いた後に使用されるデータベースの名前。デフォルト値は空のストリング。

Database プロパティは動的に更新しません。SQL 文を使って使用中のデータベースを変更すると、このプロパティは誤った値を反映することがあります。**ChangeDatabase** メソッドを使って使用中のデータベースを変更すれば、このプロパティは新しいデータベースを反映するよう更新されます。

例

以下の例は **MySqlConnection** を作成し、その読み取り専用プロパティの一部を表示します。

Visual Basic 例 :

```
Public Sub CreateMySqlConnection()
    Dim myConnString As String = _
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass"
    Dim myConnection As New MySqlConnection( myConnString )
    myConnection.Open()
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
        + ControlChars.NewLine + "Database: " + myConnection.Database )
    myConnection.ChangeDatabase( "test2" )
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.NewLine + "Database: " + myConnection.Database )
    myConnection.Close()
End Sub

```

C# 例 :

```
public void CreateMySqlConnection()
{
    string myConnString =
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass";
    MySqlConnection myConnection = new MySqlConnection( myConnString );
    myConnection.Open();
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion
        + "\nDatabase: " + myConnection.Database );
    myConnection.ChangeDatabase( "test2" );
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
        + "\nDatabase: " + myConnection.Database );
    myConnection.Close();
}

```

State

接続の状態を取得します。

戻り値 : **System.Data.ConnectionState** 値のビットごとの組み合わせ。デフォルトは **Closed**。

許可されている状態の変更は以下 :

- 接続オブジェクトの **Open** メソッドを使用する、**Closed** から **Open** まで。
- 接続オブジェクトの **Close** メソッドか **Dispose** メソッドを使用する、**Open** から **Closed** まで。

例

以下の例は **MySqlConnection** を作成して開き、そのプロパティの一部を表示、そして接続を閉じます。

Visual Basic 例 :

```
Public Sub CreateMySqlConnection(myConnString As String)

```

```
Dim myConnection As New MySqlConnection(myConnString)
myConnection.Open()
MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
+ ControlChars.Cr + "State: " + myConnection.State.ToString())
myConnection.Close()
End Sub
```

C# 例 :

```
public void CreateMySqlConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

ServerVersion

クライアントが接続された MySQL サーバのバージョンを含むストリングを取得します。

戻り値 : MySQL のインスタンスのバージョン。

例外 : 接続が開いていません。

例

以下の例は [MySqlConnection](#) を作成して開き、そのプロパティの一部を表示、そして接続を閉じます。

Visual Basic 例 :

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
+ ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# 例 :

```
public void CreateMySqlConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

Close

データベースへの接続を閉じます。この方法は、どんなオープン接続を閉じる時にも適しています。

[Close](#) メソッドはすべての保留トランザクションをロールバックします。そして接続を接続プールにリリース、または接続プールが無効になっている場合は接続を閉じます。

アプリケーションは一度以上、[Close](#) を呼び出すことができます。例外は生成されません。

例

以下の例は [MySqlConnection](#) を作成して開き、そのプロパティの一部を表示、そして接続を閉じます。

Visual Basic 例 :

```
Public Sub CreateMySqlConnection(myConnString As String)
```

```

Dim myConnection As New MySqlConnection(myConnString)
myConnection.Open()
MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
+ ControlChars.Cr + "State: " + myConnection.State.ToString())
myConnection.Close()
End Sub

```

C# 例 :

```

public void CreateMySqlConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}

```

CreateCommand

[MySqlConnection](#) に関連する [MySqlCommand](#) オブジェクトを作成し、戻します。

戻り値 : [MySqlCommand](#) オブジェクト。

BeginTransaction

データベースのトランザクションを開始します。

戻り値 : 新しいトランザクションを表すオブジェクト。

例外 : 並列トランザクションはサポートされていません。

このコマンドは、MySQL BEGIN TRANSACTION コマンドに相当します。

トランザクションは、[MySqlTransaction.Commit](#) または [MySqlTransaction.Rollback](#) メソッドを使用して、明示的にコミットするかロールバックする必要があります。

注記

隔離レベルを指定しない場合、デフォルトの隔離レベルが使用されます。[BeginTransaction](#) メソッドで隔離レベルを指定するには、[iso](#) パラメータを受け取るオーバーロードを使用します。

例

次は、[MySqlConnection](#) と [MySqlTransaction](#) を作成する例です。[BeginTransaction](#)、[MySqlTransaction.Commit](#)、[MySqlTransaction.Rollback](#) の各メソッドの使い方も示します。

Visual Basic 例 :

```

Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (101, 'Description')"

```

```

myCommand.ExecuteNonQuery()
myTrans.Commit()
Console.WriteLine("Both records are written to database.")
Catch e As Exception
Try
    myTrans.Rollback()
Catch ex As MySqlException
    If Not myTrans.Connection Is Nothing Then
        Console.WriteLine("An exception of type " + ex.GetType().ToString() + _
            " was encountered while attempting to roll back the transaction.")
    End If
End Try

Console.WriteLine("An exception of type " + e.GetType().ToString() + _
    "was encountered while inserting the data.")
Console.WriteLine("Neither record was written to database.")
Finally
    myConnection.Close()
End Try
End Sub

```

C# 例 :

```

public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "insert into Test (id, desc) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "insert into Test (id, desc) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (SqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

BeginTransaction1

隔離レベルを指定して、データベース トランザクションを開始します。

パラメータ：トランザクションを実行する分離レベル。

戻り値：新しいトランザクションを表すオブジェクト。

例外：並列トランザクションはサポートされていません。

このコマンドは、MySQL BEGIN TRANSACTION コマンドに相当します。

トランザクションは、[MySqlConnection.Commit](#) または [MySqlConnection.Rollback](#) メソッドを使用して、明示的にコミットするかロールバックする必要があります。

注記

分離レベルを指定しない場合、デフォルトの分離レベルが使用されます。[BeginTransaction](#) メソッドで分離レベルを指定するには、[iso](#) パラメータを受け取るオーバーロードを使用します。

例

次は、[MySqlConnection](#) と [MySqlConnection](#) を作成する例です。[BeginTransaction](#)、[MySqlConnection.Commit](#)、[MySqlConnection.Rollback](#) の各メソッドの使い方も示します。

Visual Basic 例：

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Both records are written to database.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " + ex.GetType().ToString() + _
                    " was encountered while attempting to roll back the transaction.")
            End If
        End Try

        Console.WriteLine("An exception of type " + e.GetType().ToString() + _
            " was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub
```

C# 例：

```
public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
```

```

MySQLTransaction myTrans;
// Start a local transaction
myTrans = myConnection.BeginTransaction();
// Must assign both transaction object and connection
// to Command object for a pending local transaction
myCommand.Connection = myConnection;
myCommand.Transaction = myTrans;
try
{
    myCommand.CommandText = "insert into Test (id, desc) VALUES (100, 'Description')";
    myCommand.ExecuteNonQuery();
    myCommand.CommandText = "insert into Test (id, desc) VALUES (101, 'Description')";
    myCommand.ExecuteNonQuery();
    myTrans.Commit();
    Console.WriteLine("Both records are written to database.");
}
catch(Exception e)
{
    try
    {
        myTrans.Rollback();
    }
    catch (SqlException ex)
    {
        if (myTrans.Connection != null)
        {
            Console.WriteLine("An exception of type " + ex.GetType() +
                " was encountered while attempting to roll back the transaction.");
        }
    }
}

Console.WriteLine("An exception of type " + e.GetType() +
    " was encountered while inserting the data.");
Console.WriteLine("Neither record was written to database.");
}
finally
{
    myConnection.Close();
}
}

```

ChangeDatabase

開いている MySqlConnection の現在のデータベースを変更します。

パラメータ : 使用するデータベースの名前。

database パラメータで提供される値は、有効なデータベース名である必要があります。 **database** パラメータは、null 値、空のストリング、または空白文字だけの文字列にはできません。

MySQL に対して接続プールを使用する場合、接続を閉じると、その接続は接続プールに返されます。次回、プールから接続を取り出した時に、接続のリセット要求が、ユーザーが操作を行う前に実行されます。

例外 : データベース名が有効ではありません。

例外 : 接続が開いていません。

例外 : データベースを変更できません。

例

以下の例は MySqlConnection を作成し、その読み取り専用プロパティの一部を表示します。

Visual Basic 例 :

```

Public Sub CreateMySQLConnection()
    Dim myConnString As String = _
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass"
    Dim myConnection As New MySqlConnection( myConnString )
    myConnection.Open()

```

```

MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
+ ControlChars.NewLine + "Database: " + myConnection.Database )
myConnection.ChangeDatabase( "test2" )
MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
+ ControlChars.NewLine + "Database: " + myConnection.Database )
myConnection.Close()
End Sub

```

C# 例 :

```

public void CreateMySQLConnection()
{
    string myConnString =
        "Persist Security Info=False;database=test;server=localhost;user id=joeuser;pwd=pass";
    MySqlConnection myConnection = new MySqlConnection( myConnString );
    myConnection.Open();
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion
+ "\nDatabase: " + myConnection.Database );
    myConnection.ChangeDatabase( "test2" );
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
+ "\nDatabase: " + myConnection.Database );
    myConnection.Close();
}

```

StateChange

接続の状態が変更した時に発生します。

[State](#) が closed から open に、または open から closed に変わるたびに、[StateChange](#) イベントが発生します。[StateChange](#) は [MySqlConnection](#) トランザクションの直後に実行します。

[StateChange](#) イベント内でイベント ハンドラが例外を投入すると、その例外は、[Open](#) メソッドまたは [Close](#) メソッドの呼び出し元に通知されます。

[StateChange](#) イベントは、[Close](#) または [Dispose](#) を明示的に呼び出さない限り発生しません。

イベント ハンドラは、このイベントに関連するデータを含む [System.Data.StateChangeEventArgs](#) タイプの引数を受け取ります。次の [StateChangeEventArgs](#) プロパティは、このイベントの固有の情報を提供します。

プロパティ	解説
System.Data.StateChangeEventArgs.CurrentState	接続の新しい状態を取得します。イベントが発生した時には、接続オブジェクトが既に新しい状態になっています。
System.Data.StateChangeEventArgs.OriginalState	接続の元の状態を取得します。

InfoMessage

MySQL が、コマンドやクエリを実行した結果として警告メッセージを返す時に発生します。

ConnectionTimeout

接続の確立の試行を開始してから、その試行を終了してエラーを生成するまでの待機時間を取得または設定します。

例外 : 設定値が 0 未満。

0 の値は限界がないことを示し、[MySqlConnection.ConnectionString](#) では接続の試行が永久に待機するため、避けるべきでしょう。

例

以下の例は MySqlConnection を作成し、そのプロパティの一部を接続ストリングに設定します。

Visual Basic 例 :

```
Public Sub CreateSqlConnection()
    Dim myConnection As New MySqlConnection()
    myConnection.ConnectionString = "Persist Security Info=False;Username=user;Password=pass;database=test1;server=localhost;Connect Timeout=30"
    myConnection.Open()
End Sub
```

C# 例 :

```
public void CreateSqlConnection()
{
    MySqlConnection myConnection = new MySqlConnection();
    myConnection.ConnectionString = "Persist Security Info=False;Username=user;»
        Password=pass;database=test1;server=localhost;Connect Timeout=30";
    myConnection.Open();
}
```

ConnectionString

MySQL Server データベースへの接続に使用されるストリングを取得または設定します。

戻った [ConnectionString](#) は元々設定されたものとやや異なるかもしれませんが、キーワード/値のペアの点では同じになります。セキュリティ情報は、Persist Security Info 値が true に設定されない限り含まれません。

[ConnectionString](#) プロパティを使用して、データベースに接続することができます。次の例は一般的な接続ストリングを示したものです。

```
"Persist Security Info=False;database=MyDB;»
server=MySqlServer;user id=myUser;Password=myPass"
```

[ConnectionString](#) プロパティを設定できるのは、接続が閉じている時だけです。接続ストリング値の多くには、対応する読み取り専用プロパティがあります。接続ストリングを設定すると、エラーが検出された場合を除いて、これらのプロパティがすべて更新されます。エラーが発生した場合は、いずれのプロパティも更新されません。[MySqlConnection](#) プロパティは、[ConnectionString](#) に含まれている設定だけを返します。

ローカル コンピュータに接続するには、サーバとして "localhost" を指定します。サーバを指定しないと、localhost がとられます。

閉じた接続に対して [ConnectionString](#) をリセットすると、パスワードを含むすべての接続ストリング値 (および関連プロパティ) がリセットされます。例えば、Database= MyDb を含む接続ストリングを設定した後で、その接続ストリングを Data Source=myserver;User Id=myUser;Password=myPass にリセットすると、[MySqlConnection.Database](#) プロパティは MyDb に設定されなくなります。

接続ストリングは、設定した直後に解析されます。解析中にシンタックスのエラーが検出された場合は、[ArgumentException](#) などの、ランタイム例外が生成されます。その他のエラーは、接続を開くときにだけ検出されます。

接続ストリングの基本的なフォーマットは、一連のキーワード/値のペアをセミコロンで区切った形となります。等号記号 (=) は、各キーワードとその値を結合します。オプションの設定値に関する追記は以下：

- セミコロン、単一引用符文字または二重引用符文字を含む値を指定するには、値を二重引用符で囲む必要があります。セミコロンと二重引用符の両方が値に含まれている場合は、単一引用符で値を囲むことができます。単一引用符は、値が二重引用符文字で始まる場合にも使用できます。逆に、値が単一引用符で始まる場合は、二重引用符を使用することができます。単一引用符と二重引用符の両方が値に含まれている場合、値を囲むために使用される引用符文字は、値の中で発生するたびに 2 つずつ使用する必要があります。
- ストリング値の前端または末尾の空白を含めるには、単一引用符または二重引用符で値を囲む必要があります。整数値、ブール変数、列挙値の先頭または末尾の空白は、引用符に囲まれていても無視されます。ただし、ストリングリテラル キーワードやストリング値内の空白は保存されます。.NET Framework Version 1.1 を使用の場合、単一引用符または二重引用符は、引用符が値の先頭または末尾の文字でない限り、区切り文字を付けずに使用できます (たとえば、Data Source= my'Server または DData Source= my"Server)。
- キーワードまたは値の中で等号記号 (=) を使用する場合は、先に等号記号をもうひとつ入れる必要があります。例えば、次のような接続ストリングでは、

```
"key==word=value"
```

"key=word" で値は "value" です。

- キーワード = 値のペアで特定のキーワードが接続ストリングの中に複数回発生する場合、最後に発生するものが値セットで使用されます。
- キーワードの大文字と小文字は区別されません。

次の表は、[ConnectionString](#) 内のキーワード値として有効な名前の一覧です。

名前	デフォルト	解説
Connect Timeout , Connection Timeout	15	接続の試行を終了し、エラーを生成するまで、サーバへの接続を待機する時間 (秒単位)。
Host , Server , Data Source , DataSource , Address , Addr , Network Address	localhost	接続する SQL Server のインスタンスの名前またはネットワークアドレス。複数のホストを、& で区切って指定することができます。この方法は、複数の MySQL サーバが複製のために構成されており、接続するサーバの正確性を特に求めている場合にも利用できます。データベースへの書き込みを同期するために、プロバイダは試行しないので、このオプションを使用する際は注意が必要になります。Mono を使用した Unix 環境においては、これは MySQL ソケット ファイル名への完全修飾パスになります。この構成では、Unix ソケットが TCP/IP ソケットの代わりに使用されます。現時点ではシングル ソケット名のみしか与えられないので、複製された環境で Unix ソケットを使用して MySQL にアクセスすることは現在サポートされていません。
Ignore Prepare	true	true の場合、 MySqlCommand.Prepare() へのすべての呼び出しを無視するようプロバイダに指示します。このオプションは、サーバ側のプリペアド ステートメントと使用する時、ステートメントの破損をとまなう発行を防ぐためのものです。サーバ側のプリペアド ステートメントを使用したい場合は、このオプションを false に設定してください。このオプションは、Connector/NET 5.0.3 および Connector/NET 1.0.9 に加えられています。
Port	3306	MySQL が接続の監視に使用するポート。この値を -1 に指定して名前付きパイプ接続 (Windows のみ) を使用します。Unix ソケットが使用されている場合、この値は無視されます。
Protocol	socket	サーバへ繋げる接続のタイプを指定。値は : MySQL 共有メモリを使用する、Unix ソケット接続メモリの名前付きパイプ コネクション unix のソケット接続パイプのソケットまたは tcp。
CharSet , Character Set		サーバに送られたすべてのクエリのエンコードに使用されるべき文字セットを指定。結果セットはいつもと同じく、戻されたデータの文字セットに返されます。
Logging	false	true の場合は、各種情報が構成されたどの TraceListeners にもアウトプットされます。
Allow Batch	true	true の場合は、複数の SQL 文を一度のコマンド実行で送信できます。注 : MySQL 4.1.1 からは、バッチ文をサーバ定義分離文字で分ける必要があります。それ以前のバージョンの MySQL に送られたコマンドは、';' で分けてください。
Encrypt	false	Connector/NET 5.0.3 以降では、true の場合は、サーバに証明書がインストールされていれば、クライアントとサーバの間で送信されるすべてのデータに SSL 暗号化を使用します。認識される値は、true、false、yes、および no です。5.0.3 より前のバージョンでは、このオプションは効果がありません。
Initial Catalog , Database	mysql	最初に使用するデータベース名。

Password, pwd		使用されているMySQL アカウントのパスワード。
Persist Security Info	false	false または no (強く推奨) に設定する時、パスワードのようなセキュリティにかかわる情報は、コネクションが開いている場合、またはオープン状態にあった場合は、接続の一部としては戻されません。接続ストリングをリセットすると、パスワードを含むすべてのストリング値がリセットされます。認識される値は、 true 、 false 、 yes 、および no です。
User Id, Username, Uid, User name		使用されている MySQL ログイン アカウント。
Shared Memory Name	MYSQL	接続プロトコルがメモリに設定される場合に、通信に使用する共有メモリ オブジェクトの名前。
Allow Zero Datetime	false	True では、 <code>MySqlDataReader.GetValue()</code> が、不正値を持つデータや日付時刻カラムに対して <code>MySqlDateTime</code> を返します。False は <code>System.DateTime</code> オブジェクトが不正値に返され、不正値に例外が投入されます。
Convert Zero Datetime	false	True では、 <code>MySqlDataReader.GetValue()</code> および <code>MySqlDataReader.GetDateTime()</code> が、不正値を持つデータまたは日付時刻カラムに対して <code>DateTime.MinValue</code> を返します。
Old Syntax, OldSyntax	false	'@' シンボルをパラメータ マーカとして使用できます。詳細は <code>MySqlCommand</code> を参照。これは、互換性のためのみに使用されます。すべての将来コードは、新しい '?' パラメータ マーカを使用するように書かれる必要があります。
Pipe Name, Pipe	mysql	名前付きパイプの名前に設定される場合、 <code>MySqlConnection</code> はその名前付きパイプ上で MySQL への接続を試行します。この設定は Windows のプラットフォームのみに適用します。
Procedure Cache	25	ストアド プロシージャ キャッシュのサイズを設定します。デフォルトでは、Connector/NET が、最後に使用された約 25 のストアド プロシージャのメタデータ (インプット/アウトプットデータ型) を保存します。ストアド プロシージャ キャッシュを無効にするには、値をゼロ (0) に設定します。このオプションは、Connector/NET 5.0.2 および Connector/NET 1.0.9 に加えられています。
Use Procedure Bodies	true	このオプションを false に設定すると、データベースに接続するユーザが、 <code>mysql.proc</code> (ストアド プロシージャ) テーブルに対する SELECT 権限を持たないということになります。 false に設定する時、Connector/NET はプロシージャが呼び出される際に、この情報に頼りません。Connector/NET はこの情報を定義することができないので、すべてのパラメータのタイプを呼び出しの前に明示的に設定し、パラメータをプロシージャ定義に現れる順にコマンドに加える必要があります。このオプションは、Connector/NET 5.0.4 および Connector/NET 1.0.10 に加えられています。

次の表は、`ConnectionString` 内のプーリング値として有効な名前の一覧です。接続プールの詳細は、MySQL Data Provider の接続プールをご覧ください。

名前	デフォルト	解説
Connection Lifetime	0	接続がプールに戻された時、その作成時が現時刻と比べられ、時間間隔 (秒単位) が Connection Lifetime で指定された値を越えている場合は、接続が破棄されます。これは、クラスター構成で、起動中のサーバとオンラインに上がったばかりのサーバの間で、ロードのバランスを取るのにも使用できます。値をゼロ (0) にすると、プールされた接続の接続タイムアウトが最長になります。
Max Pool Size	100	プールできる接続の最大数。

Min Pool Size	0	プールできる接続の最小数。
Pooling	true	true の場合、 MySQLConnection オブジェクトが適切なプールから引き出されます。または必要に応じて作成され、適切なプールに追加されます。認識される値は、true、false、yes、および no です。
Reset Pooled Connections, ResetConnections, ResetPooledConnections	true	ping とリセットが、プールされた接続が戻る前にサーバに送信されるべきか特定します。リセットによって接続がより速く開くことはなく、また一時テーブルなどのセッション アイテムが消去されることもありません。
Cache Server Configuration, CacheServerConfiguration, CacheServerConfig	false	サーバ変数が、プールされた接続が戻る時に更新されるべきか特定します。これを構成すると、接続がより速く開きますが、同時に他の接続によるサーバの変更を捉えられなくなります。

プール変数を必要とするキーワードまたは接続プーリング値を設定する時、'true' の代わりに 'yes'、'false' の代わりに 'no' を使用することができます。

注記 MySQL Data Provider は、MySQL との通信にネイティブ ソケット プロトコルを使用します。したがって、MySQL に接続する際、ODBC レイヤーを追加しないため、ODBC データソース名 (DSN) の使用はサポートしていません。

注意 このリリースでは、ユーザの入力をもとに接続ストリングを構築している時、アプリケーションには注意が必要です (例えば、ユーザ ID とパスワード情報をダイアログ ボックスから引き出す時や、接続ストリングに追加する時など)。アプリケーションは、ユーザがこれらの値に余分の接続ストリング パラメータを埋め込まないようにする必要があります (例えば、他のデータベースを付加しようとする場合、"validpassword;database=somedb" のようなパスワードを入力する)。

例

以下の例は [MySQLConnection](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateConnection()
    Dim myConnection As New MySqlConnection()
    myConnection.ConnectionString = "Persist Security Info=False;database=myDB;server=myHost;Connect Timeout=30;user id=myUser; pwd=myPass"
    myConnection.Open()
End Sub 'CreateConnection
```

C# 例 :

```
public void CreateConnection()
{
    MySqlConnection myConnection = new MySqlConnection();
    myConnection.ConnectionString = "Persist Security Info=False;database=myDB;server=myHost;Connect Timeout=30;user id=myUser; pwd=myPass";
    myConnection.Open();
}
```

例

次は、[MySQLConnection](#) を Mono をインストールした Unix 環境で作成する例です。この例では、MySQL ソケット ファイル名は "/var/lib/mysql/mysql.sock" です。実際のファイル名は、それぞれの MySQL 構成によって違います。

Visual Basic 例 :

```
Public Sub CreateConnection()
    Dim myConnection As New MySqlConnection()
    myConnection.ConnectionString = "database=myDB;server=/var/lib/mysql/mysql.sock;user id=myUser; pwd=myPass"
    myConnection.Open()
End Sub 'CreateConnection
```

C# 例 :

```
public void CreateConnection()
{
    MySqlConnection myConnection = new MySqlConnection();
    myConnection.ConnectionString = "database=myDB;server=/var/lib/mysql/mysql.sock;user id=myUser; pwd=myPass";
    myConnection.Open();
}
```

MySqlDataAdapter の使用

データセットを満たし、MySQL データベースの更新に使用されるデータ コマンドのセットとデータベース接続を表します。このクラスを継承することはできません。

[MySQLDataAdapter](#) は、データの引き出しと保存のため、[System.Data.DataSet](#) と MySQL 間のブリッジの役割を果たします。[MySQLDataAdapter](#) は、データソースに対して適切な SQL 文を使用し、[DataSet](#) 内のデータをデータソースのデータと一致するように変える [DbDataAdapter.Fill](#) と、データソースのデータを [DataSet](#) のデータと一致するように変える [DbDataAdapter.Update](#) をマッピングすることによってこのブリッジを供給します。

[MySQLDataAdapter](#) が [DataSet](#) をファイルする時、戻されたデータに必要なテーブルとカラムがまだない場合は、それらを作成します。ただし、プライマリ キー情報は、[System.Data.MissingSchemaAction](#) プロパティが [System.Data.MissingSchemaAction.AddWithKey](#) に設定にされていない限り、明示的に作成されたスキーマには含まれません。また、[MySQLDataAdapter](#) が、プライマリ キー情報を含む [DataSet](#) のスキーマを、[System.Data.Common.DbDataAdapter.FillSchema](#) を使用してデータとファイルする前に作成することもあります。

[MySQLDataAdapter](#) は、[MySqlConnection](#) と [MySqlCommand](#) の関連付けに使用され、MySQL データベースに接続する際のパフォーマンスを向上します。

[MySQLDataAdapter](#) はまた、データのロードおよび更新を促進する [MySQLDataAdapter.SelectCommand](#)、[MySQLDataAdapter.InsertCommand](#)、[MySQLDataAdapter.DeleteCommand](#)、[MySQLDataAdapter.UpdateCommand](#)、そして [DataAdapter.TableMappings](#) を含みます。

[MySQLDataAdapter](#) のインスタンスが作成される際、赤/白のプロパティが初期値に設定されます。それらの値のリストは、[MySQLDataAdapter](#) コンストラクタをご覧ください。

注記

.NET 1.0 および 1.1 での [DataColumn](#) クラスは、UInt16、UInt32、または UInt64 タイプのカラムが自動インクリメント カラムなるのを認めないので注意してください。MySQL で自動インクリメント カラムを使用したい場合は、符号付き整数カラムの使用を考慮してください。

例

The following example creates a [MySqlCommand](#) and a [MySqlConnection](#). The [MySqlConnection](#) is opened and set as the [MySqlCommand.Connection](#) for the [MySqlCommand](#). The example then calls [MySqlCommand.ExecuteNonQuery](#), and closes the connection. To accomplish this, the [ExecuteNonQuery](#) is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic 例：

```
Public Function SelectRows(dataSet As DataSet, connection As String, query As String) As DataSet
    Dim conn As New MySqlConnection(connection)
    Dim adapter As New MySqlDataAdapter()
    adapter.SelectCommand = new MySqlCommand(query, conn)
    adapter.Fill(dataSet)
    Return dataSet
End Function
```

C# 例：

```
public DataSet SelectRows(DataSet dataset,string connection,string query)
{
    MySqlConnection conn = new MySqlConnection(connection);
    MySqlDataAdapter adapter = new MySqlDataAdapter();
```

```

adapter.SelectCommand = new MySqlCommand(query, conn);
adapter.Fill(dataset);
return dataset;
}

```

クラス MySqlConnection コンストラクタ

MySqlConnection のオーバーロード メソッド

MySqlConnection クラスの新規インスタンスを初期化します。

[MySqlConnection](#) のインスタンスが作成される際、赤/白のプロパティが以下の初期値に設定されます。

プロパティ	初期値
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

これらのプロパティの値は、プロパティへの別の呼び出しを通じて変更することができます。

例

以下の例は [MySqlConnection](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```

Public Sub CreateSqlConnection()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
        "database=test")
    Dim da As MySqlConnectionAdapter = New MySqlConnectionAdapter
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.SelectCommand = New MySqlCommand("SELECT id, name FROM mytable", conn)
    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub

```

C# 例 :

```

public static void CreateSqlConnection()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlConnectionAdapter da = new MySqlConnectionAdapter();
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.SelectCommand = new MySqlCommand("SELECT id, name FROM mytable", conn);
    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}

```

}

クラス MySqlConnection コンストラクタ 形式 1

関連する [MySqlCommand](#) オブジェクトで、[MySqlConnection](#) クラスの新規インスタンスを [SelectCommand](#) プロパティとして初期化します。

パラメータ : [SELECT](#) 文、またはストアドプロシージャである [MySqlCommand](#) と、[MySqlConnection](#) の [SelectCommand](#) プロパティとして設定された [MySqlCommand](#)。

[MySqlConnection](#) のインスタンスが作成される際、赤/白のプロパティが以下の初期値に設定されます。

プロパティ	初期値
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

これらのプロパティの値は、プロパティへの別の呼び出しを通じて変更することができます。

[SelectCommand](#) (または他のコマンド プロパティ) が作成済みの [SelectCommand](#) に割り当てられた場合、[MySqlCommand](#) のクローンは生成されません。[SelectCommand](#) は、作成済みの [MySqlCommand](#) オブジェクトへの参照を維持します。

例

以下の例は [MySqlConnection](#) を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```
Public Sub CreateSqlConnection()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
        "database=test")
    Dim cmd As MySqlCommand("SELECT id, name FROM mytable", conn)
    Dim da As MySqlConnectionAdapter = New MySqlConnectionAdapter(cmd)
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub
```

C# 例 :

```
public static void CreateSqlConnection()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlCommand cmd = new MySqlCommand("SELECT id, name FROM mytable", conn);
    MySqlConnectionAdapter da = new MySqlConnectionAdapter(cmd);
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
```

```

da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}

```

クラス MySqlConnection コンストラクタ 形式 2

SelectCommand と **MySqlConnection** オブジェクトで、**MySqlDataAdapter** クラスの新規インスタンスを初期化します。

パラメータ **MySqlDataAdapter** の **SelectCommand** プロパティによって使用される、SQL **SELECT** 文またはストアドプロシージャである **String**。

パラメータ 接続を表す **MySqlConnection**。

この **MySqlDataAdapter** の実装は、**MySqlConnection** を、まだ開いていなければ開閉します。これは、ふたつ以上の **MySqlDataAdapter** オブジェクトに対して **DbDataAdapter.Fill** を呼び出す必要のあるアプリケーションでも使用できます。**MySqlConnection** がすでに開いている場合、閉じるには **MySqlConnection.Close** または **MySqlConnection.Dispose** を明示的に呼び出す必要があります。

MySqlDataAdapter のインスタンスが作成される際、赤/白のプロパティが以下の初期値に設定されます。

プロパティ	初期値
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

これらのプロパティの値は、プロパティへの別の呼び出しを通じて変更することができます。

例

以下の例は **MySqlDataAdapter** を作成し、そのプロパティの一部を設定します。

Visual Basic 例：

```

Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=localhost;" & _
        "database=test")
    Dim da As MySqlDataAdapter = New MySqlDataAdapter("SELECT id, name FROM mytable", conn)
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub

```

C# 例：

```

public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;database=test");
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT id, name FROM mytable", conn);
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
}

```

```

da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");

da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}

```

クラス MySqlConnection コンストラクタ 形式 3

SelectCommand と接続ストリングで、**MySqlConnection** クラスの新規インスタンスを初期化します。

パラメータ :**MySqlConnection** の **SelectCommand** プロパティによって使用される、SQL **SELECT** 文またはストアドプロシージャである **string**。

パラメータ : 接続ストリング

MySqlConnection のインスタンスが作成される際、赤/白のプロパティが以下の初期値に設定されます。

プロパティ	初期値
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

これらのプロパティの値は、プロパティへの別の呼び出しを通じて変更することができます。

例

以下の例は **MySqlConnection** を作成し、そのプロパティの一部を設定します。

Visual Basic 例 :

```

Public Sub CreateSqlConnection()
    Dim da As MySqlConnection = New MySqlConnection("SELECT id, name FROM mytable", "Data Source=localhost;database=test")
    Dim conn As MySqlConnection = da.SelectCommand.Connection
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, name) " & _
        "VALUES (?id, ?name)", conn)
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id, name=?name " & _
        "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name")
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original
End Sub

```

C# 例 :

```

public static void CreateSqlConnection()
{
    MySqlConnection da = new MySqlConnection("SELECT id, name FROM mytable", "Data Source=localhost;database=test");
    MySqlConnection conn = da.SelectCommand.Connection;
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name) " +
        "VALUES (?id, ?name)", conn);
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id, name=?name " +
        "WHERE id=?oldId", conn);
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
}

```



```

da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");
da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id").SourceVersion = DataRowVersion.Original;
}

```

DeleteCommand

データセットのレコードを削除するための SQL 文またはストアード プロシージャを取得または設定します。

値 : [System.Data.Common.DataAdapter.Update](#) 中に、[DataSet](#) 内の削除された行に対応するデータベース内のレコードを削除するための [MySqlCommand](#)。

[System.Data.Common.DataAdapter.Update](#) 中、プロパティが設定されておらず、プライマリ キー情報が [DataSet](#) 内に存在する場合、[SelectCommand](#) プロパティを設定し、[MySqlCommandBuilder](#) を使用すれば、[DeleteCommand](#) を自動生成することが可能です。その後、設定していない追加のコマンドは、[MySqlCommandBuilder](#) によって生成されます。この生成ロジックでは、[DataSet](#) 内にキー カラム情報が存在している必要があります。

[DeleteCommand](#) が作成済みの [SelectCommand](#) に割り当てられた場合、[MySqlCommand](#) のクローンは作成されません。[DeleteCommand](#) は、作成済みの [MySqlCommand](#) オブジェクトへの参照を維持します。

例

以下の例は [MySqlDataAdapter](#) を作成し、[SelectCommand](#) と [DeleteCommand](#) プロパティを設定します。ここでは、[MySqlConnection](#) オブジェクトがすでに作成されていることを前提にしています。

Visual Basic 例 :

```

Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter

    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the DeleteCommand.
    cmd = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    parm.SourceVersion = DataRowVersion.Original
    da.DeleteCommand = cmd
    Return da
End Function

```

C# 例 :

```

public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the DeleteCommand.
    cmd = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
    parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
    parm.SourceVersion = DataRowVersion.Original;
    da.DeleteCommand = cmd;
    return da;
}

```

InsertCommand

データ セットにレコードを挿入するための SQL 文またはストアード プロシージャを取得または設定します。

値 :[System.Data.Common.DataAdapter.Update](#) 中に、[DataSet](#) 内の新しい行に対応するデータベースにレコードを挿入するための [MySqlCommand](#)。

[System.Data.Common.DataAdapter.Update](#) 中、プロパティが設定されておらず、プライマリ キー情報が [DataSet](#) 内に存在する場合、[SelectCommand](#) プロパティを設定し、[MySqlCommandBuilder](#) を使用すれば、[InsertCommand](#) を自動生成することが可能です。その後、設定していない追加のコマンドは、[MySqlCommandBuilder](#) によって生成されます。この生成ロジックでは、[DataSet](#) 内にキー カラム情報が存在している必要があります。

[InsertCommand](#) が作成済みの [SelectCommand](#) に割り当てられた場合、[MySqlCommand](#) のクローンは作成されません。[InsertCommand](#) は、作成済みの [MySqlCommand](#) オブジェクトへの参照を維持します。

注記

このコマンドの実行によって行が返される場合、[MySqlCommand](#) オブジェクトの [MySqlCommand.UpdatedRowSource](#) プロパティの設定によっては、返された行が [DataSet](#) に追加されることがあります。

例

以下の例は [MySqlDataAdapter](#) を作成し、[SelectCommand](#) と [InsertCommand](#) プロパティを設定します。ここでは、[MySqlConnection](#) オブジェクトがすでに作成されていることを前提にしています。

Visual Basic 例：

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter
    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the InsertCommand.
    cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id, ?name)", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id")
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name")
    da.InsertCommand = cmd

    Return da
End Function
```

C# 例：

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the InsertCommand.
    cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,?name)", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id");
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name");

    da.InsertCommand = cmd;
    return da;
}
```

UpdateCommand

データ ソース内のレコードの更新に使用される SQL 文またはストアド プロシージャを取得または設定します。

値 :[System.Data.Common.DataAdapter.Update](#) 中に、[DataSet](#) からのデータで、データベース内のレコードを更新するための [MySqlCommand](#)。

`System.Data.Common.DataAdapter.Update` 中、プロパティが設定されておらず、プライマリ キー情報が `DataSet` 内に存在する場合、`SelectCommand` プロパティを設定し、`MySqlCommandBuilder` を使用すれば、`UpdateCommand` を自動生成することが可能です。その後、設定していない追加のコマンドは、`MySqlCommandBuilder` によって生成されます。この生成ロジックでは、`DataSet` 内にキー カラム情報が存在している必要があります。

`UpdateCommand` が作成済みの `SelectCommand` に割り当てられた場合、`MySqlCommand` のクローンは作成されません。`UpdateCommand` は、作成済みの `MySqlCommand` オブジェクトへの参照を維持します。

注記

このコマンドの実行によって行が返される場合、`MySqlCommand` オブジェクトの `MySqlCommand.UpdatedRowSource` プロパティの設定によっては、返された行が `DataSet` にマージされることがあります。

例

以下の例は `MySqlDataAdapter` を作成し、`SelectCommand` と `UpdateCommand` プロパティを設定します。ここでは、`MySqlConnection` オブジェクトがすでに作成されていることを前提にしています。

Visual Basic 例 :

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter

    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the UpdateCommand.
    cmd = New MySqlCommand("UPDATE mytable SET id=?id, name=?name WHERE id=?oldId", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" )
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" )

    parm = cmd.Parameters.Add("?oldId", MySqlDbType.VarChar, 15, "id")
    parm.SourceVersion = DataRowVersion.Original

    da.UpdateCommand = cmd

    Return da
End Function
```

C# 例 :

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the UpdateCommand.
    cmd = new MySqlCommand("UPDATE mytable SET id=?id, name=?name WHERE id=?oldId", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );

    parm = cmd.Parameters.Add("?oldId", MySqlDbType.VarChar, 15, "id" );
    parm.SourceVersion = DataRowVersion.Original;

    da.UpdateCommand = cmd;
    return da;
}
```

SelectCommand

データ ソース内のレコードの選択に使用される SQL 文またはストアド プロシージャを取得または設定します。

値 : `System.Data.Common.DbDataAdapter.Fill` 中に、`DataSet` に格納するレコードをデータベースから選択するための `MySqlCommand`。

`SelectCommand` が作成済みの `SelectCommand` に割り当てられた場合、`MySqlCommand` のクローンは作成されません。`SelectCommand` は、作成済みの `MySqlCommand` オブジェクトへの参照を維持します。

`SelectCommand` が行を戻さない場合、`DataSet` にテーブルは追加されず、例外も発生しません。

例

以下の例は `MySqlDataAdapter` を作成し、`SelectCommand` と `InsertCommand` プロパティを設定します。ここでは、`MySqlConnection` オブジェクトがすでに作成されていることを前提にしています。

Visual Basic 例 :

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection) As MySqlDataAdapter
    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the InsertCommand.
    cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id, ?name)", conn)
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id")
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name")
    da.InsertCommand = cmd

    Return da
End Function
```

C# 例 :

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection conn)
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND name=?name", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the InsertCommand.
    cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,?name)", conn);
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id");
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name");

    da.InsertCommand = cmd;
    return da;
}
```

MySqlDataReader の使用

`MySqlDataReader` を作成するには、コンストラクタを直接使用するのではなく、`MySqlCommand` オブジェクトの `MySqlCommand.ExecuteReader` メソッドを呼び出す必要があります。

`MySqlDataReader` が使用されている間、関連する `MySqlConnection` は `MySqlDataReader` への供給に忙しく、`MySqlConnection` で、閉じる以外の作業を行うことはできません。`MySqlDataReader` の `MySqlDataReader.Close` メソッドが呼び出されるまでは、その状態が続きます。

`MySqlDataReader` を閉じた後に呼び出せるプロパティは、`MySqlDataReader.IsClosed` および `MySqlDataReader.RecordsAffected` のみです。`RecordsAffected` プロパティは、`MySqlDataReader` が存在する間はいっでもアクセスすることが可能ですが、`RecordsAffected` の値を戻す前はいつも `Close` を呼び出し、戻り値が正確であるようにしてください。

適切にパフォーマンスを行うために、[MySqlDataReader](#) は不要なオブジェクトの作成や、不必要なデータのコピーを作ることを避けます。結果として、[MySqlDataReader.GetValue](#) をはじめとするメソッドへの複数のセルが、同じオブジェクトへ参照を返します。[GetValue](#) などのメソッドによって戻されたオブジェクトの、基礎となる値を改変している場合は、十分に注意してください。

例

次は、[MySqlConnection](#)、[MySqlCommand](#)、そして [MySqlDataReader](#) を作成する例です。この例はデータを読み取り、コンソールへ書き出します。そして最終的に [MySqlDataReader](#) を閉じ、その後 [MySqlConnection](#) を閉じます。

Visual Basic 例：

```
Public Sub ReadMyData(myConnString As String)
    Dim mySelectQuery As String = "SELECT OrderID, CustomerID FROM Orders"
    Dim myConnection As New MySqlConnection(myConnString)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
    myReader = myCommand.ExecuteReader()
    ' Always call Read before accessing data.
    While myReader.Read()
        Console.WriteLine((myReader.GetInt32(0) & ", " & myReader.GetString(1)))
    End While
    ' always call Close when done reading.
    myReader.Close()
    ' Close the connection when done with it.
    myConnection.Close()
End Sub 'ReadMyData
```

C# 例：

```
public void ReadMyData(string myConnString) {
    string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnection);
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    // Always call Read before accessing data.
    while (myReader.Read()) {
        Console.WriteLine(myReader.GetInt32(0) + ", " + myReader.GetString(1));
    }
    // always call Close when done reading.
    myReader.Close();
    // Close the connection when done with it.
    myConnection.Close();
}
```

GetBytes

[GetBytes](#) は、フィールドで使用可能なバイトの数を返します。ほとんどの場合、これはフィールドの長さと同じになります。ただし、フィールドからバイトを得るために [GetBytes](#) がすでに使用されている場合は、戻された数がフィールドの実際の長さより少ない場合があります。例えば、[MySqlDataReader](#) が大規模なデータ構造をバッファに読み込んでいる場合も、そのケースに当てはまる可能性があります。詳細は、[MySqlCommand.CommandBehavior.SequentialAccess](#) 設定を参照してください。

ヌル リファレンス (Visual Basic では [Nothing](#)) であるバッファを渡す場合は、[GetBytes](#) がフィールドの長さをバイトで返します。

変換は行われません。そのため、取り出されたデータはすでにバイト列である必要があります。

GetTimeSpan

指定したカラムの値を [TimeSpan](#) オブジェクトとして取得します。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetDateTime

指定したカラムの値を [System.DateTime](#) オブジェクトとして取得します。

注記

MySQL は、日付カラムが値 '0000-00-00' を、日付時刻カラムが値 '0000-00-00 00:00:00' を受け入れるのを許可します。DateTime 構造はこれらの値を含んだり、また表したりすることはできません。ゼロ値を含む可能性のあるカラムから日付時刻値を読み取るには、[GetMySqlDateTime](#) を使用します。このメソッドを使ってゼロ日付時刻カラムを読み取る動作は、[ZeroDateTimeBehavior](#) 接続ストリング オプションによって定義されます。このオプションの詳細は、[MySqlConnection.ConnectionString](#) を参照してください。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetMySqlDateTime

指定したカラムの値を [MySql.Data.Types.MySqlDateTime](#) オブジェクトとして取得します。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetString

指定したカラムの値を [String](#) オブジェクトとして取得します。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetDecimal

指定したカラムの値を [Decimal](#) オブジェクトとして取得します。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetDouble

指定したカラムの値を倍精度浮動小数点として取得します。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetFloat

指定したカラムの値を単精度浮動小数点として取得します。

パラメータ：カラムの 0 から始まる序数。

戻り値：指定したカラムの値。

GetGuid

指定したカラムの値をグローバル一意識別子 (GUID) として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

GetInt16

指定したカラムの値を 16 ビット符号付き整数として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

GetInt32

指定したカラムの値を 32 ビット符号付き整数として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

GetInt64

指定したカラムの値を 64 ビット符号付き整数として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

GetUInt16

指定したカラムの値を 16 ビット符号なし整数として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

GetUInt32

指定したカラムの値を 32 ビット符号なし整数として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

GetUInt64

指定したカラムの値を 64 ビット符号なし整数として取得します。

パラメータ: カラムの 0 から始まる序数。

戻り値: 指定したカラムの値。

MySqlException の使用

このクラスは、MySQL Data Provider がサーバから生成されたエラーを検出するたびに作成されます。

オープン接続は、例外が投入されても自動的に閉じません。クライアント アプリケーションが例外は致命的だと判断した場合、開いているすべての [MySqlDataReader](#) オブジェクトまたは [MySqlConnection](#) オブジェクトを閉じるようになっています。

例

次の例は、サーバの紛失に際して [MySqlException](#) を生成し、その後に例外を表示します。

Visual Basic 例：

```
Public Sub ShowException()
    Dim mySelectQuery As String = "SELECT column1 FROM table1"
    Dim myConnection As New MySqlConnection ("Data Source=localhost;Database=Sample;")
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    Try
        myCommand.Connection.Open()
    Catch e As MySqlException
        MessageBox.Show( e.Message )
    End Try
End Sub
```

C# 例：

```
public void ShowException()
{
    string mySelectQuery = "SELECT column1 FROM table1";
    MySqlConnection myConnection =
        new MySqlConnection("Data Source=localhost;Database=Sample;");
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnection);
    try
    {
        myCommand.Connection.Open();
    }
    catch (MySqlException e)
    {
        MessageBox.Show( e.Message );
    }
}
```

MySqlParameter の使用

パラメータ名の大文字と小文字は区別されません。

例

次の例は [MySqlDataAdapter](#) 内の [MySqlParameterCollection](#) コレクションを介して、[MySqlParameter](#) の複数のインスタンスを作成します。これらのパラメータはデータ ソースからデータを選択し、[DataSet](#) にそのデータを格納するために使用されます。この例は、[DataSet](#) と [MySqlDataAdapter](#) が、適切なスキーマ、コマンド、および接続ですでに作成されているという前提に基づいています。

Visual Basic 例：

```
Public Sub AddSqlParameters()
    ' ...
    ' create myDataSet and myDataAdapter
    ' ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters"
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239

    myDataAdapter.Fill(myDataSet)
End Sub 'AddSqlParameters
```

C# 例：

```
public void AddSqlParameters()
{
    // ...
    // create myDataSet and myDataAdapter
    // ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar, 80).Value = "toasters";
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Long).Value = 239;
    myDataAdapter.Fill(myDataSet);
}
```

MySqlParameterCollection の使用

コレクション内のパラメータの数は、コマンド テキスト内のパラメータ プレースホルダの数、または生成される例外の数と同じでなければなりません。

例

次の例は `MySqlDataAdapter` 内の `MySqlParameterCollection` コレクションを介して、`MySqlParameter` の複数のインスタンスを作成します。これらのパラメータはデータ ソース内のデータを選択し、`DataSet` にそのデータを格納するために使用されます。このコード、`DataSet` と `MySqlDataAdapter` が、適切なスキーマ、コマンド、および接続すでに作成されているという前提に基づいています。

Visual Basic 例 :

```
Public Sub AddParameters()
    ' ...
    ' create myDataSet and myDataAdapter
    ' ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", SqlDbType.VarChar, 80).Value = "toasters"
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", SqlDbType.Long).Value = 239

    myDataAdapter.Fill(myDataSet)
End Sub 'AddSqlParameters
```

C# 例 :

```
public void AddSqlParameters()
{
    // ...
    // create myDataSet and myDataAdapter
    // ...
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", SqlDbType.VarChar, 80).Value = "toasters";
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", SqlDbType.Long).Value = 239;
    myDataAdapter.Fill(myDataSet);
}
```

MySqlTransaction の使用

MySQL データベース内で作成される SQL トランザクションを表示。このクラスを継承することはできません。

アプリケーションは、`MySqlConnection` オブジェクト上で `MySqlConnection.BeginTransaction` を呼び出すことによって、`MySqlTransaction` オブジェクトを作成します。トランザクションに関連するすべての後続の操作 (例えば、トランザクションのコミットまたはアボート) は、`MySqlTransaction` オブジェクト上で行われます。

例

次は、`MySqlConnection` と `MySqlTransaction` を作成する例です。`MySqlConnection.BeginTransaction`、`MySqlTransaction.Commit`、`MySqlTransaction.Rollback` の各メソッドの使い方も示します。

Visual Basic 例 :

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')"
```

```

myCommand.ExecuteNonQuery()
myTrans.Commit()
Console.WriteLine("Both records are written to database.")
Catch e As Exception
Try
    myTrans.Rollback()
Catch ex As MySqlException
    If Not myTrans.Connection Is Nothing Then
        Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
            " was encountered while attempting to roll back the transaction.")
    End If
End Try

Console.WriteLine("An exception of type " & e.GetType().ToString() & _
    "was encountered while inserting the data.")
Console.WriteLine("Neither record was written to database.")
Finally
    myConnection.Close()
End Try
End Sub 'RunTransaction

```

C# 例 :

```

public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

Rollback

ロールバックは保留状態からのトランザクションです。

Rollback メソッドは、MySQL ステートメント ROLLBACK に相当します。トランザクションは、保留状態からのみロールバックできます (BeginTransaction が呼び出された後、かつ Commit が呼び出される前)。

例

次は、[MySqlConnection](#) と [MySqlTransaction](#) を作成する例です。[MySqlConnection.BeginTransaction](#)、[Commit](#)、[Rollback](#) の各メソッドの使い方も示します。

Visual Basic 例 :

```
Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Success.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                    " was encountered while attempting to roll back the transaction.")
            End If
        End Try

        Console.WriteLine("An exception of type " & e.GetType().ToString() & _
            "was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub
```

C# 例 :

```
public void RunSqlTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
}
```

```

catch(Exception e)
{
    try
    {
        myTrans.Rollback();
    }
    catch (MySqlException ex)
    {
        if (myTrans.Connection != null)
        {
            Console.WriteLine("An exception of type " + ex.GetType() +
                " was encountered while attempting to roll back the transaction.");
        }
    }

    Console.WriteLine("An exception of type " + e.GetType() +
        " was encountered while inserting the data.");
    Console.WriteLine("Neither record was written to database.");
}
finally
{
    myConnection.Close();
}
}

```

Commit

データベースのトランザクションをコミットします。

[Commit](#) メソッドは、MySQL ステートメント COMMIT に相当します。

例

次は、[MySqlConnection](#) と [MySqlTransaction](#) を作成する例です。[MySqlConnection.BeginTransaction](#)、[Commit](#)、[Rollback](#) の各メソッドの使い方も示します。

Visual Basic 例：

```

Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Success.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                    " was encountered while attempting to roll back the transaction.")
            End If
        End Try
        Console.WriteLine("An exception of type " & e.GetType().ToString() & _
            " was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    End Try
End Sub

```



```

Finally
    myConnection.Close()
End Try
End Sub

```

C# 例 :

```

public void RunSqlTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (100, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES (101, 'Description')";
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch (Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to roll back the transaction.");
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

Connector/NET に関する注記とヒント

このセクションでは、BLOB 処理、日付処理、そして Crystal Reports などの基本的なツールを使った Connector/NET の使用などを含む、Connector/NET のより一般的な使い方の例について説明します。

Connector/NET を使用した MySQL への接続

はじめに

.NET アプリケーションと MySQL サーバの間のすべてのアプリケーションは、[MySqlConnection](#) オブジェクトを通して経路指定されます。アプリケーションがサーバと交信できるようになる前に、[MySqlConnection](#) オブジェクトはインスタンスが作成され、構成され、開かれていなければなりません。

[MySqlHelper](#) クラスを使用している時でも、[MySqlConnection](#) オブジェクトはヘルパー クラスによって作成されます。

このセクションでは、[MySqlConnection](#) オブジェクトを使用して MySQL に接続する方法を説明します。

接続ストリングの作成

MySqlConnection オブジェクトは、接続ストリングを使って構成されます。接続ストリングは、セミコロンで区切られたサーバ キー/値のペアを含みます。それぞれのキー/値のペアは等号記号で結ばれます。

以下は接続ストリングのサンプルです。

```
NotInheritable Public Class MySqlCommand_
    Inherits Component_
    Implements IDbCommand, ICloneable
```

この例では、**MySqlConnection** オブジェクトは、**127.0.0.1** で MySQL サーバに接続するよう構築されています。ユーザ名は **root**、パスワードは **12345** です。すべてのステートメントのデフォルトのデータベースは **test** データベースになります。

次のオプションは通常的に使用されるものです (オプションの完全なリストは **ConnectionString** の API 使用で閲覧できます):

- **Server**: 接続する MySQL のインスタンスの名前またはネットワーク アドレス。デフォルトは **localhost**。エイリアスには **host**、**Data Source**、**DataSource**、**Address**、**Addr**、**Network Address** がある。
- **Uid**: 接続に使用する MySQL ユーザ アカウント。エイリアスには **User Id**、**Username**、**User name** がある。
- **Pwd**: 使用されている MySQL アカウントのパスワード。エイリアス **Password** も使用できる。
- **Database**: すべてのステートメントが適用されるデフォルトのデータベース。デフォルトは **mysql**。エイリアス **Initial Catalog** も使用できる。
- **Port**: MySQL が接続の監視に使用するポート。デフォルトは **3306**。この値を **-1** に指定して名前付きパイプ接続を使用する。

接続を開く

接続ストリングを作成すると、MySQL サーバへの接続を開くのに使用することができます。

次のコードは **MySqlConnection** オブジェクトの作成、接続ストリングの設定、接続の開通に使用されます。

Visual Basic 例:

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

C# 例:

```
Overloads Public Sub New()
```

また、接続ストリングを **MySqlConnection** クラスのコンストラクタに渡すこともできます。

Visual Basic 例:

```
public MySqlCommand();
```

C# 例:

```
Overloads Public Sub New( _
    ByVal cmdText As String _
)
```

Once the connection is open it can be used by the other Connector/NET classes to communicate with the MySQL server.

接続エラーの処理

外部サーバへの接続は予測不可能であるため、各 .NET アプリケーションのエラー処理を加えることが重要になります。接続でエラーが発生すると、**MySqlConnection** クラスが **MySqlException** オブジェクトを返します。このオブジェクトはエラーの処理時に重要なふたつのプロパティを持っています:

- **Message**:現在の例外を解説するメッセージ。
- **Number**:MySQL エラー番号。

処理にエラーが発生した時、エラー番号からアプリケーションの反応を予測することができます。接続で発生する一般的なエラー番号 2 種：

- **0**: サーバへ接続できません。
- **1045**: 無効なユーザ名、および/またはパスワード。

次のコードは、実際のエラーに基づいて、アプリケーションの反応に適応する方法を示します：

Visual Basic 例：

```
public MySqlCommand(
    string cmdText
);
```

C# 例：

```
Overloads Public Sub New( _
    ByVal cmdText As String, _
    ByVal connection As MySqlConnection _
)
```

重要点：複数言語データベースを使用している場合は、接続ストリングで文字セットを指定する必要があります。文字セットを指定しない場合、接続はデフォルトで **latin1** 文字セットになります。文字セットは接続ストリングの一部として指定することができます。例：

```
public MySqlCommand(
    string cmdText,
    MySqlConnection connection
);
```

プリペアド ステートメントで Connector/NET を使用する

はじめに

MySQL 4.1 から、Connector/NET でプリペアド ステートメントが使用できるようになりました。プリペアド ステートメントの使用により、一度以上実行されるクエリの性能が大きく改善されました。

準備された実行は、基本的にクエリの解析が一度だけなので、一度以上実行されるステートメントの実行を、直接実行するより素早く行います。直接実行の場合、クエリは実行のたびに解析されます。準備された実行はまた、実行のたびにパラメータにデータを送るだけで済むので、ネットワークの通信料を減らす効果があります。

プリペアド ステートメントの他の利点は、クライアントとサーバ間のデータ伝達をより効果的にするバイナリ プロトコルを使用していることです。

Connector/NET のプリペアド ステートメント

ステートメントを準備するには、コマンド オブジェクトを作成し、**.CommandText** プロパティをクエリに設定します。

ステートメントを入力した後、**MySqlCommand** オブジェクトの **.Prepare** メソッドを呼び出します。ステートメントが準備できたら、クエリの各動的要素にパラメータを加えます。

クエリとパラメータを入力した後、**.ExecuteNonQuery()**、**.ExecuteScalar()**、または **.ExecuteReader** のメソッドを使用してステートメントを実行します。

後続の実行では、パラメータの値を改変し、**execute** メソッドを呼び出すだけで済み、**.CommandText** プロパティを設定したり、パラメータを定義し直す必要はありません。

Visual Basic 例：

```
NotInheritable Public Class MySqlConnection_
    Inherits Component_
    Implements IDbConnection, ICloneable
```

C# 例 :

```
public sealed class MySqlConnection : Component, IDbConnection, ICloneable
```

Connector/NET のストアード プロシージャにアクセスする

はじめに

MySQL バージョン 5 のリリースにより、MySQL サーバは、SQL 2003 ストアド プロシージャ シンタックスでストアード プロシージャをサポートするようになりました。

ストアード プロシージャは、サーバで保管が可能な SQL 文です。これが実行されると、クライアントは各ステートメントを発行し続ける必要がなくなり、代わりにストアード プロシージャを参照します。

ストアード プロシージャは、次のような状況で特に利点があります :

- 複数のクライアントが、複数の言語で書かれている場合、または異なるプラットフォームで使用するが、同じデータベース操作を行う必要がある場合。
- セキュリティを最優先する場合。例えば銀行では、すべての通常のオペレーションにストアード プロシージャを使用します。ストアード プロシージャは安全で安定した環境を供給し、プロシージャは各オペレーションが正しくログされるよう保証します。このようなセットアップでは、アプリケーションとユーザはデータベース テーブルへ直接アクセスされる心配がなく、特定のストアード プロシージャのみを実行します。

Connector/NET は、[MySqlCommand](#) オブジェクトを通したストアード プロシージャの呼び出しをサポートします。[MySqlCommand.Parameters](#) コレクションの使用を通じて、MySQL ストアド プロシージャにデータを受け渡しすることができます。

注記

ストアード プロシージャを呼び出す際、コマンド オブジェクトは追加の [SELECT](#) 呼び出しでストアード プロシージャのパラメータを決定します。プロシージャを呼び出すユーザは、パラメータの検証を行うために、[mysql.proc](#) テーブルの [SELECT](#) 権限を取得していなければなりません。権限を持たないと、プロシージャを呼び出す際にエラーが発生します。

このセクションには、Stored Procedures の作成の詳しい情報は記載されていません。それらの詳細は、<http://dev.mysql.com/doc/mysql/en/stored-procedures.html> をご覧ください。

Connector/NET でのストアード プロシージャの使用例を示したアプリケーションのサンプルは、Connector/NET インストールの [Samples](#) ディレクトリでご覧いただけます。

Connector/NET からストアード プロシージャを作成する

MySQL のストアード プロシージャは、様々なツールを使用して作成することができます。まず、ストアード プロシージャは [mysql](#) コマンドラインのクライアントを使って作成できます。次に、[MySQL Query Browser](#) GUI クライアントでも作成できます。最後に、[MySqlCommand](#) オブジェクトの [.ExecuteNonQuery](#) メソッドでも作成することができます :

Visual Basic 例 :

```
Overloads Public Sub New()
```

C# 例 :

```
public MySqlConnection();
```

Connector/NET でストアード プロシージャを作成する際、コマンドラインや GUI クライアントとは異なり、特別な区切り符号を指定する必要がないのも特筆するべき点です。

Connector/NET からストアド プロシージャを呼び出す

Connector/NET を使用してストアド プロシージャを呼び出すには、 [MySqlCommand](#) オブジェクトを作成し、ストアド プロシージャ名を [.CommandText](#) プロパティとして渡します。 [.CommandType](#) プロパティを [CommandType.StoredProcedure](#) に設定してください。

ストアド プロシージャに名前が付けられたら、ストアド プロシージャのすべてのパラメータにつき、 [MySqlCommand](#) パラメータをひとつ作成します。 [IN](#) パラメータは、パラメータ名と値を含むオブジェクトで定義され、 [OUT](#) パラメータはパラメータ名と戻される予定のデータタイプで定義されます。すべてのパラメータは、パラメータの方向が定義されなければなりません。

パラメータを定義した後、 [MySqlCommand.ExecuteNonQuery\(\)](#) メソッドを使用して、ストアド プロシージャを呼び出してください：

Visual Basic 例：

```
Overloads Public Sub New( _
    ByVal connectionString As String _
)
```

C# 例：

```
public MySqlConnection(
    string connectionString
);
```

ストアド プロシージャが呼び出されたら、 [MySqlConnection.Parameters](#) コレクションの [.Value](#) プロパティを使って、出力パラメータの値を取り出すことができます。

Connector/NET で BLOB データを処理する

はじめに

MySQL の一般的な使い方のひとつに、 [BLOB](#) カラムのバイナリ データの保管があります。MySQL は 4 種類の BLOB データタイプをサポートしています： [TINYBLOB](#) 、 [BLOB](#) 、 [MEDIUMBLOB](#) 、および [LONGBLOB](#) 。

BLOB カラムに保管されたデータには、Connector/NET を使ってアクセスすることができ、クライアント側コードで操作することができます。BLOB データで Connector/NET を使用するための特別な条件はありません。

簡単なコードの例は、このセクションに記載されています。アプリケーションの完全なサンプルは、Connector/NET インストールの [Samples](#) ディレクトリでご覧いただけます。

MySQL サーバの作成

まず最初に、BLOB データで MySQL を使用すると、サーバが構成されます。アクセスするテーブルの作成から始めましょう。例に取り上げるファイル テーブルには、4 つのカラムがあります：ファイルを特定するプライマリ キーとして作動する、適切なサイズ (UNSIGNED SMALLINT) の AUTO_INCREMENT カラム、ファイル名を保管する VARCHAR カラム、ファイルのサイズを保管する UNSIGNED MEDIUMINT カラム、そしてファイルそのものを保管する MEDIUMBLOB カラム。例えば、次のテーブル定義を使用するとします：

```
NotOverridable Public Property ConnectionString As String _
-
    Implements IDbConnection.ConnectionString
```

テーブルを作成した後、 [max_allowed_packet](#) システム変数を変更しなければならない場合があります。この変数は、MySQL サーバに遅れるパケットの大きさ (例：1 行) を特定します。デフォルトでは、サーバーは最大 1 メガのサイズしかクライアント アプリケーションから受け入れないようになっています。1 メガを越えることがないようならば、これで問題ありません。1 メガを越えるファイルを転送する場合は、この数字を上げる必要があります。

[max_allowed_packet](#) オプションは、MySQL Administrator's Startup Variables 画面で変更することができます。Networking タブの Memory セクションにある Maximum Allowed オプションを、適切な設定に調節します。値を調節したら、Apply Changes ボタンをクリックし、MySQL Administrator の [Service Control](#) 画面でサーバを再起動し

ます。また、my.cnf ファイルで直接この値を変えることもできます (max_allowed_packet=xxM を読み取るラインを加える)。MySQL 内の SET max_allowed_packet=xxM; シンタックスを使用して値を調節できます。

BLOB データの転送には時間がかかることがあるので、max_allowed_packet の設定は慎重に行ってください。使用目的に適した値を設定し、必要に応じて値を上げるようにしてください。

データベースへのファイルの書き込み

データベースへファイルを書き込むには、ファイルをバイト列に変換し、そしてそのバイト列を INSERT クエリへのパラメータとして使用します。

次のコードは、FileStream オブジェクトを使用してファイルを開き、バイト列に読み出して、それを file テーブルに挿入します：

Visual Basic 例：

```
public string ConnectionString {get; set;}
```

C# 例：

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer _
_
Implements IDbConnection.ConnectionTimeout
```

FileStream オブジェクトの Read メソッドは、ファイルを FileStream オブジェクトの Length プロパティによってサイズ分類されたバイト列にロードするために使われます。

バイト列を MySqlCommand オブジェクトのパラメータとして割り当てた後、ExecuteNonQuery メソッドが呼び出され、BLOB が file テーブルに挿入されます。

データベースの BLOB をディスクのファイルに読み出す

フィールドが file テーブルにロードされたら、MySqlDataReader クラスを使用してそれを取り出すことができます。

次のコードは file テーブルからコードを取り出し、そしてそのデータを FileStream オブジェクトにロードしてディスクへ書き込みます：

Visual Basic 例：

```
public int ConnectionTimeout {get;}
```

C# 例：

```
NotOverridable Public ReadOnly Property Database As String _
_
Implements IDbConnection.Database
```

接続後、file テーブルのコンテンツは MySqlDataReader オブジェクトにロードされます。MySqlDataReader の GetBytes メソッドによって BLOB がバイト列にロードされ、FileStream オブジェクトを使ってディスクに書き込まれます。

MySqlDataReader の GetOrdinal メソッドで、名前付きカラムの整数インデックスを定義することができます。SELECT クエリのカラムの順が変更された場合、GetOrdinal メソッドを使用すると、エラーを防ぐことができます。

Crystal Reports で Connector/NET を使用する

はじめに

Crystal Reports は Windows アプリケーション開発者が使用する一般的なツールで、レポートとドキュメントの生成を行います。このセクションでは、MySQL および Connector/NET との Crystal Reports XI の使用方法を説明します。

データソースの作成

Crystal Reports でレポートを作成する際、レポートを設計しながら MySQL データにアクセスするためのオプションがふたつあります。

第一のオプションは、レポートを設計する時に、Connector/ODBC を ADO データとして使用する方法です。データベースをブラウズし、テーブルやフィールドを選んで、ドラッグアンドドロップでレポートを構築することができます。この方法の欠点は、レポートが示すものと一致するデータセットを生成するために、アプリケーション内で追加の操作を行わなければならないことです。

第二のオプションは、VB.NET でデータセットを作成し、XML として保存する方法です。この XML ファイルはレポートの設計に使用することができます。この方法は、アプリケーションにレポートを表示する時には非常に効果的ですが、データセットを作成する際に関連するカラムをすべて選ばなければならないので、設計時間の点で劣ります。カラムを忘れてしまった場合は、カラムをレポートに追加する前にデータセットを再作成する必要があります。

次のコードはクエリからのデータセットの作成と、ディスクへの書き込みに使用できます：

Visual Basic 例：

```
public string Database {get;}
```

C# 例：

```
Public ReadOnly Property DataSource As String
```

XML ファイルの結果物は、レポートを設計する際に ADO.NET XML データソースとして使用できます。

Connector/ODBC を使用してレポートを設計する場合は、dev.mysql.com からダウンロードすることができます。

レポートの作成

ほとんどの目的において、Standard Report ウィザードはレポートの初期作成を助けます。ウィザードを立ち上げるには、Crystal Reports を開き、File メニューから New > Standard Report オプションを選択します。

ウィザードはまず、データソースの入力を促します。Connector/ODBC をデータソースとして使用している場合、データソースを選択する際に、ODBC (RDO) ツリーではなく、OLE DB (ADO) ツリーからの ODBC オプションに OLEDB プロバイダを使用します。保存されたデータセットを使用している場合は、ADO.NET (XML) オプションを使用して、保存されたデータセットをブラウズします。

レポート作成プロセスの残部はウィザードが自動的に行います。

レポートが作成されたら、Report Options... entry of the File メニューを選びます。Save Data With Report オプションのチェックを外してください。これによって、保存されたデータがアプリケーション内のデータのロードを妨げるのを防ぎます。

レポートの表示

レポートを表示するには、まずレポートに必要なデータをデータセットに移植し、そしてレポートをロード、データセットに結合させます。最後に、レポートを crViewer コントロールに渡してユーザに表示します。

次のリファレンスはレポートを表示するプロジェクトに必要なものです：

- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

次のコードは、レポートが [データソースの作成](#) で示されているコードを使用して保存されたデータセットで作成され、フォームに [myViewer](#) と名付けられた crViewer コントロールがあるという前提で書かれています。

Visual Basic 例：

```
public string DataSource {get;}
```

C# 例：

```
Public ReadOnly Property ServerThread As Integer
```

新しいデータセットは、保存された以前のデータセットに使用された同じクエリで生成されます。データセットへの記入が終わったら、ReportDocument によってレポート ファイルがロードされ、データセットへ結合されます。ReportDocument は、crViewer の ReportSource として渡されます。

Connection/ODBC を使用して単一テーブルからレポートが作成される場合にも、同じ方法が取られます。データセットがレポートで使用されたテーブルを交換し、レポートが正しく表示されます。

Connector/ODBC を使って、複数のテーブルからレポートが作成される場合、複数のテーブルとのデータセットは、各自のアプリケーション内で作成されなければなりません。これにより、レポート データソースの各テーブルが、データセットのレポートに取り替えられます。

MySqlCommand オブジェクトに複数の **SELECT** 文を供給することにより、複数のテーブルをデータセットに移植します。これらの **SELECT** 文は、Database メニューの Show SQL Query オプションの Crystal Reports に示されている SQL クエリに基づいています。次のクエリを想定します：

```
public int ServerThread {get;}
```

このクエリは、ふたつの **SELECT** クエリに変換され、次のコードで表示されます：

Visual Basic 例：

```
Public ReadOnly Property ServerVersion As String
```

C# 例：

```
public string ServerVersion {get;}
```

SELECT クエリをアルファベット順に並べることが重要です。レポートはこの順で入るべきソース テーブルを予想します。

この方法は、Crystal Reports がクライアント側でテーブル同士を結合させる必要があり、保存済みのデータセットを使用するより遅くなるため、性能上の問題があります。

Connector/NET で日付時間情報を処理する

はじめに

MySQL と .NET 言語の日付時間情報の処理は異なり、MySQL は '0000-00-00 00:00:00' のような、.NET データ タイプでは表せない日付も使用することができます。適切に処理を行わなければ、この違いによって問題が起きることがあります。

このセクションでは、Connector/NET を使用する場合の、日付時間情報の正しい処理の方法を説明します。

無効な日付を使用する場合の問題

日付の処理の違いは、無効な日付を使用する開発者にとって問題の原因になりえます。無効な MySQL の日付を、**NULL** の日付を含む、ネイティブの .NET **DateTime** オブジェクトにロードすることはできません。

この問題により、無効な日付は **System.ArgumentOutOfRangeException** 例外の発生の原因になるので、.NET **DataSet** オブジェクトを **MySqlDataAdapter** クラスの **Fill** メソッドで作成することはできません。

無効な日付の制限

日付の問題に対する最善の解決策は、ユーザの無効な日付の使用を制限することです。これはクライアント側、またはサーバ側で行えます。

クライアント側で無効な日付を制限するのは簡単で、データの処理に常に .NET [DateTime](#) クラスを使用するだけです。[DateTime](#) クラスは、データベース内の値も有効であることを確認し、有効な日付だけを許可します。この方法の欠点は、それぞれのアプリケーションが独自の日付の発行を行う必要があり、データベースの操作に .NET および .NET コードが使われている混合環境での使用が困難な点です。

MySQL 5.0.2 以降のユーザは、無効な日付値の使用を制限する、新しい [traditional](#) SQL モードを使用することができます。[traditional](#) SQL モードの使用に関する情報は、[SQL モード](#) をご覧ください。

無効な日付の処理

.NET アプリケーションでの無効な日付の使用は避けることが推奨されますが、[MySqlDateTime](#) データタイプを用いて無効な日付を使用することは可能です。

[MySqlDateTime](#) データタイプは、MySQL サーバでサポートされている日付値と同じものを適用しています。Connector/NET のデフォルト動作は、.NET [DateTime](#) オブジェクトを有効な日付値に戻し、エラーを無効な日付に返します。このデフォルトは、Connector/NET が無効な日付に [MySqlDateTime](#) オブジェクトを返すように変更できます。

無効な日付に [MySqlDateTime](#) を返すよう Connector/NET に指示するには、次のラインを接続ストリングに追加してください：

```
Allow Zero Datetime=True
```

[MySqlDateTime](#) クラスの使用には、依然として問題があることに注意してください。以下は既知の問題の一部です：

1. 無効な日付のデータ構築は依然、エラーの原因になります (0000-00-00 などのゼロの日付ではこの問題は起こらない模様)。
2. [ToString](#) メソッドは、標準の MySQL の形式でフォーマットされた日付を返します (例えば、[2005-02-23 08:50:25](#))。これは、.NET [DateTime](#) クラスの [ToString](#) 動作とは異なるものです。
3. [MySqlDateTime](#) クラスは NULL の日付をサポートし、.NET [DateTime](#) クラスはこれをサポートしません。この差により、[MySqlDateTime](#) を [DateTime](#) に変換する時、最初に NULL を確認しておかなければエラーが発生します。

これは既知の問題であるため、有効な日付のみをアプリケーションで使用するのが最善の方法になります。

NULL データの処理

.NET [DateTime](#) データタイプは、NULL 値を処理することができません。そのため、クエリから [DateTime](#) 変数に値を設定する時、最初に値が NULL かどうかを確認する必要があります。

[MySqlDataReader](#) を使用している時、設定の前に [.IsDBNull](#) メソッドを使って、値が NULL か確認してください。

Visual Basic 例：

```
public System.Data.ConnectionState State {get;}
```

C# 例：

```
Public ReadOnly Property UseCompression As Boolean
```

NULL 値はデータセット内で作用し、特別な処理なしで結合してコントロールを形成することが可能です。

Connector/NET のサポート

Connector/NET の開発者は、ソフトウェア開発のプロセスにおいて、ユーザの声を大変貴重なものと考えています。Connector/NET が大切な機能を欠いているとお気づきになった場合、またはバグを発見し、バグ レポートを提出したい場合は、[質問またはバグの報告](#) の指示をお読みください。

Connector/NET のコミュニティー支援

- Connector/NET のコミュニティー支援は、<http://forums.mysql.com> のフォーラムで得ることができます。

- また、Connector/NET のコミュニティー支援を、<http://lists.mysql.com> のメーリング リストを介して得ることも可能です。
- MySQL AB では有料サポートも提供しています。その他の情報は <http://dev.mysql.com/support/> でご覧いただけます。

Connector/NET の不具合またはバグのレポート

Connector/NET に不具合や問題があった場合は、Connector/NET コミュニティー [Connector/NET のコミュニティー支援](#) にご連絡ください。

まず最初に、[mysql](#) クライアント プログラムか、[admindemo](#) からの同じ SQL 文やコマンドを実行してみてください。エラーが Connector/NET か MySQL で発生しているものか調べることができます。

問題をレポートする際は、メールに次の情報をご記入いただくと参考になります。

- オペレーション システムとバージョン
- Connector/NET のバージョン
- MySQL サーバのバージョン
- エラー メッセージ、または他の想定外のアウトプットのコピー。
- 簡単な再現可能サンプル

詳しい詳細を提供していただくのが問題解決には一番ですので、その旨をご理解ください。

バグの問題と思われる場合は、<http://bugs.mysql.com/> からバグをレポートするようお願いします。

Connector/ODBC 変更履歴

Connector/NET Change History (Changelog) は、MySQL のメイン Changelog に収められています。[Connector/NET Change History](#) 参照。

MySQL Visual Studio プラグイン

MySQL Visual Studio プラグインは DDEX プロバイダです。データベース構成の維持管理を可能にする Visual Studio 2005 用のプラグインで、内蔵のデータ駆動型アプリケーションをサポートします。

現行バージョンの MySQL Visual Studio プラグインには、データベース保守 ツールのみが含まれています。データ駆動型アプリケーション開発ツールはサポートされていません。

MySQL DDEX プロバイダは、Visual Studio 2005 の Server Explorer メニューを通じて使用できる Visual Studio Data Designer 機能の標準エクステンションとして作動し、MySQL データベース内のデータベース オブジェクトとデータの作成を可能にします。

MySQL Visual Studio プラグインは MySQL バージョン 5.0 との使用を目的にデザインされていますが、MySQL 4.1.1 と互換性があり、また SQL 5.1 にも制限付きで適応します。

MySQL Visual Studio プラグインのインストール

MySQL Visual Studio プラグインを利用するには、Visual Studio 2005 Professional Edition がインストールされていなければならない、従って動作環境とシステムの必要条件も同じということになります。

下記は MySQL Visual Studio プラグインをインストールする前に、あらかじめ導入しておくべきコンポーネントのリストです。

- Visual Studio 2005 Standard、Professional、または Team Developer Edition
- MySQL Server 4.1.1 以降 (同じマシンか別のサーバにインストールされたもの)
- MySQL Connector/NET 5.0.

注記

Connector/NET をインストールする際、まず Global Assembly Cache (GAC) にコネクタがインストール済みか確認してください。Connector/NET インストーラはこの作業を自動的に行うようになっていますが、カスタム インストールではこのオプションが無効になっている場合があります。

以前からMySQL サーバへ接続しているユーザーがMySQL Visual Studio プラグインの機能を利用するには、以下の権限を取得しなければなりません。

- **INFORMATION_SCHEMA** データベースに対する **SELECT** 権限
- **SHOW CREATE TABLE** 文に対する **EXECUTE** 権限
- **mysql.proc** テーブルに対する **SELECT** 権限 (ストアド プロシージャおよびファンクションの操作に必要)
- **mysql.func** テーブルに対する **SELECT** 権限 (ユーザ定義関数 (UDF) の操作に必要)
- **SHOW ENGINE STATUS** 文に対する **EXECUTE** 権限 (拡張エラー情報の読み出しに必要)
- 実行された操作に対する適切な特権 (例 : テーブルからデータを閲覧するために必要な **SELECT** 特権、等)

MySQL Visual Studio プラグインは、プロバイダのインストール、アンインストール、または再インストールに使用できる MSI パッケージとして配信されます。Windows XP か Windows Server 2003 を使用していない場合は、Windows Installer System を最新バージョンにアップグレードしてください (<http://support.microsoft.com/default.aspx?scid=kb;EN-US;292539> 参照)

MSI パッケージの名称は **MySQL.VisualStudio.msi** です。MySQL Visual Studio プラグインをインストールするには、MSI ファイルを右クリックし、**Install** を選択します。インストールの手順は次の通りです。

1. 標準の Welcome ダイアログが開きます。Next をクリックし、インストールを続けます。
2. License agreement (GNU GPL) ウィンドウが開きます。契約に合意し、**Next** をクリックして次に進みます。
3. 移動先フォルダを指定するダイアログが開きます。ここで、MySQL Visual Studio プラグインがインストールされるフォルダを指定できます。デフォルトの移動先フォルダは **%ProgramFilesDir%\MySQL\MySQL DDEX Data Provider** で、**%ProgramFilesDir%** がインストールするマシンの Program Files フォルダになります。移動先フォルダを選択したら、**Next** をクリックして次に進みます。
4. インストーラが、インストールの確認を求めます。Install をクリックしてインストールを開始してください。
5. インストールが開始します。この段階の最後に、Visual Studio コマンド テーブルが再構築されます (このプロセスの完了には数分間かかります)。
6. インストールが完了したら、**Finish** ボタンをクリックしてインストールを終了します。

MySQL Visual Studio プラグインをアンインストールする場合は、Control Panel の Add/Remove Programs コンポーネントか、同じ MSI パッケージを使用することができます。**Remove** オプションを選択すると、プロバイダは自動的にアンインストールされます。

プロバイダを修復するには、MSI パッケージを右クリックし、**Repair** オプションを選択。MySQL Visual Studio プラグインが自動的に修復されます。

インストール パッケージには次のファイルが含まれています。

- **MySQL.VisualStudio.dll** — MySQL DDEX プロバイダ アセンブリ
- **MySQL.Data.dll** — プロバイダが使用する MySQL Connector .NET を含んだアセンブリ
- **MySql.VisualStudio.dll.config** — MySQL Visual Studio プラグインの情報設定ファイル。このファイルにはプロバイダの GUI レイアウトのデフォルト値が含まれています。

注記

プロバイダを実際に使用してみるまで、このファイルは削除しないでください。

- [Register.reg](#) — 手作業でインストールする場合に、MySQL DDEX プロバイダを登録するのに使用するレジストリ登録のファイル
- [Install.js](#) — Connector .NET を machine.config ファイル内の ADO.NET データ プロバイダとして登録するために使用するスクリプト
- [Release notes.doc](#) — リリース情報のドキュメント

プロバイダを手作業でインストールするには、インストール パッケージの全ファイルを希望の保存先にコピーし、プロバイダ アセンブリにフルパスを CodeBase 項目の値として設定する。例:

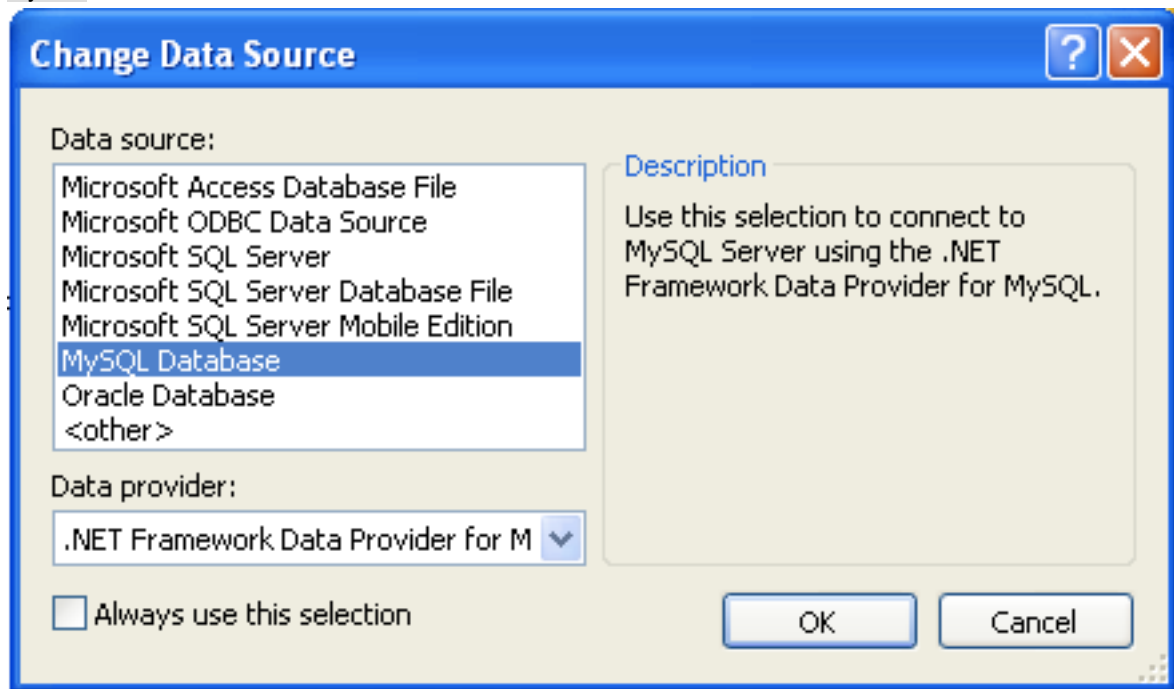
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Packages\{79A115C9-B133-4891-9E7B-242509DAD272}]\@="MySql.Data.VisualStudio.MySqlData
"InprocServer32"="C:\WINNT\system32\mscoree.dll"
"Class"="MySql.Data.VisualStudio.MySqlData.ProviderPackage"
"CodeBase"="C:\MySQLDdexProvider\MySQL.VisualStudio.dll"
```

次に、Register.reg ファイルからの情報を、ファイルをクリックしてレジストリにインポートします。確認ダイアログでは「Yes」を選択してください。次に、Command Prompt 内の devenv.exe /setup コマンドを実行し、Visual Studio コマンド テーブルを再構築します。

MySQL サーバとの接続の作成

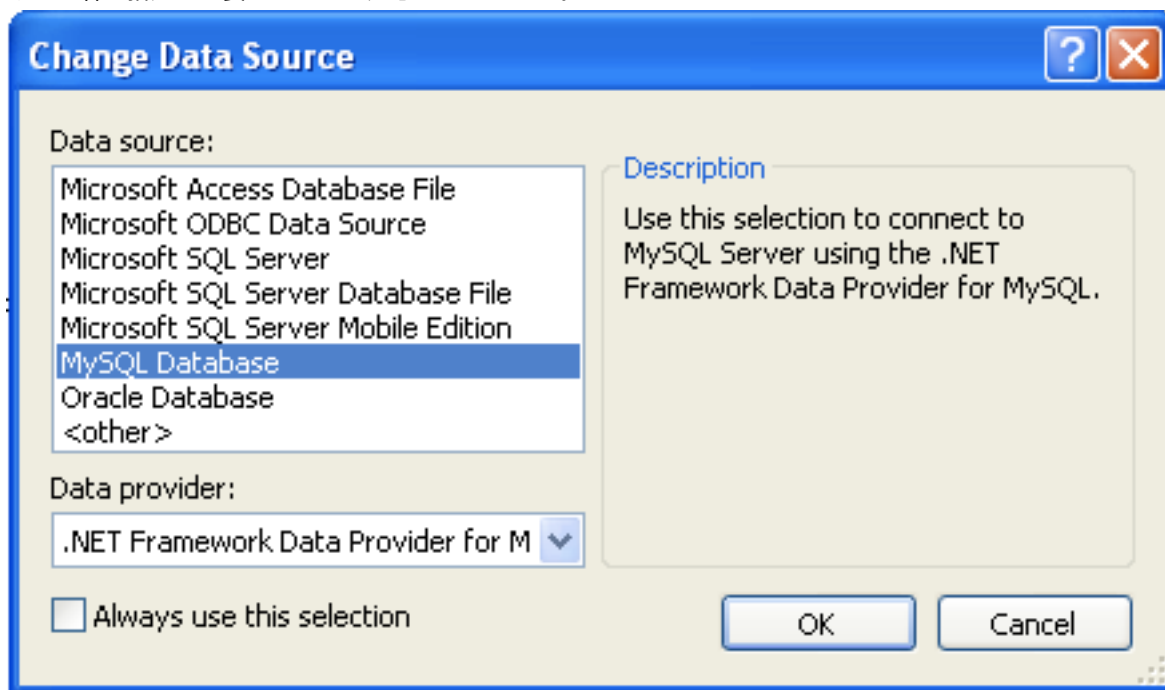
MySQL Visual Studio プラグインをインストールし終わったら、MySQL データベースとの接続を作成、変更、削除することができます。MySQL データベースとの接続を作成するには、次の手順を実行してください。

1. Visual Studio 2005 を開始し、**View** メニューから **Server Explorer** オプションを選択して、Server Explorer ウィンドウを開きます。
2. **Data Connections** ノードを右クリックして、**Add Connection** ボタンを選択します。
3. Add Connection ダイアログが開きます。**Change** ボタンを押して、MySQL Database をデータソースに指定します。
4. Change Data Source ダイアログが開きます。データソースの中から MySQL Database を選び (MySQL Database がない場合は **other** オプションを選択)、データ プロバイダの選択情報から **.NET Framework Data Provider for MySQL** を選択します。



OK を押して、選択を確認します。

5. 接続設定を入力します。サーバホスト名 (例えば、MySQL サーバがローカル マシンにインストールされる場合は localhost)、ユーザ名、パスワード、およびデフォルト データベース スキーマ。接続する際にはデフォルト スキーマ名の指定が必要になるので注意してください。



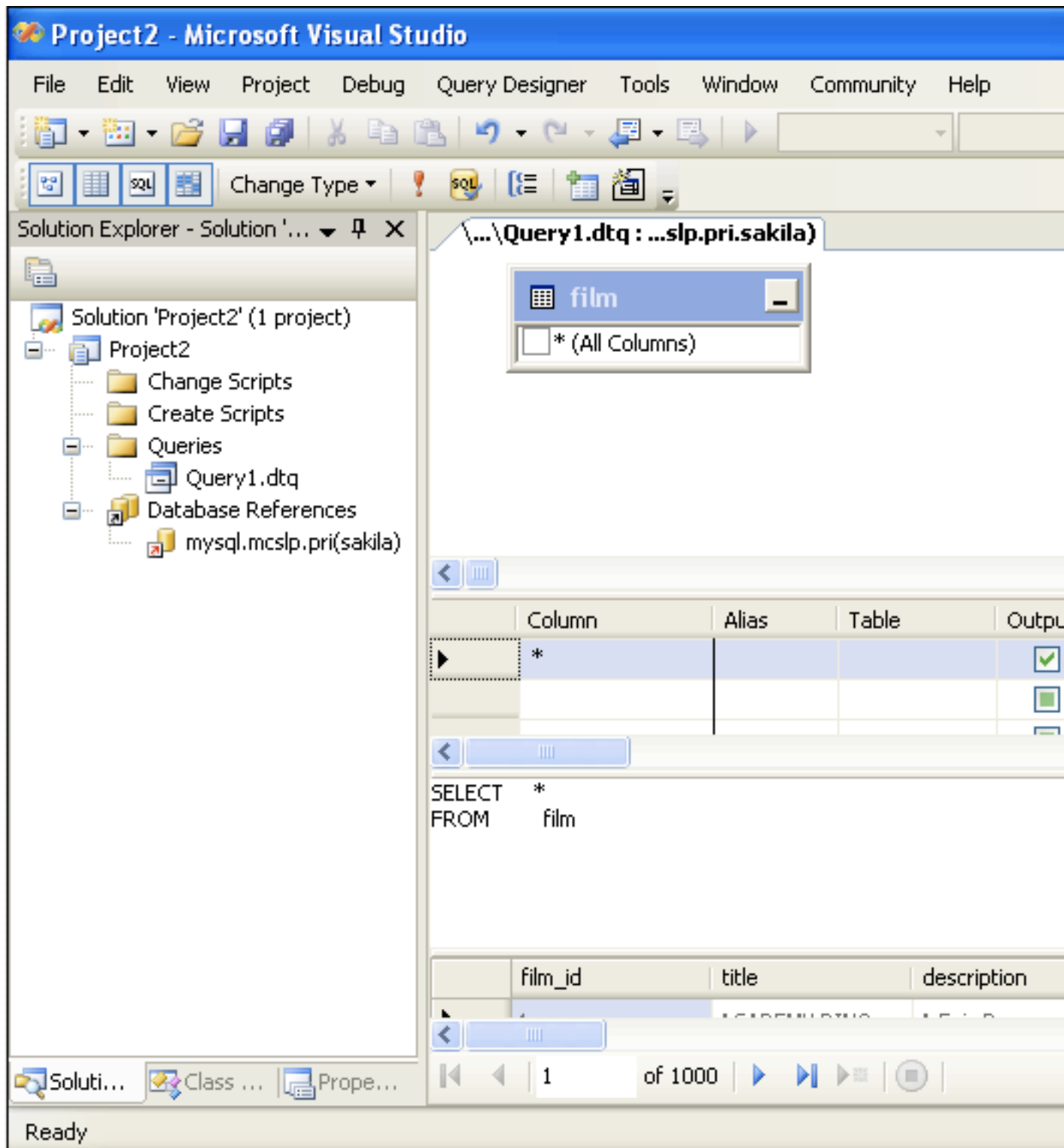
6. **Advanced** ボタンを押すと、MySQL サーバの接続に使用するポートを設定することもできます。MySQL サーバとの接続をテストするには、サーバホスト名、ユーザ名、パスワードを設定し、**Test Connection** ボタンを押します。テストが失敗した場合は、入力した接続値が正しいか、ユーザとその MySQL サーバでの権限が設定されているか確認してください。
7. すべての設定をセットし、接続をテストして問題がなければ **OK** を押します。新しく作成された接続は、**Server Explorer** に表示されます。これで、標準 **Server Explorer** インターフェイスを通して MySQL サーバを使用することができます。

接続が正常に移動したら、すべての接続設定は保存されます。次回また Visual Studio を開くと、MySQL サーバへの接続が **Server Explorer** に表示されるので、また MySQL サーバへ接続することができます。

接続を変更または削除する場合は、**Server Explorer** コンテキスト メニューを使用してノードに対応します。現行の値を新しいものに上書きするだけで、設定を変更できます。接続の変更・削除は、そのオブジェクトのエディタが開いていない時に行うよう注意してください。データが失われる恐れがあります。

MySQL Visual Studio プラグインの使用

MySQL Visual Studio プラグインを利用して MySQL サーバを使用するには、Visual Studio 2005 を立ち上げ、**Server Explorer** を開き、必要な接続を選択します。MySQL Visual Studio プラグインの作業エリアは 3 つの構成になっています。

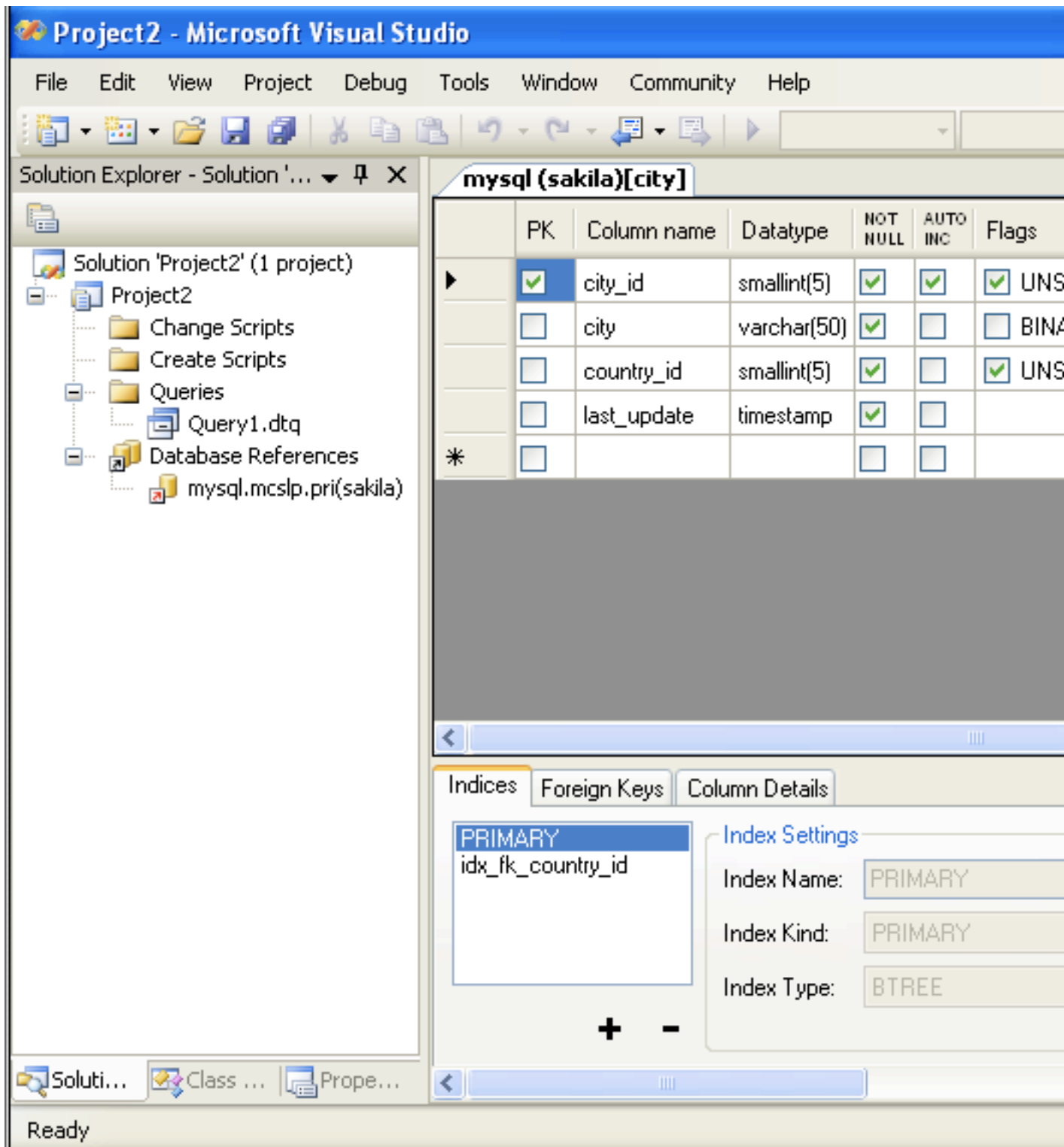


- データベース オブジェクト (テーブル、ビュー、内臓ルーチン、トリガ、ユーザ定義関数) が Server Explorer のツリーに表示されます。ここでは、オブジェクトを選択し、プロパティと定義を変更することができます。
- 選択したデータベース オブジェクトのプロパティは、Properties パネルに表示されます。プロパティのいくつかは、このウィンドウから直接変更できます。

- 編集パネルから、SQL 文や特定のオブジェクトの定義に直接アクセスすることができます。例えば、ストアド プロシージャ定義内の SQL 文もこのパネルで表示・変更できます。

テーブルの編集

Table Editor には、Server Explorer のテーブルタイプ ノードからマウス操作でアクセスすることができます。新しいテーブルを作成するには、Table ノード (接続ノード下) を右クリックし、コンテキスト メニューから Create Table コマンドを選択します。現存のテーブルを編集するには、変更したいテーブルのノードをダブル クリックするか、同じノードを右クリックしてコンテキスト メニューから Alter Table コマンドを選択します。どちらのコマンドでも Table Editor を開くことができます。



MySQL Visual Studio Plugin Table Editor は標準 Query Browser Table Editor と同様の方法で実行されますが、やや異なる部分があります。

Table Editor は以下の部位で構成されています。

- Columns Editor — カラムの作成、変更、削除

- Indexes タブ — テーブルおよびカラムのインデックス管理
- Foreign Keys タブ — 外部キーの構成
- Column Details タブ — 高度なカラム オプションの設定に使用
- Properties ウィンドウ — テーブルのプロパティの設定に使用

Table Editor での変更を保存するには、Visual Studio のメイン ツールバーにある Save または Save All ボタンを使うか、Ctrl+S を押してください。変更を保存する前に、MySQL データベース内の対象オブジェクトのアップデート承認を求める確認ダイアログが表示されます。

Column Editor

Column Editor を使用して、テーブル カラムの名称やデータ タイプ、デフォルト値、その他のプロパティを設定・変更することができます。各カラムのプロパティを設定するには、マウスでカラムを選択します。または、Tab キーと Shift+Tab を使ってグリッドからグリッドへ移動する方法もあります。

- カラムの名称やデータ タイプ、デフォルト値、コメントを設定および変更するには、該当のセルを選択し、値を編集します。
- フラグ型カラムのプロパティ (例: プライマリ キー、**NOT NULL**、自動インクリメントのもの、フラグ) を設定または設定解除するには、該当するチェックボックスをチェック、または未チェックにします。使用可能なカラム フラグはカラムのデータ タイプによって違うので注意してください。
- カラムやインデックス カラム、または外部キー カラムの順序変更を Column Editor で行うには、カラム グリッドの左側にあるセレクト カラムをクリックして、順序変更をしたいカラム全体を選択します。次に、Ctrl+Up (カラムを上に移動) や Ctrl+Down (カラムを下に移動) キーを使ってカラムを移動させます。
- カラムを削除するには、カラム グリッドの左側にあるカラム セレクトをクリックし、キーボードの Delete ボタンを押します。

Indexes タブ

インデックスの管理は Indexes タブを介して行われます。

- インデックスを加えるには、+ ボタンを押し、右側の Index Settings グループ ボックスでプロパティを設定します。インデックスの名称、種類、タイプ、インデックス カラムのセットを設定することができます。
- インデックスを取り除くには、リストから対象のインデックスを選び、- ボタンを押します。
- インデックスの設定を変更するには、対象のインデックスをリストから選択、インデックスの詳細情報は Index Settings パネルに表示されます。

テーブル カラムをドラッグ アンド ドロップでインデックス カラムにすることはできません。その代わりに、テーブルに新しいインデックス カラムを加え、Indexes タブ内に埋め込まれたエディタを使ってそれらのテーブル カラムをセットできます。

Foreign Keys タブ

Foreign Key の管理は、Foreign Keys タブを介して行われます。

- 外部キーを加えるには、+ ボタンを押し、Foreign Keys Settings パネルでプロパティを設定します。外部キーの名称、参照されるテーブルの名称、外部キー コラムと、更新と削除動作の設定をすることができます。
- 外部キーを取り除くには、該当の外部キーを選び、- ボタンを押します。
- 外部キーの設定を変更するには、該当の外部キーを選択し、Foreign Keys Settings パネルを使ってプロパティを変更します。
- 外部キーが変更されたら、MySQL Visual Studio プラグインがふたつの照会を形成します。第 1 の照会の変更されたキーの除去を、第 2 の照会新しい値の再作成を確認をします。この動作は、Bug #8377 と Bug #8919 を回避するためのものです。

注記

変更された値がなんらかの理由で矛盾し、第2の照会の失敗の原因となった場合は、影響のあるすべての外部キーが除去されます。その場合、MySQL Visual Studio プラグインは Table Editor 内にそれらのキーを新規としてマークするので、あとで再作成する必要があります。しかし、Table Editor を保存せずに閉じると、それらの外部キーは失われます。

Column Details タブ

Column Details タブはカラム オプションの設定に使用できます。Column Editor に表示されるメイン カラム プロパティに加え、Column Details タブでふたつの追加のプロパティ オプション、キャラクタセットと照合順序が設定できます。

Table Properties ウィンドウ

テーブル オプションと高度オプションはすべて同じタブにあります。すべてのテーブル オプションは、Visual Studio 2005 の Properties ウィンドウで参照・変更できます。

以下のテーブル プロパティが設定できます。

- Auto Increment
- Average Row Length
- Character Set
- Checksum for Rows
- Collation
- Comment
- Connection
- Data Directory
- Delay Key Updates
- Engine
- Index Directory
- Insert Method
- Maximum Rows
- Minimum Rows
- Name
- Pack Keys
- Password
- Row Format
- Union

これらのプロパティのいくつかは任意のテキスト値を設定することができ、その他は事前定義セットからの値を受けます。

Schema と Server のプロパティは読み取り専用です。

テーブル データの編集

Table Data Editor では、テーブルのデータの参照、作成、編集ができます。Table Data Editor は自動作成カラム付きの簡単なデータ グリッドとして実行されます。

Table Data Editor へアクセスするには、テーブルを提示しているノードが、Server Explorer のビューを右クリックします。ノードのコンテキスト メニューから、Browse か Edit Data コマンドを選択します。テーブルと更新可能なビューでは、このコマンドが編集モードで Table Data Editor を開きます。更新不可能なビューでは、このコマンドは読み取り専用モードで Table Data Editor を開きます。

編集モードの場合、表示されたテーブルの内容を直接変更することで、テーブル データを変更することができます。横列を追加するには、グリッドの最後列で希望の値を設定します。値を変更するには、適切なセルに新しい値を入力します。横列を削除するには、グリッド左側のセレクト カラムをクリックして対象列を選択し、Delete ボタンを押します。

Table Editor での変更を保存するには、Visual Studio のメイン ツールバーにある Save または Save All ボタンを使うか、Ctrl+S を押してください。確認ダイアログが、変更をデータベースに保存するか確認します。

ビューの編集

新しいビューを作成するには、Server Explorer の接続ノード下の View ノードを右クリックします。ノードのコンテキスト メニューから、Create View コマンドを選択します。このコマンドで SQL Editor が開きます。

現存のビューを編集するには、変更したいビューのノードをダブル クリックするか、同じノードを右クリックしてコンテキスト メニューから Alter Table コマンドを選択します。どちらのコマンドでも SQL Editor を開くことができます。

SQL Editor を使ってビュー定義を作成・変更するには、SQL Editor に適切な SQL 文を入力します。

注記

定義文のみを入力し、CREATE VIEW AS 序文は除外してください。

他のすべてのビュー プロパティは Properties ウィンドウで設定できます。それらのプロパティは以下です。

- Algorithm
- Check Option
- Definer
- Name
- Security Type

これらのプロパティのいくつかは任意のテキスト値を設定することができ、その他は事前定義セットからの値を受け入れます。

Is Updatable、Schema、Server プロパティは読み出し専用です。

変更を保存するには、Visual Studio のメイン ツールバーにある Save または Save All ボタンを使うか、Ctrl+S を押してください。確認ダイアログが、変更をデータベースに保存するか確認します。

ストアド プロシージャとファンクションの編集

新しいストアド プロシージャを作成するには、Server Explorer の接続ノード下の Stored Procedures ノードを右クリックします。ノードのコンテキスト メニューから、Create Routine コマンドを選択します。このコマンドで SQL Editor が開きます。

新しいストアド プロシージャを作成するには、Server Explorer の接続ノード下の Functions ノードを右クリックします。ノードのコンテキスト メニューから、Create Routine コマンドを選択します。

現存の内臓ルーチンを編集するには (プロシージャまたはファンクション)、変更したいルーチンのノードをダブル クリックするか、同じノードを右クリックしてコンテキスト メニューから Alter Routine コマンドを選択します。どちらのコマンドでも SQL Editor を開くことができます。

SQL Editor を使ってルーチン定義を作成・変更するには、標準 SQL を使用して SQL Editor にこの定義を入力します。

他のすべてのルーチン プロパティは Properties ウィンドウで設定できます。それらのプロパティは以下です。

- Comment

- Data Access
- Definer
- Is Deterministic
- Security Type

これらのプロパティのいくつかは任意のテキスト値を設定することができ、その他は事前定義セットからの値を受け入れます。

また、標準 `CREATE PROCEDURE` か `CREATE FUNCTION` 文を使用して、すべてのオプションを SQL Editor から直接設定することができます。しかしそれよりも、`Properties` ウィンドウの使用をお勧めします。

注記

ルーチン定義には決して `CREATE` 序文を加えないでください。

`Properties` ウィンドウの `Name`、`Schema`、`Server` プロパティは読み出し専用です。SQL エディタでプロシージャ名を設定または変更してください。

変更を保存するには、Visual Studio のメイン ツールバーにある `Save` または `Save All` ボタンを使うか、`Ctrl+S` を押してください。確認ダイアログが、変更をデータベースに保存するか確認します。

トリガの編集

新しいトリガを作成するには、トリガを新たに加えたいテーブルのノードを右クリックします。ノードのコンテキストメニューから、`Create Trigger` コマンドを選択します。このコマンドで SQL Editor が開きます。

既存のトリガを編集するには、変更したいトリガのノードをダブル クリックするか、同じノードを右クリックしてコンテキストメニューから `Alter Trigger` コマンドを選択します。どちらのコマンドでも SQL Editor を開くことができます。

SQL Editor を使ってトリガ定義を作成・変更するには、標準 SQL を使用して SQL Editor にトリガ文を入力します。

注記

`FOR EACH ROW` 句の後にある `CREATE TRIGGER` 問い合わせの一部の、トリガ文のみを入力してください。

他のすべてのトリガ プロパティは `Properties` ウィンドウで設定できます。それらのプロパティは以下です。

- Definer
- Event Manipulation
- Name
- Timing

これらのプロパティのいくつかは任意のテキスト値を設定することができ、その他は事前定義セットからの値を受け入れます。

`Properties` ウィンドウの `Event Table`、`Schema`、`Server` プロパティは読み出し専用です。

変更を保存するには、Visual Studio のメイン ツールバーにある `Save` または `Save All` ボタンを使うか、`Ctrl+S` を押してください。確認ダイアログが、変更をデータベースに保存するか確認します。

ユーザ定義関数 (UDF) の編集

新しいユーザ定義関数 (UDF) を作成するには、Server Explorer の接続ノード下の UDFs ノードを右クリックします。ノードのコンテキストメニューから、`Create UDF` コマンドを選択します。このコマンドで UDF Editor が開きます。

既存の UDF を編集するには、変更したい UDF のノードをダブル クリックするか、同じノードを右クリックしてコンテキストメニューから `Alter UDF` コマンドを選択します。どちらのコマンドでも UDF Editor を開くことができます。

UDF エディタでは、プロパティ パネルを介して次のプロパティを設定することができます。

- Name
- So-name (DLL name)
- Return type
- Is Aggregate

Properties ウィンドウの Server プロパティは読み出し専用です。

変更を保存するには、Visual Studio のメイン ツールバーにある **Save** または **Save All** ボタンを使うか、Ctrl+S を押してください。確認ダイアログが、変更をデータベースに保存するか確認します。

データベース オブジェクトの削除

テーブル、ビュー、内臓ルーチン、トリガ、UDF は各コンテキスト メニューからの適切な **Drop** コマンド、**Drop Table**、**Drop View**、**Drop Routine**、**Drop Trigger**、**Drop UDF** で削除することができます。

確認ダイアログが、対象の削除問い合わせを実行するか確認を求めます。

複数のオブジェクトの削除はサポートされていません。

データベース オブジェクトの複製

テーブル、ビュー、内臓ルーチン、ファンクションは各コンテキスト メニューからの、適切な **Clone** コマンド、**Clone Table**、**Clone View**、**Clone Routine** で複製することができます。複製コマンドは新しいオブジェクトのために対応エディタを開きます。Table Editor はテーブルの複製、SQL Editor はビューまたはルーチンの複製に対応します。

変更を保存するには、Visual Studio のメイン ツールバーにある **Save** または **Save All** ボタンを使うか、Ctrl+S を押してください。確認ダイアログが、変更をデータベースに保存するか確認します。

Visual Studio プラグインのサポート

感想や、バグを発見した場合は、MySQL バグ トラッキング システム (<http://bugs.mysql.com>) から問題の報告や提案をお寄せください。

Visual Studio プラグイン FAQ

Questions

- [24.3.4.1.1: \[1387\]](#) 接続を作成する時に接続詳細を入力すると、接続ウィンドウが閉じてしまいます。

Questions and Answers

24.3.4.1.1: 接続を作成する時に接続詳細を入力すると、接続ウィンドウが閉じてしまいます。

Connector/NET 5.0.2. 以前のバージョンには問題があり、Connector/NET 1.0.x は作動しないと報告されています。それらのバージョンがインストールされている場合、またはそれ以前のバージョンからアップグレードした場合は、Connector/NET を完全にアンインストールし、Connector/NET 5.0.2. をインストールしてください。

MySQL Connector/J

MySQL は、Java プログラム言語で開発されたクライアント アプリケーションに、JDBC ドライバを介する接続を提供しており、それを MySQL Connector/J と呼びます。

MySQL Connector/J は JDBC-3.0 Type 4 ドライバで、純粋 Java です。JDBC 仕様のバージョン 3.0 を導入しており、また、MySQL プロトコルを仕様して MySQL サーバと直接通信を行います。

JDBC はそのみで便利なものですが、このマニュアルの最初の数セクションを読んでみても理解しがたい場合は、無償 JDBC の使用は分かりやすい問題のみにとどめ、JDBC でしばしば必要になる反復作業や、より重要な作業には、[Hibernate](#)、[Spring's JDBC templates](#)、または [Ibatis SQL Maps](#) などの一般的なパースシステム フレームワークを使用することをお勧めします。

このセクションは、JDBC の完全なチュートリアルとして書かれたものではありません。JDBC の使用に関するさらなる情報をお求めの場合は、次のオンライン チュートリアルにより詳しい情報が掲載されています：

- [JDBC Basics](#) — JDBC の初級トピックを収録した、Sun からのチュートリアル。
- [JDBC Short Course](#) — Sun と JGuru からの、より詳細なチュートリアル。

Connector/J バージョン

現在利用できる Connector/J には 3 バージョンあります：

- Connector/J 3.0 はコア機能を提供し、MySQL 3.x または MySQL 4.1 サーバとの接続を条件に設計されていますが、それ以降のバージョンの MySQL にも基本対応が可能です。Connector/J 3.0 はサーバ側プリペアド ステートメントをサポートしていません。また、MySQL 4.1 以降のいかなる機能もサポートしていません。
- Connector/J 3.1 は、MySQL 4.1 と MySQL 5.0 のサーバへの接続を条件に設計され、MySQL 5.0 の分散トランザクション (XA) サポートを除いた全機能をサポートしています。
- Connector/J 5.0 は、分散トランザクション (XA) サポートを含む、Connector/J 3.1 のすべての機能をサポートしています。

現在推奨されている Connector/J のバージョンは 5.0 です。このガイドでは、コネクタの 3 バージョンすべてをカバーし、特定のオプションへの設定に関する注意を記載しています。

サポートされている Java のバージョン

MySQL Connector/J は、以下を含む `ava-2 JVM` をサポートしています：

- JDK 1.2.x (Connector/J 3.1.x 以前のみ)
- JDK 1.3.x
- JDK 1.4.x
- JDK 1.5.x

ソース配布物を利用して ([開発ソースツリーからのインストール](#) 参照) ソースから Connector/J を構築する場合は、Connector パッケージのコンパイルに JDK 1.4.x 以降を使用する必要があります。

MySQL Connector/J は、JDK-1.1.x または JDK-1.0.x をサポートしていません。

`java.sql.Savepoint` の導入により、クラス検証機能がオフ (Java ランタイムに `-Xverify:none` オプションを設定) にされていない限り、Connector/J 3.1.0 以降をバージョン 1.4 より前の JDK で実行することはできません。これはクラス検証機能が、セーブポイント機能を開発者が使用しない限りドライバからアクセスを受けることはないにもかかわらず、`java.sql.Savepoint` にクラス定義をロードしようとするためです。

Connector/J 3.1.0 以降で提供されているキャッシング機能もまた、JDK-1.4.0. から利用可能になった `java.util.LinkedHashMap` に頼るため、バージョン 1.4 より古い JVM では利用できません。

Connector/J のインストール

Connector/J パッケージのインストールは、バイナリ配布物がソース配布物を使用して行うことができます。バイナリ配布物の使用はより簡単なインストールの方法です。一方、ソース配布物を使用する場合は、インストールのより複雑なカスタマイズが可能になります。どちらの方法でも、Connector/J ロケーションを手動で Java `CLASSPATH` に追加する必要があります。

バイナリ配布物から Connector/J をインストールする

インストールの 2 種類の方法のうち、Connector/J パッケージの バイナリ配布物を使用する方法は極めて簡単です。バイナリ配布物は Tar/Gzip または Zip ファイルの状態で購入することができます。それをまず適当なロケーションに抽出し、オプションとして、`CLASSPATH` を変更することで、パッケージの情報を利用可能にします ([ドライバのインストールと CLASSPATH の構成](#) 参照)。

MySQL Connector/J は、ソース、クラス ファイル、[mysql-connector-java-\[version\]-bin.jar](#) と名付けられた JAR アーカイブ、そして Connector/J 3.1.8 からは、[mysql-connector-java-\[version\]-bin-g.jar](#) と名付けられたファイルにあるドライバのデバッグ ビルドを含んだ、.zip または .tar.gz アーカイブとして配布されています。

Connector/J 3.1.9 からは、JAR ファイルを構成する .class ファイルは、ドライバ JAR ファイルの一部としてのみ含まれています。

ドライバのデバッグ ビルドは実稼動環境で実行するようには設計されておらず、使用するとパフォーマンスに悪影響が出るため、問題やバグを MySQL AB に報告する際に指示されない限り使用しないでください。また、デバッグ バイナリは、Connector/J 配布物に添付された [src/lib/aspectjrt.jar](#) ファイルにある、AspectJ ランタイム ライブラリに依存します。

開発者は、適切なグラフィカル ユーティリティ、またはコマンドライン ユーティリティを使用して、配布物を未保管にする必要があります (例えば、.zip アーカイブには WinZip、そして .tar.gz アーカイブには tar)。配布物には長いファイルネームがある場合があるので、GNU tar アーカイブ フォーマットを使用しています。配布物の .tar.gz 型を解凍するには、GNU tar (または GNU tar アーカイブ フォーマットを理解するアプリケーション) を使用する必要があります。

ドライバのインストールと CLASSPATH の構成

配布物アーカイブを抽出し終えたら、[CLASSPATH](#) 環境変数にフルパスを追加するか、JVM を起動する際にコマンドライン スイッチ -cp で直接指定して、クラスパスに [mysql-connector-java-\[version\]-bin.jar](#) を据えることで、ドライバをインストールすることができます。

JDBC DriverManager でドライバを使用する場合は、[com.mysql.jdbc.Driver](#) を java.sql.Driver を実行するクラスとして利用します。

Unix、Linux、または Mac OS X 下の [CLASSPATH](#) 環境変数を、ユーザのために .profile、.login、または他のログイン ファイル内でローカルに設定することができます。また、グローバル /etc/profile ファイルを編集すれば、グローバルに設定することも可能です。

例えば、C シェル (csh、tcsh) 下では、次を使用して Connector/J ドライバを [CLASSPATH](#) に追加します：

```
shell> setenv CLASSPATH /path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

また、Bourne 互換シェル (sh、ksh、bash) では：

```
export set CLASSPATH=/path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Windows 2000、Windows XP、そして Windows Server 2003 内では、System コントロール パネルを通して環境変数を設定する必要があります。

Tomcat か JBoss などのアプリケーション サーバと MySQL Connector/J を使用する場合、ほとんどのアプリケーション サーバが [CLASSPATH](#) 環境変数を無視するため、サードパーティ クラス ライブラリの構成に関しては各ベンダーの説明資料をお読みください。J2EE アプリケーション サーバの構成例は、[J2EE および 他の Java フレームワークで Connector/J を使用する](#) を参照してください。ただし、特定のアプリケーション サーバに対する JDBC 接続プール構成情報の信頼できるソースは、そのアプリケーション サーバの資料になります。

サーブレット や JSP を開発していて、使用するアプリケーション サーバが J2EE に適合している場合、ドライバの .jar ファイルを webapp の WEB-INF/lib サブディレクトリに入れることができます。それが J2EE ウェブ アプリケーションのサードパーティ クラス ライブラリの標準ロケーションです。

また、使用する J2EE アプリケーション サーバでサポートまたは要求されていれば、[com.mysql.jdbc.jdbc2.optional](#) パッケージの [MysqlDataSource](#) か [MysqlConnectionPoolDataSource](#) を使用することもできます。Connector/J 5.0.0 からは、[javax.sql.XADataSource](#) インターフェイスは [com.mysql.jdbc.jdbc2.optional.MysqlXADataSource](#) クラスを介して導入され、MySQL サーバのバージョン 5.0 と併用すると、XA 分散トランザクションをサポートします。

様々な [MysqlDataSource](#) クラスが、以下のパラメータを (標準セット ミューテータを通して) サポートしています：

- user
- password
- serverName (フェイル オーバ ホストについて前セクションを参照)
- databaseName

- port

旧バージョンからのアップデート

MySQL AB はアップデートの手順を可能な限り簡潔に保つよう努力しておりますが、どのソフトウェアでもあるように、新しい機能のサポート、現存の機能の改良、または新たな基準の実施などのために、新しいバージョンでは変更があることもあります。

このセクションでは、ユーザが Connector/J のバージョンをアップグレードする (または JDBC 機能に関連して、MySQL サーバを新バージョンにアップグレードする) 際の注意を記述します。

MySQL Connector/J 3.0 から 3.1 へのアップグレード

Connector/J 3.1 は、Connector/J 3.0 にできるだけ後方互換性を持つよう設計されています。主な変更は MySQL-4.1 以降の新しい機能に特定されており、Unicode 文字セット、サーバ側プリペアド ステートメント、エラー メッセージでサーバによって戻される SQLState、構成プロパティを介して有効または無効にできる様々な性能強化、などがその対象です。

- Unicode 文字セット — この MySQL の新機能については、次のセクション、および [9章キャラクタセットサポート](#) をご参照ください。構成ができない場合は通常、[Illegal mix of collations](#) に似たメッセージでエラーが表示されます。
- サーバ側プリペアド ステートメント — Connector/J 3.1 は、サーバ側プリペアド ステートメントが利用可能な場合、それらを自動的に検出して使用します (MySQL サーバ バージョン 4.1.0 以降)。

バージョン 3.1.7 からは、[Connection.prepareStatement\(\)](#) のすべての変形を介して準備している SQL をドライバがスキャンし、サーバ側での準備がサポートされているタイプのステートメントかを判断します。そして、サーバでサポートされていない場合は、クライアント側のエミュレートされるプリペアド ステートメントとして準備されます。この機能は、`emulateUnsupportedPstmts=false` を JDBC URL で渡すことで無効にすることができます。

アプリケーションとサーバ側プリペアド ステートメントとの間に問題がある場合、接続プロパティ `useServerPrepStmts=false` で、4.1.0 より前の MySQL サーバでまだ使用されている旧式のクライアント側のエミュレートされたプリペアド ステートメント コードに戻すこともできます。

- Datetimes すべてゼロのコンポーネント (0000-00-00 ...) — それらの値は、Java では確実に表示されません。Connector/J 3.0.x は ResultSet から読み取る時はいつもそれらを NULL に変換します。

Connector/J 3.1 は、これらの値が JDBC と SQL の標準に沿う最も適切な動作として検出される場合に、デフォルトとして例外を投入します。この動作は `zeroDateTimeBehavior` 構成プロパティを使用して変更することができます。許容値は以下：

- `exception` (デフォルト) - SQLState の [S1009](#) で SQLException を投入。
- `convertToNull` ->データの代わりに NULL を戻す。
- `round` - データを、[0001-01-01](#) という最も近い整数に丸める。

Connector/J 3.1.7 からは、`noDatetimeStringSync=true` (デフォルト値は `false`) を介して [ResultSet.getString\(\)](#) をこの動作から切り離すことができ、それによって未修正のすべてゼロの値を String として取り出すことができます。また、これはどんなタイムゾーン変換の使用も除外し、そのためドライバは `noDatetimeStringSync` と `useTimezone` を同時に有効にすることを拒否しますのでご注意ください。

- 新しい SQLState コード — Connector/J 3.1 は、MySQL サーバ (サポートされている場合) によって戻された SQL:1999 SQLState コードを使用します。これは Connector/J 3.0 が使用する旧式の X/Open ステート コードとは異なります。MySQL-4.1.0 より前の MySQL サーバに接続している場合 (最も古いバージョンは、SQLState をエラーコードの一部として戻します)、ドライバは設定済みのマッピングを使用します。構成プロパティ `useSqlStateCodes=false` を使うと、以前のマッピングに戻すことが可能です。
- [ResultSet.getString\(\)](#) — [ResultSet.getString\(\)](#) を BLOB カラムで呼び出すと、BLOB の String 表現ではなく、`byte[]` アレイのアドレスが戻るようになりました。BLOB は文字セットを持たないため、データの紛失や破損の心配なく `java.lang.Strings` に変換することができます。

ストリングを LOB 動作で MySQL に保存するには、ドライバが `java.sql.Clob` として扱う TEXT タイプのひとつを使用します。

- デバッグ ビルド — Connector/J 3.1.8 からは、[mysql-connector-java-\[version\]bin-g.jar](#) と名付けられたファイルのドライバのデバッグ ビルドは、[mysql-connector-java-\[version\]-bin.jar](#) という名の通常バイナリの jar ファイルに沿って出荷されます。

Connector/J 3.1.9 からは、.class ファイルを細分化された状態では出荷なくなり、それらはドライバと出荷する JAR アーカイブでのみ利用できます。

ドライバのデバッグ ビルドは実稼動環境で実行するようには設計されておらず、使用するとパフォーマンスに悪影響が出るため、問題やバグを MySQL AB に報告する際に指示されない限り使用しないでください。また、デバッグ バイナリは、Connector/J 配布物に添付された [src/lib/aspectjrt.jar](#) ファイルにある、Aspect/J ランタイム ライブラリに依存します。

MySQL サーバ 4.1 以降へのアップグレードに際する JDBC の問題

- UTF-8 文字エンコードの使用 - MySQL サーバ 4.1 より古いバージョンでは、UTF-8 文字エンコードはサーバではサポートされていませんでしたが、JDBC ドライバでは使用が可能で、サーバの latin1 テーブルに複数の文字セットを保存することができました。

MySQL-4.1 からは、この機能は廃止になります。アプリケーションがこの機能を利用しており、MySQL サーバ 4.1 以降での公式の Unicode 文字サポートを使用することができない場合は、次のプロパティを接続 URL に加えてください：

[useOldUTF8Behavior=true](#)

- サーバ側プリペアド ステートメント - Connector/J 3.1 は、サーバ側プリペアド ステートメントが利用可能な場合、それらを自動的に検出して使用します (MySQL サーバ バージョン 4.1.0 以降)。アプリケーションとサーバ側プリペアド ステートメントとの間に問題がある場合、次の接続プロパティで、4.1.0 より前の MySQL サーバでまだ使用されている旧式のクライアント側のエミュレートされたプリペアド ステートメント コードに戻すこともできます：

[useServerPrepStmts=false](#)

開発ソースツリーからのインストール

注意： 当社の新しいコードのテストにご興味がおありの場合は、このセクションをお読みください。単に MySQL Connector/J をシステム上で稼働させるだけであれば、標準のリリース配布物を使用してください。

開発ソースツリーから MySQL Connector/J をインストールするには、次の前提条件が満たされている必要があります：

- リポジトリからのソースを確認する副バージョン (<http://subversion.tigris.org/> で入手可)。
- Apache Ant バージョン 1.6 以降 (<http://ant.apache.org/> で入手可)。
- JDK-1.4.2 以降。MySQL Connector/J は古い JDK にインストールすることもできますが、ソースからコンパイルするためには JDK-1.4.2 以降が必要になります。

MySQL Connector/J 用の副バージョン ソースコード リポジトリは <http://svn.mysql.com/svnpublic/connector-j> にあります。普通、リポジトリは MySQL Connector/J 用のすべてのブランチおよびタグを含んでおり、かなりの大きさになるため、全リポジトリの確認はしません。

MySQL Connector/J の特定のブランチを確認し、コンパイルするには、次の手順に従ってください：

1. この資料の作成時点では、Connector/J のアクティブ ブランチは 3 つあります :[branch_3_0](#)、[branch_3_1](#) および [branch_5_0](#)。次のコマンドで、所望のブランチから最新コードを確認 ([major] と [minor] に適切なバージョン番号を入力)。

```
shell> svn co »
http://svn.mysql.com/svnpublic/connector-j/branches/branch\_\[major\]\_\[minor\]/connector-j
```

これで、希望のブランチの最新ソースを含んだ現行ディレクトリに、[connector-j](#) が作成されます。

2. [connector-j](#) ディレクトリにロケーションを変更し、現行の作業ディレクトリにします：

```
shell> cd connector-j
```

3. 次のコマンドを発行して、ドライバをコンパイルし、インストールに適した `.jar` ファイルを作成：

```
shell> ant dist
```

これで、現行ディレクトリに `build` ディレクトリが作成され、すべてのビルドの出力先になります。構築に使用しているソースのバージョン番号を含む `build` ディレクトリに、ディレクトリが作成されます。このディレクトリはソース、コンパイルされた `.class` ファイル、そして開発に適した `.jar` ファイルを含みます。完全にパッケージされた配布物を含む、他に可能性のあるターゲットには、次のコマンドを発行します：

```
shell> ant --projecthelp
```

4. 新たに作成された、JDBC ドライバを含む `.jar` ファイルは、ディレクトリ `build/mysql-connector-java-[version]` に配置されます。

新たに作られた JDBC ドライバをインストールし、同時に、[ドライバのインストールと CLASSPATH の構成](#)にある手順で MySQL からダウンロードしたバイナリ `.jar` ファイルをインストールします。

Connector/J 例

Connector/J の使用例は、この資料の各所に含まれています。このセクションでは、これらの例の要約とリンクを紹介します。

- [例24.1 「DriverManager から接続を取得する」](#)
- [例24.2 「java.sql.Statement を使用して SELECT クエリを実行する」](#)
- [例24.3 「ストアド プロシージャ」](#)
- [例24.4 「Connection.prepareCall\(\) の使用」](#)
- [例24.5 「出力パラメータの登録」](#)
- [例24.6 「CallableStatement 入力パラメータの設定」](#)
- [例24.7 「結果と出力パラメータの値を呼び出す」](#)
- [例24.8 「Statement.getGeneratedKeys\(\) を使った AUTO_INCREMENT カラム値の呼び出し」](#)
- [例24.9 「SELECT LAST_INSERT_ID\(\) を使った AUTO_INCREMENT カラム値の呼び出し」](#)
- [例24.10 「Updatable ResultSets 内の AUTO_INCREMENT カラム値の呼び出し」](#)
- [例24.11 「J2EE アプリケーション サーバで接続プールを使用」](#)
- [例24.12 「リトライ ロジックとのトランザクション例」](#)

Connector/J (JDBC) の参考

このマニュアルの当セクションには、MySQL Connector/J の参考資料が記載されています。中には、Connector/J 構築プロセス中に自動的に生成されたものも含まれています。

Connector/J の Driver/Datasource クラス名、URL シンタックス、および構成プロパティ

MySQL Connector/J に `java.sql.Driver` を導入するクラスの名前は、`com.mysql.jdbc.Driver` です。また、`org.gjt.mm.mysql.Driver` クラス名は、MM.MySQL との後方互換性を保つのに使用できます。ドライバを登録する際、またそれ以外では MySQL Connector/J を使用するようソフトウェアを構成する時に、このクラス名を使用します。

MySQL Connector/J 用の JDBC URL フォーマットは以下。角括弧 ([、]) で閉じられたアイテムはオプションです。

```
jdbc:mysql://[host][:failoverhost...][:port]/[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

ホスト名が指定されていない場合、デフォルトで `127.0.0.1` になります。ポートが指定されていない場合は、デフォルトで `3306` になり、それが MySQL サーバのデフォルトのポート番号です。

```
jdbc:mysql://[host:port],[host:port].../[database] »
[?propertyName1]=[propertyValue1][&propertyName2]=[propertyValue2]...
```

データベースが指定されていなければ、デフォルトのデータベースなしで接続が作成されます。その場合、Connection インスタンスにある `setCatalog()` メソッドを呼び出すか、データベース名 (例: `SELECT dbname.tablename.colname FROM dbname.tablename...`) を使ってテーブル名を厳密に指定する必要があります。接続に使用するデータベースを指定しない方法は、通常、GUI データベース マネージャのように、複数のデータベースと作動するツールを構築する時のみに効果があります。

MySQL Connector/J はフェイルオーバー サポートを有しています。これによって、ドライバはどんなスレーブ ホスト番号にもフェイルオーバーでき、かつ読み込み専用クエリを実行することが可能になります。フェイルオーバーは、トランザクションの進行中は確実でないため、接続が `autoCommit(true)` 状態にある時にのみ起こります。ほとんどのアプリケーション サーバと接続プールは、毎回、トランザクション/接続の使用の最後に `autoCommit` を `true` に設定します。

フェイルオーバー機能は次の動作を行います：

- URL プロパティ `autoReconnect` が `false` の場合：フェイルオーバーは接続初期化でのみ起こり、フェイルバックはドライバが、最初のホストが再度利用可能になったと判断したときに起こります。
- URL プロパティ `autoReconnect` が `true` の場合：フェイルオーバーはドライバが、接続不能と判断した場合に起こり (すべてのクエリの前)、ホストが再度利用可能になったと判断した時、最初のホストにフェイルバックします (`queriesBeforeRetryMaster` クエリが発行された後)。

どちらのケースでも、"フェイルオーバーされた" サーバに接続するたびに、接続は読み取り専用の状態に設定され、データを改変するクエリには例外が投入されます (クエリが MySQL サーバで処理されることは絶対にありません)。

構成プロパティは、Connector/J がどのように MySQL サーバへの接続を行うか定義します。特別な指示がない限り、プロパティを DataSource オブジェクト、または Connection オブジェクトに設定することができます。

構成プロパティは、以下のどれかの方法で設定することができます：

- `java.sql.DataSource` の MySQL 実装の `set*()` メソッドを使用する (`java.sql.DataSource` の実装を使用する際に推奨される方法) 。
 - `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`
 - `com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource`
- `DriverManager.getConnection()` または `Driver.connect()` に渡された `java.util.Properties` インスタンスの値/キーペアとして。
- `java.sql.DriverManager.getConnection()`、`java.sql.Driver.connect()`、または `javax.sql.DataSource setURL()` メソッドの MySQL 実装に供給された URL 内の JDBC URL パラメータとして。

注意： JDBC URL の構成に使用する仕組みが XML ベースの場合、アンド記号 (&) は XML の予約語であるため、構成パラメータを区切るには、XML 文字リテラル `&` を使う必要があります。

次の表はプロパティの一覧です：

接続/認証.

プロパティ名	定義	デフォルト値	対応開始したバージョン
user	接続のユーザ名義		すべて
password	接続に使用するパスワード		すべて
socketFactory	サーバにソケット接続を作成するのに、ドライバが使用するべきクラスの名称。このクラスは、インターフェイス 'com.mysql.jdbc.SocketFactory' を実行し、パブリック no-args コンストラクタを有する必要があります。	com.mysql.jdbc.StandardSocketFactory	1.2.18

connectTimeout	ソケット接続のタイムアウト (ミリ秒単位)。0 はタイムアウト無し。JDK-1.4 以降でのみ有効。デフォルトで '0' になります。	0	3.0.1
socketTimeout	ネットワーク ソケット動作のタイムアウト (デフォルトの 0 ではタイムアウト無し)。	0	3.0.1
useConfigs	URL の解析、またはユーザ指定プロパティの適用の前に、構成プロパティのコンマ区切りのリストをロード。これらの構成は、資料の '構成' で説明されています。		3.1.5
interactiveClient	WAIT_TIMEOUT でなく INTERACTIVE_TIMEOUT を基に、MySQL にタイムアウトを指示する CLIENT_INTERACTIVE フラグを設定。	false	3.1.0
propertiesTransform	接続を試みる前にドライバに渡される URL プロパティの改変に、ドライバが使用する com.mysql.jdbc.ConnectionPropertiesTransform の実行。		3.1.4
useCompression	サーバとの通信中に zlib 圧縮を使用 (true/false) ? デフォルトで 'false' になる。	false	3.0.17

高可用性とクラスタ

プロパティ名	定義	デフォルト値	対応開始したバージョン
autoReconnect	ドライバは膠着した接続、および/または切断された接続の再確立を試みますか？有効になれば、ドライバは停滞または切断された接続で発行された、現行のトランザクションに属するクエリに例外を投入しますが、接続の試みは、新たなトランザクションでの接続で発行される次のクエリの前に行います。この機能の使用は、アプリケーションが SQLExceptions プロパティを扱わない場合、セッション状態とデータの一貫性に対する副次的な悪影響をひきおこします。また、切断され停滞した接続プロパティの結果として、SQLExceptions を扱うアプリケーションを構築できない時にのみ使用するように設計されているため、利用はお勧めできません。他の方法として、MySQL サーバ変数 "wait_timeout" を、デフォルトの 8 時間でなく、高い値にすることが考えられます。	false	1.1
autoReconnectForPools	接続プールに適した再接続の手段を使用 (デフォルトで 'false' になる)。	false	3.1.3
failOverReadOnly	autoReconnect モードでフェイルオーバーが起こる場合、接続は 'read-only' に設定しますか？	true	3.0.12
reconnectAtTxEnd	autoReconnect が true に設定されている場合、ドライバは毎トランザクションの最後に再接続を試みますか？	false	3.0.10
roundRobinLoadBalance	autoReconnect が有効で、failoverReadOnly が false の場合、ラウンドロビン方式で接続するホストを選びますか？	false	3.1.2
queriesBeforeRetryMaster	フェイルオーバーが起きた際、マスタへフォールバックする前に発行されるクエリの数 (マルチホストのフェイルオーバーを使用する場合) 。'queriesBeforeRetryMaster' または 'secondsBeforeRetryMaster' の、先に条件に合った方がマスタへの接続を促します。デフォルトで 50 になります。	50	3.0.2
secondsBeforeRetryMaster	フェイルオーバーが起こる時、ドライバがマスタサーバへの接続を試みるまでの待機時間は？'queriesBeforeRetryMaster' または 'secondsBeforeRetryMaster' の、先に条件に合った方がマ	30	3.0.2

	スタへの接続を促します。時間は秒単位で、デフォルトで 30 になります。		
enableDeprecatedAutoreconnect	自動再接続機能は、バージョン 3.2 からは使用停止になり、バージョン 3.3 では削除予定。このプロパティを 'true' に設定し、構成中の機能のチェックを無効にします。	false	3.2.1
resourceId	URL で使用されているホスト名からドライバが値を判断できない場合に、XAResource.isSameRM() に使用される、このデータソースまたはコネクションが接続されているソースを特定する大域的に一意な名称。		5.0.1

セキュリティ.

プロパティ名	定義	デフォルト値	対応開始したバージョン
allowMultiQueries	ひとつのステートメントの間で、複数のクエリを区切るために ';' の使用を可能にします (true/false、デフォルトの 'false' になる)	false	3.1.1
useSSL	サーバとの通信中に SSL を使用 (true/false)。デフォルトの 'false' になります。	false	3.0.2
requireSSL	useSSL=true の場合、SSL 接続は必須？ (デフォルトの 'false' になる)。	false	3.1.0
allowUrlInLocalInfile	ドライバは 'LOAD DATA LOCAL INFILE' ステートメントで URL を許可しますか？	false	3.1.4
paranoid	エラーメッセージに機密情報が表示されるのを防ぐ対策を取り、可能な場合は機密データを持つデータ構造を消去しますか？ (デフォルトの 'false' になる)	false	3.0.1

性能拡張.

プロパティ名	定義	デフォルト値	対応開始したバージョン
metadataCacheSize	cacheResultSetMetaData が 'true' に設定されている場合の、cacheResultSetMetadata へのクエリの数 (デフォルトは 50)。	50	3.1.1
prepStmtCacheSize	プリペアド ステートメントのキャッシングが有効な場合に、キャッシュされるプリペアド ステートメントの数は？	25	3.0.10
prepStmtCacheSqlLimit	ステートメントのキャッシングが有効な場合、ドライバが解析をキャッシュする最大の SQL は？	256	3.0.10
useCursorFetch	ステートメント上の MySQL > 5.0.2、および setFetchSize() > 0 に接続している場合、そのステートメントは行の取り出しにカーソル ベースのフェッチを使用しますか？	false	5.0.0
blobSendChunkSize	ServerPreparedStatements を介して BLOB/CLOB を送る時に使用するチャンク。	1048576	3.1.9
cacheCallableStmts	ドライバは CallableStatements の解析過程をキャッシュしますか？	false	3.1.2
cachePrepStmts	ドライバは、クライアント側プリペアド ステートメントの PreparedStatements、サーバ側プリペアドの適合性の "check"、そしてサーバ側プリペアド ステートメントそのものの解析過程をキャッシュしますか？	false	3.0.10

cacheResultSetMetadata	ドライバは Statements および PreparedStatements の ResultSetMetaData をキャッシュしますか？ (Req. JDK-1.4+、true/false、デフォルトは 'false')	false	3.1.1
cacheServerConfiguration	ドライバは、毎 URL ベースで 'SHOW VARIABLES' および 'SHOW COLLATION' の結果をキャッシュしますか？	false	3.1.5
defaultFetchSize	ドライバはこの値を使用して、新しく作成されたステートメントのすべてで setFetchSize(n) を呼び出します。	0	3.1.9
dontTrackOpenResources	JDBC の仕様は、ドライバが自動的にリソースを追跡し閉じることを必須としています。もしアプリケーションがステートメントまたは結果セットで上手く close() を明示的に呼び出していない場合は、これがメモリ リークの原因になることがあります。このプロパティを true に設定することで、その制約を緩和することができ、アプリケーションによってはメモリ効率を上げることにもなります。	false	3.1.7
dynamicCalendars	デフォルトのカレンダーが必要な場合、ドライバはそれを取り出す、または 毎接続/セッションでキャッシュしますか？	false	3.1.5
elideSetAutoCommits	MySQL-4.1 以降を使用している場合、サーバの状態が Connection.setAutoCommit(boolean) の要求した状態と合わない時に、ドライバは 'set autocommit=n' クエリのみを発行しますか？	false	3.1.3
holdResultsOpenOverStatementClose	JDBC 仕様で要求されているように、ドライバは Statement.close() で結果セットを閉じますか？	false	3.1.7
locatorFetchBufferSize	'emulateLocators' が 'true' に構成されている場合、getBinaryInputStream の BLOB データをフェッチする時に使用するバッファのサイズは？	1048576	3.2.1
rewriteBatchedStatements	executeBatch() が呼び出された時に、ドライバは多値インサートへの INSERT のために、プリペアドステートメントの書き換えをすると共に、マルチクエリを ("allowMultiQueries" の設定に関係なく) 使用しますか？ 普通の java.sql.Statements を使用していて、コードがインプットを正しくサニタイズしない場合、これは SQL インジェクションになりえることをご留意ください。プリペアドステートメントでは、サーバ側プリペアドステートメントは現在この書き換えオプションを利用できません。また、PreparedStatement.set*Stream() の使用時にストリーム長を指定していない場合、ドライバはバッチ当たりのパラメータの適した数を判断することができず、結果パケットが大きすぎるというエラーが出るおそれがあるので注意してください。これらの書き換えられたステートメントの Statement.getGeneratedKeys() は、全体のバッチが INSERT 文を含む場合にしか作動しません。	false	3.1.13
useFastIntParsing	過剰なオブジェクトの作成を避けるために内部 String->Integer 変換ルーチンを使用？	true	3.1.4
useJvmCharsetConverters	1 バイト文字セットをテーブルのルックアップ表を使用するより、JVM に組み込まれた文字エンコード ルーチンを常に使用する？ (このデフォルトである "true" は、より新しい JVM に適しています)	true	5.0.1
useLocalSessionState	ドライバは、データベースを問い合わせるより、Connection.setAutoCommit() および Connection.setTransactionIsolation() によって設定された、自動コミットとトランザクションの隔離の内部値を参照しますか？	false	3.1.7

useReadAheadInput	サーバから読み取る時、より新しい、最適化された非ブロッキングの、バッファされた入力ストリームを使用？	true	3.1.5

デバッグ/プロファイリング.

プロパティ名	定義	デフォルト値	対応開始したバージョン
logger	メッセージのログ先として使用される 'com.mysql.jdbc.log.Log' を実行するクラスの名称 (デフォルトは 'com.mysql.jdbc.log.StandardLogger' で、STDERR にログ) 。	com.mysql.jdbc.log.StandardLog	
profileSQL	デフォルトの false になる構成されたログ (true/false) へのクエリとその実行/フェッチ時間を追跡。	false	3.1.0
reportMetricsIntervalMillis	'gatherPerfMetrics' が有効な場合のログの頻度は (ミリ秒単位) ？	30000	3.1.2
maxQuerySizeToLog	プロファイリングまたはトレースの際に記録されるクエリの最大長/サイズを制御。	2048	3.1.3
packetDebugBufferSize	'enablePacketDebug' が true の場合に、保留するパケットの最大数。	20	3.1.3
slowQueryThresholdMillis	'logSlowQueries' が有効な場合、クエリが 'slow' と記録されるまでの長さは (ミリ秒単位) ？	2000	3.1.2
useUsageAdvisor	ドライバは、適切で効果的な JDBC および MySQL Connector/J の使用を忠告する 'usage' 警告をログに発行しますか (true/false 、デフォルトの 'false' になる) ？	false	3.1.1
autoGenerateTestcaseScript	ドライバは、サーバ側プリペアドステートメントを含む、実行中の SQL を STDERR ヘダンプしますか？	false	3.1.9
dumpMetadataOnColumnNotFound	ドライバは、ResultSet.findColumn() が履行に失敗した場合、結果セットのフィールドレベル メタデータを例外メッセージにダンプしますか？	false	3.1.13
dumpQueriesOnException	ドライバは、サーバに送られたクエリの内容を、SQLExceptions のメッセージにダンプしますか？	false	3.1.3
enablePacketDebug	有効な場合、'packetDebugBufferSize' パケットのリングバッファは維持され、ドライバのコードの重要エリアに例外が投入される際にダンプされます。	false	3.1.3
explainSlowQueries	'logSlowQueries' が有効な場合、ドライバは自動的にサーバに 'EXPLAIN' を発行し、WARN レベルで結果を構成されたログに送りますか？	false	3.1.2
logSlowQueries	'slowQueryThresholdMillis' より長くなるクエリを記録しますか？	false	3.1.2
traceProtocol	トレース レベル ネットワーク プロトコルを記録しますか？	false	3.1.2

その他.

プロパティ名	定義	デフォルト値	対応開始したバージョン
useUnicode	ドライバは、ストリングを扱う際、Unicode 文字エンコードを使用しますか？ドライバが文字セットのマッピングを識別できない、または、MySQL が元来サポートしない文字セット (例えば UTF-8) を使用するために、ドライバの 'force' を試みる場合にのみに使用する？ true/false 、デフォルトの 'true' になります。	true	1.1g

characterEncoding	'useUnicode' が true に設定されている場合、ストリングを扱う際にドライバが使用する文字エンコーディングは？ (デフォルトは 'autodetect')		1.1g
characterSetResults	戻す結果の文字セットをサーバに指示。		3.0.13
connectionCollation	設定された場合、 'set collation_connection' を介してこのコレクションを使用するようサーバに指示。		3.0.13
sessionVariables	ドライバが接続した際、SET SESSION ... としてサーバに送られる、コンマで区切られた名前/値ペアのリスト。		3.1.8
allowNaNAndInf	ドライバは、PreparedStatement.setDouble() で NaN or +/- INF 値を許容しますか？	false	3.1.5
autoClosePstmtStreams	ドライバは、set*() メソッドを介して引数として渡される .close() を、ストリーム/リーダとして自動的呼び出しますか？	false	3.1.12
autoDeserialize	ドライバは、BLOB フィールドに保管されたオブジェクトを、自動的に検出してデシリアル化しますか？	false	3.1.5
capitalizeTypeNames	DatabaseMetaData でタイプ名を大文字にしますか？ (通常は WebObjects を使用している場合にのみ有効、true/false 、デフォルトの 'false' になる)	false	2.0.7
clobCharacterEncoding	構成された接続 characterEncoding の代わりに、TEXT 、 MEDIUMTEXT 、 および LONGTEXT 値の送信と引き出しに使用する文字エンコード。		5.0.0
clobberStreamingResults	これによって、 'streaming' ResultSet は自動的に閉じられ、すべてのデータをサーバから読み取る前に他のクエリが実行された場合、まだサーバからストリーミングが続いている未決のデータは破棄されます。	false	3.0.9
continueBatchOnError	ひとつのステートメントが履行に失敗した場合、ドライバはバッチ コマンドの処理を続けますか？ JDBC 仕様はどちらも許可 (デフォルトの 'true' になる) 。	true	3.0.3
createDatabaseIfNotExist	URL で提供されるデータベースがまだなければ作成。構成されたユーザはデータベース作成の許可を取っているものとします。	false	3.1.9
emptyStringsConvertToZero	ドライバは、殻のストリング フィールドから '0' の数値への変換を許可しますか？	true	3.1.8
emulateLocators	N/A	false	3.1.0
emulateUnsupportedPstmts	ドライバは、サーバにサポートされていないプリペアドステートメントを検出し、それらをクライアント側のエミュレートされたバージョンに置き換えますか？	true	3.1.7
ignoreNonTxTables	ロールバックの非トランザクション テーブル警告を無視しますか？ (デフォルトの 'false' になる)	false	3.0.9
jdbcCompliantTruncation	警告をサポートするサーバに接続する場合 (MySQL 4.1.0 以降)、JDBC 仕様で要求されるようにデータがトランザクションされる時に、ドライバは java.sql.DataTruncation 例外を投入しますか？	true	3.1.2
maxRows	返す行の最大数 (デフォルトの 0 は、すべての行を返します) 。	-1	all versions
noAccessToProcedureBodies	CallableStatements のプロシージャ パラメータ タイプを識別する際に、接続しているユーザが "SHOW CREATE PROCEDURE" を介してプロシージャ本体にアクセスできない場合、または mysql.proc で選択できない場合、ドライバは例外を投入する代わりに、基本メタデータ (INOUT VARCHAR としてレポートされたすべてのパラメータ) を作成しますか？	false	5.0.3

noDatetimeStringSync	ResultSet.getDatetimeType().toString().equals(ResultSet.getString()) を確立しません。	false	3.1.7
noTimezoneConversionForTimeType	useTimezone='true' の場合、サーバ時間帯を使って TIME 値を変換しません。	false	5.0.0
nullCatalogMeansCurrent	DatabaseMetadataMethods が 'catalog' パラメータを要求する際、値 null は現在のカタログを使用するという意味ですか？ (これは JDBC 対応ではありませんが、前バージョンのドライバ旧来の動作に従っています)	true	3.1.8
nullNamePatternMatchesAll	*pattern パラメータを受け付ける DatabaseMetaData メソッドは、null を '%' と同様に扱いますか？ (これは JDBC 対応ではありませんが、旧バージョンのドライバは、仕様からこれを受け入れます)	true	3.1.8
overrideSupportsIntegrityEnhancementFacility	ドライバは、外部キーのシグナルサポートに "true" を戻すアプリケーションを避けるために、DatabaseMetaData.supportsIntegrityEnhancementFacility() に "true" を戻しますか？ データベースはこの関数をサポートしておらず、SQL 仕様はこの機能が外部キーのサポート以上のものを含んでいると示しています。そのような回避アプリケーションのひとつには Openoffice があります。	false	3.1.12
pedantic	例外なく JDBC 仕様に従います。	false	3.0.0
pinGlobalTxToPhysicalConnection	XAConnections を使用する時、ドライバは、与えられた XID 上の作業が常に同じ物理接続に送られるよう確認するべきですか？ これは、"XA END" が呼び出された後、XAConnection が "XA START ... JOIN" をサポートするのを許可します。	false	5.0.1
processEscapeCodesForPrepStmts	ドライバは、準備されたクエリのエスケープコードを処理しますか？	true	3.1.12
relaxAutoCommit	ドライバが接続する MySQL のバージョンがトランザクションをサポートしない場合でも、commit()、rollback()、および setAutoCommit() への呼び出しを許可しますか (true/false、デフォルトの 'false' になる)？	false	2.0.13
retainStatementAfterResultSetClose	ドライバは、ResultSet.close() が呼び出された後、ResultSet の Statement リファレンスを保持しますか？ JDBC-4.0 以降では、これは JDBC 対応ではありません)	false	3.1.11
rollbackOnPooledClose	プール内の理論接続が閉じられた後、ドライバは rollback() を発行しますか？	true	3.0.15
runningCTS13	Sun の JDBC 適応性テストスイート バージョン 1.3 のバグの回避を有効にします。	false	3.1.7
serverTimezone	timezone の検出/マッピングをオーバーライドします。サーバからの timezone が Java timezone にマップしない場合に使用されます。		3.0.2
strictFloatingPoint	適合テストの旧バージョンでのみ使用。	false	3.0.0
strictUpdates	ドライバは更新可能な結果セットに厳密なチェック (選択されたすべてのプライマリキー) を行いますか (true、false、デフォルトの 'true' になる)？	true	3.0.4
tinyInt1isBit	ドライバは datatype TINYINT(1) を BIT タイプとして扱いますか (テーブルを作成する時にサーバが BIT -> TINYINT(1) を暗黙的に変換するため)？	true	3.0.16
transformedBitsIsBoolean	ドライバが TINYINT(1) を他のタイプに変換する場合、MySQL-5.0 は BIT タイプを持っているので、今後の	false	3.1.9

	MySQL-5.0 との互換性のため、BIT でなく BOOLEAN を使用しますか？		
ultraDevHack	UltraDev が破損しており、prepareCall() for _all_ statements を発行するため、必要な場合は、prepareCall() に PreparedStatement を作成しますか？ (true/false、デフォルトの 'false' になる)	false	2.0.3
useGmtMillisForDatetimes	Date と Timestamp のインスタンスを作成する前に、セッション timezone と GMT 間を変換します ("false" の値は旧来の動作、"true" はより JDBC 対応の動作を起こします)。	false	3.1.12
useHostsInPrivileges	DatabaseMetaData.getColumn/TablePrivileges() でユーザに '@hostname' を加えます (true/false)。デフォルトの 'true' になります。	true	3.0.2
useInformationSchema	MySQL-5.0.7 以降に接続する場合、ドライバは DatabaseMetaData を使って DatabaseMetaData に使用されている情報を引き出しますか？	false	5.0.0
useJDBCCompliantTimezoneShift	java.util.Calendar 引数を取るそれらの JDBC 引数の、TIME/TIMESTAMP/DATETIME 値の timezone 情報を変換する時、ドライバは JDBC 対応 ルールを使用しますか？ (このオプションは "useTimezone=true" 構成オプション専用です)	false	5.0.0
useOldAliasMetadataBehavior	ドライバは、カラムおよびテーブルの "AS" 句に旧来の動作を使用し、エイリアス (もしあれば) を、元のカラム/テーブル名でなく、ResultSetMetaData.getColumnNames() または ResultSetMetaData.getTableName() にのみ戻しますか？	true	5.0.4
useOldUTF8Behavior	4.0 以前のサーバと通信している時、ドライバが行った UTF-8 動作を使用。	false	3.1.6
useOnlyServerErrorMessages	サーバが戻したエラー メッセージに、'standard' SQLState エラー メッセージを付加しません。	true	3.0.15
useServerPrepStmts	サーバがサポートする場合は、サーバ側プリペアド ステートメントを使用しますか？ (デフォルトの 'true' になる)。	true	3.1.0
useSqlStateCodes	'legacy' X/Open/SQL 状態コードでなく、SQL Standard 状態コードを使用 (true/false)。デフォルトは 'true'。	true	3.1.3
useStreamLengthsInPrepStmts	PreparedStatement/ResultSet.setXXXStream() メソッドコールでストリーム長パラメータを引き受けます (true/false、デフォルトの 'true' になる)？	true	3.0.2
useTimezone	クライアントとサーバの timezone 間の時間/日付タイプを変換しますか (true/false、デフォルトの 'false' になる)？	false	3.0.2
useUnbufferedInput	サーバからのデータ読み取りに BufferedInputStream を使用しません。	true	3.0.11
yearIsDateType	JDBC ドライバは MySQL タイプ "YEAR" を、java.sql.Date または SHORT として扱いますか？	true	3.1.9
zeroDateTimeBehavior	ドライバが、すべてゼロで構成された DATETIME 値 (無効な日付を表すために MySQL で使用される) に遭遇した時のリアクションは？有効値は 'exception'、'round'、および 'convertToNull'。	exception	3.1.4

Connector/J はまた、socketFactory プロパティを通して、NamedPipeSocketFactory をプラグイン ソケット ファクトリとして使用する Windows NT/2000/XP の名前付きパイプを介した MySQL へのアクセスをサポートします。namedPipePath プロパティを使わない場合、'\\.\pipe\MySQL' のデフォルトが使用されま

す。[NamedPipeSocketFactory](#) を使用する場合は、JDBC url 内のホスト名とポート番号の値は無視されます。その機能は次を使って有効にすることができます：

```
socketFactory=com.mysql.jdbc.NamedPipeSocketFactory
```

名前付きパイプは、JDBC ドライバが使用されているのと同じマシンで MySQL に接続している時にのみ作動します。簡単な作動テストでは、名前付きパイプのアクセスは、標準のアクセスより 30% から 50% 速いという結果が出ています。

[com.mysql.jdbc.NamedPipeSocketFactory](#)、または [com.mysql.jdbc.StandardSocketFactory](#) のコード例を使用して、独自のソケット ファクトリを作成することができます。

JDBC API 実装についての注記

MySQL Connector/J は一般的に利用可能な Sun の JDBC 対応テストスイートの全バージョンで、すべてのテストにパスしています。ただし、JDBC 仕様は、特定の機能をどのように実装するか、もしくは、その仕様は実装でリレーを許可するかについての点で曖昧な部分が多くみられます。

このセクションでは、特定の实装の決定が MySQL Connector/J の使用方法におよぼす影響の詳細を、インターフェイスごとに説明します。

- Blob

Connector/J 3.1.0 からは、プロパティ 'emulateLocators=true' を JDBC URL に加えることで、ロケータで Blob をエミュレートすることができます。この方法を使うと、他のデータを取り出すまでドライバは実際の Blob データのローディングを遅らせ、その後 blob データ ストリームで抽出メソッド ([getInputStream\(\)](#)、[getBytes\(\)](#)、など) を使用します。

これを実行するには、実際の Blob の名称に カラムの値を伴うカラム エイリアスを使用する必要があります。その例は：

```
SELECT id, data as 'data' from blobtable
```

これを行うには、次のルールに従う必要があります：

- [SELECT](#) はまた、ただひとつのテーブルを参照していなければならない、そのテーブルはプライマリ キーを持っている必要があります。
- [SELECT](#) はプライマリ キーを組成するすべてのカラムをカバーする必要があります。

Blob 実装はインプレース変更を許可しません ([DatabaseMetaData.locatorsUpdateCopies\(\)](#) メソッドで報告されたように、それらはコピーです)。このため、対応する [PreparedStatement.setBlob\(\)](#) または [ResultSet.updateBlob\(\)](#) (更新可能な結果セットの場合) メソッドを使い、変更をデータベースに戻して保存するようにしてください。

- CallableStatement

Connector/J 3.1.1 からは、ストアード プロシージャが、[CallableStatement](#) インターフェイスを介して MySQL バージョン 5.0 以降に接続している時にサポートされるようになりました。現在は、[CallableStatement](#) の [getParameterMetaData\(\)](#) メソッドはサポートされていません。

- Clob

Clob 実装はインプレース変更を許可しません ([DatabaseMetaData.locatorsUpdateCopies\(\)](#) メソッドで報告されたように、それらはコピーです)。このため、[PreparedStatement.setClob\(\)](#) メソッドを使用して、変更をデータベースに戻して保存するようにしてください。JDBC API は [ResultSet.updateClob\(\)](#) メソッドを持っていません。

- Connection

MM.MySQL の旧バージョンとは異なり、[isClosed\(\)](#) メソッドはサーバを ping して生きているかを判断しません。JDBC 仕様に従い、[closed\(\)](#) が接続で呼び出された時にのみ true を返します。接続がまだ有効か確認したい場合は、[SELECT 1](#) などの簡単なクエリを発行してください。接続が無効になっていれば、ドライバは例外を投入します。

- DatabaseMetaData

外部キー情報 (`getImportedKeys()`/`getExportedKeys()`) および `getCrossReference()`) は InnoDB テーブルでのみ入手可能です。ただし、ドライバは `SHOW CREATE TABLE` を使用してこの情報を取り出すので、他の保存エンジンが外部キーをサポートする場合、ドライバはそれらも透過的にサポートします。

- PreparedStatement

PreparedStatements は、MySQL がプリペアド ステートメント機能を持たないため、ドライバによって実装されます。これにより、ドライバがクライアントの完全な SQL のパーサを持つことが要求されるため、ドライバは `getParameterMetaData()` または `getMetaData()` を実装しません。

MySQL Connector/J 3.1.0 からは、サーバ側プリペアド ステートメントおよびバイナリ エンコードされた結果セットは、サーバがそれらをサポートする場合に使用されます。

`setBinaryStream()`、`setAsciiStream()`、`setUnicodeStream()`、`setBlob()`、または `setClob()` を介して設定された `large` パラメータとサーバ側プリペアド ステートメントを使用する際は注意してください。非 `large` パラメータに変更された `large` パラメータとステートメントを再実行したい場合、`clearParameters()` を呼び出し、すべてのパラメータを再度設定する必要があります。その理由は以下です：

- サーバ側プリペアド ステートメントとクライアント側エミュレーションの双方の間、大きなデータは `PreparedStatement.execute()` が呼び出された時にのみ交換されます。
- それが終了したら、クライアント側のデータの読み取りに使用されたストリームは閉じられ (JDBC 仕様により)、再度読み取ることはできません。
- パラメータが `large` から 非 `large` へ変更する場合、ドライバはプリペアド ステートメントのサーバ側の状況をリセットし、変更するパラメータが前の大きな値になり代わるの許可する必要があります。これにより、サーバに送られたすべての大きなデータが取り除かれ、したがってデータの再送を `setBinaryStream()`、`setAsciiStream()`、`setUnicodeStream()`、`setBlob()`、または `setClob()` を介して要求します。

結果的に、パラメータのタイプを非 `large` にしたい場合は、再実行される前に、`clearParameters()` を呼び出し、プリペアド ステートメントのすべてのパラメータを再度設定する必要があります。

- ResultSet

デフォルトにより、ResultSets は完全に摘出され、メモリに保存されます。ほとんどの場合において、これは最も効果的な操作方法であり、MySQL の設計により、ネットワーク プロトコルはより簡単に実装できます。多大な行や大きな値を持つ ResultSets を扱っていて、要求されるメモリのために JVM に十分なスペースを割り振れない場合は、ドライバに結果を一行ごとにストリームし戻すよう指示することができます。

この機能を有効にするには、次の方法で Statement インスタンスを作成する必要があります。

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                             java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
```

フェッチのサイズが `Integer.MIN_VALUE` の、前進専用、読み取り専用のコンビネーションは、行ごとに結果セットをストリームするようドライバに指示する信号として機能します。この後、このステートメントで作成された結果セットは行ごとに摘出されます。

このアプローチには注意点がいくつかあります。接続にクエリを発行する前に、結果セットのすべての行を読まなければならない、さもなければ例外が投入されます。

これらのステートメントが保持するロックが解除されうる最も早い時点は (InnoDB などの他の保存エンジンの MyISAM テーブル レベル ロック、または 行レベル ロックであっても)、ステートメントが完遂された時です。

ステートメントがトランザクションの範囲内にある時は、ロックはトランザクションが完遂した時に解除されます (つまり、ステートメントが先に完遂される必要があります)。他のほとんどのデータベースと同じく、ステートメントで保留になっているすべての結果が読み取られるまで、またはステートメントのアクティブな結果セットが閉じられるまで、ステートメントは完遂されません。

したがって、結果のストリーミングを使用する場合、結果セットを作成しているステートメントが参照するテーブルへの並行アクセスを維持したいのであれば、それらをできるだけ速く処理してください。

- `ResultSetMetaData`

`isAutoIncrement()` メソッドは MySQL サーバ 4.0 以降を使用している時にのみ作動します。

- `Statement`

JDBC ドライバの 3.2.1 以前のバージョンを使用し、5.0.3 以前のバージョンのサーバに接続している場合、上記のように結果セットのストリーミングをトグルする以外では、`setFetchSize()` は無効です。

Connector/J 5.0.0 以降は、`Statement.cancel()` と `Statement.setQueryTimeout()` 両方へのサポートを含みます。両方ともに MySQL 5.0.0 以降が必要で、`KILL QUERY` 文を発行するために別の接続が要求されます。`setQueryTimeout()` の場合、その実装はタイムアウト機能を扱うために追加のスレッドを作成します。

注記

タイムアウトの期限切れによって取り消されるクエリを実行するスレッドのブロックを解除し、例外として投入する方法が現在はないため、`setQueryTimeout()` へのステートメントの取り消しの失敗は、そのまま失敗するのではなく、それ自体を `RuntimeException` として宣言する場合があります。

MySQL は SQL カーソルをサポートせず、JDBC ドライバはそれらをエミュレートしないので、`"setCursorName()"` は無効です。

Java、JDBC および MySQL のタイプ

MySQL Connector/J は、MySQL データタイプと Java データタイプ間の変換の扱い方に柔軟に対応します。

丸めやオーバーフロー、もしくは精度の損失がおこることはありますが、一般的に、どんな MySQL データタイプでも `java.lang.String` に変換することが可能で、また、いかなる数値タイプも Java 数値タイプに自由に変換することができます。

Connector/J 3.1.0 からは、プロパティ `jdbcCompliantTruncation` を `false` に設定して接続の構成に使用し、その動作を規制しない限り、JDBC ドライバは JDBC 仕様で要求される通り警告を発行、もしくは `DataTruncation` 例外を投入します。

常に作動が保証されている変換は、次の表の通りです：

接続プロパティ - その他.

これらの MySQL データタイプは	常時以下の Java タイプへの変換が可能
<code>CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET</code>	<code>java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob</code>
<code>FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT</code>	<code>java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal</code>
<code>DATE, TIME, DATETIME, TIMESTAMP</code>	<code>java.lang.String, java.sql.Date, java.sql.Timestamp</code>

注記

変換の対象になる MySQL データタイプより低い精度または容量を持つ Java 数値データタイプを選択した場合、丸め、オーバーフロー、もしくは精度の損失が起こることがあります。

`ResultSet.getObject()` メソッドは、適切な JDBC 仕様にしたがって、MySQL と Java タイプ間のタイプ変換を使用します。`ResultSetMetaData.getColumnClassName()` によって戻される値も以下に記載されています。`java.sql.Types` のクラスに関する詳細は、[Java 2 Platform Types](#) をご覧ください。

`ResultSet.getObject()` の MySQL タイプ から Java タイプへ.

MySQL タイプ名	<code>getColumnClassName()</code> の戻り値	Java クラスとして返還

BIT(1) (MySQL-5.0 から)	BIT	java.lang.Boolean
BIT(> 1) (MySQL-5.0 から)	BIT	byte[]
TINYINT	TINYINT	java.lang.Boolean 構成プロパティ tinyInt1isBit が true (デフォルト) に設定され、格納サイズが 1 の場合。それ以外では java.lang.Integer
BOOL, BOOLEAN	TINYINT	上記 TINYINT 参照。TINYINT(1) のエイリアス。現行。
SMALLINT[(M)] [UNSIGNED]	SMALLINT [UNSIGNED]	java.lang.Integer (UNSIGNED である場合、ない場合両方)
MEDIUMINT[(M)] [UNSIGNED]	MEDIUMINT [UNSIGNED]	java.lang.Integer , UNSIGNED であれば java.lang.Long
INT, INTEGER[(M)] [UNSIGNED]	INTEGER [UNSIGNED]	java.lang.Integer , UNSIGNED であれば java.lang.Long
BIGINT[(M)] [UNSIGNED]	BIGINT [UNSIGNED]	java.lang.Long , UNSIGNED であれば java.math.BigInteger
FLOAT[(M,D)]	FLOAT	java.lang.Float
DOUBLE[(M,B)]	DOUBLE	java.lang.Double
DECIMAL[(M[,D])]	DECIMAL	java.math.BigDecimal
DATE	DATE	java.sql.Date
DATETIME	DATETIME	java.sql.Timestamp
TIMESTAMP[(M)]	TIMESTAMP	java.sql.Timestamp
TIME	TIME	java.sql.Time
YEAR[(2 4)]	YEAR	yearIsDateType 構成プロパティが false に設定されている場合、戻されるオブジェクトのタイプは java.sql.Short 。true (デフォルト) の場合のオブジェクトタイプは java.sql.Date (日付設定は 1 月 1 日の深夜まで)。
CHAR(M)	CHAR	java.lang.String (カラムの文字セットが BINARY の場合は byte[] が戻される)
VARCHAR(M) [BINARY]	VARCHAR	java.lang.String (カラムの文字セットが BINARY の場合は byte[] が戻される)
BINARY(M)	BINARY	byte[]
VARBINARY(M)	VARBINARY	byte[]
TINYBLOB	TINYBLOB	byte[]
TINYTEXT	VARCHAR	java.lang.String
BLOB	BLOB	byte[]
TEXT	VARCHAR	java.lang.String
MEDIUMBLOB	MEDIUMBLOB	byte[]
MEDIUMTEXT	VARCHAR	java.lang.String
LOBLOB	LOBLOB	byte[]
LONGTEXT	VARCHAR	java.lang.String
ENUM('value1','value2',...)	CHAR	java.lang.String
SET('value1','value2',...)	CHAR	java.lang.String

文字セットと Unicode の使用

JDBC ドライバからサーバへ送られた、[Statement.execute\(\)](#)、[Statement.executeUpdate\(\)](#)、[Statement.executeQuery\(\)](#) を介して送られたクエリ、そして、[setBytes\(\)](#)、[setBinaryStream\(\)](#)、[setAsciiStream\(\)](#)

、[setUnicodeStream\(\)](#)、および [set- Blob\(\)](#) を使用するパラメータ セットを除く [Prepared- Statement](#) および [CallableStatement](#) パラメータを含むすべてのストリングは、元来の Java Unicode 型から クライアント側の文字エンコードに自動的に変換されます。

MySQL Server 4.1 より前では、Connector/J は、サーバ構成から自動的に検出が可能な、もしくは [useUnicode](#) および [characterEncoding](#) プロパティを通じてユーザが構成することができる、接続当たりの単一文字エンコードをサポートしていました。

MySQL Server 4.1 からは、Connector/J はクライアントとサーバ間の単一文字エンコードと、[Result- Sets](#) でサーバによってクライアントに返されたデータの文字エンコードのすべての数をサポートしています。

クライアントとサーバ間の文字エンコードは、接続に基づいて自動的に検出されます。ドライバに使用されるエンコードは、4.1.0 より前のバージョンのサーバでは [character_set](#) システム変数、4.1.0 以降では [character_set_server](#) を介して、サーバで特定されます。詳細は [サーバのキャラクタセットおよび照合順序](#) をご覧ください。

クライアント側の自動検出エンコードをオーバーライドするには、サーバへの接続に使用される URL の [characterEncoding](#) プロパティを使用します。

クライアント側で文字エンコードを特定する際、Java スタイルの名称を使用してください。次の表は MySQL 文字セット用の Java スタイルの名称のリストです：

MySQL から Java エンコード名への変換。

MySQL 文字セット名	Java スタイル文字エンコード名
ascii	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
cp932	Cp932 or MS932 (MySQL Server > 4.1.11)
gb2312	EUC_CN
ujis	EUC_JP
euckr	EUC_KR
latin1	ISO8859_1
latin2	ISO8859_2
greek	ISO8859_7
hebrew	ISO8859_8
cp866	Cp866
tis620	TIS620
cp1250	Cp1250
cp1251	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

注意. ドライバは文字セットの変更を検出できず、最初の接続セットアップで検出された文字セットを使い続けるため、Connector/J で 'set names' クエリを発行しないでください。

クライアントからの複数の文字セットの送信を許可するには、UTF-8 の使用が必要です。[utf8](#) をデフォルトのサーバ文字セットとして構成するか、[characterEncoding](#) を通して UTF-8 を使用するよう JDBC ドライバを構成してください。

SSL を使用して安全に接続する

MySQL Connector/J の SSL は、JDBC ドライバとサーバ間のすべてのデータ (最初のハンドシェイク以外) を暗号化します。SSL を有効にした場合の性能上の影響として、クエリの処理時間が、クエリのサイズと戻すデータの量によって、35% から 50% 長くなります。

SSL Support の作動には、以下が必要になります：

- JDK-1.4.1 以降などの、JSSE (Java Secure Sockets Extension) を高める JDK。SSL は現在、次のバグにより、JDK-1.2.x または JDK-1.3.x などの JESSE を追加できる JDK とは作動しません：<http://developer.java.sun.com/developer/bugParade/bugs/4273544.html>
- SSL をサポートし、そのためにコンパイルおよび構成された、MySQL-4.0.4 以降の MySQL サーバ。詳細は [接続安全](#) をご覧ください。
- クライアント証明書 (このセクション内であとに説明)。

まず最初に、MySQL サーバ CA Certificate を Java truststore にインポートしてください。CA Certificate のサンプルは、MySQL ソース配布物の [SSL](#) サブディレクトリにあります。SSL はこれを使用して、安全な MySQL サーバと交信しているかを判断します。

Java の [keytool](#) を使って truststore を現行のディレクトリに作成し、サーバの CA 証明書 ([cacert.pem](#)) をインポートするには、次の手順で行うことができます ([keytool](#) がパスにあることを前提としています。[keytool](#) は JDK または JRE の [bin](#) サブディレクトリにあります)：

```
shell> keytool -import -alias mysqlServerCACert \
               -file cacert.pem -keystore truststore
```

Keytool は次の情報で応答します：

```
Enter keystore password: *****
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus,
      O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus,
        O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Serial number: 0
Valid from:
  Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
  MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
  SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]: yes
Certificate was added to keystore
```

その後、クライアント証明書を生成し、安全なクライアントと交信していることをMySQL サーバに知らせます：

```
shell> keytool -genkey -keyalg rsa \
               -alias mysqlClientCertificate -keystore keystore
```

Keytool は次の情報のプロンプトを出し、[keystore](#) と名付けられた keystore を現行のディレクトリに作成します。

その状況に適した情報で応答してください：

```
Enter keystore password: *****
What is your first and last name?
[Unknown]: Matthews
What is the name of your organizational unit?
[Unknown]: Software Development
What is the name of your organization?
[Unknown]: MySQL AB
What is the name of your City or Locality?
[Unknown]: Flossmoor
What is the name of your State or Province?
[Unknown]: IL
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
L=Flossmoor, ST=IL, C=US> correct?
```

```
[no]: y
```

```
Enter key password for <mysqlClientCertificate>
(RETURN if same as keystore password):
```

最後に、生成した keystore と truststore を JSSE に使わせるには、JVM を起動する時に次のシステム プロパティを設定し、path_to_keystore_file を作成した鍵ストアへの完全パスに、path_to_truststore_file を作成した truststore に置き換え、各プロパティに適したパスワード値を使用する必要があります。

```
-Djavax.net.ssl.keyStore=path_to_keystore_file
-Djavax.net.ssl.keyStorePassword=*****
-Djavax.net.ssl.trustStore=path_to_truststore_file
-Djavax.net.ssl.trustStorePassword=*****
```

また、`useSSL=true` を URL に加えるか、`DriverManager.getConnection()` に渡す `java.util.Properties` インスタンスでプロパティ `useSSL` を `true` に設定して、MySQL Connector/J の接続パラメータで `useSSL` を `true` に設定する必要があります。

JESSE デバッグ (下記参照) を有効にし、次のキー イベントを探することで、SSL が作動しているかテストすることができます：

```
...
*** ClientHello, v3.1
RandomCookie: GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, »
  54, 244, 0, 168, 55, 103, 215, 64, 16, 138, 225, 190, 132, 153, 2, »
  217, 219, 239, 202, 19, 121, 78 }
Session ID: {}
Cipher Suites: { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
Compression Methods: { 0 }
***
[write] MD5 and SHA1 hashes: len = 59
0000: 01 00 00 37 03 01 3D B6 90 FA C7 94 B4 D7 4A 0C ...7..=.....J.
0010: 36 F4 00 A8 37 67 D7 40 10 8A E1 BE 84 99 02 D9 6...7g.@.....
0020: DB EF CA 13 79 4E 00 00 10 00 05 00 04 00 09 00 ....yN.....
0030: 0A 00 12 00 13 00 03 00 11 01 00 .....
main, WRITE: SSL v3.1 Handshake, length = 59
main, READ: SSL v3.1 Handshake, length = 74
*** ServerHello, v3.1
RandomCookie: GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, »
  202, 79, 185, 178, 100, 215, 66, 254, 21, 83, 187, 190, 42, 170, 3, »
  132, 110, 82, 148, 160, 92 }
Session ID: {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, »
  182, 158, 30, 11, 150, 79, 170, 76, 255, 92, 15, 226, 24, 17, 177, »
  219, 158, 177, 187, 143}
Cipher Suite: { 0, 5 }
Compression Method: 0
***
%% Created: [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes: len = 74
0000: 02 00 00 46 03 01 3D B6 43 98 74 32 04 67 19 64 ...F..=.C.t2.g.d
0010: 3A CA 4F B9 B2 64 D7 42 FE 15 53 BB BE 2A AA 03 ..O..d.B..S.*..
0020: 84 6E 52 94 A0 5C 20 A3 E3 54 35 51 7F FC FE B2 ..nR..\..T5Q....
0030: B3 44 3F B6 9E 1E 0B 96 4F AA 4C FF 5C 0F E2 18 ..D?.....O.L...\
0040: 11 B1 DB 9E B1 BB 8F 00 05 00 .....
main, READ: SSL v3.1 Handshake, length = 1712
...
```

次のシステム プロパティを設定する時、JSSE は (STDOUT に) デバッグを提供します：`-Djavax.net.debug=all` これは、どの鍵ストアおよび truststore を使用しているか、SSL ハンドシェイクと証明書交換の間に何が起きているかを報告します。SSL 接続を開きたい時に、何が作動していないか判断するのに便利です。

ReplicationConnection でマスタ/スレーブ複製を使用する

Connector/J 3.1.7 からは、クエリを自動的に読込/書込マスタに送るドライバの変異型が、`Connection.getReadOnly()` の状況によってフェイルオーバーかラウンドロビンのロードバランスされたスレーブのセットを利用できるようにしています。

アプリケーションは、`Connection.setReadOnly(true)` の呼び出しによって、トランザクションを読み出し専用にする信号を出します。この複製を考慮した接続は、ラウンドロビン スキーマを使用するロードバランスされた per-vm で

あるスレーブ接続のひとつを使用します (供給された接続は、スレーブがサービスから削除されない限り、スレーブに付着します)。書込トランザクションがある場合、または時間に敏感な読み込みがある場合 (MySQL での複製は非同期なので注意)、[Connection.setReadOnly\(false\)](#) を呼び出し、接続が読み込み専用にならないよう設定すると、ドライバは以後の呼び出しが確実にマスタ MySQL サーバに送られるようにします。ドライバは、このロードバランス機能の完遂に使用するすべての接続間の自動コミット、隔離レベル、およびカタログの現状の伝搬を行います。

この機能を有効にするには、アプリケーション サーバの接続プールを構成する時か、スタンドアロン アプリケーションに JDBC ドライバのインスタンスを作成する時に、"[com.mysql.jdbc.ReplicationDriver](#)" クラスを使用します。URL フォーマットを標準の MySQL JDBC ドライバとして受け入れるので、[ReplicationDriver](#) は現在、それが [DriverManager](#) に登録された唯一の MySQL JDBC ドライバでない限り、[java.sql.DriverManager](#) ベースの接続作成とは作動しません。

以下は、[ReplicationDriver](#) のスタンドアロン アプリケーションでの、手短で簡単な使用例です：

```
import java.sql.Connection; import java.sql.ResultSet; import java.util.Properties;

import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {

    public static void main(String[] args) throws Exception { ReplicationDriver driver = new ReplicationDriver();

        Properties props = new Properties();

        // We want this for failover on the slaves props.put("autoReconnect", "true");

        // We want to load balance between the slaves props.put("roundRobinLoadBalance", "true");

        props.put("user", "foo"); props.put("password", "bar");

        // // Looks like a normal MySQL JDBC url, with a // comma-separated list of hosts, the first // being the 'master', the rest being any number // of slaves that the driver

        Connection conn = driver.connect("jdbc:mysql://master,slave1,slave2,slave3/test", props);

        // // Perform read/write work on the master // by setting the read-only flag to "false" //

        conn.setReadOnly(false); conn.setAutoCommit(false); conn.createStatement().executeUpdate("UPDATE some_table ...."); conn.commit();

        // // Now, do a query from a slave, the driver automatically picks one // from the list //

        conn.setReadOnly(true);

        ResultSet rs = conn.createStatement().executeQuery("SELECT a,b FROM alt_table");

        ..... }}
```

Connector/J に関する注記とヒント

JDBC の基本コンセプト

このセクションでは、JDBC の一般的な背景を説明します。

[DriverManager](#) インターフェイスを使用して MySQL に接続する

アプリケーション サーバの外で JDBC を使用している場合、[DriverManager](#) クラスは Connections の確立を管理します。

[DriverManager](#) は、Connections をどの JDBC ドライバ で作成するべきかの指示を必要とします。この最も簡単な方法は、[java.sql.Driver](#) インターフェイスを実装するクラスで [Class.forName\(\)](#) を使用することです。MySQL Connector/J では、このクラスの名称は [com.mysql.jdbc.Driver](#) になります。このメソッドで、外部構成ファイルを使用して、データベースへの接続に使用するドライバ クラス名とドライバ パラメータを供給することが可能です。

次のセクションの Java コードは、アプリケーションの [main\(\)](#) メソッドから MySQL Connector/J を登録する方法を表しています。

```
import java.sql.Connection;
import java.sql.DriverManager;
```



```
import java.sql.SQLException;

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
    }
}
```

ドライバが [DriverManager](#) に登録されたら、[DriverManager.getConnection\(\)](#) を呼び出すことによって、特定のデータベースに接続される [Connection](#) インスタンスを取得することができます：

例24.1 [DriverManager](#) から接続を取得する

この例は、[DriverManager](#) から [Connection](#) インスタンスを取得する方法を示しています。[getConnection\(\)](#) メソッドにはいくつかの異なる署名があります。JDK に添付されている API 資料で、詳しい使用方法を確認してください。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

...
try {
    Connection conn =
        DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                                    "user=monty&password=greatsqldb");

    // Do something with the Connection

    ...
} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

[Connection](#) が確立されたら、[Statement](#) と [PreparedStatement](#) のオブジェクトの作成、そしてデータベースに関するメタデータの抽出に使用できます。これについては次のセクションで説明されます。

ステートメントを使用して SQL を実行する

[Statement](#) オブジェクトは、後で説明されるように、基本的な SQL クエリを実行し、[ResultSet](#) クラスを通しての結果の抽出を可能にします。

[Statement](#) を作成するには、前で説明した [DriverManager.getConnection\(\)](#) か [Data-Source.getConnection\(\)](#) メソッドのひとつを介して抽出した [Connection](#) オブジェクトで [createStatement\(\)](#) メソッドを呼び出します。

[Statement](#) インスタンスを得たら、使いたい SQL で [executeQuery\(String\)](#) メソッドを呼び出し、[SELECT](#) クエリを実行することができます。

データベースのデータを更新するには、[executeUpdate\(String SQL\)](#) メソッドを使用します。このメソッドは、update 文に影響を受けた行の数を返します。

SQL 文が [SELECT](#)、または [UPDATE/INSERT](#) になるかが事前に分からない場合は、[execute\(String SQL\)](#) メソッドを使用することができます。このメソッドは、SQL クエリが [SELECT](#) の場合は true、[UPDATE](#)、[INSERT](#)、もしくは [DELETE](#) 文の場合は false を返します。ステートメントが [SELECT](#) クエリの場合は、[getResultSet\(\)](#) メソッドを呼び出すことで結果を抽出できます。ステートメントが [UPDATE](#)、[INSERT](#)、もしくは [DELETE](#) 文であれば、[Statement](#) インスタンスで [getUpdateCount\(\)](#) を呼び出すことによって、影響を受けた行の数を呼び出すことができます。

例24.2 java.sql.Statement を使用して **SELECT** クエリを実行する

```
// assume that conn is an already created JDBC connection
Statement stmt = null;
ResultSet rs = null;

try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...

    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }

    // Now do something with the ResultSet ....
} finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { // ignore }

        rs = null;
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { // ignore }

        stmt = null;
    }
}
```

CallableStatements を使用してストアド プロシージャを実行する

MySQL サーバ バージョン 5.0 からは、Connector/J 3.1.1 以降と使用する場合、`java.sql.CallableStatement` インターフェイスは `getParameterMetaData()` メソッドを除いて完全に実装されています。

MySQL ストアド プロシージャの詳細は、<http://dev.mysql.com/doc/mysql/en/stored-procedures.html> をご覧ください。

Connector/J は、JDBC の `CallableStatement` インターフェイスを通して、ストアド プロシージャ機能を露出します。

注意： MySQL サーバの現行バージョンは、JDBC ドライバが呼び出し可能なステートメントに結果セット メタデータを提供するための十分な情報を返しません。つまり、`CallableStatement` を使用すると、`ResultSetMetaData` は `NULL` を返す場合があります。

次の例は、1 増やされた `inOutParam` の値を戻すストアド プロシージャと、`ResultSet` として `inputParam` を介して渡されたストリングを示しています：

例24.3 ストアド プロシージャ

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                        INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
```

END

`demoSp` プロシージャを Connector/J で使用するには、次の手順に従ってください：

1. `Connection.prepareCall()` を使用して、呼び出し可能なステートメントを準備

JDBC エスケープ シンタックスを使用する必要がある、またパラメータ プレースホルダを囲む丸括弧 (()) はオプションではないので注意。

例24.4 `Connection.prepareCall()` の使用

```
import java.sql.CallableStatement;
...

//
// Prepare a call to the stored procedure 'demoSp'
// with two parameters
//
// Notice the use of JDBC-escape syntax ({call ...})
//

CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");

cStmt.setString(1, "abcdefg");
```

注意. 出力パラメータのサポートのためにドライバが行うメタデータの取り出しにより、`Connection.prepareCall()` は拡張可能なメソッドです。性能上の理由から、コード内で `CallableStatement` インスタンスを使用して、`Connection.prepareCall()` への不要な呼び出しを最小限に抑えてください。

2. 出力パラメータ (ある場合は) を登録

出力パラメータ (ストアード プロシージャの作成時に `OUT` または `INOUT` と特定されたパラメータ) の値を取り出すには、JDBC は、`CallableStatement` インターフェイスの様々な `registerOutputParameter()` メソッドを使用して、ステートメントの実行の前にそれらを特定することを要求します：

例24.5 出力パラメータの登録

```
import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//
//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter(2, Types.INTEGER);

//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter("inOutParam", Types.INTEGER);
```

...

3. 入力パラメータ (ある場合は) を設定

入力および in/out パラメータは、[PreparedStatement](#) オブジェクトを対象として設定されます。しかし、[CallableStatement](#) はまた、名前によってパラメータの設定をサポートします：

例24.6 [CallableStatement](#) 入力パラメータの設定

...

```
//
// Set a parameter by index
//

cStmt.setString(1, "abcdefg");

//
// Alternatively, set a parameter using
// the parameter name
//

cStmt.setString("inputParameter", "abcdefg");

//
// Set the 'in/out' parameter using an index
//

cStmt.setInt(2, 1);

//
// Alternatively, set the 'in/out' parameter
// by name
//

cStmt.setInt("inOutParam", 1);
```

...

4. [CallableStatement](#) を実行し、すべての結果セット、もしくは出力パラメータを呼び出す

[CallableStatement](#) はいかなる [Statement](#) execute メソッド ([executeUpdate\(\)](#)、[executeQuery\(\)](#)、または [execute\(\)](#)) の呼び出しもサポートしますが、最も呼び出しやすいメソッドは [execute\(\)](#) で、ストアド プロシージャが結果セットを返すが事前に分からなくても問題ありません：

例24.7 結果と出力パラメータの値を呼び出す

...

```
boolean hadResults = cStmt.execute();

//
// Process all returned result sets
//

while (hadResults) {
    ResultSet rs = cStmt.getResultSet();

    // process result set
    ...

    hadResults = rs.getMoreResults();
}

//
// Retrieve output parameters
//
// Connector/J supports both index-based and
// name-based retrieval
//

int outputValue = cStmt.getInt(2); // index-based
```

```
outputValue = cStmt.getInt("inOutParam"); // name-based
```

```
...
```

AUTO_INCREMENT カラム値の呼び出し

JDBC API のバージョン 3.0 より前では、自動インクリメント、または識別カラムをサポートするデータベースからキー値を呼び出す標準の方法がありませんでした。MySQL に古い JDBC ドライバを使用すると、[Statement](#) インターフェイスでいつでも MySQL 特有のメソッドを使用でき、また、[AUTO_INCREMENT](#) キーを持つテーブルに [INSERT](#) を発行した後で、クエリ [SELECT LAST_INSERT_ID\(\)](#) を発行することができました。MySQL 特有メソッド呼び出しの使用はポータブルではなく、[AUTO_INCREMENT](#) キーの値を得るために [SELECT](#) を発行するには、データベースまでもう一度往復する必要があり、最も効率的とはいえません。次のコード部品は、[AUTO_INCREMENT](#) 値を呼び出す 3 つの方法を実証します。まず、[AUTO_INCREMENT](#) キーを呼び出し、JDBC-3.0 にアクセスする必要がある場合に推奨されるメソッドである、新しい JDBC-3.0 メソッド [getGeneratedKeys\(\)](#) 使用を実証します。その次の例では、標準の [SELECT LAST_INSERT_ID\(\)](#) クエリを使用して、同じ値を呼び出す方法を挙げます。最後の例では、[insertRow\(\)](#) メソッドを使用する場合に、更新可能な結果セットが [AUTO_INCREMENT](#) 値を呼び出す方法を示します。

例24.8 [Statement.getGeneratedKeys\(\)](#) を使った [AUTO_INCREMENT](#) カラム値の呼び出し

```
Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets assuming you have a
    // Connection 'conn' to a MySQL database already
    // available

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')",
        Statement.RETURN_GENERATED_KEYS);

    //
    // Example of using Statement.getGeneratedKeys()
    // to retrieve the value of an auto-increment
    // value
    //

    int autoIncKeyFromApi = -1;

    rs = stmt.getGeneratedKeys();

    if (rs.next()) {
        autoIncKeyFromApi = rs.getInt(1);
    } else {

        // throw an exception from here
    }
}
```

```
rs.close();

rs = null;

System.out.println("Key returned from getGeneratedKeys():"
    + autoIncKeyFromApi);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

例24.9 SELECT LAST_INSERT_ID() を使った AUTO_INCREMENT カラム値の呼び出し

```
Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.

    stmt = conn.createStatement();

    //
    // Issue the DDL queries for the table for this example
    //
    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //
    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')");

    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //

    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();
```



```

        System.out.println("Key returned from " +
            "SELECT LAST_INSERT_ID()": " +
            autoIncKeyFromFunc);
    } finally {

        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException ex) {
                // ignore
            }
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException ex) {
                // ignore
            }
        }
    }
}

```

例24.10 Updatable ResultSets 内の AUTO_INCREMENT カラム値の呼び出し

```

Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
        java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //

    rs = stmt.executeQuery("SELECT priKey, dataField "
        + "FROM autoIncTutorial");

    rs.moveToInsertRow();

    rs.updateString("dataField", "AUTO INCREMENT here?");
    rs.insertRow();

    //
    // the driver adds rows at the end
    //

    rs.last();

    //
    // We should now be on the row we just inserted
    //

```

```

int autoIncKeyFromRS = rs.getInt("priKey");

rs.close();

rs = null;

System.out.println("Key returned for inserted row: "
    + autoIncKeyFromRS);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

上記の例のコードを使用する時は、次の出力を取得してください: `getGeneratedKeys()` から戻されたキー: 1 `SELECT LAST_INSERT_ID()` から戻されたキー: 1 挿入された行に戻されたキー: 2 関数の値は接続ヘスコープされているため、`SELECT LAST_INSERT_ID()` クエリの使用がしにくい場合がありますので注意してください。したがって、他のクエリが同じ接続上で発生する場合、値は上書きされます。一方、`getGeneratedKeys()` メソッドは `Statement` インスタンスにスコープされているので、他のクエリが同じ接続上で発生しても使用が可能です、同じ `Statement` インスタンス上では使用できません。

J2EE および 他の Java フレームワークで Connector/J を使用する

このセクションでは、複数の状況での Connector/J の使用方法を説明します。

J2EE の一般的コンセプト

このセクションでは、Connector/J の使用に関連する J2EE コンセプトの一般的な背景を説明します。

接続プーリングを理解する

接続プーリングは、すべてのスレッドが必要な接続をすぐに使用できるよう、接続のプールを作成し管理する方法です。

接続をプールする方法は、ほとんどのアプリケーションは、普通数ミリ秒で完遂できるトランザクションを積極的に処理している時に、スレッドが JDBC 接続にアクセスすることだけを必要とするという事実に基づいています。トランザクションを処理していない時、接続はそうでなければ待機します。代わりに、接続プールは他のスレッドに、待機接続を有効に利用することを許可します。

実際には、スレッドが MySQL や、JDBC を使用する他のデータベースに対して作業を行う時、プールからの接続を要求します。スレッドは接続を使い終えたらそれをプールに戻し、他のスレッドが使用できるようにします。

接続がプールから貸し出されると、要求したスレッドが独占的にそれを使用します。プログラミングの観点からは、スレッドが JDBC 接続が必要になるたびに `DriverManager.getConnection()` を呼び出すのと同じことですが、しかし接続プーリングを使えば、スレッドは新しい、または既存の接続を使用する可能性があります。

接続プーリングは、全体的なリソースの使用を削減しつつ、Java アプリケーションの能率を著しく向上させます。接続プーリングの主な利点:

- 接続作成時間の短縮

他のデータベースに比べ、MySQL が提供する高速接続 setup に対してはこれはあまり関係ありませんが、新しい JDBC 接続の作成ではネットワーキングと JDBC ドライバのオーバーヘッドが増えます。これは接続を再利用することと避けることができます。

- 簡略化されたプログラム モデル

接続プーリングを使用する場合、各スレッドは専用の JDBC 接続を作成したかのような動作が可能になり、直接的な JDBC プログラミング テクニックを使用することができるようになります。

- 制御されたリソースの使用

接続プーリングを使用せず、代わりにスレッドが接続を必要とするたびに新しいものを作成する場合、アプリケーションのリソースの使用はきわめて無駄なものになり、荷重を受けて予想外の動作に出る可能性があります。

MySQL への各接続は、クライアント側とサーバ側両方にオーバーヘッド (メモリ、CPU、コンテキスト スイッチ、など) を持つことを覚えておいてください。すべての接続は、アプリケーションと MySQL で利用できるリソースの数を制限します。これらのリソースの多くは、接続が有効な作業をしていなくてもかまわず使用されます！

リソースの利用を、ただ遅いだけでなく、アプリケーションが失効しかけるまで押さえながら、接続プールを有効にして最大限に機能させることができます。

幸運にも、Sun は JDBC の接続プーリングのコンセプトを、JDBC-2.0 Optional インターフェイスを通して標準化し、すべての主要なアプリケーション サーバは MySQL Connector/J と正常に機能するこれらの API を実装しています。

通常、接続プールはアプリケーション サーバの構成ファイルで構成し、Java Naming または Directory Interface (JNDI) を介してアクセスします。次のコードは、J2EE アプリケーション サーバで展開されたアプリケーションから、接続プールを使用する例を示しています：

例24.11 J2EE アプリケーション サーバで接続プールを使用

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;

public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         * lookup the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */

        InitialContext ctx = new InitialContext();

        /*
         * Lookup the DataSource, which will be backed by a pool
         * that the application server provides. DataSource instances
         * are also a good candidate for caching as an instance
         * variable, as JNDI lookups can be expensive as well.
         */

        DataSource ds =
```

```

        (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");

/*
 * The following code is what would actually be in your
 * Servlet, JSP or EJB 'service' method...where you need
 * to work with a JDBC connection.
 */

Connection conn = null;
Statement stmt = null;

try {
    conn = ds.getConnection();

    /*
     * Now, use normal JDBC programming to work with
     * MySQL, making sure to close each resource when you're
     * finished with it, which allows the connection pool
     * resources to be recovered as quickly as possible
     */

    stmt = conn.createStatement();
    stmt.execute("SOME SQL QUERY");

    stmt.close();
    stmt = null;

    conn.close();
    conn = null;
} finally {
    /*
     * close any jdbc instances here that weren't
     * explicitly closed during normal code path, so
     * that we don't 'leak' resources...
     */

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlex) {
            // ignore -- as we can't do anything about it here
        }

        stmt = null;
    }

    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException sqlex) {
            // ignore -- as we can't do anything about it here
        }

        conn = null;
    }
}
}

```

上記の例で示されているように、JNDI InitialContext を取得し、DataSource をルックアップした後では、残りのコードは、JDBC プログラミングの経験がある人には見覚えのあるものになります。

接続プーリングを使用する際の最も重要な注意点は、コードにながら起こっても（例えばフロー制御など）、接続とそれによって作成されたすべてのもの（ステートメントもしくは結果セット等）を必ず閉じることです。そうすると、それらを再使用することができますが、これを怠ればそれらは座礁してしまい、それらが表す MySQL サーバリソース（バッファ、ロック、もしくはソケットなど）が、良い場合でもしばらくの間、ひどい時は永久に停滞する可能性があります。

接続プールの適切なサイズは？

構成におけるすべての経験則と同じく、答えは：場合によって異なる。適したサイズは予測されたロードとデータベース トランザクション時間の平均によりますが、最適な接続プールのサイズは予想するより小さなものになりま

す。Sun の Java Petstore ブループリント アプリケーションを例に挙げると、許容可能な応答時間での MySQL または Tomcat を使用して、15-20 接続の接続プールは比較的緩やかなロード (600 平行ユーザ) をサーブすることができます。

アプリケーションへの接続プールのサイズを正しく計るには、Apache JMeter または The Grinder のようなロードテスト スクリプトを作成し、アプリケーションをロードテストしてください。

開始ポイントを判断する簡単な方法は、接続プールの接続の最大数を非有界なよう構成し、ロードテストを実行、同時に使用された接続の最大数を計ります。そこから逆に作業して、プールされた接続の最小、または最大のどんな値が、特定のアプリケーションに最高の性能を与えるか判断することができます。

Tomcat で Connector/J を使用する

次の指示は Tomcat-5.x のインストラクションに基づいており、この資料が書かれた時の現行版だったものは <http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html> で見ることができます。

まず、`$CATALINA_HOME/common/lib` 内の Connector/J についてくる .jar ファイルをインストールし、コンテナにインストールされたすべてのアプリケーションが利用できるようにします。

次に、ウェブ アプリケーションを定義するコンテキストで、宣言リソースを `$CATALINA_HOME/conf/server.xml` に追加することにより、JNDI DataSource を構築します：

```
<Context ....>

...

<Resource name="jdbc/MySQLDB"
    auth="Container"
    type="javax.sql.DataSource"/>

<!-- The name you used above, must match _exactly_ here!

    The connection pool will be bound into JNDI with the name
    "java:/comp/env/jdbc/MySQLDB"
-->

<ResourceParams name="jdbc/MySQLDB">
  <parameter>
    <name>factory</name>
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>

  <!-- Don't set this any higher than max_connections on your
    MySQL server, usually this should be a 10 or a few 10's
    of connections, not hundreds or thousands -->

  <parameter>
    <name>maxActive</name>
    <value>10</value>
  </parameter>

  <!-- You don't want to many idle connections hanging around
    if you can avoid it, only enough to soak up a spike in
    the load -->

  <parameter>
    <name>maxIdle</name>
    <value>5</value>
  </parameter>

  <!-- Don't use autoReconnect=true, it's going away eventually
    and it's a crutch for older connection pools that couldn't
    test connections. You need to decide whether your application
    is supposed to deal with SQLExceptions (hint, it should), and
    how much of a performance penalty you're willing to pay
    to ensure 'freshness' of the connection -->

  <parameter>
    <name>validationQuery</name>
    <value>SELECT 1</value>
  </parameter>
```

```
<!-- The most conservative approach is to test connections
before they're given to your application. For most applications
this is okay, the query used above is very small and takes
no real server resources to process, other than the time used
to traverse the network.

If you have a high-load application you'll need to rely on
something else. -->

<parameter>
  <name>testOnBorrow</name>
  <value>true</value>
</parameter>

<!-- Otherwise, or in addition to testOnBorrow, you can test
while connections are sitting idle -->

<parameter>
  <name>testWhileIdle</name>
  <value>true</value>
</parameter>

<!-- You have to set this value, otherwise even though
you've asked connections to be tested while idle,
the idle evictor thread will never run -->

<parameter>
  <name>timeBetweenEvictionRunsMillis</name>
  <value>10000</value>
</parameter>

<!-- Don't allow connections to hang out idle too long,
never longer than what wait_timeout is set to on the
server...A few minutes or even fraction of a minute
is sometimes okay here, it depends on your application
and how much spikey load it will see -->

<parameter>
  <name>minEvictableIdleTimeMillis</name>
  <value>60000</value>
</parameter>

<!-- Username and password used when connecting to MySQL -->

<parameter>
  <name>username</name>
  <value>someuser</value>
</parameter>

<parameter>
  <name>password</name>
  <value>somepass</value>
</parameter>

<!-- Class name for the Connector/J driver -->

<parameter>
  <name>driverClassName</name>
  <value>com.mysql.jdbc.Driver</value>
</parameter>

<!-- The JDBC connection url for connecting to MySQL, notice
that if you want to pass any other MySQL-specific parameters
you should pass them here in the URL, setting them using the
parameter tags above will have no effect, you will also
need to use & to separate parameter values as the
ampersand is a reserved character in XML -->

<parameter>
  <name>url</name>
  <value>jdbc:mysql://localhost:3306/test</value>
</parameter>

</ResourceParams>
```



```
</Context>
```

通常は、Tomcat のデータソースを構成する方法は随時変化し、XML ファイルで間違ったシンタックスを使用している場合は残念ながら、以下に似た例外が投入される可能性が高いので、Tomcat のバージョンについてくるインストール インストラクションにしたがってください。

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null' SQL
state: null
```

JBoss で Connector/J を使用する

このインストラクションは JBoss-4.x をカバーしています。アプリケーション サーバで JDBC ドライバ クラスを利用可能にするには、Connector/J についてくる .jar ファイルを、サーバ構築のために、[lib](#) ディレクトリにコピーします (これは通常 [default](#) と呼ばれる)。そして、同じ構築ディレクトリで、deploy と名付けられたサブディレクトリで、"-ds.xml" で終わるデータソース 構築ファイルを作成します。"-ds.xml" はこのファイルを JDBC Datasource としてデプロイするよう JBoss に指示します。ファイルには次の内容が含まれています：

```
<datasources>
  <local-tx-datasource>
    <!-- This connection pool will be bound into JNDI with the name
         "java:/MySQLDB" -->

    <jndi-name>MySQLDB</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>user</user-name>
    <password>pass</password>

    <min-pool-size>5</min-pool-size>

    <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

    <max-pool-size>20</max-pool-size>

    <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->

    <idle-timeout-minutes>5</idle-timeout-minutes>

    <!-- If you're using Connector/J 3.1.8 or newer, you can use
         our implementation of these to increase the robustness
         of the connection pool. -->

    <exception-sorter-class-name>com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter</exception-sorter-class-name>
    <valid-connection-checker-class-name>com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker</valid-connection-checker-class-name>

  </local-tx-datasource>
</datasources>
```

よくある問題と解決法

MySQL Connector/J のユーザがよく直面する問題がいくつかあります。このセクションでは、それらの兆候と解決法を説明します。

Questions

- [24.4.5.3.1: \[1422\]](#) MySQL Connector/J でデータベースに接続しようとする、次の例外が出ます：

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

これは为什么呢？MySQL コマンドライン クライアントとは問題なく接続できます。

- [24.4.5.3.2: \[1422\]](#) アプリケーションが SQLException 'No Suitable Driver' を投入するのですが、なぜでしょうか？

- [24.4.5.3.3: \[1422\]](#) アプレットまたはアプリケーションで MySQL Connector/J を使用したいのですが、次に似た例外が出ます：

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

- [24.4.5.3.4: \[1423\]](#) 1 日は問題なく動くのですが、夜の間に停止してしまうサーブレット/アプリケーションがあります。
- [24.4.5.3.5: \[1425\]](#) JDBC-2.0 の更新可能な結果セットを使おうとすると、その結果セットは更新不可能という例外が出ます。
- [24.4.5.3.6: \[1425\]](#) Connector/J を使って MySQL サーバに接続できません。接続パラメータが間違っているようです。

Questions and Answers

24.4.5.3.1: MySQL Connector/J でデータベースに接続しようとする、次の例外が出ます：

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

これは为什么呢？MySQL コマンドライン クライアントとは問題なく接続できます。

java は Unix Domain Sockets をサポートしないため、MySQL Connector/J は MySQL への接続に TCP/IP ソケットを使用する必要があります。そのため、MySQL Connector/J が MySQL に接続する場合、MySQL サーバのセキュリティマネージャはその grant テーブルを使用して、接続が許可されるべきかを決定します。

MySQL サーバに [GRANT](#) 文を使用して、これが起きるように、必要なセキュリティ証明書を MySQL サーバに追加する必要があります。詳細は [GRANT 構文](#) を参照。

注意. [mysql](#) コマンドライン クライアントでの接続のテストは、[--host](#) フラグを加え、[localhost](#) 以外をホストに使用しなければ作動しません。[mysql](#) コマンドライン クライアントは、特別なホスト名 [localhost](#) を使っている場合、Unix ドメイン ソケットを使用しません。[localhost](#) への接続をテストしている場合、ホスト名に [127.0.0.1](#) を代わりに使用してください。

注意. MySQL で権限や許可を不適切に変更すると、サーバが適したセキュリティプロパティを持たないでインストールされる恐れがあります。

24.4.5.3.2: アプリケーションが SQLException 'No Suitable Driver' を投入するのですが、なぜでしょうか？

このエラーには 3 つの原因が考えられます：

- Connector/J driver が [CLASSPATH](#) にない。[Connector/J のインストール](#) 参照。
- 接続 URL のフォーマットが間違っている、または誤った JDBC ドライバを参照している。
- DriverManager を使用している場合、[jdbc.drivers](#) システム プロパティに Connector/J ドライバのロケーションが取り込まれていない。

24.4.5.3.3: アプレットまたはアプリケーションで MySQL Connector/J を使用したいのですが、次に似た例外が出ます：

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

アプレットを起動しているか、MySQL サーバが "--skip-networking" オプション セットとインストールされている、または MySQL サーバの前にファイアウォールがあるということが考えられます。

アプレットは、そのアプレットに .class ファイルをサブしたウェブ サーバを実行するマシンにだけネットワーク接続を確立し戻すことができます。つまり、これを実行するには、MySQL を同じマシン上で起動する必要があります (もしくは何らかのポート リダイレクトが必要)。これはまた、ローカル ファイル システムからはアプレットをテストすることができないということで、それらをいつもウェブ サーバにデプロイする必要があります。

Java は Unix ドメイン ソケットをサポートしないので、MySQL Connector/J は TCP/IP を使用して、MySQL とだけ交信することができます。MySQL を "--skip-networking" フラグで起動した場合、またはファイアウォールされている場合は、TCP/IP の MySQL との交信は影響を受けている可能性があります。

MySQL が "--skip-networking" オプション セットと起動された場合 (例えば MySQL サーバの Debian Linux パッケージはこれを行いません)、file /etc/mysql/my.cnf or /etc/my.cnf でそれをコメントアウトする必要があります。当然、my.cnf file もMySQL サーバの [data](#) ディレクトリか他の場所にある場合があります (MySQL がシステムのためにどのようにコンパイルされたかによる)。MySQL AB で作成されたバイナリは常に、/etc/my.cnf and [datadir]/my を調査します。MySQL サーバがファイアウォールされている場合、MySQL が監視しているポートの MySQL サーバへ Java コードを実行しているホストからの TCP/IP を許可するよう、ファイアウォールを構成する必要があります。(デフォルトでは 3306)。

24.4.5.3.4: 1 日は問題なく動くのですが、夜の間に停止してしまうサーバレット/アプリケーションがあります。

MySQL は、8 時間動きがなければ接続を閉じてしまいます。停滞した接続を扱う接続プールを使用するか、"autoReconnect" パラメータを使用する必要があります ([Connector/J の Driver/Datasource クラス名、URL シンタックス、および構成プロパティ](#) 参照)。

また、アプリケーションが終了するまでSQLExceptions をプロンプトするより、アプリケーションでキャッチして対処すべきでしょう。プログラミングのよい練習です。MySQL Connector/J は、クエリ処理中にネットワーク接続の問題に直面した場合、SQLState (APIDocs 内の [java.sql.SQLException.getSQLState\(\)](#) 参照) を "08S01" に設定します。アプリケーション コードは、この時点で MySQL への再接続を試行します。

次の (単純な) 例は、これらの例外を扱えるコードはどんな様子かを表しています。

例24.12 リトライ ロジックとのトランザクション例

```
public void doBusinessOp() throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    //
    // How many times do you want to retry the transaction
    // (or at least _getting_ a connection)?
    //
    int retryCount = 5;

    boolean transactionCompleted = false;

    do {
        try {
            conn = getConnection(); // assume getting this from a
                                   // javax.sql.DataSource, or the
                                   // java.sql.DriverManager

            conn.setAutoCommit(false);

            //
            // Okay, at this point, the 'retry-ability' of the
            // transaction really depends on your application logic,
            // whether or not you're using autocommit (in this case
            // not), and whether you're using transactional storage
            // engines
            //
            // For this example, we'll assume that it's _not_ safe
            // to retry the entire transaction, so we set retry
            // count to 0 at this point
            //
            // If you were using exclusively transaction-safe tables,
```

```
// or your application could recover from a connection going
// bad in the middle of an operation, then you would not
// touch 'retryCount' here, and just let the loop repeat
// until retryCount == 0.
//
retryCount = 0;

stmt = conn.createStatement();

String query = "SELECT foo FROM bar ORDER BY baz";

rs = stmt.executeQuery(query);

while (rs.next()) {
}

rs.close();
rs = null;

stmt.close();
stmt = null;

conn.commit();
conn.close();
conn = null;

transactionCompleted = true;
} catch (SQLException sqlEx) {

    //
    // The two SQL states that are 'retry-able' are 08S01
    // for a communications error, and 40001 for deadlock.
    //
    // Only retry if the error was due to a stale connection,
    // communications problem or deadlock
    //

    String sqlState = sqlEx.getSQLState();

    if ("08S01".equals(sqlState) || "40001".equals(sqlState)) {
        retryCount--;
    } else {
        retryCount = 0;
    }
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) {
            // You'd probably want to log this . . .
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) {
            // You'd probably want to log this as well . . .
        }
    }

    if (conn != null) {
        try {
            //
            // If we got here, and conn is not null, the
            // transaction should be rolled back, as not
            // all work has been done

            try {
                conn.rollback();
            } finally {
                conn.close();
            }
        } catch (SQLException sqlEx) {
            //

```

```

// If we got an exception here, something
// pretty serious is going on, so we better
// pass it up the stack, rather than just
// logging it. . .

    throw sqlEx;
}
}
} while (!transactionCompleted && (retryCount > 0));
}

```

注意. 接続状態やデータベース状態の情報が破損する危険なしに MySQL サーバに再接続できる安全なメソッドはないので、[autoReconnect](#) オプションの使用はお勧めしません。その代わりに、プールからの利用可能な接続を使用して、MySQL サーバに接続するアプリケーションを有効にする接続プールを使いましょう。[autoReconnect](#) 機能は廃止になり、今後のリリースでは除外される可能性があります。

24.4.5.3.5: JDBC-2.0 の更新可能な結果セットを使おうとすると、その結果セットは更新不可能という例外が出ます。

MySQL は行識別子を持たないため、MySQL Connector/J は、少なくともプライマリ キーをひとつ持つテーブルのクエリからの結果セットしか更新できず、クエリはすべてのプライマリ キーと、テーブルをひとつだけスパンすることができるクエリを選択する必要があります (結合はなし)。これは JDBC 仕様に概要があります。

この問題は更新可能な結果セットを使用する時にだけ起こり、Connector/J は、各行の一意な参照なしで更新される結果セット内の正しい行を特定することができるとは保証できません。[WHERE](#) 句を使って、整合される基準値をそれぞれ特定することのできるテーブルで [UPDATE](#) または [DELETE](#) 文を使用している場合、テーブルに一意的なフィールドを持つための必須条件はありません。

24.4.5.3.6: Connector/J を使って MySQL サーバに接続できません。接続パラメータが間違っているようです。

サーバで [skip-networking](#) オプションが有効になっていないことを確かめてください。Connector/J は TCP/IP でサーバと通信しなければなりません。名前付きソケットはサポートされていません。また、接続を Firewall や他のネットワーク セキュリティ システムに通さないように注意してください。詳細は [Can't connect to \[local\] MySQL server](#) をご覧ください。

Connector/J のサポート

Connector/J のコミュニティ支援

MySQL AB はメーリング リストを用いて、ユーザ コミュニティーを後援します。Connector/J 関連の問題については、MySQL と Java メーリングリストから、経験豊富なユーザに援助を求めることができます。アーカイブと加入情報はオンラインで、<http://lists.mysql.com/java> に閲覧できます。

MySQL メーリングリストへの参加、リスト アーカイブの参照については、<http://lists.mysql.com/>、[MySQL メーリングリスト](#) 参照。

経験豊富なユーザからのコミュニティ支援は、[JDBC Forum](#) でも得ることができます。また、<http://forums.mysql.com> にあるもうひとつの MySQL Forum でも、他のユーザとの情報交換が行えます。[MySQL フォーラムにおける MySQL コミュニティサポート](#) 参照。

Connector/J のバグまたは不具合のレポート

通常のバグのレポートは、弊社のバグ データベース <http://bugs.mysql.com/> で行ってください。このデータベースは一般公開されており、どなたでも参照、検索することができます。このシステムにログインすると、新しいレポートを投稿することもできます。

MySQL のセキュリティに深刻なバグを発見した場合は、security_at_mysql.com まで E メールでお知らせください。

正確なバグ レポートを書くのは時間がかかるものですが、レポートを一度で済ませることで、報告者と弊社、双方の時間を節約することができます。そのバグの完全なテスト ケースを含むバグ レポートを提供していただければ、次のリリースではその問題を修正できる可能性が高くなります。

このセクションでは、ユーザの時間短縮と効果的な情報提供のため、レポートの正しい書き方を紹介します。

再現可能なバグ レポートがある場合は、<http://bugs.mysql.com/> のバグ データベースへ報告してください。弊社で再現可能なバグであれば、次回の MySQL リリースで修正される可能性が高くなります。

他の問題の報告には、MySQL メーリングリストのいずれかをご利用ください。

多大な情報をお寄せいただくのは特に問題はありませんが、情報があまりにも少ないと対処が難しくなりますのでご了承ください。問題に直接関係しないと判断し、詳細を省略して報告されることがしばしばありますのでご注意ください。

ご留意いただきたい点は以下です : 報告に含むべきか迷う情報があれば、それも加えてください。最初の報告で情報が足りない場合は後日お尋ねすることになりますので、はじめのレポートにできるだけ詳細を書いていただくと、作業がより速く簡潔になります。

バグのレポートによくある間違いは、(a) 使用した Connector/J または MySQL のバージョン番号の書き忘れ、(b) Connector/J がインストールされたプラットフォームの説明が不完全なこと (MySQL そのものがインストールされた JVM のバージョン、プラットフォームのタイプ、バージョン番号、等)、です。

これは非常に重要な情報で、99% の場合、この情報なしでは報告を活用することができません。「なぜうまく作動しないのか？」という質問が頻繁に寄せられます。そのほとんどは、希望の機能がその MySQL バージョンに実装されていないことが原因で、もしくは、報告にあるバグは新しい MySQL のバージョンではすでに修正されています。

エラーの原因はプラットフォームにあることもあります。その場合、オペレーション システムとプラットフォームのバージョン番号の情報なしで解決することはほぼ不可能です。

もしできれば、再現可能で、サードパーティ クラスが関与しない独立したテストケースを作成してください。

このプロセスを能率化するため、弊社では、'[com.mysql.jdbc.util.BaseBugReport](#)' と名付けられたテストケース用のベースクラスを Connector/J と共に発送しています。このクラスを使用して、Connector/J のテストケースを作成するには、[com.mysql.jdbc.util.BaseBugReport](#) から継承する独自のクラスを作成し、メソッド [setUp\(\)](#)、[tearDown\(\)](#)、および [runTest\(\)](#) をオーバーライドしてください。

[setUp\(\)](#) メソッドで、テーブルを作るコードを作成し、バグの証明に必要なデータを取り込みます。

[runTest\(\)](#) メソッドで、[setUp](#) メソッドで作ったテーブルとデータを使用して、バグを証明するコードを作成します。

[tearDown\(\)](#) メソッドで、[setUp\(\)](#) メソッドで作ったテーブルのいずれかを削除します。

上記の 3 つのメソッドのどれかで、[getConnection\(\)](#) メソッドの変異型のひとつを使用して、MySQL への JDBC 接続を作成してください :

- [getConnection\(\)](#) - [getUrl\(\)](#) で特定された JDBC URL への接続を提供。接続がすでにある場合、その接続は戻され、それでなければ新たな接続が作成される。
- [getNewConnection\(\)](#) - バグ レポートのために新しい接続が必要な場合はこれを使用 (つまり、ひとつ以上の接続が関与する)。
- [getConnection\(String url\)](#) - 与えられた URL を使用して接続を戻す。
- [getConnection\(String url, Properties props\)](#) - 与えられた URL とプロパティを使用して接続を戻す。

'jdbc:mysql:///test' とは異なる JDBC URL を使う必要がある場合、メソッド [getUrl\(\)](#) もオーバーライドします。

[assertTrue\(boolean expression\)](#) および [assertTrue\(String failureMessage, boolean expression\)](#) メソッドを使用して、証明したい動作のテストケースにあう条件を作成します (観察している動作と比べ、それによってほぼバグ レポートを送ることになります)。

最後に、テストケースの新しいインスタンスを作る [main\(\)](#) メソッドを作成し、[run](#) メソッドを呼び出します :

```
public static void main(String[] args) throws Exception {
    new MyBugReport().run();
}
```

テストケースを終え、報告したいバグが証明されていることを確認したら、バグ レポートを添えて <http://bugs.mysql.com/> にアップロードします。

Connector/J 変更履歴

Connector/J Change History (Changelog) は、MySQL のメイン Changelog に収められています。[MySQL Connector/J Change History](#) 参照。

Connector/PHP

PHP とその資料は、PHP ウェブサイトから入手することができます。MySQL では Windows のオペレーティング システムでの使用を対象に、MySQL バージョン 4.1.16 以降と MySQL 5.0.18 、そして MySQL 5.1 に対応する [mysql](#) エクステンションと [mysqli](#) エクステンションを <http://dev.mysql.com/downloads/connector/php/> で提供しています。MySQL が提供するエクステンションの使用の推奨に関しては、同ウェブページをご覧ください。Windows 以外のプラットフォームでは、PHP ソースに同梱されている [mysql](#) エクステンションまたは [mysqli](#) エクステンションを使用してください。詳しくは[MySQL PHP API](#)を参照してください。

第25章 Extending MySQL

目次

MySQL Internals	1429
MySQL Threads	1429
MySQL Test Suite	1430
The MySQL Plugin Interface	1430
Characteristics of the Plugin Interface	1431
Full-Text Parser Plugins	1432
INSTALL PLUGIN Syntax	1433
UNINSTALL PLUGIN Syntax	1433
Writing Plugins	1434
Adding New Functions to MySQL	1446
Features of the User-Defined Function Interface	1447
CREATE FUNCTION Syntax	1447
DROP FUNCTION Syntax	1447
Adding a New User-Defined Function	1448
Adding a New Native Function	1455
Adding New Procedures to MySQL	1456
Procedure Analyse	1456
Writing a Procedure	1456

MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. If you plan to contribute to MySQL development, want to have access to the bleeding-edge versions of the code, or just want to keep track of development, follow the instructions in [開発ソース ツリーからのインストール](#). If you are interested in MySQL internals, you should also subscribe to our [internals](#) mailing list. This list has relatively low traffic. For details on how to subscribe, please see [MySQL メーリング リスト](#). All developers at MySQL AB are on the [internals](#) list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

MySQL Threads

The MySQL server creates the following threads:

- One thread manages TCP/IP file connection requests and creates a new dedicated thread to handle the authentication and SQL statement processing for each connection. (On Unix, this thread also manages Unix socket file connection requests.) On Windows, a similar thread manages shared-memory connection requests, and on Windows NT-based systems, a thread manages named-pipe connection requests. Every client connection has its own thread, although the manager threads try to avoid creating threads by consulting the thread cache first to see whether a cached thread can be used for a new connection.
- On Windows NT, there is a named pipe handler thread that does the same work as the TCP/IP connection thread on named pipe connect requests.
- On a master replication server, slave server connections are like client connections: There is one thread per connected slave.
- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.
- The signal thread handles all signals. This thread also normally handles alarms and calls [process_alarm\(\)](#) to force timeouts on connections that have been idle too long.

- If `mysqld` is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
 - If the server is started with the `--flush_time=val` option, a dedicated thread is created to flush all tables every `val` seconds.
 - Each table for which `INSERT DELAYED` statements are issued gets its own thread.
 - If the event scheduler is active, there is one thread for the scheduler, and a thread for each event currently running.
- `mysqladmin processlist` only shows the connection, `INSERT DELAYED`, replication threads, and event threads.

MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (`mysqltest`), a shell script to run all tests (`mysql-test-run.pl`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make test` from the source root directory, or change location to the `mysql-test` directory and type `./mysql-test-run.pl`. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run `./mysql-test-run.pl`. All tests should succeed. If any do not, you should try to find out why and report the problem if it indicates a bug in MySQL. See [質問またはバグの報告](#).

If one test fails, you should run `mysql-test-run.pl` with the `--force` option to check whether any other tests fail.

If you have a copy of `mysqld` running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports `9306` or `9307`. If either of those ports is taken, you should edit `mysql-test-run.pl` and change the values of the master or slave port to one that is available.

In the `mysql-test` directory, you can run an individual test case with `./mysql-test-run test_name`.

You can use the `mysqltest` language to write your own test cases. This is documented in the MySQL Test Framework manual, available at <http://dev.mysql.com/doc/>.

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL [internals](#) mailing list. See [MySQL メーリング リスト](#). This list does not accept attachments, so you should FTP all the relevant files to: <ftp://ftp.mysql.com/pub/mysql/upload/>

The MySQL Plugin Interface

MySQL 5.1 and up supports a plugin API that allows the loading and unloading of server components at runtime, without restarting the server. Currently, the plugin API supports creation of full-text parser plugins. Such a plugin can be used to replace or augment the built-in full-text parser. For example, a plugin can parse text into words using rules that differ from those used by the built-in parser. This can be useful if you need to parse text with characteristics different from those expected by the built-in parser.

The plugin interface is intended as the successor to the older user-defined function (UDF) interface. The plugin interface eventually will include an API for creating UDFs, and it is intended this plugin UDF API will replace the older non-plugin UDF API. After that point, it will be possible for UDFs to be revised for use as plugin UDFs so that they can take advantage of the better security and versioning capabilities of the plugin API. Eventually, support for the older UDF API will be phased out.

The plugin interface requires the `plugin` table in the `mysql` database. This table is created as part of the MySQL installation process. If you are upgrading from an older version to MySQL 5.1, you should run the `mysql_upgrade` command to create this table. See [mysql_upgrade — MySQL アップグレードのテーブル チェック](#).

Characteristics of the Plugin Interface

In some respects, the plugin API is similar to the older user-defined function (UDF) API that it supersedes, but the plugin API has several advantages over the older interface:

- The plugin framework is extendable to accommodate different kinds of plugins.

Some aspects of the plugin API are common to all types of plugins, but the API also allows for type-specific interface elements so that different types of plugins can be created. A plugin with one purpose can have an interface most appropriate to its own requirements and not the requirements of some other plugin type.

Although only the interface for full-text parser plugins is implemented currently, others can be added, such as an interface for UDF plugins.

- The plugin API includes versioning information.

The version information included in the plugin API enables a plugin library and each plugin that it contains to be self-identifying with respect to the API version that was used to build the library. If the API changes over time, the version numbers will change, but a server can examine a given plugin library's version information to determine whether it supports the plugins in the library.

There are two types of version numbers. The first is the version for the general plugin framework itself. Each plugin library includes this kind of version number. The second type of version applies to individual plugins. Each specific type of plugin has a version for its interface, so each plugin in a library has a type-specific version number. For example, library containing a full-text parsing plugin has a general plugin API version number, and the plugin has a version number specific to the full-text plugin interface.

- Plugin security is improved relative to the UDF interface.

The older interface for writing non-plugin UDFs allowed libraries to be loaded from any directory searched by the system's dynamic linker, and the symbols that identified the UDF library were relatively non-specific. The newer rules are more strict. A plugin library must be installed in a specific dedicated directory for which the location is controlled by the server and cannot be changed at runtime. Also, the library must contain specific symbols that identify it as a plugin library. The server will not load something as a plugin if it was not built as a plugin.

The newer plugin interface eliminates the security issues of the older UDF interface. When a UDF plugin type is implemented, that will allow non-plugin UDFs to be brought into the plugin framework and the older interface to be phased out.

The plugin implementation includes the following components:

Source files (the locations given indicate where the files are found in a MySQL source distribution):

- [include/mysql/plugin.h](#) exposes the public plugin API. This file should be examined by anyone who wants to write a plugin library.
- [sql/sql_plugin.h](#) and [sql/sql_plugin.cc](#) comprise the internal plugin implementation. These files need not be consulted by plugin writers. They may be of interest for those who want to know more about how the server handles plugins.

System table:

- The [plugin](#) table in the [mysql](#) database lists each installed plugin and is required for plugin use. For new MySQL installations, this table is created during the installation process. If you are upgrading from a version older than MySQL 5.1, you should run [mysql_upgrade](#) to update your system tables and create the [plugin](#) table (see [mysql_upgrade — MySQL アップグレードのテーブル チェック](#)).

SQL statements:

- [INSTALL PLUGIN](#) registers a plugin in the [plugin](#) table and loads the plugin code.
- [UNINSTALL PLUGIN](#) unregisters a plugin from the [plugin](#) table and unloads the plugin code.
- The [WITH PARSER](#) clause for full-text index creation associates a full-text parser plugin with a given [FULLTEXT](#) index.

- [SHOW PLUGINS](#) displays information about known plugins. The [PLUGINS](#) table in [INFORMATION_SCHEMA](#) also contains plugin information.

System variable:

- [plugin_dir](#) indicates the location of the directory where all plugins must be installed. The value of this variable can be specified at server startup with a `--plugin_dir=path` option.

Full-Text Parser Plugins

MySQL has a built-in parser that it uses by default for full-text operations (parsing text to be indexed, or parsing a query string to determine the terms to be used for a search). For full-text processing, 「parsing」 means extracting words from text or a query string based on rules that define which character sequences make up a word and where word boundaries lie.

When parsing for indexing purposes, the parser passes each word to the server, which adds it to a full-text index. When parsing a query string, the parser passes each word to the server, which accumulates the words for use in a search.

The parsing properties of the built-in full-text parser are described in [全文検索関数](#). These properties include rules for determining how to extract words from text. The parser is influenced by certain system variables such as [ft_min_word_len](#) and [ft_max_word_len](#) that cause words shorter or longer to be excluded, and by the stopword list that identifies common words to be ignored.

The plugin API enables you to provide a full-text parser of your own so that you have control over the basic duties of a parser. A parser plugin can operate in either of two roles:

- The plugin can replace the built-in parser. In this role, the plugin reads the input to be parsed, splits it up into words, and passes the words to the server (either for indexing or for word accumulation).

One reason to use a parser this way is that you need to use different rules from those of the built-in parser for determining how to split up input into words. For example, the built-in parser considers the text 「case-sensitive」 to consist of two words 「case」 and 「sensitive」 whereas an application might need to treat the text as a single word.

- The plugin can act in conjunction with the built-in parser by serving as a front end for it. In this role, the plugin extracts text from the input and passes the text to the parser, which splits up the text into words using its normal parsing rules. In particular, this parsing will be affected by the [ft_XXX](#) system variables and the stopword list.

One reason to use a parser this way is that you need to index content such as PDF documents, XML documents, or [.doc](#) files. The built-in parser is not intended for those types of input but a plugin can pull out the text from these input sources and pass it to the built-in parser.

It is also possible for a parser plugin to operate in both roles. That is, it could extract text from non-plaintext input (the front end role), and also parse the text into words (thus replacing the built-in parser).

A full-text plugin is associated with full-text indexes on a per-index basis. That is, when you install a parser plugin initially, that does not cause it to be used for any full-text operations. It simply becomes available. For example, a full-text parser plugin becomes available to be named in a [WITH PARSE](#)R clause when creating individual [FULLTEXT](#) indexes. To create such an index at table-creation time, do this:

```
CREATE TABLE t
(
  doc CHAR(255),
  FULLTEXT INDEX (doc) WITH PARSE
```

[R](#) my_parser
);

Or you can add the index after the table has been created:

```
ALTER TABLE t ADD FULLTEXT INDEX (doc) WITH PARSE
```

[R](#) my_parser;

The only SQL change for associating the parser with the index is the [WITH PARSE](#)R clause. Searches are specified as before, with no changes needed for queries.

When you associate a parser plugin with a [FULLTEXT](#) index, the plugin is required for using the index. If the parser plugin is dropped, any index associated with it becomes unusable. Any attempt to use it a table for which a plugin is not available results in an error, although [DROP TABLE](#) is still possible.

INSTALL PLUGIN Syntax

```
INSTALL PLUGIN plugin\_name SONAME 'plugin\_library'
```

This statement installs a plugin.

[plugin_name](#) is the name of the plugin as defined in the plugin declaration structure contained in the library file. Plugin name case sensitivity is determined by the host system filename semantics.

[plugin_library](#) is the name of the shared library that contains the plugin code. The name includes the filename extension (for example, [libmyplugin.so](#) or [libmyplugin.dylib](#)).

The shared library must be located in the plugin directory (that is, the directory named by the [plugin_dir](#) system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, [plugin_dir](#) is the directory named by the [pkglibdir](#) configuration variable, but it can be changed by setting the value of [plugin_dir](#) at server startup. For example, set its value in a [my.cnf](#) file:

```
[mysqld]
plugin\_dir=/path/to/plugin/directory
```

If the value of [plugin_dir](#) is a relative pathname, it is taken to be relative to the MySQL base directory (the value of the [basedir](#) system variable).

[INSTALL PLUGIN](#) adds a line to the [mysql.plugin](#) table that describes the [plugin](#). This table contains the plugin name and library filename.

[INSTALL PLUGIN](#) also loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used.

To use [INSTALL PLUGIN](#), you must have the [INSERT privilege](#) for the [mysql.plugin](#) table.

At server startup, the server loads and initializes any plugin that is listed in the [mysql.plugin](#) table. This means that a plugin is installed with [INSTALL PLUGIN](#) only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the [--skip-grant-tables](#) option.

When the server shuts down, it executes the deinitialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

To remove a plugin entirely, use the [UNINSTALL PLUGIN](#) statement:

To see what plugins are installed, use the [SHOW PLUGIN](#) statement.

If you recompile a plugin library and need to reinstall it, you can use either of the following procedures:

- Use [UNINSTALL PLUGIN](#) to uninstall all plugins in the library, install the new plugin library file in the plugin directory, and then use [INSTALL PLUGIN](#) to install all plugins in the library. This procedure has the advantage that it can be used without stopping the server. However, if the plugin library contains many plugins, you must issue many [INSTALL PLUGIN](#) and [UNINSTALL PLUGIN](#) statements.
- Alternatively, stop the server, install the new plugin library file in the plugin directory, and then restart the server.

UNINSTALL PLUGIN Syntax

```
UNINSTALL PLUGIN plugin\_name
```

This statement removes an installed plugin. You cannot uninstall a plugin if any table that uses it is open.

[plugin_name](#) must be the name of some plugin that is listed in the [mysql.plugin](#) table. The server executes the plugin's deinitialization function and removes the row for the plugin from the [mysql.plugin](#) table, so that subsequent server restarts will not load and initialize the plugin. [UNINSTALL PLUGIN](#) does not remove the plugin's shared library file.

To use [UNINSTALL PLUGIN](#), you must have the [DELETE](#) privilege for the [mysql.plugin](#) table.

Plugin removal has implications for the use of associated tables. For example, if a full-text parser plugin is associated with a [FULLTEXT](#) index on the table, uninstalling the plugin makes the table unusable. Any attempt to access the table results in an error. The table cannot even be opened, so you cannot drop an index for which the plugin is used. This means that uninstalling a plugin is something to do with care unless you do not care about the table contents. If you are uninstalling a plugin with no intention of reinstalling it later and you care about the table contents, you should dump the table with [mysqldump](#) and remove the [WITH PARSER](#) clause from the dumped [CREATE TABLE](#) statement so that you can reload the table later. If you do not care about the table, [DROP TABLE](#) can be used even if any plugins associated with the table are missing.

Writing Plugins

This section describes the general and type-specific parts of the plugin API. It also provides a step-by-step guide to creating a plugin library. For example plugin source code, see the [plugin/fulltext](#) directory of a MySQL source distribution.

You can write plugins in C or C++ (or another language that can use C calling conventions). Plugins are loaded and unloaded dynamically, so your operating system must support dynamic loading and you must have compiled [mysqld](#) dynamically (not statically).

A plugin contains code that becomes part of the running server, so when you write a plugin, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the [libstdc++](#) library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to plugins that were originally written for older servers. For information about these constraints, see [典型的な configure オプション](#), and [MySQL のコンパイルに関する問題](#).

General Plugin Structures and Functions

Every plugin must have a general plugin declaration. The declaration corresponds to the [st_mysql_plugin](#) structure in the [plugin.h](#) file:

```
struct st_mysql_plugin
{
    int type;           /* the plugin type (a MYSQL_XXX_PLUGIN value) */
    void *info;         /* pointer to type-specific plugin descriptor */
    const char *name;    /* plugin name */
    const char *author;  /* plugin author (for SHOW PLUGINS) */
    const char *descr;   /* general descriptive text (for SHOW PLUGINS) */
    int (*init)(void);  /* the function to invoke when plugin is loaded */
    int (*deinit)(void); /* the function to invoke when plugin is unloaded */
    unsigned int version; /* plugin version (for SHOW PLUGINS) */
    struct st_mysql_show_var *status_vars;
};
```

The [st_mysql_plugin](#) structure is common to every type of plugin. Its members should be filled in as follows:

- [type](#)

The plugin type. This must be one of the plugin-type values from [plugin.h](#). For a full-text parser plugin, the [type](#) value is [MYSQL_FTPARSER_PLUGIN](#).

- [info](#)

A pointer to the descriptor for the plugin. Unlike the general plugin declaration structure, this descriptor's structure depends on the particular type of plugin. Each descriptor has a version number that indicates the API version for that type of plugin, plus any other members needed. The descriptor for full-text plugins is described in [Type-Specific Plugin Structures and Functions](#).

- [name](#)

The plugin name. This is the name that will be listed in the [plugin](#) table and by which you refer to the plugin in SQL statements such as [INSTALL PLUGIN](#) and [UNINSTALL PLUGIN](#).

- [author](#)

The plugin author. This can be whatever you like.

- `desc`

A general description of the plugin. This can be whatever you like.

- `init`

A once-only initialization function. This is executed when the plugin is loaded, which happens for `INSTALL PLUGIN` or, for plugins listed in the `plugin` table, at server startup. The function takes no arguments. It returns zero for success and non-zero for failure. If an `init` function is unneeded for a plugin, it can be specified as 0.

- `deinit`

A once-only deinitialization function. This is executed when the plugin is unloaded, which happens for `UNINSTALL PLUGIN` or, for plugins listed in the `plugin` table, at server shutdown. The function takes no arguments. It returns zero for success and non-zero for failure. If a `deinit` function is unneeded for a plugin, it can be specified as 0.

- `version`

The plugin version number. When the plugin is installed, this value can be retrieved from the `INFORMATION_SCHEMA.PLUGINS` table. The value includes major and minor numbers. If you write the value as a hex constant, the format is `0xMMNN`, where `MM` and `NN` are the major and minor numbers, respectively. For example, `0x0302` represents version 3.2.

- `status_vars`

A pointer to a structure for status variables associated with the plugin, or 0 if there are no such variables. When the plugin is installed, these variables are displayed in the output of the `SHOW STATUS` statement.

The `init` and `deinit` functions in the general plugin declaration are invoked only when loading and unloading the plugin. They have nothing to do with use of the plugin such as happens when an SQL statement causes the plugin to be invoked.

The `status_vars` member, if not 0, points to an array of `st_mysql_show_var` structures, each of which describes one status variable, followed by a structure with all members set to 0. The `st_mysql_show_var` structure has this definition:

```
struct st_mysql_show_var {
    const char *name;
    char *value;
    enum enum_mysql_show_type type;
};
```

When the plugin is installed, the plugin name and the `name` value are joined with an underscore to form the name displayed by `SHOW STATUS`.

The following table shows the allowable status variable `type` values and what the corresponding variable should be:

Type	Meaning
<code>SHOW_BOOL</code>	Pointer to a boolean variable
<code>SHOW_INT</code>	Pointer to an integer variable
<code>SHOW_LONG</code>	Pointer to a long integer variable
<code>SHOW_LONGLONG</code>	Pointer to a longlong integer variable
<code>SHOW_CHAR</code>	A string
<code>SHOW_CHAR_PTR</code>	Pointer to a string
<code>SHOW_ARRAY</code>	Pointer to another <code>st_mysql_show_var</code> array
<code>SHOW_FUNC</code>	Pointer to a function

For the `SHOW_FUNC` type, the function is called and fills in its `out` parameter, which then provides information about the variable to be displayed. The function has this calling sequence:

```
#define SHOW_VAR_FUNC_BUFF_SIZE 1024
```

```
typedef int (*mysql_show_var_func) (void *thd,
                                   struct st_mysql_show_var *out,
                                   char *buf);
```

Plugins should consider the `thd` parameter to be read-only.

Type-Specific Plugin Structures and Functions

In the `st_mysql_plugin` structure that defines a plugin's general declaration, the `info` member points to a type-specific plugin descriptor. For a full-text parser plugin, the descriptor corresponds to the `st_mysql_ftparser` structure in the `plugin.h` file:

```
struct st_mysql_ftparser
{
    int interface_version;
    int (*parse)(MYSQL_FTPARSER_PARAM *param);
    int (*init)(MYSQL_FTPARSER_PARAM *param);
    int (*deinit)(MYSQL_FTPARSER_PARAM *param);
};
```

As shown by the structure definition, the descriptor has a version number (`MYSQL_FTPARSER_INTERFACE_VERSION` for full-text parser plugins) and contains pointers to three functions. The `init` and `deinit` members should point to a function or be set to 0 if the function is not needed. The `parse` member must point to the function that performs the parsing.

A full-text parser plugin is used in two different contexts, indexing and searching. In both contexts, the server calls the initialization and deinitialization functions at the beginning and end of processing each SQL statement that causes the plugin to be invoked. However, during statement processing, the server calls the main parsing function in context-specific fashion:

- For indexing, the server calls the parser for each column value to be indexed.
- For searching, the server calls the parser to parse the search string. The parser might also be called for rows processed by the statement. In natural language mode, there is no need for the server to call the parser. For boolean mode phrase searches or natural language searches with query expansion, the parser is used to parse column values for information that is not in the index. Also, if a boolean mode search is done for a column that has no `FULLTEXT` index, the built-in parser will be called. (Plugins are associated with specific indexes. If there is no index, no plugin is used.)

Note that the plugin declaration in the plugin library descriptor has initialization and deinitialization functions, and so does the plugin descriptor to which it points. These pairs of functions have different purposes and are invoked for different reasons:

- For the plugin declaration in the plugin library descriptor, the initialization and deinitialization functions are invoked when the plugin is loaded and unloaded.
- For the plugin descriptor, the initialization and deinitialization functions are invoked per SQL statement for which the plugin is used.

Each interface function named in the plugin descriptor should return zero for success or non-zero for failure, and each of them receives an argument that points to a `MYSQL_FTPARSER_PARAM` structure containing the parsing context. The structure has this definition:

```
typedef struct st_mysql_ftparser_param
{
    int (*mysql_parse)(struct st_mysql_ftparser_param *,
                     char *doc, int doc_len);
    int (*mysql_add_word)(struct st_mysql_ftparser_param *,
                       char *word, int word_len,
                       MYSQL_FTPARSER_BOOLEAN_INFO *boolean_info);
    void *ftparser_state;
    void *mysql_ftparam;
    struct charset_info_st *cs;
    char *doc;
    int length;
    int flags;
    enum enum_ftparser_mode mode;
```

```
} MYSQL_FTPARSER_PARAM;
```

Note: The definition shown is current as of MySQL 5.1.12. It is incompatible with versions of MySQL 5.1 older than 5.1.12.

The structure members are used as follows:

- [mysql_parse](#)

A pointer to a callback function that invokes the server's built-in parser. Use this callback when the plugin acts as a front end to the built-in parser. That is, when the plugin parsing function is called, it should process the input to extract the text and pass the text to the [mysql_parse](#) callback.

The first parameter for this callback function should be the [param](#) value itself:

```
param->mysql_parse(param, ...);
```

A front end plugin can extract text and pass it all at once to the built-in parser, or it can extract and pass text to the built-in parser a piece at a time. However, in this case, the built-in parser treats the pieces of text as though there are implicit word breaks between them.

- [mysql_add_word](#)

A pointer to a callback function that adds a word to a full-text index or to the list of search terms. Use this callback when the parser plugin replaces the built-in parser. That is, when the plugin parsing function is called, it should parse the input into words and invoke the [mysql_add_word](#) callback for each word.

The first parameter for this callback function should be the [param](#) value itself:

```
param->mysql_add_word(param, ...);
```

- [ftparser_state](#)

This is a generic pointer. The plugin can set it to point to information to be used internally for its own purposes.

- [mysql_ftparam](#)

This is set by the server. It is passed as the first argument to the [mysql_parse](#) or [mysql_add_word](#) callback.

- [cs](#)

A pointer to information about the character set of the text, or 0 if no information is available.

- [doc](#)

A pointer to the text to be parsed.

- [length](#)

The length of the text to be parsed, in bytes.

- [flags](#)

Parser flags. This is zero if there are no special flags. Currently, the only non-zero flag is [MYSQL_FTFLAGS_NEED_COPY](#), which means that [mysql_add_word\(\)](#) must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.) This member was added in MySQL 5.1.12.

This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the [mysql_parse\(\)](#) function.

- [mode](#)

The parsing mode. This value will be one of the following constants:

- [MYSQL_FTPARSER_SIMPLE_MODE](#)

Parse in fast and simple mode, which is used for indexing and for natural language queries. The parser should pass to the server only those words that should be indexed. If the parser uses length limits or a stopword list to determine which words to ignore, it should not pass such words to the server.

- [MYSQL_FTPARSER_WITH_STOPWORDS](#)

Parse in stopword mode. This is used in boolean searches for phrase matching. The parser should pass all words to the server, even stopwords or words that are outside any normal length limits.

- [MYSQL_FTPARSER_FULL_BOOLEAN_INFO](#)

Parse in boolean mode. This is used for parsing boolean query strings. The parser should recognize not only words but also boolean-mode operators and pass them to the server as tokens via the [mysql_add_word](#) callback. To tell the server what kind of token is being passed, the plugin needs to fill in a [MYSQL_FTPARSER_BOOLEAN_INFO](#) structure and pass a pointer to it.

If the parser is called in boolean mode, the `param->mode` value will be [MYSQL_FTPARSER_FULL_BOOLEAN_INFO](#). The [MYSQL_FTPARSER_BOOLEAN_INFO](#) structure that the parser uses for passing token information to the server looks like this:

```
typedef struct st_mysql_ftparser_boolean_info
{
    enum enum_ft_token_type type;
    int yesno;
    int weight_adjust;
    bool wasign;
    bool trunc;
    /* These are parser state and must be removed. */
    byte prev;
    byte *quot;
} MYSQL_FTPARSER_BOOLEAN_INFO;
```

The parser should fill in the structure members as follows:

- [type](#)

The token type. This should be one of values shown in the following table:

Type	Meaning
FT_TOKEN_EOF	End of data
FT_TOKEN_WORD	A regular word
FT_TOKEN_LEFT_PAREN	The beginning of a group or subexpression
FT_TOKEN_RIGHT_PAREN	The end of a group or subexpression
FT_TOKEN_STOPWORD	A stopwords

- [yesno](#)

Whether the word must be present for a match to occur. 0 means that the word is optional but increases the match relevance if it is present. Values larger than 0 mean that the word must be present. Values smaller than 0 mean that the word must not be present.

- [weight_adjust](#)

A weighting factor that determines how much a match for the word counts. It can be used to increase or decrease the word's importance in relevance calculations. A value of zero indicates no weight adjustment. Values greater than or less than zero mean higher or lower weight, respectively. The examples at [ブール全文検索](#), that use the `<` and `>` operators illustrate how weighting works.

- [wasign](#)

The sign of the weighting factor. A negative value acts like the `~` boolean-search operator, which causes the word's contribution to the relevance to be negative.

- [trunc](#)

Whether matching should be done as if the boolean-mode `*` truncation operator had been given.

Plugins should not use the [prev](#) and [quot](#) members of the `MYSQL_FTPARSER_BOOLEAN_INFO` structure.

Creating a Plugin Library

This section provides a step-by-step procedure for creating a plugin library. It shows how to develop a library that contains a full-text parsing plugin named [simple_parser](#). This plugin performs parsing based on simpler rules than those used by the MySQL built-in full-text parser: Words are non-empty runs of whitespace characters.

Each plugin library has the following contents:

- A plugin library descriptor that indicates the version number of the general plugin API that the library uses and that contains a general declaration for each plugin in the library.
- Each plugin general declaration contains information that is common to all types of plugin: A value that indicates the plugin type; the plugin name, author, description, and license type; and pointers to the initialization and deinitialization functions that the server invokes when it loads and unloads the plugin.
- The plugin general declaration also contains a pointer to a type-specific plugin descriptor. The structure of these descriptors can vary from one plugin type to another, because each type of plugin can have its own API. A plugin descriptor contains a type-specific API version number and pointers to the functions that are needed to implement that plugin type. For example, a full-text parser plugin has initialization and deinitialization functions, and a main parsing function. The server invokes these functions when it uses the plugin to parse text.
- The plugin library contains the interface functions that are referenced by the library descriptor and by the plugin descriptors.

The easiest way to follow the instructions in this section is to use the source code in the [plugin/fulltext](#) directory of a MySQL source distribution. The instructions assume that you make a copy of that directory and use it to build the plugin library. To make a copy of the directory, use the following commands, which assume that the MySQL source tree is in a directory named [mysql-5.1](#) under your current directory:

```
shell> mkdir fulltext_plugin
shell> cp mysql-5.1/plugin/fulltext/* fulltext_plugin
```

If you are copying files from a BitKeeper source tree, `cp` will display an error message about the [SCCS](#) directory, which you can ignore.

After copying the source files, use the following procedure to create a plugin library:

1. Change location into the [fulltext_plugin](#) directory:

```
shell> cd fulltext_plugin
```

2. The plugin source file should include the header files that the plugin library needs. The [plugin.h](#) file is required, and the library might require other files as well. For example:

```
#include <stdlib.h>
#include <ctype.h>
#include <mysql/plugin.h>
```

3. Set up the plugin library file descriptor.

Every plugin library must include a library descriptor that must define two symbols:

- [_mysql_plugin_interface_version](#) specifies the version number of the general plugin framework. This is given by the `MYSQL_PLUGIN_INTERFACE_VERSION` symbol, which is defined in the [plugin.h](#) file.
- [_mysql_plugin_declarations](#) defines an array of plugin declarations, terminated by a declaration with all members set to 0. Each declaration is an instance of the `st_mysql_plugin` structure (also defined in [plugin.h](#)). There must be one of these for each plugin in the library.

If the server does not find these two symbols in a library, it does not accept it as a legal plugin library and rejects it with an error. This prevents use of a library for plugin purposes unless it was built specifically as a plugin library.

The standard (and most convenient) way to define the two required symbols is by using the `mysql_declare_plugin` and `mysql_declare_plugin_end` macros from the `plugin.h` file:

```
mysql_declare_plugin
... one or more plugin declarations here ...
mysql_declare_plugin_end;
```

For example, the library descriptor for a library that contains a single plugin named `simple_parser` looks like this:

```
mysql_declare_plugin
{
  MYSQL_FTPARSER_PLUGIN, /* type */
  &simple_parser_descriptor, /* descriptor */
  "simple_parser", /* name */
  "MySQL AB", /* author */
  "Simple Full-Text Parser", /* description */
  PLUGIN_LICENSE_GPL, /* plugin license */
  simple_parser_plugin_init, /* init function (when loaded) */
  simple_parser_plugin_deinit, /* deinit function (when unloaded) */
  0x0001, /* version */
  simple_status /* status variables */
}
mysql_declare_plugin_end;
```

For a full-text parser plugin, the type must be `MYSQL_FTPARSER_PLUGIN`. This is the value that identifies the plugin as being legal for use in a `WITH PARSER` clause when creating a `FULLTEXT` index. (No other plugin type is legal for this clause.)

The `mysql_declare_plugin` and `mysql_declare_plugin_end` macros are defined in `plugin.h` like this:

```
#ifndef MYSQL_DYNAMIC_PLUGIN
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
int VERSION= MYSQL_PLUGIN_INTERFACE_VERSION; \
int PSIZE= sizeof(struct st_mysql_plugin); \
struct st_mysql_plugin DECLS[]={
#else
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
int _mysql_plugin_interface_version= MYSQL_PLUGIN_INTERFACE_VERSION; \
int _mysql_sizeof_struct_st_plugin= sizeof(struct st_mysql_plugin); \
struct st_mysql_plugin _mysql_plugin_declarations[]={
#endif

#define mysql_declare_plugin(NAME) \
__MYSQL_DECLARE_PLUGIN(NAME, \
    builtin_ ## NAME ## _plugin_interface_version, \
    builtin_ ## NAME ## _sizeof_struct_st_plugin, \
    builtin_ ## NAME ## _plugin)

#define mysql_declare_plugin_end ,{0,0,0,0,0,0,0,0,0}}
```

One point to note about those definitions is that the `_mysql_plugin_interface_version` symbol is defined only if the `MYSQL_DYNAMIC_PLUGIN` symbol is defined. This means that you'll need to provide `-DMYSQL_DYNAMIC_PLUGIN` as part of the compilation command when you build the plugin.

When the macros are used as just shown, they expand to the following code, which defines both of the required symbols (`_mysql_plugin_interface_version` and `_mysql_plugin_declarations`):

```
int _mysql_plugin_interface_version= MYSQL_PLUGIN_INTERFACE_VERSION;
struct st_mysql_plugin _mysql_plugin_declarations[]={
{
  MYSQL_FTPARSER_PLUGIN, /* type */
  &simple_parser_descriptor, /* descriptor */
  "simple_parser", /* name */
  "MySQL AB", /* author */
  "Simple Full-Text Parser", /* description */
  PLUGIN_LICENSE_GPL, /* plugin license */
  simple_parser_plugin_init, /* init function (when loaded) */
  simple_parser_plugin_deinit, /* deinit function (when unloaded) */
  0x0001, /* version */
  simple_status /* status variables */
}
```

```

simple_parser_plugin_deinit, /* deinit function (when unloaded) */
0x0001, /* version */
simple_status /* status variables */
}
,{0,0,0,0,0,0,0,0}
};

```

The preceding example declares a single plugin in the library descriptor, but it is possible to declare multiple plugins. List the declarations one after the other between `mysql_declare_plugin` and `mysql_declare_plugin_end`, separated by commas.

MySQL plugins can be written in C or C++ (or another language that can use C calling conventions). One feature of C++ is that you can use non-constant variables to initialize global structures. However, if you write a C++ plugin, you should not use this feature. Members of structures such as the `st_mysql_plugin` structure should be initialized with constant variables. See the discussion at the end of this section that describes some legal and illegal initializers for plugins.

4. Set up the plugin descriptor.

Each plugin declaration in the library descriptor points to a type-specific descriptor for the corresponding plugin. In the `simple_parser` declaration, that descriptor is indicated by `&simple_parser_descriptor`. The descriptor specifies the version number for the full-text plugin interface (as given by `MYSQL_FTPARSER_INTERFACE_VERSION`), and the plugin's parsing, initialization, and deinitialization functions:

```

static struct st_mysql_ftparser simple_parser_descriptor=
{
    MYSQL_FTPARSER_INTERFACE_VERSION, /* interface version */
    simple_parser_parse, /* parsing function */
    simple_parser_init, /* parser init function */
    simple_parser_deinit /* parser deinit function */
};

```

5. Set up the plugin interface functions.

The general plugin declaration in the library descriptor names the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. For `simple_parser`, these functions do nothing but return zero to indicate that they succeeded:

```

static int simple_parser_plugin_init(void)
{
    return(0);
}

static int simple_parser_plugin_deinit(void)
{
    return(0);
}

```

Because those functions do not actually do anything, you could omit them and specify 0 for each of them in the plugin declaration.

The type-specific plugin descriptor for `simple_parser` names the initialization, deinitialization, and parsing functions that the server invokes when the plugin is used. For `simple_parser`, the initialization and deinitialization functions do nothing:

```

static int simple_parser_init(MYSQL_FTPARSER_PARAM *param)
{
    return(0);
}

static int simple_parser_deinit(MYSQL_FTPARSER_PARAM *param)
{
    return(0);
}

```

Here too, because those functions do nothing, you could omit them and specify 0 for each of them in the plugin descriptor.

The main parsing function, `simple_parser_parse()`, acts as a replacement for the built-in full-text parser, so it needs to split text into words and pass each word to the server. The parsing function's first argument is a pointer to a structure that contains the parsing context. This structure has a `doc` member that points to the text to be parsed, and a `length` member that indicates how long the text is. The simple parsing done by the plugin considers non-empty runs of whitespace characters to be words, so it identifies words like this:

```
static int simple_parser_parse(MYSQL_FTPARSER_PARAM *param)
{
    char *end, *start, *docend= param->doc + param->length;

    for (end= start= param->doc;; end++)
    {
        if (end == docend)
        {
            if (end > start)
                add_word(param, start, end - start);
            break;
        }
        else if (isspace(*end))
        {
            if (end > start)
                add_word(param, start, end - start);
            start= end + 1;
        }
    }
    return(0);
}
```

As the parser finds each word, it invokes a function `add_word()` to pass the word to the server. `add_word()` is a helper function only; it is not part of the plugin interface. The parser passes the parsing context pointer to `add_word()`, as well as a pointer to the word and a length value:

```
static void add_word(MYSQL_FTPARSER_PARAM *param, char *word, size_t len)
{
    MYSQL_FTPARSER_BOOLEAN_INFO bool_info=
        { FT_TOKEN_WORD, 0, 0, 0, 0, ' ', 0 };

    param->mysql_add_word(param, word, len, &bool_info);
}
```

For boolean-mode parsing, `add_word()` fills in the members of the `bool_info` structure as described in [Type-Specific Plugin Structures and Functions](#).

- Set up the status variables, if there are any. For the `simple_parser` plugin, the following status variable array sets up one status variable with a value that is static text, and another with a value that is stored in a long integer variable:

```
long number_of_calls= 0;

struct st_mysql_show_var simple_status[]=
{
    {"static", (char *)"just a static text", SHOW_CHAR},
    {"called", (char *)&number_of_calls, SHOW_LONG},
    {0,0,0}
};
```

When the plugin is installed, the plugin name and the `name` value are joined with an underscore to form the name displayed by `SHOW STATUS`. For the array just shown, the resulting status variable names are `simple_parser_static` and `simple_parser_called`. This convention means that you can easily display the variables for a plugin using its name:

```
mysql> SHOW STATUS LIKE 'simple_parser%';
+-----+-----+
| Variable_name | Value                |
+-----+-----+
| simple_parser_static | just a static text |
| simple_parser_called | 0                    |
+-----+-----+
```

- Compile the plugin library as a shared library and install it in the plugin directory.

Note: As mentioned earlier, be sure to specify `-DMYSQL_DYNAMIC_PLUGIN` as part of the compilation command when you build the plugin.

The procedure for compiling shared objects varies from system to system. If you build your library using the GNU autotools, `libtool` should be able to generate the correct compilation commands for your system. If the library is named `mypluglib`, you should end up with a shared object file that has a name something like `libmypluglib.so`. (The filename might have a different extension on your system.)

To use the autotools, you'll need to make a few changes to the configuration files at this point to enable the plugin to be compiled and installed. Assume that your MySQL distribution is installed at a base directory of `/usr/local/mysql` and that its header files are located in the `include` directory under the base directory.

Edit `Makefile.am`, which should look something like this:

```
#Makefile.am example for a plugin

pkglibdir=$(libdir)/mysql
INCLUDES=-I$(top_builddir)/include -I$(top_srcdir)/include
#noinst_LTLIBRARIES= mypluglib.la
pkglib_LTLIBRARIES= mypluglib.la
mypluglib_la_SOURCES= plugin_example.c
mypluglib_la_LDFLAGS= -module -rpath $(pkglibdir)
mypluglib_la_CFLAGS= -DMYSQL_DYNAMIC_PLUGIN
```

The `mypluglib_la_CFLAGS` line takes care of passing the `-DMYSQL_DYNAMIC_PLUGIN` flag to the compilation command.

Adjust the `INCLUDES` line to specify the pathname to the installed MySQL header files. Edit it to look like this:

```
INCLUDES= -I/usr/local/mysql/include
```

Make sure that the `noinst_LTLIBRARIES` line is commented out or remove it. Make sure that the `pkglib_LTLIBRARIES` line is not commented out; it enables the `make install` command.

Set up the files needed for the `configure` command, invoke it, and run `make`:

```
shell> autoreconf --force --install --symlink
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

The `--prefix` option to `configure` indicates the MySQL base directory under which the plugin should be installed. You can see what value to use for this option with `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'basedir';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| base          | /usr/local/mysql |
+-----+-----+
```

The location of the plugin directory where you should install the library is given by the `plugin_dir` system variable. For example:

```
mysql> SHOW VARIABLES LIKE 'plugin_dir';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| plugin_dir    | /usr/local/mysql/lib/mysql |
+-----+-----+
```

To install the plugin library, use `make`:

```
shell> make install
```

Verify that `make install` installed the plugin library in the proper directory. After installing it, make sure that the library permissions allow it to be executed by the server.

8. Register the plugin with the server.

The `INSTALL PLUGIN` statement causes the server to list the plugin in the `plugin` table and to load the plugin code from the library file. Use that statement to register `simple_parser` with the server, and then verify that the plugin is listed in the `plugin` table:

```
mysql> INSTALL PLUGIN simple_parser SONAME 'libmypluglib.so';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM mysql.plugin;
+-----+-----+
| name      | dl      |
+-----+-----+
| simple_parser | libmypluglib.so |
+-----+-----+
1 row in set (0.00 sec)
```

9. Try the plugin.

Create a table that contains a string column and associate the parser plugin with a `FULLTEXT` index on the column:

```
mysql> CREATE TABLE t (c VARCHAR(255),
-> FULLTEXT (c) WITH PARSER simple_parser);
Query OK, 0 rows affected (0.01 sec)
```

Insert some text into the table and try some searches. These should verify that the parser plugin treats all non-whitespace characters as word characters:

```
mysql> INSERT INTO t VALUES
-> ('latin1_general_cs is a case-sensitive collation'),
-> ('I\'d like a case of oranges'),
-> ('this is sensitive information'),
-> ('another row'),
-> ('yet another row');
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT c FROM t;
+-----+
| c      |
+-----+
| latin1_general_cs is a case-sensitive collation |
| I'd like a case of oranges |
| this is sensitive information |
| another row |
| yet another row |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('case') FROM t;
+-----+
| MATCH(c) AGAINST('case') |
+-----+
| 0 |
| 1.2968142032623 |
| 0 |
| 0 |
| 0 |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('sensitive') FROM t;
+-----+
| MATCH(c) AGAINST('sensitive') |
+-----+
| 0 |
| 0 |
| 1.3253291845322 |
| 0 |
| 0 |
+-----+
```



```

5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('case-sensitive') FROM t;
+-----+
| MATCH(c) AGAINST('case-sensitive') |
+-----+
| 1.3109166622162 |
| 0 |
| 0 |
| 0 |
| 0 |
+-----+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('l\l'd') FROM t;
+-----+
| MATCH(c) AGAINST('l\l'd') |
+-----+
| 0 |
| 1.2968142032623 |
| 0 |
| 0 |
| 0 |
+-----+
5 rows in set (0.01 sec)

```

Note how neither 「case」 nor 「insensitive」 match 「case-insensitive」 the way that they would for the built-in parser.

MySQL plugins can be written in C or C++ (or another language that can use C calling conventions). One feature of C++ is that you can use non-constant variables to initialize global structures. However, if you write a C++ plugin, you should not use this feature. Members of structures such as the [st_mysql_plugin](#) structure should be initialized with constant variables. The [simple_parser](#) descriptor shown earlier is allowable in a C++ plugin because it satisfies that requirement:

```

mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    "simple_parser", /* name */
    "MySQL AB", /* author */
    "Simple Full-Text Parser", /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;

```

Here is another valid way to write the descriptor. It uses constant variables to indicate the plugin name, author, and description:

```

const char *simple_parser_name = "simple_parser";
const char *simple_parser_author = "MySQL AB";
const char *simple_parser_description = "Simple Full-Text Parser";

mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    simple_parser_name, /* name */
    simple_parser_author, /* author */
    simple_parser_description, /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;

```

However, the following descriptor is invalid. It uses structure members to indicate the plugin name, author, and description, but structures are not considered constant initializers in C++:

```
typedef struct
{
    const char *name;
    const char *author;
    const char *description;
} plugin_info;

plugin_info parser_info = {
    "simple_parser",
    "MySQL AB",
    "Simple Full-Text Parser"
};

mysql_declare_plugin
{
    MYSQL_FTPARSER_PLUGIN, /* type */
    &simple_parser_descriptor, /* descriptor */
    parser_info.name, /* name */
    parser_info.author, /* author */
    parser_info.description, /* description */
    PLUGIN_LICENSE_GPL, /* plugin license */
    simple_parser_plugin_init, /* init function (when loaded) */
    simple_parser_plugin_deinit, /* deinit function (when unloaded) */
    0x0001, /* version */
    simple_status /* status variables */
}
mysql_declare_plugin_end;
```

Adding New Functions to MySQL

There are two ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the [CREATE FUNCTION](#) and [DROP FUNCTION](#) statements. See [CREATE FUNCTION Syntax](#).
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

Each method has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as [ABS\(\)](#) or [SOUNDEX\(\)](#).

Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is described in [17章ストアドプロシージャとファンクション](#).

See [関数名の構文解析と名前解決](#), for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the [sql/udf_example.c](#) file that is provided in MySQL source distributions.

Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number and types of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns [NULL](#) or that an error occurred.

CREATE FUNCTION Syntax

```
CREATE [AGGREGATE] FUNCTION function\_name RETURNS {STRING|INTEGER|REAL|DECIMAL}
SONAME shared\_library\_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as [ABS\(\)](#) or [CONCAT\(\)](#).

[function_name](#) is the name that should be used in SQL statements to invoke the function. The [RETURNS](#) clause indicates the type of the function's return value. [DECIMAL](#) is a legal value after [RETURNS](#), but currently [DECIMAL](#) functions return string values and should be written like [STRING](#) functions.

[shared_library_name](#) is the basename of the shared object file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the [plugin_dir](#) system variable. (Note: This a change in MySQL 5.1. For earlier versions of MySQL, the shared object can be located in any directory that is searched by your system's dynamic linker.)

To create a function, you must have the [INSERT](#) and privilege for the [mysql](#) database. This is necessary because [CREATE FUNCTION](#) adds a row to the [mysql.func](#) system table that records the function's name, type, and shared library name. If you do not have this table, you should run the [mysql_upgrade](#) command to create it. See [mysql_upgrade — MySQL アップグレードのテーブル チェック](#).

An active function is one that has been loaded with [CREATE FUNCTION](#) and not removed with [DROP FUNCTION](#). All active functions are reloaded each time the server starts, unless you start [mysqld](#) with the [--skip-grant-tables](#) option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see [Adding a New User-Defined Function](#). For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled [mysqld](#) dynamically (not statically).

An [AGGREGATE](#) function works exactly like a native MySQL aggregate (summary) function such as [SUM](#) or [COUNT\(\)](#). For [AGGREGATE](#) to work, your [mysql.func](#) table must contain a [type](#) column. If your [mysql.func](#) table does not have this column, you should run the [mysql_upgrade](#) program to create it (see [mysql_upgrade — MySQL アップグレードのテーブル チェック](#)).

DROP FUNCTION Syntax

```
DROP FUNCTION function\_name
```

This statement drops the user-defined function (UDF) named [function_name](#).

To drop a function, you must have the [DELETE](#) privilege for the [mysql](#) database. This is because [DROP FUNCTION](#) removes a row from the [mysql.func](#) system table that records the function's name, type, and shared library name.

Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The MySQL source distribution includes a file `sql/udf_example.c` that defines 5 new functions. Consult this file to see how UDF calling conventions work. UDF-related symbols and data structures are defined in the `include/mysql_com.h` header file. (You need not include this header file directly because it is included by `mysql.h`.)

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see [典型的な configure オプション](#), and [MySQL のコンパイルに関する問題](#).

To be able to use UDFs, you need to link `mysqld` dynamically. Don't configure MySQL using `--with-mysqld-ldflags=-all-static`. If you want to use a UDF that needs to access symbols from `mysqld` (for example, the `metaphone` function in `sql/udf_example.c` that uses `default_charset_info`), you must link the program with `-rdynamic` (see [man dlopen](#)). If you plan to use UDFs, the rule of thumb is to configure MySQL with `--with-mysqld-ldflags=-rdynamic` unless you have a very good reason not to.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name 「xxx」 is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the interface for `XXX()` are:

- `xxx()` (required)

The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here:

SQL Type	C/C++ Type
<code>STRING</code>	<code>char *</code>
<code>INTEGER</code>	<code>long long</code>
<code>REAL</code>	<code>double</code>

It is also possible to declare a `DECIMAL` function, but currently the value is returned as a string, so you should write the UDF as though it were a `STRING` function. `ROW` functions are not implemented.

- `xxx_init()` (optional)

The initialization function for `xxx()`. It can be used for the following purposes:

- To check the number of arguments to `XXX()`.
- To check that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the types you want when the main function is called.
- To allocate any memory required by the main function.
- To specify the maximum length of the result.
- To specify (for `REAL` functions) the maximum number of decimal places in the result.
- To specify whether the result can be `NULL`.

- `xxx_deinit()` (optional)

The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `XXX()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, MySQL aborts the SQL

statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function `xxx()` once for each row. After all rows have been processed, MySQL calls the deinitialization function `xxx_deinit()` so that it can perform any required cleanup.

For aggregate functions that work like `SUM()`, you must also provide the following functions:

- `xxx_clear()` (required in 5.1)

Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()` (required)

Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each new row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat 3-5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. [Adding a New User-Defined Function](#), describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
           char *is_null, char *error);
```

`DECIMAL` functions return string values and should be declared the same way as `STRING` functions. `ROW` functions are not implemented.

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
```

```
void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `my_bool maybe_null`

`xxx_init()` should set `maybe_null` to 1 if `xxx()` can return `NULL`. The default value is 1 if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. (For example, if the function is passed 1.34, 1.345, and 1.3, the default would be 3, because 1.345 has 3 decimal digits.)

- `unsigned int max_length`

The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- `char *ptr`

A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

- `my_bool const_item`

`xxx_init()` should set `const_item` to 1 if `xxx()` always returns the same value and to 0 otherwise.

UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. [Adding a New User-Defined Function](#), describes the order in which MySQL calls these functions.

- `xxx_reset()`

This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given `UDF_ARGS` argument as the first value in your internal summary value for the group. Declare `xxx_reset()` as follows:

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
```

`xxx_reset()` is not needed or used in MySQL 5.1, in which the UDF interface uses `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work with older versions of the server. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

- `xxx_clear()`

This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:


```
char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

`is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

`xxx_clear()` is required by MySQL 5.1.

- `xxx_add()`

This function is called for all rows that belong to the same group, except for the first row. You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The `xxx()` function for an aggregate UDF should be declared the same way as for a non-aggregate UDF. See [UDF Calling Sequences for Simple Functions](#).

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a non-aggregate UDF. See [UDF Return Values and Error Handling](#).

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for non-aggregate UDFs. See [UDF Argument Processing](#).

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error!` `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

- `enum Item_result *arg_type`

A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, `REAL_RESULT`, and `DECIMAL_RESULT`.

To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
}
```

```
return 1;
}
```

Arguments of type `DECIMAL_RESULT` are passed as strings, so you should handle them like `STRING_RESULT` values.

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

Exact-value decimal arguments such as `1.3` or `DECIMAL` column values are passed with a type of `DECIMAL_RESULT`. However, the values are passed as strings. If you want to receive a number, use the initialization function to specify that the argument should be coerced to a `REAL_RESULT` value:

```
args->arg_type[2] = REAL_RESULT;
```

- `char **args`

`args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a non-constant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)`. A non-constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non-constant arguments.

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

If argument `i` represents `NULL`, `args->args[i]` is a null pointer (`0`). If the argument is not `NULL`, functions can refer to it as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. You should not assume that strings are null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

```
double real_val;
real_val = *((double*) args->args[i]);
```

- For an argument of type `DECIMAL_RESULT`, the value is passed as a string and should be handled like a `STRING_RESULT` value.
 - `ROW_RESULT` arguments are not implemented.
- `unsigned long *lengths`

For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

UDF Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer

is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*result` and `*length` to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The `result` buffer that is passed to the `xxx()` function is 255 bytes long. If your result fits in this, you don't have to worry about memory allocation for results.

If your string function needs to return a string longer than 255 bytes, you must allocate the space for it with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See [UDF Calling Sequences for Simple Functions](#).

To indicate a return value of `NULL` in the main function, set `*is_null` to 1:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to 1:

```
*error = 1;
```

If `xxx()` sets `*error` to 1 for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

Compiling and Installing User-Defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.c` that is included in the MySQL source distribution.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.c` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it's more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.c
```

If you are using `gcc` with `configure` and `libtool` (which is how MySQL is configured), you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example.la
```

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.c` using `gcc` directly produces a file named `udf_example.so`. Compiling the shared object

using `make` produces a file named something like `udf_example.so.0.0.0` in the `.libs` directory (the exact name may vary from platform to platform). Copy the shared object to the server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` system variable. (Note: This a change in MySQL 5.1. For earlier versions of MySQL, the shared object can be located in any directory that is searched by your system's dynamic linker.)

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. You need to obtain the BitKeeper source repository for MySQL 5.1. See [開発ソース ツリーからのインストール](#).
2. You must obtain the CMake build utility from <http://www.cmake.org>. (Version 2.4.2 or later is required).
3. In the source repository, look in the `sql` directory. There are files named `udf_example.def` `udf_example.c` there. Copy both files from this directory to your working directory.
4. Create a CMake `makefile` with these contents:

```
PROJECT(udf_example)

# Path for MySQL include directory
INCLUDE_DIRECTORIES("c:/mysql/include")

ADD_DEFINITIONS("-DHAVE_DLOPEN")
ADD_LIBRARY(udf_example MODULE udf_example.c udf_example.def)
TARGET_LINK_LIBRARIES(udf_example wsock32)
```

5. Create the VC project and solution files:

```
cmake -G "<Generator>"
```

Invoking `cmake --help` shows you a list of valid Generators.

6. Create `udf_example.dll`:

```
devenv udf_example.sln /build Release
```

After the shared object file has been installed, notify `mysqld` about the new functions with these statements:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
-> RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
-> RETURNS REAL SONAME 'udf_example.so';
```

Functions can be deleted using **DROP FUNCTION**:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The **CREATE FUNCTION** and **DROP FUNCTION** statements update the `func` system table in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the **INSERT** and **DELETE** privileges for the `mysql` database to create and drop functions.

You should not use **CREATE FUNCTION** to add a function that has previously been created. If you need to reinstall a function, you should remove it with **DROP FUNCTION** and then reinstall it with **CREATE FUNCTION**. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with [CREATE FUNCTION](#) and not removed with [DROP FUNCTION](#). All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

User-Defined Function Security Precautions

MySQL takes the following measures to prevent misuse of user-defined functions.

You must have the [INSERT](#) privilege to be able to use [CREATE FUNCTION](#) and the [DELETE](#) privilege to be able to use [DROP FUNCTION](#). This is necessary because these statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. `mysqld` also supports an `--allow-suspicious-udfs` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` option. Otherwise, you should avoid enabling this capability.

UDF object files cannot be placed in arbitrary directories. They must be located in the server's plugin directory. This directory is given by the value of the `plugin_dir` system variable. (Note: This is a change in MySQL 5.1. For earlier versions of MySQL, the shared object can be located in any directory that is searched by your system's dynamic linker.)

Adding a New Native Function

The procedure for adding a new native function is described here. Note that you cannot add native functions to a binary distribution because the procedure involves modifying MySQL source code. You must compile MySQL yourself from a source distribution. Also note that if you migrate to another version of MySQL (for example, when a new version is released), you need to repeat the procedure with the new version.

To add a new native MySQL function, follow these steps:

1. Add one line to `lex.h` that defines the function name in the `sql_functions[]` array.
2. If the function prototype is simple (just takes zero, one, two or three arguments), you should in `lex.h` specify `SYM(FUNC_ARGN)` (where `N` is the number of arguments) as the second argument in the `sql_functions[]` array and add a function that creates a function object in `item_create.cc`. Take a look at "ABS" and `create_funcs_abs()` for an example of this.

If the function prototype is complicated (for example, if it takes a variable number of arguments), you should add two lines to `sql_yacc.yy`. One indicates the preprocessor symbol that `yacc` should define (this should be added at the beginning of the file). Then define the function parameters and add an "item" with these parameters to the `simple_expr` parsing rule. For an example, check all occurrences of `ATAN` in `sql_yacc.yy` to see how this is done.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.
4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double Item_func_newname::val()
longlong Item_func_newname::val_int()
String *Item_func_newname::Str(String *str)
```

If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. You can take a look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, don't use any global or static variables in the functions without protecting them with mutexes)

If you want to return `NULL`, from `::val()`, `::val_int()` or `::str()` you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are some additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)
- The `::str()` function should return the string that holds the result or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

Adding New Procedures to MySQL

In MySQL, you can define a procedure in C++ that can access and modify the data in a query before it is sent to the client. The modification can be done on a row-by-row or `GROUP BY` level.

We have created an example procedure to show you what can be done.

Additionally, we recommend that you take a look at `mylua`. With this you can use the LUA language to load a procedure at runtime into `mysqld`.

Procedure Analyse

`analyse([max_elements],[max_memory])`

This procedure is defined in the `sql/sql_analyse.cc` file. It examines the result from a query and returns an analysis of the results that suggests optimal data types for each column. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements],[max_memory])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- `max_elements` (default 256) is the maximum number of distinct values that `analyse` notices per column. This is used by `analyse` to check whether the optimal data type should be of type `ENUM`.
- `max_memory` (default 8192) is the maximum amount of memory that `analyse` should allocate per column while trying to find all distinct values.

Writing a Procedure

For the moment, the only documentation for this is the source.

You can find all information about procedures by examining the following files:

- [sql/sql_analyse.cc](#)
- [sql/procedure.h](#)
- [sql/procedure.cc](#)
- [sql/sql_select.cc](#)

付録A Frequently Asked Questions About MySQL 5.1

目次

MySQL 5.1 FAQ — General	1459
MySQL 5.1 FAQ — Storage Engines	1460
MySQL 5.1 FAQ — Server SQL Mode	1461
MySQL 5.1 FAQ — Stored Procedures	1462
MySQL 5.1 FAQ — Triggers	1464
MySQL 5.1 FAQ — Stored Routines, Triggers, and Replication	1466
MySQL 5.1 FAQ — Views	1467
MySQL 5.0 FAQ — INFORMATION_SCHEMA	1468
MySQL 5.1 FAQ — Migration	1469
MySQL 5.1 FAQ — Security	1469
MySQL 5.1 FAQ — MySQL Cluster	1470
MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets	1478
MySQL 5.1 FAQ — Connectors & APIs	1489

MySQL 5.1 FAQ — General

Questions

- [A.1.1: \[1459\]](#) When did MySQL 5.1 become production-ready (GA)?
- [A.1.2: \[1459\]](#) Can MySQL 5.1 do subqueries?
- [A.1.3: \[1459\]](#) Can MySQL 5.1 perform multi-table inserts, updates, and deletes?
- [A.1.4: \[1460\]](#) Does MySQL 5.1 have a Query Cache? Does it work on Server, Instance or Database?
- [A.1.5: \[1460\]](#) Does MySQL 5.1 have Sequences?
- [A.1.6: \[1460\]](#) Does MySQL 5.1 have a [NOW\(\)](#) function with fractions of seconds?
- [A.1.7: \[1460\]](#) Does MySQL 5.1 work with multi-core processors?
- [A.1.8: \[1460\]](#) Is there a hot backup tool for MyISAM like InnoDB Hot Backup?
- [A.1.9: \[1460\]](#) Have there been any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?
- [A.1.10: \[1460\]](#) Can MySQL 5.1 perform ACID transactions?

Questions and Answers

A.1.1: When did MySQL 5.1 become production-ready (GA)?

MySQL 5.0.15 was released for production use on 19 October 2005. We are now working on MySQL 5.1, which is currently in beta.

A.1.2: Can MySQL 5.1 do subqueries?

Yes. See [サブクエリ構文](#).

A.1.3: Can MySQL 5.1 perform multi-table inserts, updates, and deletes?

Yes. For the syntax required to perform multi-table updates, see [UPDATE 構文](#); for that required to perform multi-table deletes, see [DELETE 構文](#).

A multi-table insert can be accomplished using a trigger whose [FOR EACH ROW](#) clause contains multiple [INSERT](#) statements within a [BEGIN ... END](#) block. See [トリガの使用](#).

A.1.4: Does MySQL 5.1 have a Query Cache? Does it work on Server, Instance or Database?

Yes. The Query Cache operates on the server level, caching complete result sets matched with the original query string. If an exactly identical query is made (which often happens, particularly in web applications), no parsing or execution is necessary; the result is sent directly from the cache. Various tuning options are available. See [MySQL クエリ キャッシュ](#).

A.1.5: Does MySQL 5.1 have Sequences?

No. However, MySQL has an [AUTO_INCREMENT](#) system, which in MySQL 5.1 can also handle inserts in a multi-master replication setup. With the `--auto-increment-increment` and `--auto-increment-offset` startup options, you can set each server to generate auto-increment values that don't conflict with other servers. The `--auto-increment-increment` value should be greater than the number of servers, and each server should have a unique offset.

A.1.6: Does MySQL 5.1 have a [NOW\(\)](#) function with fractions of seconds?

No. This is on the MySQL roadmap as a 「rolling feature」. This means that it is not a flagship feature, but will be implemented, development time permitting. Specific customer demand may change this scheduling.

However, MySQL does parse time strings with a fractional component. See [TIME タイプ](#).

A.1.7: Does MySQL 5.1 work with multi-core processors?

Yes. MySQL is fully multi-threaded, and will make use of multiple CPUs, provided that the operating system supports them.

A.1.8: Is there a hot backup tool for MyISAM like InnoDB Hot Backup?

This is currently under development for a future MySQL release.

A.1.9: Have there been any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?

The foreign key support in [InnoDB](#) has seen improvements in each major version of MySQL. Foreign key support generic to all storage engines is scheduled for MySQL 5.2; this should resolve any inadequacies in the current storage engine specific implementation.

A.1.10: Can MySQL 5.1 perform ACID transactions?

Yes. All current MySQL versions support transactions. The [InnoDB](#) storage engine offers full ACID transactions with row-level locking, multi-versioning, non-locking repeatable reads, and all four SQL standard isolation levels.

The [NDB](#) storage engine supports the [READ COMMITTED](#) transaction isolation level only.

MySQL 5.1 FAQ — Storage Engines

Questions

- [A.2.1: \[1460\]](#) Where can I obtain complete documentation for MySQL storage engines and the pluggable storage engine architecture?
- [A.2.2: \[1461\]](#) Are there any new storage engines in MySQL 5.1?
- [A.2.3: \[1461\]](#) Have any storage engines been removed in MySQL 5.1?
- [A.2.4: \[1461\]](#) What are the unique benefits of the [ARCHIVE](#) storage engine?
- [A.2.5: \[1461\]](#) Do the new features in MySQL 5.1 apply to all storage engines?

Questions and Answers

A.2.1: Where can I obtain complete documentation for MySQL storage engines and the pluggable storage engine architecture?

See [13章ストレージエンジンとテーブルタイプ](#). That chapter contains information about all MySQL storage engines except for the [NDB](#) storage engine used for MySQL Cluster; [NDB](#) is covered in [14章MySQL Cluster](#).

A.2.2: Are there any new storage engines in MySQL 5.1?

There have been significant improvements in existing storage engines, in particular for the [NDB](#) storage engine that forms the basis for MySQL Cluster.

A.2.3: Have any storage engines been removed in MySQL 5.1?

Yes. MySQL 5.1 no longer supports the [BDB](#) storage engine. Any existing [BDB](#) tables should be converted to another storage engine before upgrading to MySQL 5.1.

A.2.4: What are the unique benefits of the [ARCHIVE](#) storage engine?

The [ARCHIVE](#) storage engine is ideally suited for storing large amounts of data without indexes; it has a very small footprint, and performs selects using table scans. See [ARCHIVE ストレージエンジン](#), for details.

A.2.5: Do the new features in MySQL 5.1 apply to all storage engines?

The general new features such as views, stored procedures, triggers, [INFORMATION_SCHEMA](#), precision math ([DECIMAL](#) column type), and the [BIT](#) column type, apply to all storage engines. There are also additions and changes for specific storage engines.

MySQL 5.1 FAQ — Server SQL Mode

Questions

- [A.3.1: \[1461\]](#) What are server SQL modes?
- [A.3.2: \[1461\]](#) How many server SQL modes are there?
- [A.3.3: \[1461\]](#) How do you determine the server SQL mode?
- [A.3.4: \[1461\]](#) Is the mode dependent on the database or connection?
- [A.3.5: \[1462\]](#) Can the rules for strict mode be extended?
- [A.3.6: \[1462\]](#) Does strict mode impact performance?
- [A.3.7: \[1462\]](#) What is the default server SQL mode when My SQL 5.1 is installed?

Questions and Answers

A.3.1: What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see [SQL モード](#).

A.3.2: How many server SQL modes are there?

Each mode can be independently switched on and off. See [SQL モード](#), for a complete list of available modes.

A.3.3: How do you determine the server SQL mode?

You can set the default SQL mode (for `mysqld` startup) with the `--sql-mode` option. Using the statement `SET [SESSION|GLOBAL] sql_mode='modes'`, you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement.

A.3.4: Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using `SET [SESSION|GLOBAL] sql_mode='modes'`.

A.3.5: Can the rules for strict mode be extended?

When we refer to strict mode, we mean a mode where at least one of the modes [TRADITIONAL](#), [STRICT_TRANS_TABLES](#), or [STRICT_ALL_TABLES](#) is enabled. Options can be combined, so you can add additional restrictions to a mode. See [SQL モード](#), for more information.

A.3.6: Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However — if you do require it — strict mode can provide such validation.

A.3.7: What is the default server SQL mode when MySQL 5.1 is installed?

By default, no special modes are enabled. See [SQL モード](#), for information about all available modes and MySQL's default behavior.

MySQL 5.1 FAQ — Stored Procedures

Questions

- [A.4.1: \[1463\]](#) Does MySQL 5.1 support stored procedures?
- [A.4.2: \[1463\]](#) Where can I find documentation for MySQL stored procedures and stored functions?
- [A.4.3: \[1463\]](#) Is there a discussion forum for MySQL stored procedures?
- [A.4.4: \[1463\]](#) Where can I find the ANSI SQL 2003 specification for stored procedures?
- [A.4.5: \[1463\]](#) How do you manage stored routines?
- [A.4.6: \[1463\]](#) Is there a way to view all stored procedures and stored functions in a given database?
- [A.4.7: \[1463\]](#) Where are stored procedures stored?
- [A.4.8: \[1463\]](#) Is it possible to group stored procedures or stored functions into packages?
- [A.4.9: \[1463\]](#) Can a stored procedure call another stored procedure?
- [A.4.10: \[1463\]](#) Can a stored procedure call a trigger?
- [A.4.11: \[1463\]](#) Can a stored procedure access tables?
- [A.4.12: \[1464\]](#) Do stored procedures have a statement for raising application errors?
- [A.4.13: \[1464\]](#) Do stored procedures provide exception handling?
- [A.4.14: \[1464\]](#) Can MySQL 5.1 stored routines return result sets?
- [A.4.15: \[1464\]](#) Is [WITH RECOMPILE](#) supported for stored procedures?
- [A.4.16: \[1464\]](#) Is there a MySQL equivalent to using [mod_plsql](#) as a gateway on Apache to talk directly to a stored procedure in the database?
- [A.4.17: \[1464\]](#) Can I pass an array as input to a stored procedure?
- [A.4.18: \[1464\]](#) Can I pass a cursor as an [IN](#) parameter to a stored procedure?
- [A.4.19: \[1464\]](#) Can I return a cursor as an [OUT](#) parameter from a stored procedure?
- [A.4.20: \[1464\]](#) Can I print out a variable's value within a stored routine for debugging purposes?
- [A.4.21: \[1464\]](#) Can I commit or roll back transactions inside a stored procedure?

Questions and Answers

A.4.1: Does MySQL 5.1 support stored procedures?

Yes. MySQL 5.1 supports two types of stored routines — stored procedures and stored functions.

A.4.2: Where can I find documentation for MySQL stored procedures and stored functions?

See [17章ストアードプロシージャとファンクション](#).

A.4.3: Is there a discussion forum for MySQL stored procedures?

Yes. See <http://forums.mysql.com/list.php?98>.

A.4.4: Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books — such as SQL-99 Complete, Really by Peter Gultzan and Trudy Pelzer — which give a comprehensive overview of the standard, including coverage of stored procedures.

A.4.5: How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with [CREATE \[FUNCTION|PROCEDURE\]](#), [ALTER \[FUNCTION|PROCEDURE\]](#), [DROP \[FUNCTION|PROCEDURE\]](#), and [SHOW CREATE \[FUNCTION|PROCEDURE\]](#). You can obtain information about existing stored procedures using the [ROUTINES](#) table in the [INFORMATION_SCHEMA](#) database (see [INFORMATION_SCHEMA ROUTINES テーブル](#)).

A.4.6: Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named [dbname](#), use this query on the [INFORMATION_SCHEMA.ROUTINES](#) table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see [INFORMATION_SCHEMA ROUTINES テーブル](#).

The body of a stored routine can be viewed using [SHOW CREATE FUNCTION](#) (for a stored function) or [SHOW CREATE PROCEDURE](#) (for a stored procedure). See [SHOW CREATE PROCEDURE と SHOW CREATE FUNCTION 構文](#), for more information.

A.4.7: Where are stored procedures stored?

In the [proc](#) table of the [mysql](#) system database. However, you should not access the tables in the system database directly. Instead, use [SHOW CREATE FUNCTION](#) to obtain information about stored functions, and [SHOW CREATE PROCEDURE](#) to obtain information about stored procedures. See [SHOW CREATE PROCEDURE と SHOW CREATE FUNCTION 構文](#), for more information about these statements.

You can also query the [ROUTINES](#) table in the [INFORMATION_SCHEMA](#) database — see [INFORMATION_SCHEMA ROUTINES テーブル](#), for information about this table.

A.4.8: Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 5.1.

A.4.9: Can a stored procedure call another stored procedure?

Yes.

A.4.10: Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an [UPDATE](#), that causes a trigger to fire.

A.4.11: Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

A.4.12: Do stored procedures have a statement for raising application errors?

Not in MySQL 5.1. We intend to implement the SQL standard [SIGNAL](#) and [RESIGNAL](#) statements in a future MySQL release.

A.4.13: Do stored procedures provide exception handling?

MySQL implements [HANDLER](#) definitions according to the SQL standard. See [DECLARE ハンドラ](#), for details.

A.4.14: Can MySQL 5.1 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary [SELECT](#) inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or above) client-server protocol for this to work. This means that — for instance — in PHP, you need to use the [mysqli](#) extension rather than the old [mysql](#) extension.

A.4.15: Is [WITH RECOMPILE](#) supported for stored procedures?

Not in MySQL 5.1.

A.4.16: Is there a MySQL equivalent to using [mod_plsql](#) as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 5.1.

A.4.17: Can I pass an array as input to a stored procedure?

Not in MySQL 5.1.

A.4.18: Can I pass a cursor as an [IN](#) parameter to a stored procedure?

In MySQL 5.1, cursors are available inside stored procedures only.

A.4.19: Can I return a cursor as an [OUT](#) parameter from a stored procedure?

In MySQL 5.1, cursors are available inside stored procedures only. However, if you do not open a cursor on a [SELECT](#), the result will be sent directly to the client. You can also [SELECT INTO](#) variables. See [SELECT 構文](#).

A.4.20: Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a stored procedure, but not in a stored function. If you perform an ordinary [SELECT](#) inside a stored procedure, the result set is returned directly to the client. You will need to use the MySQL 4.1 (or above) client-server protocol for this to work. This means that — for instance — in PHP, you need to use the [mysqli](#) extension rather than the old [mysql](#) extension.

A.4.21: Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

MySQL 5.1 FAQ — Triggers

Questions

- [A.5.1: \[1465\]](#) Where can I find the documentation for MySQL 5.1 triggers?
- [A.5.2: \[1465\]](#) Is there a discussion forum for MySQL Triggers?
- [A.5.3: \[1465\]](#) Does MySQL 5.1 have statement-level or row-level triggers?
- [A.5.4: \[1465\]](#) Are there any default triggers?
- [A.5.5: \[1465\]](#) How are triggers managed in MySQL?
- [A.5.6: \[1465\]](#) Is there a way to view all triggers in a given database?

- [A.5.7: \[1465\]](#) Where are triggers stored?
- [A.5.8: \[1465\]](#) Can a trigger call a stored procedure?
- [A.5.9: \[1465\]](#) Can triggers access tables?
- [A.5.10: \[1466\]](#) Can triggers call an external application through a UDF?
- [A.5.11: \[1466\]](#) Is possible for a trigger to update tables on a remote server?

Questions and Answers

A.5.1: Where can I find the documentation for MySQL 5.1 triggers?

See [18章トリガ](#).

A.5.2: Is there a discussion forum for MySQL Triggers?

Yes. It is available at <http://forums.mysql.com/list.php?99>.

A.5.3: Does MySQL 5.1 have statement-level or row-level triggers?

In MySQL 5.1, all triggers are **FOR EACH ROW** — that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 5.1 does not support triggers using **FOR EACH STATEMENT**.

A.5.4: Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some **TIMESTAMP** columns, as well as for columns which are defined using **AUTO_INCREMENT**.

A.5.5: How are triggers managed in MySQL?

In MySQL 5.1, triggers can be created using the **CREATE TRIGGER** statement, and dropped using **DROP TRIGGER**. See [CREATE TRIGGER 構文](#), and [DROP TRIGGER 構文](#), for more about these statements.

Information about triggers can be obtained by querying the **INFORMATION_SCHEMA.TRIGGERS** table. See [INFORMATION_SCHEMA TRIGGERS テーブル](#).

A.5.6: Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database **dbname** using a query on the **INFORMATION_SCHEMA.TRIGGERS** table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
FROM INFORMATION_SCHEMA.TRIGGERS
WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see [INFORMATION_SCHEMA TRIGGERS テーブル](#).

You can also use the **SHOW TRIGGERS** statement, which is specific to MySQL. See [SHOW TRIGGERS 構文](#).

A.5.7: Where are triggers stored?

Triggers are currently stored in **.TRG** files, with one such file one per table. In other words, a trigger belongs to a table.

In the future, we plan to change this so that trigger information will be included in the **.FRM** file that defines the structure of the table. We also plan to make triggers database-level objects — rather than table-level objects as they are now — to bring them into compliance with the SQL standard.

A.5.8: Can a trigger call a stored procedure?

Yes.

A.5.9: Can triggers access tables?

A trigger can access both old and new data in its own table. Through a stored procedure, or a multi-table update or delete statement, a trigger can also affect other tables.

A.5.10: Can triggers call an external application through a UDF?

No, not at present.

A.5.11: Is possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the [FEDERATED](#) storage engine. (See [FEDERATED ストレージエンジン](#)).

MySQL 5.1 FAQ — Stored Routines, Triggers, and Replication

Questions

- [A.6.1: \[1466\]](#) Do MySQL 5.1 stored procedures and functions work with replication?
- [A.6.2: \[1466\]](#) Are stored procedures and functions created on a master server replicated to a slave?
- [A.6.3: \[1466\]](#) How are actions that take place inside stored procedures and functions replicated?
- [A.6.4: \[1466\]](#) Are there special security requirements for using stored procedures and functions together with replication?
- [A.6.5: \[1466\]](#) What limitations exist for replicating stored procedure and function actions?
- [A.6.6: \[1467\]](#) Do the preceding limitations affect MySQL's ability to do point-in-time recovery?
- [A.6.7: \[1467\]](#) What will MySQL do to correct the aforementioned limitations?
- [A.6.8: \[1467\]](#) Do triggers work with replication?
- [A.6.9: \[1467\]](#) How are actions carried out through triggers on a master replicated to a slave?

Questions and Answers

A.6.1: Do MySQL 5.1 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in [ストアドルーチンとトリガのバイナリログ](#).

A.6.2: Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. [ALTER](#) and [DROP](#) statements for stored procedures and functions are also replicated.

A.6.3: How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

A.6.4: Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the [SUPER](#) privilege.
2. Alternatively, a DBA can set the [log_bin_trust_function_creators](#) system variable to 1, which enables anyone with the standard [CREATE ROUTINE](#) privilege to create stored functions.

A.6.5: What limitations exist for replicating stored procedure and function actions?

Non-deterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Note that declaring stored functions to be [DETERMINISTIC](#) or setting the [log_bin_trust_function_creators](#) system variable to 0 will not allow random-valued operations to be invoked.

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, non-transactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the [IGNORE](#) keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

A.6.6: Do the preceding limitations affect MySQL's ability to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

A.6.7: What will MySQL do to correct the aforementioned limitations?

As of MySQL 5.1.5, you can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Beginning with MySQL 5.1.8, mixed replication is also available (by starting the server with [--binlog-format=mixed](#)). This hybrid, 「smart」 form of replication 「knows」 whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see [レプリケーション フォーマット](#).

A.6.8: Do triggers work with replication?

Triggers and replication in MySQL 5.1 work in the same way as in most other database engines: Actions carried out through triggers on a master are not replicated to a slave server. Instead, triggers that exist on tables that reside on a MySQL master server need to be created on the corresponding tables on any MySQL slave servers so that the triggers activate on the slaves as well as the master.

A.6.9: How are actions carried out through triggers on a master replicated to a slave?

First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table [EMP](#) that has an [AFTER](#) insert trigger, which exists on a master MySQL server. The same [EMP](#) table and [AFTER](#) insert trigger exist on the slave server as well. The replication flow would be:

1. An [INSERT](#) statement is made to [EMP](#).
2. The [AFTER](#) trigger on [EMP](#) activates.
3. The [INSERT](#) statement is written to the binary log.
4. The replication slave picks up the [INSERT](#) statement to [EMP](#) and executes it.
5. The [AFTER](#) trigger on [EMP](#) that exists on the slave activates.

For answers to some general questions about MySQL stored procedures and stored functions, see [MySQL 5.1 FAQ — Stored Procedures](#). Some common questions concerning MySQL triggers are addressed in [MySQL 5.1 FAQ — Triggers](#).

MySQL 5.1 FAQ — Views

Questions

- [A.7.1: \[1468\]](#) Where can I find documentation covering MySQL Views?
- [A.7.2: \[1468\]](#) Is there a discussion forum for MySQL Views?
- [A.7.3: \[1468\]](#) What happens to a view if an underlying table is dropped or renamed?
- [A.7.4: \[1468\]](#) Does MySQL 5.1 have table snapshots?
- [A.7.5: \[1468\]](#) Does MySQL 5.1 have materialized views?
- [A.7.6: \[1468\]](#) Can you insert into views that are based on joins?

Questions and Answers

A.7.1: Where can I find documentation covering MySQL Views?

See [20章ビュー](#).

A.7.2: Is there a discussion forum for MySQL Views?

Yes. See <http://forums.mysql.com/list.php?100>

A.7.3: What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the [CHECK TABLE](#) statement. (See [CHECK TABLE 構文](#).)

A.7.4: Does MySQL 5.1 have table snapshots?

No.

A.7.5: Does MySQL 5.1 have materialized views?

No.

A.7.6: Can you insert into views that are based on joins?

It is possible, provided that your [INSERT](#) statement has a column list that makes it clear there's only one table involved.

You cannot insert into multiple tables with a single insert on a view.

MySQL 5.0 FAQ — INFORMATION_SCHEMA

Questions

- [A.8.1: \[1468\]](#) Where can I find documentation for the MySQL [INFORMATION_SCHEMA](#) database?
- [A.8.2: \[1468\]](#) Is there a discussion forum for [INFORMATION_SCHEMA](#)?
- [A.8.3: \[1469\]](#) Where can I find the ANSI SQL 2003 specification for [INFORMATION_SCHEMA](#)?
- [A.8.4: \[1469\]](#) What is the difference between the Oracle Data Dictionary and MySQL's [INFORMATION_SCHEMA](#)?
- [A.8.5: \[1469\]](#) Can I add to or otherwise modify the tables found in the [INFORMATION_SCHEMA](#) database?

Questions and Answers

A.8.1: Where can I find documentation for the MySQL [INFORMATION_SCHEMA](#) database?

See [21章INFORMATION_SCHEMA データベース](#)

A.8.2: Is there a discussion forum for [INFORMATION_SCHEMA](#)?

See <http://forums.mysql.com/list.php?101>.

A.8.3: Where can I find the ANSI SQL 2003 specification for [INFORMATION_SCHEMA](#)?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available — such as SQL-99 Complete, Really by Peter Gultzan and Trudy Pelzer — which give a comprehensive overview of the standard, including [INFORMATION_SCHEMA](#).

A.8.4: What is the difference between the Oracle Data Dictionary and MySQL's [INFORMATION_SCHEMA](#)?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. MySQL's implementation is more similar to those found in DB2 and SQL Server, which also support [INFORMATION_SCHEMA](#) as defined in the SQL standard.

A.8.5: Can I add to or otherwise modify the tables found in the [INFORMATION_SCHEMA](#) database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, MySQL AB cannot support bugs or other issues which result from modifying [INFORMATION_SCHEMA](#) tables or data.

MySQL 5.1 FAQ — Migration

Questions

- [A.9.1: \[1469\]](#) Where can I find information on how to migrate from MySQL 5.0 to MySQL 5.1?
- [A.9.2: \[1469\]](#) How has storage engine (table type) support changed in MySQL 5.1 from previous versions?

Questions and Answers

A.9.1: Where can I find information on how to migrate from MySQL 5.0 to MySQL 5.1?

For detailed upgrade information, see [MySQL のアップグレード](#). We recommend that you do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but it will save you time and trouble — if you encounter problems during the upgrade, their origin will be easier to identify, either by you or — if you have a MySQL Network subscription — by MySQL support.

A.9.2: How has storage engine (table type) support changed in MySQL 5.1 from previous versions?

Storage engine support has changed as follows:

- Support for [ISAM](#) tables was removed in MySQL 5.0 and you should now use the [MyISAM](#) storage engine in place of [ISAM](#). To convert a table [tblname](#) from [ISAM](#) to [MyISAM](#), simply issue a statement such as this one:

```
ALTER TABLE tblname ENGINE=MYISAM;
```

- Internal [RAID](#) for [MyISAM](#) tables was also removed in MySQL 5.0. This was formerly used to allow large tables in filesystems that did not support file sizes greater than 2GB. All modern filesystems allow for larger tables; in addition, there are now other solutions such as [MERGE](#) tables and views.
- The [VARCHAR](#) column type now retains trailing spaces in all storage engines.
- [MEMORY](#) tables (formerly known as [HEAP](#) tables) can also contain [VARCHAR](#) columns.

MySQL 5.1 FAQ — Security

Questions

- [A.10.1: \[1470\]](#) Where can I find documentation that addresses security issues for MySQL?
- [A.10.2: \[1470\]](#) Does MySQL 5.1 have native support for SSL?
- [A.10.3: \[1470\]](#) Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?
- [A.10.4: \[1470\]](#) Does MySQL 5.1 have built-in authentication against LDAP directories?

- [A.10.5: \[1470\]](#) Does MySQL 5.1 include support for Roles Based Access Control (RBAC)?

Questions and Answers

A.10.1: Where can I find documentation that addresses security issues for MySQL?

The best place to start is [セキュリティ問題](#).

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- [セキュリティ ガイドライン](#).
- [MySQL のクラッカー対策](#).
- [How to Reset the Root Password](#).
- [ユーザによる MySQL の実行](#).
- [User-Defined Function Security Precautions](#).
- [セキュリティ 関連の mysqld オプション](#).
- [LOAD DATA LOCAL のセキュリティ関連事項](#).
- [インストール後の設定とテスト](#).
- [SELinux の注釈](#).
- [SSL の基本概念](#).

A.10.2: Does MySQL 5.1 have native support for SSL?

Most 5.1 binaries have support for SSL connections between the client and server. We can't currently build with the new YaSSL library everywhere, as it's still quite new and does not compile on all platforms yet. See [接続安全](#).

You can also tunnel a connection via SSH, if (for instance) if the client application doesn't support SSL connections. For an example, see [SSH で Windows からリモート接続](#).

A.10.3: Is SSL support be built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 5.1 binaries have SSL enabled for client-server connections that are secured, authenticated, or both. However, the YaSSL library currently does not compile on all platforms. See [接続安全](#), for a complete listing of supported and unsupported platforms.

A.10.4: Does MySQL 5.1 have built-in authentication against LDAP directories?

No. Support for external authentication methods is on the MySQL roadmap as a 「rolling feature」, which means that we plan to implement it in the future, but we have not yet determined when this will be done.

A.10.5: Does MySQL 5.1 include support for Roles Based Access Control (RBAC)?

No. Support for roles is on the MySQL roadmap as a 「rolling feature」, which means that we plan to implement it in the future, but we have not yet determined when this will be done.

MySQL 5.1 FAQ — MySQL Cluster

In the following section, we provide answers to questions that are most frequently asked about MySQL Cluster and the [NDB](#) storage engine.

Questions

- [A.11.1: \[1472\]](#) What does 「NDB」 mean?

- [A.11.2: \[1472\]](#) What's the difference in using Cluster vs using replication?
- [A.11.3: \[1472\]](#) Do I need to do any special networking to run Cluster? How do computers in a cluster communicate?
- [A.11.4: \[1472\]](#) How many computers do I need to run a cluster, and why?
- [A.11.5: \[1472\]](#) What do the different computers do in a MySQL Cluster?
- [A.11.6: \[1472\]](#) With which operating systems can I use Cluster?
- [A.11.7: \[1473\]](#) What are the hardware requirements for running MySQL Cluster?
- [A.11.8: \[1473\]](#) How much RAM do I need? Is it possible to use disk memory at all?
- [A.11.9: \[1474\]](#) What filesystems can I use with MySQL Cluster? What about network filesystems or network shares?

- [A.11.10: \[1474\]](#) I'm trying to populate a Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: The table 'my_cluster_table' is full
```

Why is this happening?

- [A.11.11: \[1474\]](#) MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?
- [A.11.12: \[1474\]](#) Do I have to learn a new programming or query language to use MySQL Cluster?
- [A.11.13: \[1475\]](#) How do I find out what an error or warning message means when using MySQL Cluster?
- [A.11.14: \[1475\]](#) Is MySQL Cluster transaction-safe? What isolation levels are supported?
- [A.11.15: \[1475\]](#) What storage engines are supported by MySQL Cluster?
- [A.11.16: \[1475\]](#) Which versions of the MySQL software support Cluster? Do I have to compile from source?
- [A.11.17: \[1475\]](#) In the event of a catastrophic failure — say, for instance, the whole city loses power and my UPS fails — would I lose all my data?
- [A.11.18: \[1475\]](#) Is it possible to use [FULLTEXT](#) indexes with Cluster?
- [A.11.19: \[1475\]](#) Can I run multiple nodes on a single computer?
- [A.11.20: \[1476\]](#) Can I add nodes to a cluster without restarting it?
- [A.11.21: \[1476\]](#) Are there any limitations that I should be aware of when using MySQL Cluster?
- [A.11.22: \[1476\]](#) How do I import an existing MySQL database into a cluster?
- [A.11.23: \[1476\]](#) How do cluster nodes communicate with one another?
- [A.11.24: \[1477\]](#) What is an arbitrator?
- [A.11.25: \[1477\]](#) What data types are supported by MySQL Cluster?
- [A.11.26: \[1477\]](#) How do I start and stop MySQL Cluster?
- [A.11.27: \[1478\]](#) What happens to cluster data when the cluster is shut down?
- [A.11.28: \[1478\]](#) Is it helpful to have more than one management node for a cluster?
- [A.11.29: \[1478\]](#) Can I mix different kinds of hardware and operating systems in one MySQL Cluster?
- [A.11.30: \[1478\]](#) Can I run two data nodes on a single host? Two SQL nodes?
- [A.11.31: \[1478\]](#) Can I use hostnames with MySQL Cluster?

- [A.11.32: \[1478\]](#) How do I handle MySQL users in a Cluster having multiple MySQL servers?

Questions and Answers

A.11.1: What does 「NDB」 mean?

This stands for 「Network Database」. [NDB](#) (also known as [NDB Cluster](#) or [NDBCLUSTER](#)) is the storage engine that enables clustering in MySQL.

A.11.2: What's the difference in using Cluster vs using replication?

In a replication setup, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as [InnoDB](#) is being used, a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is asynchronous, MySQL Cluster is synchronous.

We have implemented (asynchronous) replication for Cluster in MySQL 5.1. This includes the capability to replicate both between two clusters, and from a MySQL cluster to a non-Cluster MySQL server. See [MySQL Cluster レプリケーション](#).

A.11.3: Do I need to do any special networking to run Cluster? How do computers in a cluster communicate?

MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting via TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.

The faster SCI protocol is also supported, but requires special hardware. See [MySQL Cluster での高速インターコネクトを使用する](#), for more information about SCI.

A.11.4: How many computers do I need to run a cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum recommended number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

A.11.5: What do the different computers do in a MySQL Cluster?

A MySQL Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as nodes, and a computer housing a cluster node is sometimes referred to as a cluster host. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- Management node (MGM node): Provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application [ndb_mgmd](#); the management client used to control MySQL Cluster via the MGM node is [ndb_mgm](#).
- Data node: Stores and replicates data. Data node functionality is handled by an instance of the NDB data node process [ndbd](#).
- SQL node: This is simply an instance of MySQL Server ([mysqld](#)) that is built with support for the [NDB Cluster](#) storage engine and started with the [--ndb-cluster](#) option to enable the engine.

A.11.6: With which operating systems can I use Cluster?

MySQL Cluster is supported on most Unix-like operating systems, including Linux, Mac OS X, and Solaris. Beginning with MySQL Cluster NDB 6.4, it is also possible to run MySQL Cluster on Windows platforms on an experimental basis; we hope to release a GA version for Windows in MySQL Cluster NDB 7.1

For more detailed information concerning the level of support which is offered for MySQL Cluster on various operating system versions, OS distributions, and hardware platforms, please refer to <http://www.mysql.com/support/supportedplatforms.html>.

A.11.7: What are the hardware requirements for running MySQL Cluster?

Cluster should run on any platform for which NDB-enabled binaries are available. Naturally, faster CPUs and more memory will improve performance, and 64-bit CPUs will likely be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see [How much RAM do I Need?](#) for more information). Nodes can communicate via a standard TCP/IP network and hardware. For SCI support, special networking hardware is required.

A.11.8: How much RAM do I need? Is it possible to use disk memory at all?

Previous to MySQL 5.1, Cluster was in-memory only. This meant that all table data (including indexes) was stored in RAM. If your data took up 1GB of space and you wanted to replicate it once in the cluster, you needed 2GB of memory to do so (1 GB per replica). This was in addition to the memory required by the operating system and any applications running on the cluster computers. This is still true of in-memory tables.

If a data node's memory usage exceeds what is available in RAM, then the system will attempt to use swap space up to the limit set for [DataMemory](#). However, this will at best result in severely degraded performance, and may cause the node to be dropped due to slow response time (missed heartbeats). We do not recommend on relying on disk swapping in a production environment for this reason. In any case, once the [DataMemory](#) limit is reached, any operations requiring additional memory (such as inserts) will fail.

[NDB Cluster](#) in MySQL 5.1 includes support for Disk Data, which helps to alleviate these issues. See [MySQL Cluster ディスク データ ストレージ](#), for more information.

You can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

```
(SizeofDatabase × NumberOfReplicas × 1.1) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see [データタイプが必要とする記憶容量](#), for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an [NDBCluster](#) table requires 21–25 bytes per record. These indexes use [IndexMemory](#).
- Each ordered index requires 10 bytes storage per record, using [DataMemory](#).
- Creating a primary key or unique index also creates an ordered index, unless this index is created with [USING HASH](#). In other words:
 - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
 - However, if the primary key or unique index is created with [USING HASH](#), then it requires only 21 to 25 bytes per record.

Note that creating MySQL Cluster tables with [USING HASH](#) for all primary keys and unique indexes will generally cause table updates to run more quickly — in some cases by as much as 20 to 30 percent faster than updates on tables where [USING HASH](#) was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the [ndb_size.pl](#) utility which is available in recent MySQL 5.1 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on

how much space that database would require if it used the [NDBCluster](#) storage engine. For more information, see [ndb_size.pl — NDBCluster サイズ仕様エスエイメーター](#).

It is especially important to keep in mind that every MySQL Cluster table must have a primary key. The [NDB](#) storage engine creates a primary key automatically if none is defined, and this primary key is created without [USING HASH](#).

There is no easy way to determine exactly how much memory is being used for storage of Cluster indexes at any given time; however, warnings are written to the Cluster log when 80% of available [DataMemory](#) or [IndexMemory](#) is in use, and again when use reaches 85%, 90%, and so on.

A.11.9: What filesystems can I use with MySQL Cluster? What about network filesystems or network shares?

Generally, any filesystem that is native to the host operating system should work well with MySQL Cluster. If you find that a given filesystem works particularly well (or not so especially well) with MySQL Cluster, we invite you to discuss your findings in the [MySQL Cluster Forums](#).

We do not test MySQL Cluster with [FAT](#) or [VFAT](#) filesystems on Linux. Because of this, and due to the fact that these are not very useful for any purpose other than sharing disk partitions between Linux and Windows operating systems on multi-boot computers, we do not recommend their use with MySQL Cluster.

MySQL Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network filesystems is not supported for MySQL Cluster. This also applies to shared storage devices such as SANs.

A.11.10: I'm trying to populate a Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: The table 'my_cluster_table' is full
```

Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, including the primary key required by the [NDB](#) storage engine and automatically created in the event that the table definition does not include the definition of a primary key.

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. In other words, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote only 1GB to clustering.

A.11.11: MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is very unlikely that a cluster would perform reliably under such conditions, as MySQL Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet — preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in a MySQL Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.)

A.11.12: Do I have to learn a new programming or query language to use MySQL Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL queries and commands are required for the following operations:

- Creating, altering, and dropping tables (including Disk Data tables and related objects)
- Inserting, updating, and deleting table data

- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up a MySQL Cluster — see [設定ファイル](#), for information about these.

A few simple commands are used in the MySQL Cluster management client for tasks such as starting and stopping cluster nodes. See [マネジメント クライアントのコマンド](#).

A.11.13: How do I find out what an error or warning message means when using MySQL Cluster?

There are two ways in which this can be done:

-
- From a system shell prompt, use `pererror --ndb error_code`.

A.11.14: Is MySQL Cluster transaction-safe? What isolation levels are supported?

Yes: For tables created with the [NDB](#) storage engine, transactions are supported. In MySQL 5.1, Cluster supports only the [READ COMMITTED](#) transaction isolation level.

A.11.15: What storage engines are supported by MySQL Cluster?

Clustering in MySQL is supported only by the [NDB](#) storage engine. That is, in order for a table to be shared between nodes in a cluster, it must be created using `ENGINE=NDB` (or `ENGINE=NDBCLUSTER`, which is equivalent).

It is possible to create tables using other storage engines (such as [MyISAM](#) or [InnoDB](#)) on a MySQL server being used for clustering, but these non-[NDB](#) tables will not participate in the cluster; they are local to the individual MySQL server instance on which they are created.

A.11.16: Which versions of the MySQL software support Cluster? Do I have to compile from source?

Cluster is supported in all server binaries in the 5.1 release series for operating systems on which MySQL Cluster is available. See [mysqld — MySQL サーバ](#). You can determine whether your server has NDB support using either the `SHOW VARIABLES LIKE 'have_%'` or `SHOW ENGINES` statement.

You can also obtain NDB support by compiling MySQL from source, but it is not necessary to do so simply to use MySQL Cluster. To download the latest binary, RPM, or source distribution in the MySQL 5.1 series, visit <http://dev.mysql.com/downloads/mysql/5.1.html>.

A.11.17: In the event of a catastrophic failure — say, for instance, the whole city loses power and my UPS fails — would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

A.11.18: Is it possible to use [FULLTEXT](#) indexes with Cluster?

[FULLTEXT](#) indexing is not supported by the [NDB](#) storage engine in MySQL 5.1, or by any storage engine other than [MyISAM](#). We are working to add this capability in a future release.

A.11.19: Can I run multiple nodes on a single computer?

It is possible but not advisable. One of the chief reasons to run a cluster is to provide redundancy. To enjoy the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. Given that MySQL Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 200 MHz Pentium CPU and sufficient RAM for the operating system plus a small amount of overhead for the `ndb_mgmd` and `ndb_mgm` processes.

It is acceptable to run multiple cluster data nodes on a single host for learning about MySQL Cluster, or for testing purposes; however, this is not supported for production use.

A.11.20: Can I add nodes to a cluster without restarting it?

Not at present. A simple restart is all that is required for adding new MGM or SQL nodes to a Cluster. When adding data nodes the process is more complex, and requires the following steps:

1. Make a complete backup of all Cluster data.
2. Completely shut down the cluster and all cluster node processes.
3. Restart the cluster, using the `--initial` startup option.
4. Restore all cluster data from the backup.

In a future MySQL Cluster release series, we hope to implement a 「hot」 reconfiguration capability for MySQL Cluster to minimize (if not eliminate) the requirement for restarting the cluster when adding new nodes. However, this is not planned for MySQL 5.1.

A.11.21: Are there any limitations that I should be aware of when using MySQL Cluster?

Limitations on [NDB](#) tables in MySQL 5.1 include:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- The only types of user-defined partitioning supported for [NDB](#) tables are [KEY](#) and [LINEAR KEY](#). (Beginning with MySQL 5.1.12, attempting to create an [NDB](#) table using any other partitioning type fails with an error.)
- [FULLTEXT](#) indexes and index prefixes are not supported. Only complete columns may be indexed.
- Spatial data types are not supported. See [16章 Spatial Extensions](#).
- Only complete rollbacks for transactions are supported. Partial rollbacks and rollbacks to save points are not supported.
- The maximum number of attributes allowed per table is 128, and attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- The maximum size for a table row is 8 kilobytes, not counting [BLOB](#) values. There is no set limit for the number of rows per table. Table size limits depend on a number of factors, in particular on the amount of RAM available to each data node.
- The [NDB](#) engine does not support foreign key constraints. As with [MyISAM](#) tables, these are ignored.

For a complete listing of limitations in MySQL Cluster, see [MySQL Cluster の既知の制限](#).

A.11.22: How do I import an existing MySQL database into a cluster?

You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ and in [MySQL Cluster の既知の制限](#), the only other special requirement is that any tables to be included in the cluster must use the [NDB](#) storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables using other storage engines to [NDB Cluster](#) using one or more `ALTER TABLE` statement, but this requires an additional workaround. See [MySQL Cluster の既知の制限](#), for details.

A.11.23: How do cluster nodes communicate with one another?

Cluster nodes can communicate via any of three different protocols: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental in MySQL 5.1. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See [MySQL Cluster での高速インターコネクトを使用する](#), for more about using SCI as a transport mechanism in MySQL Cluster.

A.11.24: What is an arbitrator?

If one or more nodes in a cluster fail, it is possible that not all cluster nodes will be able to 「see」 one another. In fact, it is possible that two sets of nodes might become isolated from one another in a network partitioning, also known as a 「split brain」 scenario. This type of situation is undesirable because each set of nodes tries to behave as though it is the entire cluster.

When cluster nodes go down, there are two possibilities. If more than 50% of the remaining nodes can communicate with each other, we have what is sometimes called a 「majority rules」 situation, and this set of nodes is considered to be the cluster. The arbitrator comes into play when there is an even number of nodes: in such cases, the set of nodes to which the arbitrator belongs is considered to be the cluster, and nodes not belonging to this set are shut down.

The preceding information is somewhat simplified. A more complete explanation taking into account node groups follows:

When all nodes in at least one node group are alive, network partitioning is not an issue, because no one portion of the cluster can form a functional cluster. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the 「split-brain」 scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the [ArbitrationRank](#) configuration parameter for MySQL Server and management server nodes. (See [マネジメント サーバーの定義](#), for details.) It should also be noted that the role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

A.11.25: What data types are supported by MySQL Cluster?

MySQL Cluster supports all of the usual MySQL data types, with the exception of those associated with MySQL's spatial extensions. (See [16章 Spatial Extensions](#).) In addition, there are some differences with regard to indexes when used with [NDB](#) tables. Note: MySQL Cluster Disk Data tables (that is, tables created with [TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER](#)) have only fixed-width rows. This means that (for example) each Disk Data table record containing a [VARCHAR\(255\)](#) column requires space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein.

See [MySQL Cluster の既知の制限](#), for more information about these issues.

A.11.26: How do I start and stop MySQL Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node with the [ndb_mgmd](#) command.
2. Start each data node with the [ndbd](#) command.
3. Start each MySQL server (SQL node) using [mysqld_safe --user=mysql &](#).

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine — a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the MGM management client [ndb_mgm](#) on the machine housing the MGM node and issuing the [SHOW](#) or [ALL STATUS](#) command.

To shut down a running cluster, issue the command [SHUTDOWN](#) in the MGM client. Alternatively, you may enter the following command in a system shell on the machine hosting the MGM node:

```
shell> ndb_mgm -e "SHUTDOWN"
```

(Note that the quotation marks are optional here; the [SHUTDOWN](#) command itself is not case-sensitive.)

Either of these commands causes the [ndb_mgm](#), [ndb_mgm](#), and any [ndbd](#) processes to terminate gracefully. MySQL servers running as Cluster SQL nodes can be stopped using [mysqldadmin shutdown](#).

For more information, see [マネジメント クライアントのコマンド](#), and [安全なシャットダウンと再起動](#).

A.11.27: What happens to cluster data when the cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

A.11.28: Is it helpful to have more than one management node for a cluster?

It can be helpful as a fail-safe. Only one MGM node controls the cluster at any given time, but it is possible to configure one MGM as primary, and one or more additional management nodes to take over in the event that the primary MGM node fails.

See [設定ファイル](#), for information on how to configure MySQL Cluster management nodes.

A.11.29: Can I mix different kinds of hardware and operating systems in one MySQL Cluster?

Yes, so long as all machines and operating systems have the same 「endianness」 (all big-endian or all little-endian). It is also possible to use different MySQL Cluster releases on different nodes. However, we recommend this be done only as part of a rolling upgrade procedure (see [クラスタのローリング再起動の実行](#)).

A.11.30: Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port. However, running more than one cluster node of a given type per machine is not supported for production use.

A.11.31: Can I use hostnames with MySQL Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires 「five nines」 availability, we recommend using fixed IP addresses. Making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional points of failure, and the fewer of these, the better.

A.11.32: How do I handle MySQL users in a Cluster having multiple MySQL servers?

MySQL user accounts and privileges are not automatically propagated between different MySQL servers accessing the same MySQL Cluster. Therefore, you must make sure that these are copied between the SQL nodes yourself.

MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

Questions

- [A.12.1: \[1479\]](#) I have inserted CJK characters into my table. Why does `SELECT` display them as 「？」 characters?
- [A.12.2: \[1480\]](#) What GB (Chinese) character sets does MySQL support?
- [A.12.3: \[1481\]](#) What problems should I be aware of when working with the Big5 Chinese character set?
- [A.12.4: \[1481\]](#) Why do Japanese character set conversions fail?
- [A.12.5: \[1482\]](#) What should I do if I want to convert SJIS 81CA to cp932?
- [A.12.6: \[1482\]](#) How does MySQL represent the Yen (¥) sign?
- [A.12.7: \[1482\]](#) Do MySQL plan to make a separate character set where 5C is the Yen sign, as at least one other major DBMS does?
- [A.12.8: \[1482\]](#) Of what issues should I be aware when working with Korean character sets in MySQL?
- [A.12.9: \[1482\]](#) Why do I get `Data truncated` error messages?

- [A.12.10: \[1483\]](#) Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?
- [A.12.11: \[1484\]](#) I've upgraded to MySQL 5.1. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?
- [A.12.12: \[1485\]](#) Why do some [LIKE](#) and [FULLTEXT](#) searches with CJK characters fail?
- [A.12.13: \[1485\]](#) What CJK character sets are available in MySQL?
- [A.12.14: \[1486\]](#) How do I know whether character [X](#) is available in all character sets?
- [A.12.15: \[1487\]](#) Why don't CJK strings sort correctly in Unicode? (I)
- [A.12.16: \[1488\]](#) Why don't CJK strings sort correctly in Unicode? (II)
- [A.12.17: \[1488\]](#) Why are my supplementary characters rejected by MySQL?
- [A.12.18: \[1488\]](#) Shouldn't it be 「CJKV」?
- [A.12.19: \[1489\]](#) Does MySQL allow CJK characters to be used in database and table names?
- [A.12.20: \[1489\]](#) Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?
- [A.12.21: \[1489\]](#) Where can I get help with CJK and related issues in MySQL?

Questions and Answers

A.12.1: I have inserted CJK characters into my table. Why does [SELECT](#) display them as 「？」 characters?

This problem is usually due to a setting in MySQL that doesn't match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- Be certain of what MySQL version you are using.

Use the statement [SELECT VERSION\(\)](#); to determine this.

- Make sure that the database is actually using the desired character set.

People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of [SHOW CREATE TABLE tablename](#) or — better — yet by using this statement:

```
SELECT character_set_name, collation_name
FROM information_schema.columns
WHERE table_schema = your_database_name
AND table_name = your_table_name
AND column_name = your_column_name;
```

- Determine the hexadecimal value of the character or characters that are not being displayed correctly.

You can obtain this information for a column [column_name](#) in the table [table_name](#) using the following query:

```
SELECT HEX(column_name)
FROM table_name;
```

[3F](#) is the encoding for the [?](#) character; this means that [?](#) is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- Make sure that a round trip possible — that is, when you select [literal](#) (or [_introducer hexadecimal-value](#)), you obtain [literal](#) as a result.

For example, the Japanese Katakana character Pe (ペ) exists in all CJK character sets, and has the code point value (hexadecimal coding) [0x30da](#). To test a round trip for this character, use this query:

```
SELECT 'ペ' AS `ペ`; /* or SELECT _ucs2 0x30da; */
```

If the result is not also ㄹ, then the round trip has failed.

For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ㄹ');`. Then we can determine whether the client encoding is correct.

- Make sure that the problem is not with the browser or other application, rather than with MySQL.

Use the `mysql` client program (on Windows: `mysql.exe`) to accomplish this task. If `mysql` displays correctly but your application doesn't, then your problem is probably due to system settings.

To find out what your settings are, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysql/charsets/ |
+-----+-----+
8 rows in set (0.03 sec)
```

These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set and a default for MySQL).

Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it's often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is what's suitable.

If you cannot control the server settings, and you have no idea what your underlying computer is, then try changing to a common character set for the country that you're in (`euckr` = Korea; `gb2312` or `gbk` = People's Republic of China; `big5` = Taiwan; `sjis`, `ujis`, `cp932`, or `eucjpms` = Japan; `ucs2` or `utf8` = anywhere). Usually it is necessary to change only the client and connection and results settings. There is a simple statement which changes all three at once: `SET NAMES`. For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing `my.cnf` or `my.ini`. For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see [Why does my GUI front end or browser not display CJK characters correctly...?](#) for more information.

A.12.2: What GB (Chinese) character sets does MySQL support?

MySQL supports the two common variants of the GB (Guojia Biaozhun, or National Standard) character sets which are official in the People's Republic of China: `gb2312` and `gbk`. Sometimes people try to insert `gbk` characters into `gb2312`, and it works most of the time because `gbk` is a superset of `gb2312` — but eventually they try to insert a rarer Chinese character and it doesn't work. (See Bug #16072 for an example).

Here, we try to clarify exactly what characters are legitimate in `gb2312` or `gbk`, with reference to the official documents. Please check these references before reporting `gb2312` or `gbk` bugs.

- For a complete listing of the `gb2312` characters, ordered according to the `gb2312_chinese_ci` collation: http://d.udm.net/bar/~bar/charts/gb2312_chinese_ci.html.

- MySQL's `gbk` is in reality 「Microsoft code page 936」. This differs from the official `gbk` for characters `A1A4` (middle dot), `A1AA` (em dash), `A6E0-A6F5`, and `A8BB-A8C0`. For a listing of the differences, see <http://recode.progiciels-bpi.ca/showfile.html?name=dist/libiconv/gbk.h>.
- For a listing of `gbk`/Unicode mappings, see <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT>.
- For MySQL's listing of `gbk` characters, see http://d.udm.net/bar/~bar/charts/gbk_chinese_ci.html.

A.12.3: What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). MySQL's `big5` is in reality Microsoft code page 950, which is very similar to the original `big5` character set. We changed to this character set starting with MySQL version 4.1.16 / 5.0.16 (as a result of Bug #12476). For example, the following statements work in current versions of MySQL, but not in old versions:

```
mysql> CREATE TABLE big5 (BIG5 CHAR(1) CHARACTER SET BIG5);
Query OK, 0 rows affected (0.13 sec)

mysql> INSERT INTO big5 VALUES (0xf9dc);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM big5;
+-----+
| big5 |
+-----+
| 嫫 |
+-----+
1 row in set (0.02 sec)
```

A feature request for adding `HKSCS` extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

A.12.4: Why do Japanese character set conversions fail?

MySQL supports the `sjis`, `ujis`, `cp932`, and `eucjpms` character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with `sjis` or `ujis`) and a Windows client (typically with `cp932`).

In the following conversion table, the `ucs2` column represents the source, and the `sjis`, `cp932`, `ujis`, and `eucjpms` columns represent the destinations — that is, the last 4 columns provide the hexadecimal result when we use `CONVERT(ucs2)` or we assign a `ucs2` column containing the value to an `sjis`, `cp932`, `ujis`, or `eucjpms` column.

Character Name	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH ""	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2

Character Name	ucs2	sjis	cp932	ujis	eucjpms
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider this portion of the table:

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This means that MySQL converts the **NOT SIGN** (Unicode **U+00AC**) to **sjis** code point **0x81CA** and to **cp932** code point **3F**. (**3F** is the question mark (「?」) — this is what is always used when the conversion cannot be performed.

A.12.5: What should I do if I want to convert SJIS **81CA** to **cp932**?

Our answer is: 「?」. There are serious complaints about this: many people would prefer a 「loose」 conversion, so that **81CA (NOT SIGN)** in **sjis** becomes **81CA (FULLWIDTH NOT SIGN)** in **cp932**. We are considering a change to this behavior.

A.12.6: How does MySQL represent the Yen (¥) sign?

A problem arises because some versions of Japanese character sets (both **sjis** and **euc**) treat **5C** as a reverse solidus (\ — also known as a backslash), and others treat it as a yen sign (¥).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, **5C** is always the reverse solidus (\).

A.12.7: Do MySQL plan to make a separate character set where **5C** is the Yen sign, as at least one other major DBMS does?

This is one possible solution to the Yen sign issue; however, this will not happen in MySQL 5.1 or 5.2.

A.12.8: Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the **euckr** (Extended Unix Code Korea) character set, only one problem has been noted.

We use the 「ASCII」 variant of EUC-KR, in which the code point **0x5c** is REVERSE SOLIDUS, that is \, instead of the 「KS-Roman」 variant of EUC-KR, in which the code point **0x5c** is **WON SIGN(₩)**. This means that you cannot convert Unicode **U+20A9** to **euckr**:

```
mysql> SELECT
->   CONVERT('₩' USING euckr) AS euckr,
->   HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-----+
| euckr | hexeuckr |
+-----+
| ?    | 3F      |
+-----+
1 row in set (0.00 sec)
```

MySQL's graphic Korean chart is here: http://d.udm.net/bar/~bar/charts/euckr_korean_ci.html.

A.12.9: Why do I get **Data truncated** error messages?

For illustration, we'll create a table with one Unicode ([ucs2](#)) column and one Chinese ([gb2312](#)) column.

```
mysql> CREATE TABLE ch
-> (ucs2 CHAR(3) CHARACTER SET ucs2,
-> gb2312 CHAR(3) CHARACTER SET gb2312);
Query OK, 0 rows affected (0.05 sec)
```

We'll try to place the rare character 洵 in both columns.

```
mysql> INSERT INTO ch VALUES ('A洵B','A洵B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

Ah, there's a warning. Let's see what it is.

```
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'gb2312' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

So it's a warning about the [gb2312](#) column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-----+-----+-----+
| ucs2 | HEX(ucs2) | gb2312 | HEX(gb2312) |
+-----+-----+-----+
| A洵B | 00416C4C0042 | A?B | 413F42 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

There are several things that need explanation here.

1. The fact that it's a 「warning」 rather than an 「error」 is characteristic of MySQL. We like to try to do what we can, to get the best fit, rather than give up.
2. The 洵 character isn't in the [gb2312](#) character set. We described that problem earlier.
3. Admittedly the message is misleading. We didn't 「truncate」 in this case, we replaced with a question mark. We've had a complaint about this message (See Bug #9337). But until we come up with something better, just accept that error/warning code 2165 can mean a variety of things.
4. With [SQL_MODE=TRADITIONAL](#), there would be an error message, but instead of error 2165 you would see: [ERROR 1406 \(22001\): Data too long for column 'gb2312' at row 1](#).

A.12.10: Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the [mysql](#) client (Windows: [mysql.exe](#)), and try the same query there. If [mysql](#) responds correctly, then the trouble may be that your application interface requires initialization. Use [mysql](#) to tell you what character set or sets it uses with the statement [SHOW VARIABLES LIKE 'char%'](#). If you are using Access, then you are most likely connecting with MyODBC. In this case, you should check [Connector/ODBC の構成](#). If, for instance, you use [big5](#), you would enter [SET NAMES 'big5'](#). (Note that no ; is required in this case). If you are using ASP, you might need to add [SET NAMES](#) in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
& "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than [latin1](#) with Connector/NET, then you must specify the character set in the connection string. See [Connector/NET を使用した MySQL への接続](#), for more information.

If you are using PHP, try this:

```
<?php
$link = mysql_connect($host, $usr, $pwd);

mysql_select_db($db);

if( mysql_error() ) { print "Database ERROR: " . mysql_error(); }
mysql_query("SET NAMES 'utf8'", $link);
?>
```

In this case, we used `SET NAMES` to change `character_set_client` and `character_set_connection` and `character_set_results`.

We encourage the use of the newer `mysqli` extension, rather than `mysql`. Using `mysqli`, the previous example could be rewritten as shown here:

```
<?php
$link = new mysqli($host, $usr, $pwd, $db);

if( mysqli_connect_errno() )
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$link->query("SET NAMES 'utf8'");
?>
```

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, you should include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` of the HTML page.

If you are using Connector/J, see [文字セットと Unicode の使用](#).

A.12.11: I've upgraded to MySQL 5.1. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single 「global」 character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a 「handshake」, as described in [接続のキャラクタセットおよび照合順序](#):

When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some of our Asian customers have said that prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set that it wants to use — however, the server ignores this request from the client.

By way of example, suppose that your favorite server character set is `latin1` (unlikely in a CJK area, but this is the default value). Suppose further that the client uses `utf8` because this is what the client's operating system supports. Now, start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The current settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
+-----+-----+
+-----+-----+
```

```

+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysql/charsets/ |
+-----+-----+
8 rows in set (0.01 sec)

```

Now stop the client, and then stop the server using `mysqldadmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the current settings:

```

mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | latin1 |
| character_set_connection | latin1 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | latin1 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysql/charsets/ |
+-----+-----+
8 rows in set (0.01 sec)

```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` is used.

A.12.12: Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

There is a very simple problem with `LIKE` searches on `BINARY` and `BLOB` columns: we need to know the end of a character. With multi-byte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but `ㄅ` requires three bytes, as shown here:

```

+-----+-----+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'ㄅ') |
+-----+-----+
| 1 | 3 |
+-----+-----+
1 row in set (0.00 sec)

```

If we don't know where the first character ends, then we don't know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a regular CJK character set in the first place, or to convert to a CJK character set before comparing.

This is one reason why MySQL cannot allow encodings of nonexistent characters. If it is not strict about rejecting bad input, then it has no way of knowing where characters end.

For `FULLTEXT` searches, we need to know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary — the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

A.12.13: What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the `eucljpm` character set was not supported prior to MySQL 5.0.3. However, since the name of the applicable language appears in the

[DESCRIPTION](#) column for every entry in the [INFORMATION_SCHEMA.CHARACTER_SETS](#) table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
-> FROM INFORMATION_SCHEMA.CHARACTER_SETS
-> WHERE DESCRIPTION LIKE '%Chinese%'
-> OR DESCRIPTION LIKE '%Japanese%'
-> OR DESCRIPTION LIKE '%Korean%'
-> ORDER BY CHARACTER_SET_NAME;
```

CHARACTER_SET_NAME	DESCRIPTION
big5	Big5 Traditional Chinese
cp932	SJIS for Windows Japanese
eucjpms	UJIS for Windows Japanese
euckr	EUC-KR Korean
gb2312	GB2312 Simplified Chinese
gbk	GBK Simplified Chinese
sjis	Shift-JIS Japanese
ujis	EUC-JP Japanese

8 rows in set (0.01 sec)

(See [INFORMATION_SCHEMA.CHARACTER_SETS テーブル](#), for more information.)

A.12.14: How do I know whether character **X** is available in all character sets?

The majority of simplified Chinese and basic non-halfwidth Japanese Kana characters appear in all CJK character sets. This stored procedure accepts a [UCS-2](#) Unicode character, converts it to all other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
(ucs2 CHAR(1) character set ucs2,
 utf8 CHAR(1) character set utf8,
 big5 CHAR(1) character set big5,
 cp932 CHAR(1) character set cp932,
 eucjpms CHAR(1) character set eucjpms,
 euckr CHAR(1) character set euckr,
 gb2312 CHAR(1) character set gb2312,
 gbk CHAR(1) character set gbk,
 sjis CHAR(1) character set sjis,
 ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
 big5=ucs2,
 cp932=ucs2,
 eucjpms=ucs2,
 euckr=ucs2,
 gb2312=ucs2,
 gbk=ucs2,
 sjis=ucs2,
 ujis=ucs2;

/* If there's a conversion problem, UPDATE will produce a warning. */

SELECT hex(ucs2) AS ucs2,
 hex(utf8) AS utf8,
 hex(big5) AS big5,
 hex(cp932) AS cp932,
 hex(eucjpms) AS eucjpms,
 hex(euckr) AS euckr,
 hex(gb2312) AS gb2312,
 hex(gbk) AS gbk,
 hex(sjis) AS sjis,
 hex(ujis) AS ujis
FROM tj;
```



```
DROP TABLE tj;

END//
```

The input can be any single [ucs2](#) character, or it can be the code point value (hexadecimal representation) of that character. For example, from Unicode's list of [ucs2](#) encodings and names (<http://www.unicode.org/Public/UNIDATA/UnicodeData.txt>), we know that the Katakana character Pe appears in all CJK character sets, and that its code point value is [0x30da](#). If we use this value as the argument to [p_convert\(\)](#), the result is as shown here:

```
mysql> CALL p_convert(0x30da)//
+-----+-----+-----+-----+-----+
| ucs2 | utf8 | big5 | cp932 | eucjpm | euckr | gb2312 | gbk | sjis | ujis |
+-----+-----+-----+-----+-----+
| 30DA | E3839A | C772 | 8379 | A5DA | ABDA | A5DA | A5DA | 8379 | A5DA |
+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

Since none of the column values is [3F](#) — that is, the question mark character (?) — we know that every conversion worked.

A.12.15: Why don't CJK strings sort correctly in Unicode? (I)

Sometimes people observe that the result of a [utf8_unicode_ci](#) or [ucs2_unicode_ci](#) search, or of an [ORDER BY](#) sort is not what they think a native would expect. Although we never rule out the possibility that there is a bug, we have found in the past that many people do not read correctly the standard table of weights for the Unicode Collation Algorithm. MySQL uses the table found at <http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt>. This is not the first table you will find by navigating from the [unicode.org](#) home page, because MySQL uses the older 4.0.0 「allkeys」 table, rather than the more recent 4.1.0 table. This is because we are very wary about changing ordering which affects indexes, lest we bring about situations such as that reported in Bug #16526, illustrated as follows:

```
mysql< CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_unicode_ci);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO tj VALUES ('が'),('か');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM tj WHERE s1 = 'か';
+-----+
| s1 |
+-----+
| が |
| か |
+-----+
2 rows in set (0.00 sec)
```

The character in the first result row is not the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the [ucs2](#) version of the characters:

```
mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
+-----+-----+
| s1 | HEX(CONVERT(s1 USING ucs2)) |
+-----+-----+
| が | 304C |
| か | 304B |
+-----+-----+
2 rows in set (0.03 sec)
```

Now we search for [304B](#) and [304C](#) in the 4.0.0 [allkeys](#) table, and find these lines:

```
304B ; [1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C ; [1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETTER GA; QQCM
```

The official Unicode names (following the 「#」 mark) tell us the Japanese syllabary (Hiragana), the informal classification (letter, digit, or punctuation mark), and the Western identifier ([KA](#) or [GA](#), which happen to be voiced and unvoiced components of the same letter pair). More importantly, the primary weight (the first hexadecimal number inside the square brackets) is [1E57](#) on both lines. For comparisons in both searching and sorting, MySQL pays attention to the primary weight only, ignoring all the other numbers. This means that we are sorting [が](#) and [か](#) correctly

according to the Unicode specification. If we wanted to distinguish them, we'd have to use a non-UCA (Unicode Collation Algorithm) collation ([utf8_unicode_bin](#) or [utf8_general_ci](#)), or to compare the [HEX\(\)](#) values, or use [ORDER BY CONVERT\(s1 USING sjis\)](#). Being correct 「according to Unicode」 isn't enough, of course: the person who submitted the bug was equally correct. We plan to add another collation for Japanese according to the JIS X 4061 standard, in which voiced/unvoiced letter pairs like [KA/GA](#) are distinguishable for ordering purposes.

A.12.16: Why don't CJK strings sort correctly in Unicode? (II)

If you are using Unicode ([ucs2](#) or [utf8](#)), and you know what the Unicode sort order is (see [MySQL 5.1 FAQ — MySQL Chinese, Japanese, and Korean Character Sets](#)), but MySQL still seems to sort your table incorrectly, then you should first verify the table character set:

```
mysql> SHOW CREATE TABLE t1G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Since the character set appears to be correct, let's see what information the [INFORMATION_SCHEMA.COLUMNS](#) table can provide about this column:

```
mysql> SELECT COLUMN_NAME, CHARACTER_SET_NAME, COLLATION_NAME
-> FROM INFORMATION_SCHEMA.COLUMNS
-> WHERE COLUMN_NAME = 's1'
-> AND TABLE_NAME = 't';
+-----+-----+-----+
| COLUMN_NAME | CHARACTER_SET_NAME | COLLATION_NAME |
+-----+-----+-----+
| s1          | ucs2               | ucs2_general_ci |
+-----+-----+-----+
1 row in set (0.01 sec)
```

(See [INFORMATION_SCHEMA COLUMNS テーブル](#), for more information.)

You can see that the collation is [ucs2_general_ci](#) instead of [ucs2_unicode_ci](#). The reason why this is so can be found using [SHOW CHARSET](#), as shown here:

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+
| ucs2    | UCS-2 Unicode | ucs2_general_ci   | 2      |
+-----+-----+-----+
1 row in set (0.00 sec)
```

For [ucs2](#) and [utf8](#), the default collation is 「general」. To specify a Unicode collation, use [COLLATE ucs2_unicode_ci](#).

A.12.17: Why are my supplementary characters rejected by MySQL?

MySQL does not support supplementary characters — that is, characters which need more than 3 bytes — for [UTF-8](#). We support only what Unicode calls the Basic Multilingual Plane / Plane 0. Only a few very rare Han characters are supplementary; support for them is uncommon. This has led to reports such as that found in [Bug #12600](#), which we rejected as 「not a bug」. With [utf8](#), we must truncate an input string when we encounter bytes that we don't understand. Otherwise, we wouldn't know how long the bad multi-byte character is.

One possible workaround is to use [ucs2](#) instead of [utf8](#), in which case the 「bad」 characters are changed to question marks; however, no truncation takes place. You can also change the data type to [BLOB](#) or [BINARY](#), which perform no validity checking.

We intend at some point in the future to add support for [UTF-16](#), which would solve such issues by allowing 4-byte characters. However, we have as yet set no definite timetable for doing so.

A.12.18: Shouldn't it be 「CJKV」?

No. The term 「CJKV」 (Chinese Japanese Korean Vietnamese) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL has no plan to support the old Vietnamese script using Han characters. MySQL does of course support the modern Vietnamese script with Western characters.

Bug #4745 is a request for a specialized Vietnamese collation, which we might add in the future if there is sufficient demand for it.

A.12.19: Does MySQL allow CJK characters to be used in database and table names?

This issue is fixed in MySQL 5.1, by automatically rewriting the names of the corresponding directories and files.

For example, if you create a database named 椿 on a server whose operating system does not support CJK in directory names, MySQL creates a directory named @0w@00a5@00ae. which is just a fancy way of encoding E6A5AE — that is, the Unicode hexadecimal representation for the 椿 character. However, if you run a SHOW DATABASES statement, you can see that the database is listed as 椿.

A.12.20: Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

A Simplified Chinese version of the Manual, current for MySQL 5.1.12, can be found at <http://dev.mysql.com/doc/#chinese-5.1>. The Japanese translation of the MySQL 4.1 manual can be downloaded from <http://dev.mysql.com/doc/#japanese-4.1>.

A.12.21: Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at <http://dev.mysql.com/user-groups/>.
- You can contact a sales engineer at the MySQL KK Japan office using any of the following:

Tel: +81(0)3-5326-3133
Fax: +81(0)3-5326-3001
Email: dsaito@mysql.com

- View feature requests relating to character set issues at <http://tinyurl.com/y6xcuf>.
- Visit the MySQL [Character Sets](#), [Collation](#), [Unicode Forum](#). We are also in the process of adding foreign-language forums at <http://forums.mysql.com/>.

MySQL 5.1 FAQ — Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- [C APIを使うときよく尋ねられる質問と問題](#)
- [MySQLとPHPに対する共通問題](#)
- [Connector/ODBC に関する注釈とヒント](#)
- [Connector/NET に関する注記とヒント](#)
- [Connector/J に関する注記とヒント](#)

付録B Errors, Error Codes, and Common Problems

目次

Problems and Common Errors	1491
How to Determine What Is Causing a Problem	1491
Common Errors When Using MySQL Programs	1492
Installation-Related Issues	1503
Administration-Related Issues	1505
Query-Related Issues	1510
Optimizer-Related Issues	1515
Table Definition-Related Issues	1515
Known Issues in MySQL	1517
Server Error Codes and Messages	1519
Client Error Codes and Messages	1559

This appendix lists common problems and errors that may occur and potential resolutions, in addition to listing the errors that may appear when you call MySQL from any host language. The first section covers problems and resolutions. Detailed information on errors is provided; The first list displays server error messages. The second list displays client program messages.

Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
 - The keyboard doesn't work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light doesn't change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
 - The mouse pointer doesn't move.
 - The machine doesn't answer to a remote machine's pings.
 - Other programs that are not related to MySQL don't behave correctly.
 - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as [glibc](#)) are up to date.

It's always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See [MySQL サーバ ログ](#).

- If you don't think you have hardware problems, you should try to find out which program is causing problems. Try using [top](#), [ps](#), Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use [top](#), [df](#), or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it doesn't want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it's time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the 「copy and paste」 method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only 「the system doesn't work.」 This doesn't provide us with any information about what could be the problem.

If a program fails, it's always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with [top](#). Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the [mysqld](#) server is causing problems, can you get any response from it with [mysqladmin -u root ping](#) or [mysqladmin -u root processlist](#)?
- What does a client program say when you try to connect to the MySQL server? (Try with [mysql](#), for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in [質問またはバグの報告](#).

Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

Access denied

An [Access denied](#) error can have many causes. Often the problem is related to the MySQL accounts that the server allows client programs to use when connecting. See [Access denied エラーの原因](#), and [権限システムの機能](#).

Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the [mysqld](#) server in two different ways: By using a Unix socket file to connect through a file in the filesystem (default [/tmp/mysql.sock](#)), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you don't specify a hostname or if you specify the special hostname [localhost](#).

If the MySQL server is running on Windows 9x or Me, you can connect only via TCP/IP. If the server is running on Windows NT, 2000, XP, or 2003 and is started with the [--enable-named-pipe](#) option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is [MySQL](#) by default. If you don't give a hostname when connecting to [mysqld](#), a MySQL client first tries to connect to the named pipe. If that doesn't work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using [.](#) as the hostname.

The error (2002) [Can't connect to ...](#) normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket filename or TCP/IP port number when trying to connect to the server.

The error (2003) [Can't connect to MySQL server on 'server' \(10061\)](#) indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, the network port you specified is the one configured on the server, and that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See [MySQL サーバの起動とトラブルシューティング](#).

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket filename might be different in your setup. `host_ip` represents the IP number of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=socket --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotes with the `hostname` command; these cause the output of `hostname` (that is, the current hostname) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the hostname of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Here are some reasons the [Can't connect to local MySQL server](#) error might occur:

- `mysqld` is not running. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: [Connection to MySQL Server Failing on Windows](#).
- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, `mysqld` uses the MIT-pthreads package. See [MySQL Community Server がサポートしているオペレーティング システム](#). However, not all MIT-pthreads versions support Unix socket files. On a system without socket file support, you must always specify the hostname explicitly when connecting to the server. Try using this command to check the connection to the server:

```
shell> mysqladmin -h `hostname` version
```

- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See [How to Protect or Change the MySQL Unix Socket File](#).
- You have started the `mysqld` server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket pathname for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

```
shell> netstat -ln | grep mysql
```

See [How to Protect or Change the MySQL Unix Socket File](#).

- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill` or with the `mysql_zap` script) before you can restart the MySQL server. See [What to Do If MySQL Keeps Crashing](#).
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` option that specifies a socket filename in a directory where the server can create it and where client programs can access it.

If you get the error message [Can't connect to MySQL server on some_host](#), you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- Make sure that your `mysqld` server was not started with the `--skip-networking` option. If it was, you cannot connect to it using TCP/IP.
- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default).

Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked.

Under Windows, applications such as ZoneAlarm and the Windows XP personal firewall may need to be configured to allow external access to a MySQL server.

- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows doesn't allow for enough ephemeral (short-lived) ports to serve those connections.

By default, Windows allows 5000 ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. This status allows the connection to be reused at a much lower cost than reinitializing a brand new connection. However, the port will not be available again until this time expires.

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

IMPORTANT: The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore the registry if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: <http://support.microsoft.com/kb/256986/EN-US/>.

1. Start Registry Editor (`Regedt32.exe`).
2. Locate the following key in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

3. On the **Edit** menu, click **Add Value**, and then add the following registry value:

```
Value Name: MaxUserPort
Data Type: REG_DWORD
Value: 65534
```

This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the **Edit** menu, click **Add Value**, and then add the following registry value:

```
Value Name: TcpTimedWaitDelay
```

```
Data Type: REG_DWORD
Value: 30
```

This sets the number of seconds to hold a TCP port connection in [TIME_WAIT](#) state before closing. The valid range is between 0 (zero) and 300 (decimal). The default value is 0x78 (120 decimal).

5. Quit Registry Editor.
6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

Client does not support authentication protocol

MySQL 5.1 uses an authentication protocol based on a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. If you upgrade the server from 4.0, attempts to connect to it with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To solve this problem, you should use one of the following approaches:

- Upgrade all client programs to use a 4.1.1 or newer client library.
- When connecting to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the [SET PASSWORD](#) statement and the [OLD_PASSWORD\(\)](#) function:

```
mysql> SET PASSWORD FOR
-> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

Alternatively, use [UPDATE](#) and [FLUSH PRIVILEGES](#):

```
mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
-> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

Substitute the password you want to use for `'newpwd'` in the preceding examples. MySQL cannot tell you what the original password was, so you'll need to pick a new one.

- Tell the server to use the older password hashing algorithm:
 1. Start [mysqld](#) with the [--old-passwords](#) option.
 2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

```
mysql> SELECT Host, User, Password FROM mysql.user
-> WHERE LENGTH>Password) > 16;
```

For each account record displayed by the query, use the [Host](#) and [User](#) values and assign a password using the [OLD_PASSWORD\(\)](#) function and either [SET PASSWORD](#) or [UPDATE](#), as described earlier.

Note: In older versions of PHP, the [mysql](#) extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the [mysql](#) extension with MySQL 4.1 or newer, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The [mysqli](#) extension (stands for "MySQL, Improved"; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the [mysqli](#) extension, see <http://php.net/mysqli>.

It may also be possible to compile the older [mysql](#) extension against the new MySQL client library. This is beyond the scope of this Manual; consult the PHP documentation for more information. You also be able to obtain assistance with these issues in our [MySQL with PHP forum](#).

For additional background on password hashing and authentication, see [MySQL 4.1 のパスワードハッシュ](#).

Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

Host 'host_name' is blocked

If you get the following error, it means that `mysqld` has received many connect requests from the host `'host_name'` that have been interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The number of interrupted connect requests allowed is determined by the value of the `max_connect_errors` system variable. After `max_connect_errors` failed requests, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you execute a `mysqladmin flush-hosts` command or issue a `FLUSH HOSTS` statement. See [システム変数](#).

By default, `mysqld` blocks a host after 10 connection errors. You can adjust the value by starting the server like this:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

If you get this error message for a given host, you should first verify that there isn't anything wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` variable.

Too many connections

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that all available connections are in use by other clients.

The number of connections allowed is controlled by the `max_connections` system variable. Beginning with MySQL 5.1.15, its default value is 151 to improve performance when MySQL is used with the Apache Web server. (Previously, the default was 100.) If you need to support more connections, you should restart `mysqld` with a larger value for this variable.

MySQL Enterprise

Subscribers to the MySQL Network Monitoring and Advisory Service receive advice on dynamically configuring the `max_connections` variable — avoiding failed connection attempts. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

`mysqld` actually allows `max_connections+1` clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the `SUPER` privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See [SHOW PROCESSLIST 構文](#).

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform. Linux or Solaris should be able to support 500-1000 simultaneous connections, depending on how much RAM you have and what your clients are doing. Static Linux binaries provided by MySQL AB can support up to 4000 connections.

Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

MySQL server has gone away

This section also covers the related [Lost connection to server during query](#) error.

The most common reason for the [MySQL server has gone away](#) error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent):

Error Code	Description
CR_SERVER_GONE_ERROR	The client couldn't send a question to the server.
CR_SERVER_LOST	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start `mysqld`. See [システム変数](#).

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the [MySQL server has gone away](#) error are:

- You (or the db administrator) has killed the running thread with a [KILL](#) statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

The problem on Windows is that in some cases MySQL doesn't get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

Prior to MySQL 5.1.8, even if the `reconnect` flag in the `MYSQL` structure is equal to 1, MySQL does not automatically reconnect and re-issue the query as it doesn't know if the server did get the original query or not.

The solution to this is to either do a `mysql_ping` on the connection if there has been a long time since the last query (this is what `MyODBC` does) or set `wait_timeout` on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the

connection. If you need big queries (for example, if you are working with big [BLOB](#) columns), you can increase the query limit by setting the server's `max_allowed_packet` variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in [Packet too large](#).

An [INSERT](#) or [REPLACE](#) statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per [INSERT](#) or [REPLACE](#).

- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.
- It is also possible to see this error if hostname lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working — from MySQL's point of view the problem is indistinguishable from any other network timeout.

You may also see the [MySQL server has gone away](#) error if MySQL is started with the `--skip-networking` option.

Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.
- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing [mysqladmin version](#) and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See [What to Do If MySQL Keeps Crashing](#).

You can get more information about the lost connections by starting `mysqld` with the `--log-warnings=2` option. This logs some of the disconnected errors in the `hostname.err` file. See [エラー ログ](#).

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See [What to Do If MySQL Keeps Crashing](#).
- If a specific query kills `mysqld` and the tables involved were checked with [CHECK TABLE](#) before you ran the query, can you provide a reproducible test case? See [Making a Test Case If You Experience Table Corruption](#).
- What is the value of the `wait_timeout` system variable in the MySQL server? ([mysqladmin variables](#) gives you the value of this variable.)
- Have you tried to run `mysqld` with the `--log` option to determine whether the problem query appears in the log?

See also [Communication Errors and Aborted Connections](#), and [質問またはバグの報告](#).

Packet too large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

The largest possible packet that can be transmitted to or from a MySQL 5.1 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` bytes, it issues a [Packet too large](#) error and closes the connection. With some clients, you may also get a [Lost connection to MySQL server during query](#) error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

Communication Errors and Aborted Connections

The server error log can be a useful source of information about connection problems. See [エラー ログ](#). If you start the server with the `--log-warnings` option, you might find messages like this in your error log:

```
010301 14:38:23 Aborted connection 854 to db: 'users' user: 'josh'
```

If [Aborted connections](#) messages appear in the error log, the cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See [システム変数](#).
- The client program ended abruptly in the middle of a data transfer.

When any of these things happen, the server increments the `Aborted_clients` status variable.

The server increments the `Aborted_connects` status variable when the following things happen:

- A client doesn't have privileges to connect to a database.
- A client uses an incorrect password.
- A connection packet doesn't contain the right information.
- It takes more than `connect_timeout` seconds to get a connect packet. See [システム変数](#).

If these kinds of things happen, it might indicate that someone is trying to break into your server!

MySQL Enterprise

For reasons of security and performance the advisors provided by the MySQL Network Monitoring and Advisory Service pay special attention to the `Aborted_connections` status variable. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

Other reasons for problems with aborted clients or aborted connections:

- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file via FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. The only solution is switching the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and testing the results to determine the best setting.
- Some problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.
- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See [Packet too large](#).

See also [MySQL server has gone away](#).

The table is full

The maximum effective table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	File-size Limit
Win32 w/ FAT/FAT32	2GB/4GB
Win32 w/ NTFS	2TB (possibly larger)
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 filesystem) 4TB
Solaris 9/10	16TB
MacOS X w/ HFS+	2TB
NetWare w/NSS filesystem	8TB

Windows users, please note that FAT and VFAT (FAT32) are not considered suitable for production use with MySQL. Use NTFS instead.

On Linux 2.2, you can get [MyISAM](#) tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 filesystem. Most current Linux distributions are based on kernel 2.4 or higher and include all the required LFS patches. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). With JFS and XFS, petabyte and larger files are possible on Linux.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's Large File Support in Linux page at http://www.suse.de/~aj/linux_lfs.html.

If you do encounter a full-table error, there are several reasons why it might have occurred:

- You are using a MySQL server older than 3.23 and an in-memory temporary table becomes larger than `tmp_table_size` bytes. To avoid this problem, you can use the `--tmp_table_size=val` option to make `mysqld` increase the temporary table size or use the SQL option `SQL_BIG_TABLES` before you issue the problematic query. See [SET 構文](#).

You can also start `mysqld` with the `--big-tables` option. This is exactly the same as using `SQL_BIG_TABLES` for all queries.

As of MySQL 3.23, this problem should not occur. If an in-memory temporary table becomes larger than `tmp_table_size`, the server automatically converts it to a disk-based [MyISAM](#) table.

- The [InnoDB](#) storage engine maintains [InnoDB](#) tables within a tablespace that can be created from several files. This allows a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which allows extremely large tables. The maximum tablespace size is 64TB.

If you are using [InnoDB](#) tables and run out of room in the [InnoDB](#) tablespace. In this case, the solution is to extend the [InnoDB](#) tablespace. See [InnoDB データとログ ファイルの追加と削除](#).

- You are using [MyISAM](#) tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a [MyISAM](#) table and the space required for the table exceeds what is allowed by the internal pointer size. [MyISAM](#) creates tables to allow up to 256GB by default, but this limit can be changed up to the maximum allowable size of 65,536TB ($256^7 - 1$ bytes).

If you need a [MyISAM](#) table that is larger than the default limit and your operating system supports large files, the [CREATE TABLE](#) statement supports [AVG_ROW_LENGTH](#) and [MAX_ROWS](#) options. See [CREATE TABLE 構文](#). The server uses these options to determine how large a table to allow.

If the pointer size is too small for an existing table, you can change the options with [ALTER TABLE](#) to increase a table's maximum allowable size. See [ALTER TABLE 構文](#).

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

You have to specify [AVG_ROW_LENGTH](#) only for tables with [BLOB](#) or [TEXT](#) columns; in this case, MySQL can't optimize the space required based only on the number of rows.

To change the default size limit for [MyISAM](#) tables, set the [mysam_data_pointer_size](#), which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the [MAX_ROWS](#) option. The value of [mysam_data_pointer_size](#) can be from 2 to 7. A value of 4 allows tables up to 4GB; a value of 6 allows tables up to 256TB.

You can check the maximum data and index sizes by using this statement:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

You also can use [mysamchk -dv /path/to/table-index-file](#). See [SHOW 構文](#), or [mysamchk — MyISAM テーブル メンテナンス ユーティリティ](#).

Other ways to work around file-size limits for [MyISAM](#) tables are as follows:

- If your large table is read-only, you can use [mysampack](#) to compress it. [mysampack](#) usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. [mysampack](#) also can merge multiple tables into a single table. See [mysampack — 圧縮された、読み取り専用MyISAM テーブルを作成する。](#)
- MySQL includes a [MERGE](#) library that allows you to handle a collection of [MyISAM](#) tables that have identical structure as a single [MERGE](#) table. See [MERGE ストレージエンジン](#).
- You are using the [NDB](#) storage engine, in which case you need to increase the values for the [DataMemory](#) and [IndexMemory](#) configuration parameters in your [config.ini](#) file. See [データノードの設定パラメータ](#).
- You are using the [MEMORY \(HEAP\)](#) storage engine; in this case you need to increase the value of the [max_heap_table_size](#) system variable. See [システム変数](#).

Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file 'sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start [mysqld](#) with the [--tmpdir](#) option or to add the option to the [\[mysqld\]](#) section of your option file. For example, to specify a directory of [C:\temp](#), use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See [オプションファイルの使用](#).

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with [perror](#). One reason the server cannot write to a table is that the filesystem is full:

```
shell> perror 28
Error code 28: No space left on device
```

Commands out of sync

If you get [Commands out of sync; you can't run this command now](#) in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

Ignoring user

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

[Found wrong password for user 'some_user'@'some_host'; ignoring user](#)

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see whether the `Password` column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The account has an old password (eight characters long) and you didn't start `mysqld` with the `--old-protocol` option. Update the account in the `user` table to have a new password or restart `mysqld` with the `--old-protocol` option.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use `mysql` to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` function:

```
mysql> UPDATE user SET Password=PASSWORD('newpwd')
-> WHERE User='some_user' AND Host='some_host';
```

Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a filesystem that has case-sensitive filenames.
- Even for filesystems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See [SHOW 構文](#).

Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset\_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running [configure](#) with the `--with-charset=charset_name` or `--with-extra-charsets=charset_name` option. See [典型的な configure オプション](#).

All standard MySQL binaries are compiled with `--with-extra-character-sets=complex`, which enables support for all multi-byte character sets. See [データおよびソート用キャラクタ セット](#).

- The character set is a simple character set that is not compiled into [mysqld](#), and the character set definition files are not in the place where the client expects to find them.

In this case, you need to use one of the following methods to solve the problem:

- Recompile the client with support for the character set. See [典型的な configure オプション](#).
- Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
- Copy the character definition files to the path where the client expects them to be.

'File' Not Found and Similar Errors

If you get [ERROR '...' not found \(errno: 23\)](#), [Can't open file: ... \(errno: 24\)](#), or any other error with [errno 23](#) or [errno 24](#) from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the [perlor](#) utility to get a description of what the error number means:

```
shell> perlor 23
Error code 23: File table overflow
shell> perlor 24
Error code 24: Too many open files
shell> perlor 11
Error code 11: Resource temporarily unavailable
```

The problem here is that [mysqld](#) is trying to keep open too many files simultaneously. You can either tell [mysqld](#) not to open so many files at once or increase the number of file descriptors available to [mysqld](#).

To tell [mysqld](#) to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the [table_open_cache](#) system variable (the default value is 64). Reducing the value of [max_connections](#) also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to [mysqld](#), you can use the `--open-files-limit` option to [mysqld_safe](#) or (as of MySQL 3.23.30) set the [open_files_limit](#) system variable. See [システム変数](#). The easiest way to set these values is to add an option to your option file. See [オプションファイルの使用](#). If you have an old version of [mysqld](#) that doesn't support setting the open files limit, you can edit the [mysqld_safe](#) script. There is a commented-out line `ulimit -n 256` in the script. You can remove the '#' character to uncomment this line, and change the number 256 to set the number of file descriptors to be made available to [mysqld](#).

`--open-files-limit` and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a 「hard」 limit that can be overridden only if you start [mysqld_safe](#) or [mysqld](#) as [root](#) (just remember that you also need to start the server with the `--user` option in this case so that it does not continue to run as [root](#) after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note: If you run the [tcsh](#) shell, `ulimit` does not work! [tcsh](#) also reports incorrect values when you ask for the current limits. In this case, you should start [mysqld_safe](#) using [sh](#).

Installation-Related Issues

Problems Linking to the MySQL Client Library

When you are linking an application program to use the MySQL client library, you might get undefined reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x57): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-Ldir_path -lmysqlclient` at the end of your link command, where `dir_path` represents the pathname of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

If you get `undefined reference` errors for a function that should exist on your system, such as `connect`, check the manual page for the function in question to determine which libraries you should add to the link command.

You might get `undefined reference` errors such as the following for functions that don't exist on your system:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

This usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours. In this case, you should download the latest MySQL source distribution and compile MySQL yourself. See [ソースのディストリビューションを使用した MySQL のインストール](#).

You might get undefined reference errors at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `mysqlclient` library can't be found, it means that your system can't find the shared `libmysqlclient.so` library. The fix for this is to tell your system to search for shared libraries where the library is located. Use whichever of the following methods is appropriate for your system:

- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` environment variable.
- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY` environment variable.
- Copy `libmysqlclient.so` to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`.

Another way to solve this problem is by linking your program statically with the `-static` option, or by removing the dynamic MySQL libraries before linking your code. Before trying the second method, you should be sure that no other programs are using the dynamic libraries.

Problems with File Permissions

If you have problems with file permissions, the `UMASK` environment variable might be set incorrectly when `mysqld` starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` value is `0660`. You can change this behavior by starting `mysqld_safe` as follows:

```
shell> UMASK=384 # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database directories with an access permission value of `0700`. You can modify this behavior by setting the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504 # = 770 in octal
shell> export UMASK_DIR
```



```
shell> mysqld_safe &
```

In MySQL 3.23.25 and above, MySQL assumes that the value for `UMASK` and `UMASK_DIR` is in octal if it starts with a zero.

See [環境変数](#).

Administration-Related Issues

How to Reset the Root Password

If you have never set a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, it is recommended to set a password for each account. See [セキュリティ ガイドライン](#).

If you set a `root` password previously, but have forgotten what it was, you can set a new password. The following procedure is for Windows systems. The procedure for Unix systems is given later in this section.

The procedure under Windows:

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager:

```
Start Menu -> Control Panel -> Administrative Tools -> Services
```

Then find the MySQL service in the list, and stop it.

If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file and place the following command within it on a single line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

Save the file with any name. For this example the file will be `C:\mysql-init.txt`.

4. Open a console window to get to the DOS command prompt:

```
Start Menu -> Run -> cmd
```

5. We are assuming that you installed MySQL to `C:\mysql`. If you installed MySQL to another location, adjust the following commands accordingly.

At the DOS command prompt, execute this command:

```
C:\> C:\mysql\bin\mysqld-nt --init-file=C:\mysql-init.txt
```

The contents of the file named by the `--init-file` option are executed at server startup, changing the `root` password. After the server has started successfully, you should delete `C:\mysql-init.txt`.

If you install MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld-nt.exe"
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.1\my.ini"
--init-file=C:\mysql-init.txt
```

The appropriate `--defaults-file` setting can be found using the Services Manager:

```
Start Menu -> Control Panel -> Administrative Tools -> Services
```

Find the MySQL service in the list, right-click on it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` setting.

6. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.
7. You should be able to connect using the new password.

In a Unix environment, the procedure for resetting the `root` password is as follows:

1. Log on to your system as either the Unix `root` user or as the same user that the `mysqld` server runs as.
2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, hostname, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the filename has the extension of `.pid` and begins with either `mysqld` or your system's hostname.

You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process, using the pathname of the `.pid` file in the following command:

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

Note the use of backticks rather than forward quotes with the `cat` command; these cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file and place the following command within it on a single line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

Save the file with any name. For this example the file will be `~/mysql-init`.

4. Restart the MySQL server with the special `--init-file=~/mysql-init` option:

```
shell> mysqld_safe --init-file=~/mysql-init &
```

The contents of the init-file are executed at server startup, changing the root password. After the server has started successfully you should delete `~/mysql-init`.

5. You should be able to connect using the new password.

Alternatively, on any platform, you can set the new password using the `mysql` client (but this approach is less secure):

1. Stop `mysqld` and restart it with the `--skip-grant-tables --user=root` options (Windows users omit the `--user=root` portion).
2. Connect to the `mysqld` server with this command:

```
shell> mysql -u root
```

3. Issue the following statements in the `mysql` client:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')
-> WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

Replace `'newpwd'` with the actual `root` password that you want to use.

4. You should be able to connect using the new password.

What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This doesn't mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See [エラー ログ](#).

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `resolve_stack_dump` program. See [Using a Stack Trace](#). Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `--delay-key-write`, in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that doesn't support good filesystem locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it's at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all MyISAM tables, and restart `mysqld`. This ensures that you are running from a clean state. See [4章データベース管理](#).
- Start `mysqld` with the `--log` option and try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See [一般クエリ ログ](#). If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See [質問またはバグの報告](#).
- Try to make a test case that we can use to repeat the problem. See [Making a Test Case If You Experience Table Corruption](#).
- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See [MySQL Test Suite](#). They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening. Reconfigure MySQL with the `--with-debug` or `--with-debug=full` option to `configure` and then recompile. See [Debugging a MySQL Server](#).
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while you run your other queries.
- Try the following:
 1. Start `mysqld` from `gdb` (or another debugger). See [Debugging mysqld under gdb](#).
 2. Run your test scripts.
 3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

```
backtrace
info local
up
info local
up
info local
```

With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where `N` is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See [質問またはバグの報告](#). Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

The current dynamic row code has been in use at MySQL AB for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.

- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to both RAMS and hard-drives when troubleshooting hardware.

How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as 「no space left on device」), and to quota-exceeded errors (such as 「write failed」 or 「user block limit reached」).

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to 「row」 and 「record」 should be understood to mean 「event」.

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several 「locked」 threads, killing the one thread that is waiting on the disk-full condition allows the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of these statements may create

large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for **ALTER TABLE**, the old table is left unchanged.

Where MySQL Stores Temporary Files

MySQL uses the value of the **TMPDIR** environment variable as the pathname of the directory in which to store temporary files. If you don't have **TMPDIR** set, MySQL uses the system default, which is normally **/tmp**, **/var/tmp**, or **/usr/tmp**. If the filesystem containing your temporary file directory is too small, you can use the **--tmpdir** option to **mysqld** to specify a directory in a filesystem where you have enough space.

In MySQL 5.1, the **--tmpdir** option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (':') on Unix and semicolon characters(';') on Windows, NetWare, and OS/2. Note: To spread the load effectively, these paths should be located on different physical disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should not set **--tmpdir** to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or **LOAD DATA INFILE** operations. If files in the temporary file directory are lost when the server restarts, replication fails.

MySQL creates all temporary files as hidden files. This ensures that the temporary files are removed if **mysqld** is terminated. The disadvantage of using hidden files is that you do not see a big temporary file that fills up the filesystem in which the temporary file directory is located.

MySQL Enterprise

Advisors provided by the MySQL Network Monitoring and Advisory Service automatically detect excessive temporary table storage to disk. For more information see, <http://www.mysql.com/products/enterprise/advisors.html>.

When sorting (**ORDER BY** or **GROUP BY**), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some **SELECT** queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form **SQL_***.

ALTER TABLE creates a temporary table in the same directory as the original table.

How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is **/tmp/mysql.sock**. (For some distribution formats, the directory might be different, such as **/var/lib/mysql** for RPMs.)

On some versions of Unix, anyone can delete files in the **/tmp** directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your **/tmp** directory so that files can be deleted only by their owners or the superuser (**root**). To do this, set the **sticky** bit on the **/tmp** directory by logging in as **root** and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the **sticky** bit is set by executing **ls -ld /tmp**. If the last permission character is **t**, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in **/etc/my.cnf**:

```
[mysqld]
socket=/path/to/socket

[client]
socket=/path/to/socket
```

See [オプションファイルの使用](#).

- Specify a `--socket` option on the command line to `mysqld_safe` and when you run client programs.
- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `--with-unix-socket-path` option when you run `configure`. See [典型的な configure オプション](#).

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` returns the wrong value. This should be done for the environment in which the server runs; for example, in `mysqld_safe` or `mysql.server`. See [環境変数](#).

You can set the time zone for the server with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The allowable values for `--timezone` or `TZ` are system-dependent. Consult your operating system documentation to see what values are acceptable.

Query-Related Issues

Case Sensitivity in Searches

By default, MySQL searches are not case sensitive (although there are some character sets that are never case insensitive, such as `czech`). This means that if you search with `col_name LIKE 'a%'`, you get all column values that start with `A` or `a`. If you want to make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation. For example:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See [CREATE TABLE 構文](#).

Simple comparison operations (`>=`, `>`, `=`, `<`, `<=`, sorting, and grouping) are based on each character's 「sort value.」 Characters with the same sort value (such as `'E'`, `'e'`, and `'Ã#Ã©'`) are treated as the same character.

Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is allowed. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
mysql> SELECT * FROM tbl_name WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context (and vice versa). It is also smart enough to allow a 「relaxed」 string form when updating and in a `WHERE` clause that compares a date to a `TIMESTAMP`, `DATE`, or `DATETIME` column. (「Relaxed form」 means that any punctuation

character may be used as the separator between parts. For example, '2004-08-15' and '2004#08#15' are equivalent.) MySQL can also convert a string containing no separators (such as '20040815'), provided it makes sense as a date.

When you compare a [DATE](#), [TIME](#), [DATETIME](#), or [TIMESTAMP](#) to a constant string with the [<](#), [<=](#), [=](#), [>=](#), [>](#), or [BETWEEN](#) operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more 「relaxed」 string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a [DATE](#), [TIME](#), [DATETIME](#), or [TIMESTAMP](#) column to an expression
- When you use any other comparison method than those just listed, such as [IN](#) or [STRCMP\(\)](#).

For these exceptional cases, the comparison is done by converting the objects to strings and performing a string comparison.

To keep things safe, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special date '0000-00-00' can be stored and retrieved as '0000-00-00'. When using a '0000-00-00' date through MyODBC, it is automatically converted to [NULL](#) in MyODBC 2.50.12 and above, because ODBC can't handle this kind of date.

Because MySQL performs the conversions described above, the following statements work:

```
mysql> INSERT INTO tbl_name (idate) VALUES (19970505);
mysql> INSERT INTO tbl_name (idate) VALUES ('19970505');
mysql> INSERT INTO tbl_name (idate) VALUES ('97-05-05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997.05.05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997 05 05');
mysql> INSERT INTO tbl_name (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM tbl_name WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT idate FROM tbl_name WHERE idate >= '19970505';
```

However, the following does not work:

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate,'20030505')=0;
```

[STRCMP\(\)](#) is a string function, so it converts [idate](#) to a string in 'YYYY-MM-DD' format and performs a string comparison. It does not convert '20030505' to the date '2003-05-05' and perform a date comparison.

If you are using the [ALLOW_INVALID_DATES](#) SQL mode, MySQL allows you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12.

This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

If you are not using the [NO_ZERO_IN_DATE](#) SQL mode, the day or month part can be zero. This is convenient if you want to store a birthdate in a [DATE](#) column and you know only part of the date.

If you are not using the [NO_ZERO_DATE](#) SQL mode, MySQL also allows you to store '0000-00-00' as a 「dummy date.」 This is in some cases more convenient than using [NULL](#) values.

If the date cannot be converted to any reasonable value, a 0 is stored in the [DATE](#) column, which is retrieved as '0000-00-00'. This is both a speed and a convenience issue. We believe that the database server's responsibility is to retrieve the same date you stored (even if the data was not logically correct in all cases). We think it is up to the application and not the server to check the dates.

If you want MySQL to check all dates and accept only legal dates (unless overridden by [IGNORE](#)), you should set [sql_mode](#) to "NO_ZERO_IN_DATE,NO_ZERO_DATE".

Problems with [NULL](#) Values

The concept of the **NULL** value is a common source of confusion for newcomers to SQL, who often think that **NULL** is the same thing as an empty string `"`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

Both statements insert a value into the `phone` column, but the first inserts a **NULL** value and the second inserts an empty string. The meaning of the first can be regarded as 「phone number is not known」 and the meaning of the second can be regarded as 「the person is known to have no phone, and thus no phone number.」

To help with **NULL** handling, you can use the **IS NULL** and **IS NOT NULL** operators and the **IFNULL()** function.

In SQL, the **NULL** value is never true in comparison to any other value, even **NULL**. An expression that contains **NULL** always produces a **NULL** value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return **NULL**:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

If you want to search for column values that are **NULL**, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for **NULL** values, you must use the **IS NULL** test. The following statements show how to find the **NULL** phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

See [Working with NULL Values](#), for additional information and examples.

You can add an index on a column that can have **NULL** values if you are using the **MyISAM**, **InnoDB**, or **MEMORY** storage engine. Otherwise, you must declare an indexed column **NOT NULL**, and you cannot insert **NULL** into the column.

When reading data with **LOAD DATA INFILE**, empty or missing columns are updated with `"`. If you want a **NULL** value in a column, you should use `\N` in the data file. The literal word 「**NULL**」 may also be used under some circumstances. See [LOAD DATA INFILE 構文](#).

When using **DISTINCT**, **GROUP BY**, or **ORDER BY**, all **NULL** values are regarded as equal.

When using **ORDER BY**, **NULL** values are presented first, or last if you specify **DESC** to sort in descending order.

Aggregate (summary) functions such as **COUNT()**, **MIN()**, and **SUM()** ignore **NULL** values. The exception to this is **COUNT(*)**, which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-**NULL** values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles **NULL** values specially. If you insert **NULL** into a **TIMESTAMP** column, the current date and time is inserted. If you insert **NULL** into an integer column that has the **AUTO_INCREMENT** attribute, the next number in the sequence is inserted.

Problems with Column Aliases

You can use an alias to refer to a column in **GROUP BY**, **ORDER BY**, or **HAVING** clauses. Aliases can also be used to give columns better names:

```
SELECT SQRT(a*b) AS root FROM tbl_name GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL doesn't allow you to refer to a column alias in a [WHERE](#) clause. This restriction is imposed because when the [WHERE](#) code is executed, the column value may not yet be determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl\_name WHERE cnt > 0 GROUP BY id;
```

The [WHERE](#) statement is executed to determine which rows should be included in the [GROUP BY](#) part, whereas [HAVING](#) is used to decide which rows from the result set should be used.

Rollback Failure for Non-Transactional Tables

If you receive the following message when trying to perform a [ROLLBACK](#), it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These non-transactional tables are not affected by the [ROLLBACK](#) statement.

If you were not deliberately mixing transactional and non-transactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your [mysqld](#) server (or that was disabled with a startup option). If [mysqld](#) doesn't support a storage engine, it instead creates the table as a [MyISAM](#) table, which is non-transactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl\_name';
SHOW CREATE TABLE tbl\_name;
```

See [SHOW TABLE STATUS 構文](#), and [SHOW CREATE TABLE 構文](#).

You can check which storage engines your [mysqld](#) server supports by using this statement:

```
SHOW ENGINES;
```

You can also use the following statement, and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the [InnoDB](#) storage engine is available, check the value of the [have_innodb](#) variable.

See [SHOW ENGINES 構文](#), and [SHOW VARIABLES 構文](#).

Deleting Rows from Related Tables

If the total length of the [DELETE](#) statement for [related_table](#) is more than 1MB (the default value of the [max_allowed_packet](#) system variable), you should split it into smaller parts and execute multiple [DELETE](#) statements. You probably get the fastest [DELETE](#) by specifying only 100 to 1,000 [related_column](#) values per statement if the [related_column](#) is indexed. If the [related_column](#) isn't indexed, the speed is independent of the number of arguments in the [IN](#) clause.

Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that doesn't return any rows, you should use the following procedure to find out what is wrong:

1. Test the query with [EXPLAIN](#) to check whether you can find something that is obviously wrong. See [EXPLAIN を使用して、クエリを最適化する](#).
2. Select only those columns that are used in the [WHERE](#) clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are large, it's a good idea to use [LIMIT 10](#) with the query.

4. Issue a [SELECT](#) for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing [FLOAT](#) or [DOUBLE](#) columns with numbers that have decimals, you can't use equality (=) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the [FLOAT](#) to a [DOUBLE](#) fixes this. See [Problems with Floating-Point Comparisons](#).
6. If you still can't figure out what's wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your [SELECT](#) statement at the end of the file.

Verify that the test file demonstrates the problem by executing these commands:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Attach the test file to a bug report, which you can file using the instructions in [質問またはバグの報告](#).

Problems with Floating-Point Comparisons

Floating-point numbers sometimes cause confusion because they are approximate. That is, they are not stored as exact values inside computer architecture. What you can see on the screen usually is not the exact value of the number. The [FLOAT](#) and [DOUBLE](#) data types are such. For [DECIMAL](#) columns, MySQL performs operations with a precision of 64 decimal digits, which should solve most common inaccuracy problems.

The following example demonstrates the problem using [DOUBLE](#). It shows that are calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
-> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
-> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
-> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
-> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
-> (6, 0.00, 0.00), (6, -51.40, 0.00);
```

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

```
+-----+-----+-----+
| i | a | b |
+-----+-----+-----+
| 1 | 21.4 | 21.4 |
| 2 | 76.8 | 76.8 |
| 3 | 7.4 | 7.4 |
| 4 | 15.4 | 15.4 |
| 5 | 7.2 | 7.2 |
| 6 | -51.4 | 0 |
+-----+-----+-----+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of [a](#) and [b](#) do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns [d1](#) and [d2](#) had been defined as [DECIMAL](#) rather than [DOUBLE](#), the result of the [SELECT](#) query would have contained only one row — the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
```

```

+-----+-----+
| i | a | b |
+-----+-----+
| 6 | -51.4 | 0 |
+-----+-----+
1 row in set (0.00 sec)

```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+-----+-----+
| i | a | b |
+-----+-----+
| 1 | 21.4 | 21.4 |
| 2 | 76.8 | 76.8 |
| 3 | 7.4 | 7.4 |
| 4 | 15.4 | 15.4 |
| 5 | 7.2 | 7.2 |
+-----+-----+
5 rows in set (0.03 sec)

```

Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL doesn't have enough information about the data at hand and has to make 「educated」 guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the [EXPLAIN](#) statement to get information about how MySQL processes a query. To use it, just add the keyword [EXPLAIN](#) to the front of your [SELECT](#) statement:

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

[EXPLAIN](#) is discussed in more detail in [EXPLAINを使用して、クエリを最適化する](#).

- Use [ANALYZE TABLE tbl_name](#) to update the key distributions for the scanned table. See [ANALYZE TABLE 構文](#).
- Use [FORCE INDEX](#) for the scanned table to tell MySQL that table scans are very expensive compared to using the given index. See [SELECT 構文](#).

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

[USE INDEX](#) and [IGNORE INDEX](#) may also be useful.

- Global and table-level [STRAIGHT_JOIN](#). See [SELECT 構文](#).
- You can tune global or thread-specific system variables. For example, Start [mysqld](#) with the [--max-seeks-for-key=1000](#) option or use [SET max_seeks_for_key=1000](#) to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [システム変数](#).

Table Definition-Related Issues

Problems with [ALTER TABLE](#)

[ALTER TABLE](#) changes a table to the current character set. If you get a duplicate-key error during [ALTER TABLE](#), the cause is either that the new character sets maps two keys to the same value or that the table is corrupted. In the latter case, you should run [REPAIR TABLE](#) on the table.

If [ALTER TABLE](#) dies with the following error, the problem may be that MySQL crashed during an earlier [ALTER TABLE](#) operation and there is an old table named [A-xxx](#) or [B-xxx](#) lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with **A-** or **B-**. (You may want to move them elsewhere instead of deleting them.)

ALTER TABLE works in the following way:

- Create a new table named **A-xxx** with the requested structural changes.
- Copy all rows from the original table to **A-xxx**.
- Rename the original table to **B-xxx**.
- Rename **A-xxx** to your original table name.
- Delete **B-xxx**.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as **B-xxx**. A simple rename of the table files at the system level should get your data back.

If you use **ALTER TABLE** on a transactional table or if you are using Windows or OS/2, **ALTER TABLE** unlocks the table if you had done a **LOCK TABLE** on it. This is done because **InnoDB** and these operating systems cannot drop a table that is in use.

How to Change the Order of Columns in a Table

First, consider whether you really need to change the column order in a table. The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in which you wish to retrieve your data. The first of the following statements returns columns in the order **col_name1**, **col_name2**, **col_name3**, whereas the second returns them in the order **col_name1**, **col_name3**, **col_name2**:

```
mysql> SELECT col_name1, col_name2, col_name3 FROM tbl_name;
mysql> SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

If you decide to change the order of table columns anyway, you can do so as follows:

1. Create a new table with the columns in the new order.
2. Execute this statement:

```
mysql> INSERT INTO new_table
-> SELECT columns-in-new-order FROM old_table;
```

3. Drop or rename **old_table**.
4. Rename the new table to the original name:

```
mysql> ALTER TABLE new_table RENAME old_table;
```

SELECT * is quite suitable for testing queries. However, in an application, you should never rely on using **SELECT *** and retrieving the columns based on their position. The order and position in which columns are returned does not remain the same if you add, move, or delete columns. A simple change to your table structure could cause your application to fail.

TEMPORARY TABLE Problems

The following list indicates limitations on the use of **TEMPORARY** tables:

- A **TEMPORARY** table can only be of type **MEMORY**, **MyISAM**, **MERGE**, or **InnoDB**.

Temporary tables are not supported for MySQL Cluster.

- You cannot refer to a **TEMPORARY** table more than once in the same query. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```


- The [SHOW TABLES](#) statement does not list [TEMPORARY](#) tables.
- You cannot use [RENAME](#) to rename a [TEMPORARY](#) table. However, you can use [ALTER TABLE](#) instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

- There are known issues in using temporary tables with replication. See [レプリケーション機能と既知問題](#), for more information.

Known Issues in MySQL

This section is a list of the known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in [オペレーティング システムに特化した注釈](#), and [Debugging and Porting MySQL](#).

Open Issues in MySQL

The following problems are known and fixing them is a high priority:

- MySQL Cluster fails to recover from an out-of-disk failure when using disk data. (Bug #17614)
- If you compare a [NULL](#) value to a subquery using [ALL/ANY/SOME](#) and the subquery returns an empty result, the comparison might evaluate to the non-standard result of [NULL](#) rather than to [TRUE](#) or [FALSE](#). This will be fixed in MySQL 5.1.
- Subquery optimization for [IN](#) is not as effective as for [=](#).
- Even if you use [lower_case_table_names=2](#) (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function [DATABASE\(\)](#) or within the various logs (on case-insensitive systems).
- Dropping a [FOREIGN KEY](#) constraint doesn't work in replication because the constraint may have another name on the slave.
- [REPLACE](#) (and [LOAD DATA](#) with the [REPLACE](#) option) does not trigger [ON DELETE CASCADE](#).
- [DISTINCT](#) with [ORDER BY](#) doesn't work inside [GROUP_CONCAT\(\)](#) if you don't use all and only those columns that are in the [DISTINCT](#) list.
- If one user has a long-running transaction and another user drops a table that is updated in the transaction, there is small chance that the binary log may contain the [DROP TABLE](#) command before the table is used in the transaction itself. We plan to fix this by having the [DROP TABLE](#) command wait until the table is not being used in any transaction.
- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- [FLUSH TABLES WITH READ LOCK](#) does not block [COMMIT](#) if the server is running without binary logging, which may cause a problem (of consistency between tables) when doing a full backup.
- [ANALYZE TABLE](#), [OPTIMIZE TABLE](#), and [REPAIR TABLE](#) may cause problems on tables for which you are using [INSERT DELAYED](#).
- Performing [LOCK TABLE ...](#) and [FLUSH TABLES ...](#) doesn't guarantee that there isn't a half-finished transaction in progress on the table.
- Replication uses query-level logging: The master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases.

It is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is non-deterministic (generally not a recommended practice, even outside of replication).

For example:

- [CREATE ... SELECT](#) or [INSERT ... SELECT](#) statements that insert zero or [NULL](#) values into an [AUTO_INCREMENT](#) column.
- [DELETE](#) if you are deleting rows from a table that has foreign keys with [ON DELETE CASCADE](#) properties.
- [REPLACE ... SELECT](#), [INSERT IGNORE ... SELECT](#) if you have duplicate key values in the inserted data.

If and only if the preceding queries have no [ORDER BY](#) clause guaranteeing a deterministic order.

For example, for [INSERT ... SELECT](#) with no [ORDER BY](#), the [SELECT](#) may return rows in a different order (which results in a row having different ranks, hence getting a different number in the [AUTO_INCREMENT](#) column), depending on the choices made by the optimizers on the master and slave.

A query is optimized differently on the master and slave only if:

- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use [InnoDB](#) on the master, but [MyISAM](#) on the slave if the slave has less available disk space.)
- MySQL buffer sizes ([key_buffer_size](#), and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using [mysqlbinlog|mysql](#).

The easiest way to avoid this problem is to add an [ORDER BY](#) clause to the aforementioned non-deterministic queries to ensure that the rows are always stored or modified in the same order.

In future MySQL versions, we will automatically add an [ORDER BY](#) clause when needed.

The following issues are known and will be fixed in due time:

- Log filenames are based on the server hostname (if you don't specify a filename with the startup option). You have to use options such as [--log-bin=old_host_name-bin](#) if you change your hostname to something else. Another option is to rename the old files to reflect your hostname change (if these are binary logs, you need to edit the binary log index file and fix the binlog names there as well). See [コマンド オプション](#).
- [mysqlbinlog](#) does not delete temporary files left after a [LOAD DATA INFILE](#) command. See [mysqlbinlog — バイナリ ログファイル処理するためのユーティリティ](#).
- [RENAME](#) doesn't work with [TEMPORARY](#) tables or tables used in a [MERGE](#) table.
- Due to the way table format ([.frm](#)) files are stored, you cannot use character 255 ([CHAR\(255\)](#)) in table names, column names, or enumerations. This is scheduled to be fixed in version 5.1 when we implement new table definition format files.
- When using [SET CHARACTER SET](#), you can't use translated characters in database, table, and column names.
- You can't use [_](#) or [%](#) with [ESCAPE](#) in [LIKE ... ESCAPE](#).
- If you have a [DECIMAL](#) column in which the same number is stored in different formats (for example, [+01.00](#), [1.00](#), [01.00](#)), [GROUP BY](#) may regard each value as a different value.
- You cannot build the server in another directory when using MIT-pthreads. Because this requires changes to MIT-pthreads, we are not likely to fix this. See [MIT-pthreads ノート](#).
- [BLOB](#) and [TEXT](#) values can't reliably be used in [GROUP BY](#), [ORDER BY](#) or [DISTINCT](#). Only the first [max_sort_length](#) bytes are used when comparing [BLOB](#) values in these cases. The default value of [max_sort_length](#) is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with [BIGINT](#) or [DOUBLE](#) (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with [BIGINT](#) precision, [IF](#) and [ELT\(\)](#) with [BIGINT](#) or [DOUBLE](#) precision, and the rest with [DOUBLE](#) precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.

- You can have up to 255 [ENUM](#) and [SET](#) columns in one table.
- In [MIN\(\)](#), [MAX\(\)](#), and other aggregate functions, MySQL currently compares [ENUM](#) and [SET](#) columns by their string value rather than by the string's relative position in the set.
- [mysqld_safe](#) redirects all messages from [mysqld](#) to the [mysqld](#) log. One problem with this is that if you execute [mysqladmin refresh](#) to close and reopen the log, [stdout](#) and [stderr](#) are still redirected to the old log. If you use [--log](#) extensively, you should edit [mysqld_safe](#) to log to [host_name.err](#) instead of [host_name.log](#) so that you can easily reclaim the space for the old log by deleting it and executing [mysqladmin refresh](#).
- In an [UPDATE](#) statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments [KEY](#) by 2, not 1:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following doesn't work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;  
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle [DISTINCT](#) differently when you are using 「hidden」 columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns don't participate in the [DISTINCT](#) comparison. We will probably change this in the future to never compare the hidden columns when executing [DISTINCT](#).

An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads  
WHERE userid = 9 ORDER BY id DESC;
```

and

```
SELECT DISTINCT band_downloads.mp3id  
FROM band_downloads,band_mp3  
WHERE band_downloads.userid = 9  
AND band_mp3.id = band_downloads.mp3id  
ORDER BY band_downloads.id DESC;
```

In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden [id](#) column may differ).

Note that this happens only for queries where that do not have the [ORDER BY](#) columns in the result.

- If you execute a [PROCEDURE](#) on a query that returns an empty set, in some cases the [PROCEDURE](#) does not transform the columns.
- Creation of a table of type [MERGE](#) doesn't check whether the underlying tables are compatible types.
- If you use [ALTER TABLE](#) to add a [UNIQUE](#) index to a table used in a [MERGE](#) table and then add a normal index on the [MERGE](#) table, the key order is different for the tables if there was an old, non-[UNIQUE](#) key in the table. This is because [ALTER TABLE](#) puts [UNIQUE](#) indexes before normal indexes to be able to detect duplicate keys as early as possible.

Server Error Codes and Messages

MySQL programs have access to several types of error information when the server returns an error. For example, the [mysql](#) client program displays errors using the following format:

```
shell> SELECT * FROM no_such_table;  
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

The message displayed contains three types of information:

- A numeric error value ([1146](#)). This number is MySQL-specific and is not portable to other database systems.

- A five-character SQLSTATE value ('42S02'). The values are specified by ANSI SQL and ODBC and are more standardized. Not all MySQL error numbers are mapped to SQLSTATE error codes. The value 'HY000' (general error) is used for unmapped errors.
- A string that provides a textual description of the error.

Server error information comes from the following source files. For details about the way that error information is defined, see the MySQL Internals manual, available at <http://dev.mysql.com/doc/>.

- Error message information is listed in the [share/errmsg.txt](#) file. %d and %s represent numbers and strings, respectively, that are substituted into the Message values when they are displayed.
- The Error values listed in [share/errmsg.txt](#) are used to generate the definitions in the [include/mysqld_error.h](#) and [include/mysqld_ename.h](#) MySQL source files.
- The SQLSTATE values listed in [share/errmsg.txt](#) are used to generate the definitions in the [include/sql_state.h](#) MySQL source file.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: 1000 SQLSTATE: HY000 (ER_HASHCHK)
Message: hashchk
- Error: 1001 SQLSTATE: HY000 (ER_NISAMCHK)
Message: isamchk
- Error: 1002 SQLSTATE: HY000 (ER_NO)
Message: NO
- Error: 1003 SQLSTATE: HY000 (ER_YES)
Message: YES
- Error: 1004 SQLSTATE: HY000 (ER_CANT_CREATE_FILE)
Message: '%s' ファイルが作れません (errno: %d)
- Error: 1005 SQLSTATE: HY000 (ER_CANT_CREATE_TABLE)
Message: '%s' テーブルが作れません.(errno: %d)
- Error: 1006 SQLSTATE: HY000 (ER_CANT_CREATE_DB)
Message: '%s' データベースが作れません (errno: %d)
- Error: 1007 SQLSTATE: HY000 (ER_DB_CREATE_EXISTS)
Message: '%s' データベースが作れません.既にそのデータベースが存在します
- Error: 1008 SQLSTATE: HY000 (ER_DB_DROP_EXISTS)
Message: '%s' データベースを破棄できません. そのデータベースがないのです.
- Error: 1009 SQLSTATE: HY000 (ER_DB_DROP_DELETE)
Message: データベース破棄エラー ('%s' を削除できません, errno: %d)
- Error: 1010 SQLSTATE: HY000 (ER_DB_DROP_RMDIR)
Message: データベース破棄エラー ('%s' を rmdir できません, errno: %d)
- Error: 1011 SQLSTATE: HY000 (ER_CANT_DELETE_FILE)

Message: '%s' の削除がエラー (errno: %d)

- Error: 1012 SQLSTATE: HY000 (ER_CANT_FIND_SYSTEM_REC)

Message: system table のレコードを読む事ができませんでした

- Error: 1013 SQLSTATE: HY000 (ER_CANT_GET_STAT)

Message: '%s' のステータスが得られません. (errno: %d)

- Error: 1014 SQLSTATE: HY000 (ER_CANT_GET_WD)

Message: working directory を得る事ができませんでした (errno: %d)

- Error: 1015 SQLSTATE: HY000 (ER_CANT_LOCK)

Message: ファイルをロックできません (errno: %d)

- Error: 1016 SQLSTATE: HY000 (ER_CANT_OPEN_FILE)

Message: '%s' ファイルを開く事ができません (errno: %d)

- Error: 1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND)

Message: '%s' ファイルを見付ける事ができません.(errno: %d)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

Message: '%s' ディレクトリが読めません.(errno: %d)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

Message: '%s' ディレクトリに chdir できません.(errno: %d)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD)

Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL)

Message: Disk full (%s). 誰かが何かを減らすまでまってください...

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY)

Message: table '%s' に key が重複していて書きこめません

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

Message: Error on close of '%s' (errno: %d)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

Message: '%s' ファイルの読み込みエラー (errno: %d)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

Message: '%s' を '%s' に rename できません (errno: %d)

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)

Message: '%s' ファイルを書く事ができません (errno: %d)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED)

Message: '%s' はロックされています

- Error: **1028** SQLSTATE: **HY000** (**ER_FILSORT_ABORT**)
Message: Sort 中断
- Error: **1029** SQLSTATE: **HY000** (**ER_FORM_NOT_FOUND**)
Message: View '%s' が '%s' に定義されていません
- Error: **1030** SQLSTATE: **HY000** (**ER_GET_ERRNO**)
Message: Got error %d from table handler
- Error: **1031** SQLSTATE: **HY000** (**ER_ILLEGAL HA**)
Message: Table handler for '%s' doesn't have this option
- Error: **1032** SQLSTATE: **HY000** (**ER_KEY_NOT_FOUND**)
Message: '%s'のなかにレコードが見付かりません
- Error: **1033** SQLSTATE: **HY000** (**ER_NOT_FORM_FILE**)
Message: ファイル '%s' の info が間違っているようです
- Error: **1034** SQLSTATE: **HY000** (**ER_NOT_KEYFILE**)
Message: '%s' テーブルの key file が間違っているようです. 修復をしてください
- Error: **1035** SQLSTATE: **HY000** (**ER_OLD_KEYFILE**)
Message: '%s' テーブルは古い形式の key file のようです; 修復をしてください
- Error: **1036** SQLSTATE: **HY000** (**ER_OPEN_AS_READONLY**)
Message: '%s' は読み込み専用です
- Error: **1037** SQLSTATE: **HY001** (**ER_OUTOFMEMORY**)
Message: Out of memory. デーモンをリスタートしてみてください (%d bytes 必要)
- Error: **1038** SQLSTATE: **HY001** (**ER_OUT_OF_SORTMEMORY**)
Message: Out of sort memory. sort buffer size が足りないようです.
- Error: **1039** SQLSTATE: **HY000** (**ER_UNEXPECTED_EOF**)
Message: '%s' ファイルを読み込み中に EOF が予期せぬ所で現れました. (errno: %d)
- Error: **1040** SQLSTATE: **08004** (**ER_CON_COUNT_ERROR**)
Message: 接続が多すぎます
- Error: **1041** SQLSTATE: **HY000** (**ER_OUT_OF_RESOURCES**)
Message: Out of memory; mysqld かその他のプロセスがメモリーを全て使っているか確認してください. メモリーを使い切っていない場合、'ulimit' を設定して mysqld のメモリー使用限界量を多くするか、swap space を増やしてみてください
- Error: **1042** SQLSTATE: **08S01** (**ER_BAD_HOST_ERROR**)
Message: その address の hostname が引けません.
- Error: **1043** SQLSTATE: **08S01** (**ER_HANDSHAKE_ERROR**)
Message: Bad handshake
- Error: **1044** SQLSTATE: **42000** (**ER_DBACCESS_DENIED_ERROR**)

Message: ユーザー '%s'@'%s' の '%s' データベースへのアクセスを拒否します

- Error: [1045](#) SQLSTATE: [28000](#) ([ER_ACCESS_DENIED_ERROR](#))

Message: ユーザー '%s'@'%s' を拒否します.uUsing password: %s)

- Error: [1046](#) SQLSTATE: [3D000](#) ([ER_NO_DB_ERROR](#))

Message: データベースが選択されていません.

- Error: [1047](#) SQLSTATE: [08S01](#) ([ER_UNKNOWN_COM_ERROR](#))

Message: そのコマンドは何？

- Error: [1048](#) SQLSTATE: [23000](#) ([ER_BAD_NULL_ERROR](#))

Message: Column '%s' は null にはできないのです

- Error: [1049](#) SQLSTATE: [42000](#) ([ER_BAD_DB_ERROR](#))

Message: '%s' なんてデータベースは知りません.

- Error: [1050](#) SQLSTATE: [42S01](#) ([ER_TABLE_EXISTS_ERROR](#))

Message: Table '%s' は既にあります

- Error: [1051](#) SQLSTATE: [42S02](#) ([ER_BAD_TABLE_ERROR](#))

Message: table '%s' はありません.

- Error: [1052](#) SQLSTATE: [23000](#) ([ER_NON_UNIQ_ERROR](#))

Message: Column: '%s' in %s is ambiguous

- Error: [1053](#) SQLSTATE: [08S01](#) ([ER_SERVER_SHUTDOWN](#))

Message: Server を shutdown 中...

- Error: [1054](#) SQLSTATE: [42S22](#) ([ER_BAD_FIELD_ERROR](#))

Message: '%s' column は '%s' にはありません.

- Error: [1055](#) SQLSTATE: [42000](#) ([ER_WRONG_FIELD_WITH_GROUP](#))

Message: '%s' isn't in GROUP BY

- Error: [1056](#) SQLSTATE: [42000](#) ([ER_WRONG_GROUP_FIELD](#))

Message: Can't group on '%s'

- Error: [1057](#) SQLSTATE: [42000](#) ([ER_WRONG_SUM_SELECT](#))

Message: Statement has sum functions and columns in same statement

- Error: [1058](#) SQLSTATE: [21S01](#) ([ER_WRONG_VALUE_COUNT](#))

Message: Column count doesn't match value count

- Error: [1059](#) SQLSTATE: [42000](#) ([ER_TOO_LONG_IDENT](#))

Message: Identifier name '%s' は長すぎます

- Error: [1060](#) SQLSTATE: [42S21](#) ([ER_DUP_FIELDNAME](#))

Message: '%s' という column 名は重複しています

- Error: [1061](#) SQLSTATE: [42000](#) ([ER_DUP_KEYNAME](#))
Message: '%s' という key の名前は重複しています
- Error: [1062](#) SQLSTATE: [23000](#) ([ER_DUP_ENTRY](#))
Message: '%s' は key %d において重複しています
- Error: [1063](#) SQLSTATE: [42000](#) ([ER_WRONG_FIELD_SPEC](#))
Message: Incorrect column specifier for column '%s'
- Error: [1064](#) SQLSTATE: [42000](#) ([ER_PARSE_ERROR](#))
Message: %s : '%s' 付近 : %d 行目
- Error: [1065](#) SQLSTATE: [42000](#) ([ER_EMPTY_QUERY](#))
Message: Query が空です.
- Error: [1066](#) SQLSTATE: [42000](#) ([ER_NONUNIQ_TABLE](#))
Message: '%s' は一意の table/alias 名ではありません
- Error: [1067](#) SQLSTATE: [42000](#) ([ER_INVALID_DEFAULT](#))
Message: Invalid default value for '%s'
- Error: [1068](#) SQLSTATE: [42000](#) ([ER_MULTIPLE_PRI_KEY](#))
Message: 複数の primary key が定義されました
- Error: [1069](#) SQLSTATE: [42000](#) ([ER_TOO_MANY_KEYS](#))
Message: key の指定が多すぎます. key は最大 %d までです
- Error: [1070](#) SQLSTATE: [42000](#) ([ER_TOO_MANY_KEY_PARTS](#))
Message: Too many key parts specified; max %d parts allowed
- Error: [1071](#) SQLSTATE: [42000](#) ([ER_TOO_LONG_KEY](#))
Message: key が長すぎます. key の長さは最大 %d です
- Error: [1072](#) SQLSTATE: [42000](#) ([ER_KEY_COLUMN_DOES_NOT_EXISTS](#))
Message: Key column '%s' がテーブルにありません.
- Error: [1073](#) SQLSTATE: [42000](#) ([ER_BLOB_USED_AS_KEY](#))
Message: BLOB column '%s' can't be used in key specification with the used table type
- Error: [1074](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_FIELDLENGTH](#))
Message: column '%s' は、確保する column の大きさが多すぎます. (最大 %lu まで). BLOB をかわりに使用してください.
- Error: [1075](#) SQLSTATE: [42000](#) ([ER_WRONG_AUTO_KEY](#))
Message: テーブルの定義が違います; there can be only one auto column and it must be defined as a key
- Error: [1076](#) SQLSTATE: [HY000](#) ([ER_READY](#))
Message: %s: 準備完了 Version: '%s' socket: '%s' port: %d"
- Error: [1077](#) SQLSTATE: [HY000](#) ([ER_NORMAL_SHUTDOWN](#))

Message: %s: Normal shutdown

- Error: [1078](#) SQLSTATE: [HY000](#) ([ER_GOT_SIGNAL](#))

Message: %s: Got signal %d. 中断!

- Error: [1079](#) SQLSTATE: [HY000](#) ([ER_SHUTDOWN_COMPLETE](#))

Message: %s: Shutdown 完了

- Error: [1080](#) SQLSTATE: [08S01](#) ([ER_FORCING_CLOSE](#))

Message: %s: スレッド %ld 強制終了 user: '%s'

- Error: [1081](#) SQLSTATE: [08S01](#) ([ER_IPSOCK_ERROR](#))

Message: IP socket が作れません

- Error: [1082](#) SQLSTATE: [42S12](#) ([ER_NO_SUCH_INDEX](#))

Message: Table '%s' はそのような index を持っていません(CREATE INDEX 実行時に指定されていません). テーブルを作り直してください

- Error: [1083](#) SQLSTATE: [42000](#) ([ER_WRONG_FIELD_TERMINATORS](#))

Message: Field separator argument is not what is expected; check the manual

- Error: [1084](#) SQLSTATE: [42000](#) ([ER_BLOBS_AND_NO_TERMINATED](#))

Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'.

- Error: [1085](#) SQLSTATE: [HY000](#) ([ER_TEXTFILE_NOT_READABLE](#))

Message: ファイル '%s' は databse の directory にあるか全てのユーザーが読めるように許可されていなければなりません.

- Error: [1086](#) SQLSTATE: [HY000](#) ([ER_FILE_EXISTS_ERROR](#))

Message: File '%s' は既に存在します

- Error: [1087](#) SQLSTATE: [HY000](#) ([ER_LOAD_INFO](#))

Message: レコード数: %ld 削除: %ld Skipped: %ld Warnings: %ld

- Error: [1088](#) SQLSTATE: [HY000](#) ([ER_ALTER_INFO](#))

Message: レコード数: %ld 重複: %ld

- Error: [1089](#) SQLSTATE: [HY000](#) ([ER_WRONG_SUB_KEY](#))

Message: Incorrect prefix key; the used key part isn't a string or the used length is longer than the key part

- Error: [1090](#) SQLSTATE: [42000](#) ([ER_CANT_REMOVE_ALL_FIELDS](#))

Message: ALTER TABLE で全ての column は削除できません. DROP TABLE を使用してください

- Error: [1091](#) SQLSTATE: [42000](#) ([ER_CANT_DROP_FIELD_OR_KEY](#))

Message: '%s' を破棄できませんでした; check that column/key exists

- Error: [1092](#) SQLSTATE: [HY000](#) ([ER_INSERT_INFO](#))

Message: レコード数: %ld 重複数: %ld Warnings: %ld

- Error: [1093](#) SQLSTATE: [HY000](#) ([ER_UPDATE_TABLE_USED](#))

Message: You can't specify target table '%s' for update in FROM clause

- Error: [1094](#) SQLSTATE: [HY000](#) ([ER_NO_SUCH_THREAD](#))

Message: thread id: %lu はありません

- Error: [1095](#) SQLSTATE: [HY000](#) ([ER_KILL_DENIED_ERROR](#))

Message: thread %lu のオーナーではありません

- Error: [1096](#) SQLSTATE: [HY000](#) ([ER_NO_TABLES_USED](#))

Message: No tables used

- Error: [1097](#) SQLSTATE: [HY000](#) ([ER_TOO_BIG_SET](#))

Message: Too many strings for column %s and SET

- Error: [1098](#) SQLSTATE: [HY000](#) ([ER_NO_UNIQUE_LOGFILE](#))

Message: Can't generate a unique log-filename %s.(1-999)

- Error: [1099](#) SQLSTATE: [HY000](#) ([ER_TABLE_NOT_LOCKED_FOR_WRITE](#))

Message: Table '%s' は READ lock になっていて、更新はできません

- Error: [1100](#) SQLSTATE: [HY000](#) ([ER_TABLE_NOT_LOCKED](#))

Message: Table '%s' は LOCK TABLES によってロックされていません

- Error: [1101](#) SQLSTATE: [42000](#) ([ER_BLOB_CANT_HAVE_DEFAULT](#))

Message: BLOB column '%s' can't have a default value

- Error: [1102](#) SQLSTATE: [42000](#) ([ER_WRONG_DB_NAME](#))

Message: 指定した database 名 '%s' が間違っています

- Error: [1103](#) SQLSTATE: [42000](#) ([ER_WRONG_TABLE_NAME](#))

Message: 指定した table 名 '%s' はまちがっています

- Error: [1104](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_SELECT](#))

Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET MAX_JOIN_SIZE=# if the SELECT is okay

- Error: [1105](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_ERROR](#))

Message: Unknown error

- Error: [1106](#) SQLSTATE: [42000](#) ([ER_UNKNOWN_PROCEDURE](#))

Message: Unknown procedure '%s'

- Error: [1107](#) SQLSTATE: [42000](#) ([ER_WRONG_PARAMCOUNT_TO_PROCEDURE](#))

Message: Incorrect parameter count to procedure '%s'

- Error: [1108](#) SQLSTATE: [HY000](#) ([ER_WRONG_PARAMETERS_TO_PROCEDURE](#))

Message: Incorrect parameters to procedure '%s'

- Error: [1109](#) SQLSTATE: [42S02](#) ([ER_UNKNOWN_TABLE](#))

Message: Unknown table '%s' in %s

- Error: [1110](#) SQLSTATE: [42000](#) ([ER_FIELD_SPECIFIED_TWICE](#))
Message: Column '%s' specified twice
- Error: [1111](#) SQLSTATE: [HY000](#) ([ER_INVALID_GROUP_FUNC_USE](#))
Message: Invalid use of group function
- Error: [1112](#) SQLSTATE: [42000](#) ([ER_UNSUPPORTED_EXTENSION](#))
Message: Table '%s' uses an extension that doesn't exist in this MySQL version
- Error: [1113](#) SQLSTATE: [42000](#) ([ER_TABLE_MUST_HAVE_COLUMNS](#))
Message: テーブルは最低 1 個の column が必要です
- Error: [1114](#) SQLSTATE: [HY000](#) ([ER_RECORD_FILE_FULL](#))
Message: table '%s' はいっぱいです
- Error: [1115](#) SQLSTATE: [42000](#) ([ER_UNKNOWN_CHARACTER_SET](#))
Message: character set '%s' はサポートしていません
- Error: [1116](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_TABLES](#))
Message: テーブルが多すぎます; MySQL can only use %d tables in a join
- Error: [1117](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_FIELDS](#))
Message: column が多すぎます
- Error: [1118](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_ROWSIZE](#))
Message: row size が大きすぎます. BLOB を含まない場合の row size の最大は %ld です. いくつかの field を BLOB に変えてください.
- Error: [1119](#) SQLSTATE: [HY000](#) ([ER_STACK_OVERRUN](#))
Message: Thread stack overrun: Used: %ld of a %ld stack. スタック領域を多くとりたい場合、'mysqld -O thread_stack=#' と指定してください
- Error: [1120](#) SQLSTATE: [42000](#) ([ER_WRONG_OUTER_JOIN](#))
Message: Cross dependency found in OUTER JOIN; examine your ON conditions
- Error: [1121](#) SQLSTATE: [42000](#) ([ER_NULL_COLUMN_IN_INDEX](#))
Message: Table handler doesn't support NULL in given index. Please change column '%s' to be NOT NULL or use another handler
- Error: [1122](#) SQLSTATE: [HY000](#) ([ER_CANT_FIND_UDF](#))
Message: function '%s' をロードできません
- Error: [1123](#) SQLSTATE: [HY000](#) ([ER_CANT_INITIALIZE_UDF](#))
Message: function '%s' を初期化できません; %s
- Error: [1124](#) SQLSTATE: [HY000](#) ([ER_UDF_NO_PATHS](#))
Message: shared library へのパスが通っていません
- Error: [1125](#) SQLSTATE: [HY000](#) ([ER_UDF_EXISTS](#))
Message: Function '%s' は既に定義されています

- Error: **1126** SQLSTATE: **HY000** (**ER_CANT_OPEN_LIBRARY**)
Message: shared library '%s' を開く事ができません (errno: %d %s)
- Error: **1127** SQLSTATE: **HY000** (**ER_CANT_FIND_DL_ENTRY**)
Message: function '%s' をライブラリー中に見付ける事ができません
- Error: **1128** SQLSTATE: **HY000** (**ER_FUNCTION_NOT_DEFINED**)
Message: Function '%s' は定義されていません
- Error: **1129** SQLSTATE: **HY000** (**ER_HOST_IS_BLOCKED**)
Message: Host '%s' は many connection error のため、拒否されました. 'mysqladmin flush-hosts' で解除してください
- Error: **1130** SQLSTATE: **HY000** (**ER_HOST_NOT_PRIVILEGED**)
Message: Host '%s' は MySQL server に接続を許可されていません
- Error: **1131** SQLSTATE: **42000** (**ER_PASSWORD_ANONYMOUS_USER**)
Message: MySQL を anonymous users で使用している状態では、パスワードの変更はできません
- Error: **1132** SQLSTATE: **42000** (**ER_PASSWORD_NOT_ALLOWED**)
Message: 他のユーザーのパスワードを変更するためには, mysql データベースに対して update の許可がなければなりません.
- Error: **1133** SQLSTATE: **42000** (**ER_PASSWORD_NO_MATCH**)
Message: Can't find any matching row in the user table
- Error: **1134** SQLSTATE: **HY000** (**ER_UPDATE_INFO**)
Message: 一致数(Rows matched): %ld 変更: %ld Warnings: %ld
- Error: **1135** SQLSTATE: **HY000** (**ER_CANT_CREATE_THREAD**)
Message: 新規にスレッドが作れませんでした (errno %d). もし最大使用許可メモリー数を越えていないのにエラーが発生しているなら, マニュアルの中から 'possible OS-dependent bug' という文字を探してみてください.
- Error: **1136** SQLSTATE: **21S01** (**ER_WRONG_VALUE_COUNT_ON_ROW**)
Message: Column count doesn't match value count at row %ld
- Error: **1137** SQLSTATE: **HY000** (**ER_CANT_REOPEN_TABLE**)
Message: Can't reopen table: '%s'
- Error: **1138** SQLSTATE: **22004** (**ER_INVALID_USE_OF_NULL**)
Message: NULL 値の使用方法が不適切です
- Error: **1139** SQLSTATE: **42000** (**ER_REGEXP_ERROR**)
Message: Got error '%s' from regexp
- Error: **1140** SQLSTATE: **42000** (**ER_MIX_OF_GROUP_FUNC_AND_FIELDS**)
Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause
- Error: **1141** SQLSTATE: **42000** (**ER_NONEXISTING_GRANT**)

Message: ユーザー '%s' (ホスト '%s' のユーザー) は許可されていません

- Error: [1142](#) SQLSTATE: [42000](#) ([ER_TABLEACCESS_DENIED_ERROR](#))

Message: コマンド %s は ユーザー '%s'@'%s', テーブル '%s' に対して許可されていません

- Error: [1143](#) SQLSTATE: [42000](#) ([ER_COLUMNACCESS_DENIED_ERROR](#))

Message: コマンド %s は ユーザー '%s'@'%s' カラム '%s' テーブル '%s' に対して許可されていません

- Error: [1144](#) SQLSTATE: [42000](#) ([ER_ILLEGAL_GRANT_FOR_TABLE](#))

Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used.

- Error: [1145](#) SQLSTATE: [42000](#) ([ER_GRANT_WRONG_HOST_OR_USER](#))

Message: The host or user argument to GRANT is too long

- Error: [1146](#) SQLSTATE: [42S02](#) ([ER_NO_SUCH_TABLE](#))

Message: Table '%s.%s' doesn't exist

- Error: [1147](#) SQLSTATE: [42000](#) ([ER_NONEXISTING_TABLE_GRANT](#))

Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: [1148](#) SQLSTATE: [42000](#) ([ER_NOT_ALLOWED_COMMAND](#))

Message: The used command is not allowed with this MySQL version

- Error: [1149](#) SQLSTATE: [42000](#) ([ER_SYNTAX_ERROR](#))

Message: Something is wrong in your syntax

- Error: [1150](#) SQLSTATE: [HY000](#) ([ER_DELAYED_CANT_CHANGE_LOCK](#))

Message: Delayed insert thread couldn't get requested lock for table %s

- Error: [1151](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_DELAYED_THREADS](#))

Message: Too many delayed threads in use

- Error: [1152](#) SQLSTATE: [08S01](#) ([ER_ABORTING_CONNECTION](#))

Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: [1153](#) SQLSTATE: [08S01](#) ([ER_NET_PACKET_TOO_LARGE](#))

Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: [1154](#) SQLSTATE: [08S01](#) ([ER_NET_READ_ERROR_FROM_PIPE](#))

Message: Got a read error from the connection pipe

- Error: [1155](#) SQLSTATE: [08S01](#) ([ER_NET_FCNTL_ERROR](#))

Message: Got an error from fcntl()

- Error: [1156](#) SQLSTATE: [08S01](#) ([ER_NET_PACKETS_OUT_OF_ORDER](#))

Message: Got packets out of order

- Error: [1157](#) SQLSTATE: [08S01](#) ([ER_NET_UNCOMPRESS_ERROR](#))

Message: Couldn't uncompress communication packet

- Error: **1158** SQLSTATE: **08S01** (**ER_NET_READ_ERROR**)
 Message: Got an error reading communication packets
- Error: **1159** SQLSTATE: **08S01** (**ER_NET_READ_INTERRUPTED**)
 Message: Got timeout reading communication packets
- Error: **1160** SQLSTATE: **08S01** (**ER_NET_ERROR_ON_WRITE**)
 Message: Got an error writing communication packets
- Error: **1161** SQLSTATE: **08S01** (**ER_NET_WRITE_INTERRUPTED**)
 Message: Got timeout writing communication packets
- Error: **1162** SQLSTATE: **42000** (**ER_TOO_LONG_STRING**)
 Message: Result string is longer than 'max_allowed_packet' bytes
- Error: **1163** SQLSTATE: **42000** (**ER_TABLE_CANT_HANDLE_BLOB**)
 Message: The used table type doesn't support BLOB/TEXT columns
- Error: **1164** SQLSTATE: **42000** (**ER_TABLE_CANT_HANDLE_AUTO_INCREMENT**)
 Message: The used table type doesn't support AUTO_INCREMENT columns
- Error: **1165** SQLSTATE: **HY000** (**ER_DELAYED_INSERT_TABLE_LOCKED**)
 Message: INSERT DELAYED can't be used with table '%s', because it is locked with LOCK TABLES
- Error: **1166** SQLSTATE: **42000** (**ER_WRONG_COLUMN_NAME**)
 Message: Incorrect column name '%s'
- Error: **1167** SQLSTATE: **42000** (**ER_WRONG_KEY_COLUMN**)
 Message: The used table handler can't index column '%s'
- Error: **1168** SQLSTATE: **HY000** (**ER_WRONG_MRG_TABLE**)
 Message: All tables in the MERGE table are not defined identically
- Error: **1169** SQLSTATE: **23000** (**ER_DUP_UNIQUE**)
 Message: Can't write, because of unique constraint, to table '%s'
- Error: **1170** SQLSTATE: **42000** (**ER_BLOB_KEY_WITHOUT_LENGTH**)
 Message: BLOB column '%s' used in key specification without a key length
- Error: **1171** SQLSTATE: **42000** (**ER_PRIMARY_CANT_HAVE_NULL**)
 Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead
- Error: **1172** SQLSTATE: **42000** (**ER_TOO_MANY_ROWS**)
 Message: Result consisted of more than one row
- Error: **1173** SQLSTATE: **42000** (**ER_REQUIRES_PRIMARY_KEY**)
 Message: This table type requires a primary key
- Error: **1174** SQLSTATE: **HY000** (**ER_NO_RAID_COMPILED**)
 Message: This version of MySQL is not compiled with RAID support

- Error: **1175** SQLSTATE: **HY000** (**ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE**)
 Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column
- Error: **1176** SQLSTATE: **42000** (**ER_KEY_DOES_NOT_EXISTS**)
 Message: Key '%s' doesn't exist in table '%s'
- Error: **1177** SQLSTATE: **42000** (**ER_CHECK_NO_SUCH_TABLE**)
 Message: Can't open table
- Error: **1178** SQLSTATE: **42000** (**ER_CHECK_NOT_IMPLEMENTED**)
 Message: The handler for the table doesn't support %s
- Error: **1179** SQLSTATE: **25000** (**ER_CANT_DO_THIS_DURING_AN_TRANSACTION**)
 Message: You are not allowed to execute this command in a transaction
- Error: **1180** SQLSTATE: **HY000** (**ER_ERROR_DURING_COMMIT**)
 Message: Got error %d during COMMIT
- Error: **1181** SQLSTATE: **HY000** (**ER_ERROR_DURING_ROLLBACK**)
 Message: Got error %d during ROLLBACK
- Error: **1182** SQLSTATE: **HY000** (**ER_ERROR_DURING_FLUSH_LOGS**)
 Message: Got error %d during FLUSH_LOGS
- Error: **1183** SQLSTATE: **HY000** (**ER_ERROR_DURING_CHECKPOINT**)
 Message: Got error %d during CHECKPOINT
- Error: **1184** SQLSTATE: **08S01** (**ER_NEW_ABORTING_CONNECTION**)
 Message: Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)
- Error: **1185** SQLSTATE: **HY000** (**ER_DUMP_NOT_IMPLEMENTED**)
 Message: The handler for the table does not support binary table dump
- Error: **1186** SQLSTATE: **HY000** (**ER_FLUSH_MASTER_BINLOG_CLOSED**)
 Message: Binlog closed, cannot RESET MASTER
- Error: **1187** SQLSTATE: **HY000** (**ER_INDEX_REBUILD**)
 Message: Failed rebuilding the index of dumped table '%s'
- Error: **1188** SQLSTATE: **HY000** (**ER_MASTER**)
 Message: Error from master: '%s'
- Error: **1189** SQLSTATE: **08S01** (**ER_MASTER_NET_READ**)
 Message: Net error reading from master
- Error: **1190** SQLSTATE: **08S01** (**ER_MASTER_NET_WRITE**)
 Message: Net error writing to master
- Error: **1191** SQLSTATE: **HY000** (**ER_FT_MATCHING_KEY_NOT_FOUND**)

Message: Can't find FULLTEXT index matching the column list

- Error: **1192** SQLSTATE: **HY000** (**ER_LOCK_OR_ACTIVE_TRANSACTION**)

Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: **1193** SQLSTATE: **HY000** (**ER_UNKNOWN_SYSTEM_VARIABLE**)

Message: Unknown system variable '%s'

- Error: **1194** SQLSTATE: **HY000** (**ER_CRASHED_ON_USAGE**)

Message: Table '%s' is marked as crashed and should be repaired

- Error: **1195** SQLSTATE: **HY000** (**ER_CRASHED_ON_REPAIR**)

Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: **1196** SQLSTATE: **HY000** (**ER_WARNING_NOT_COMPLETE_ROLLBACK**)

Message: Some non-transactional changed tables couldn't be rolled back

- Error: **1197** SQLSTATE: **HY000** (**ER_TRANS_CACHE_FULL**)

Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: **1198** SQLSTATE: **HY000** (**ER_SLAVE_MUST_STOP**)

Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: **1199** SQLSTATE: **HY000** (**ER_SLAVE_NOT_RUNNING**)

Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: **1200** SQLSTATE: **HY000** (**ER_BAD_SLAVE**)

Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: **1201** SQLSTATE: **HY000** (**ER_MASTER_INFO**)

Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: **1202** SQLSTATE: **HY000** (**ER_SLAVE_THREAD**)

Message: Could not create slave thread; check system resources

- Error: **1203** SQLSTATE: **42000** (**ER_TOO_MANY_USER_CONNECTIONS**)

Message: User %s already has more than 'max_user_connections' active connections

- Error: **1204** SQLSTATE: **HY000** (**ER_SET_CONSTANTS_ONLY**)

Message: You may only use constant expressions with SET

- Error: **1205** SQLSTATE: **HY000** (**ER_LOCK_WAIT_TIMEOUT**)

Message: Lock wait timeout exceeded; try restarting transaction

- Error: **1206** SQLSTATE: **HY000** (**ER_LOCK_TABLE_FULL**)

Message: The total number of locks exceeds the lock table size

- Error: **1207** SQLSTATE: **25000** (**ER_READ_ONLY_TRANSACTION**)

Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: [1208](#) SQLSTATE: [HY000](#) ([ER_DROP_DB_WITH_READ_LOCK](#))
Message: DROP DATABASE not allowed while thread is holding global read lock
- Error: [1209](#) SQLSTATE: [HY000](#) ([ER_CREATE_DB_WITH_READ_LOCK](#))
Message: CREATE DATABASE not allowed while thread is holding global read lock
- Error: [1210](#) SQLSTATE: [HY000](#) ([ER_WRONG_ARGUMENTS](#))
Message: Incorrect arguments to %s
- Error: [1211](#) SQLSTATE: [42000](#) ([ER_NO_PERMISSION_TO_CREATE_USER](#))
Message: '%s'@'%s' is not allowed to create new users
- Error: [1212](#) SQLSTATE: [HY000](#) ([ER_UNION_TABLES_IN_DIFFERENT_DIR](#))
Message: Incorrect table definition; all MERGE tables must be in the same database
- Error: [1213](#) SQLSTATE: [40001](#) ([ER_LOCK_DEADLOCK](#))
Message: Deadlock found when trying to get lock; try restarting transaction
- Error: [1214](#) SQLSTATE: [HY000](#) ([ER_TABLE_CANT_HANDLE_FT](#))
Message: The used table type doesn't support FULLTEXT indexes
- Error: [1215](#) SQLSTATE: [HY000](#) ([ER_CANNOT_ADD_FOREIGN](#))
Message: Cannot add foreign key constraint
- Error: [1216](#) SQLSTATE: [23000](#) ([ER_NO_REFERENCED_ROW](#))
Message: Cannot add or update a child row: a foreign key constraint fails
- Error: [1217](#) SQLSTATE: [23000](#) ([ER_ROW_IS_REFERENCED](#))
Message: Cannot delete or update a parent row: a foreign key constraint fails
- Error: [1218](#) SQLSTATE: [08S01](#) ([ER_CONNECT_TO_MASTER](#))
Message: Error connecting to master: %s
- Error: [1219](#) SQLSTATE: [HY000](#) ([ER_QUERY_ON_MASTER](#))
Message: Error running query on master: %s
- Error: [1220](#) SQLSTATE: [HY000](#) ([ER_ERROR_WHEN_EXECUTING_COMMAND](#))
Message: Error when executing command %s: %s
- Error: [1221](#) SQLSTATE: [HY000](#) ([ER_WRONG_USAGE](#))
Message: Incorrect usage of %s and %s
- Error: [1222](#) SQLSTATE: [21000](#) ([ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT](#))
Message: The used SELECT statements have a different number of columns
- Error: [1223](#) SQLSTATE: [HY000](#) ([ER_CANT_UPDATE_WITH_READLOCK](#))
Message: Can't execute the query because you have a conflicting read lock
- Error: [1224](#) SQLSTATE: [HY000](#) ([ER_MIXING_NOT_ALLOWED](#))
Message: Mixing of transactional and non-transactional tables is disabled

- Error: [1225](#) SQLSTATE: [HY000](#) ([ER_DUP_ARGUMENT](#))
Message: Option '%s' used twice in statement
- Error: [1226](#) SQLSTATE: [42000](#) ([ER_USER_LIMIT_REACHED](#))
Message: User '%s' has exceeded the '%s' resource (current value: %ld)
- Error: [1227](#) SQLSTATE: [42000](#) ([ER_SPECIFIC_ACCESS_DENIED_ERROR](#))
Message: Access denied; you need the %s privilege for this operation
- Error: [1228](#) SQLSTATE: [HY000](#) ([ER_LOCAL_VARIABLE](#))
Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL
- Error: [1229](#) SQLSTATE: [HY000](#) ([ER_GLOBAL_VARIABLE](#))
Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL
- Error: [1230](#) SQLSTATE: [42000](#) ([ER_NO_DEFAULT](#))
Message: Variable '%s' doesn't have a default value
- Error: [1231](#) SQLSTATE: [42000](#) ([ER_WRONG_VALUE_FOR_VAR](#))
Message: Variable '%s' can't be set to the value of '%s'
- Error: [1232](#) SQLSTATE: [42000](#) ([ER_WRONG_TYPE_FOR_VAR](#))
Message: Incorrect argument type to variable '%s'
- Error: [1233](#) SQLSTATE: [HY000](#) ([ER_VAR_CANT_BE_READ](#))
Message: Variable '%s' can only be set, not read
- Error: [1234](#) SQLSTATE: [42000](#) ([ER_CANT_USE_OPTION_HERE](#))
Message: Incorrect usage/placement of '%s'
- Error: [1235](#) SQLSTATE: [42000](#) ([ER_NOT_SUPPORTED_YET](#))
Message: This version of MySQL doesn't yet support '%s'
- Error: [1236](#) SQLSTATE: [HY000](#) ([ER_MASTER_FATAL_ERROR_READING_BINLOG](#))
Message: Got fatal error %d from master when reading data from binary log: '%s'
- Error: [1237](#) SQLSTATE: [HY000](#) ([ER_SLAVE_IGNORED_TABLE](#))
Message: Slave SQL thread ignored the query because of replicate-*-table rules
- Error: [1238](#) SQLSTATE: [HY000](#) ([ER_INCORRECT_GLOBAL_LOCAL_VAR](#))
Message: Variable '%s' is a %s variable
- Error: [1239](#) SQLSTATE: [42000](#) ([ER_WRONG_FK_DEF](#))
Message: Incorrect foreign key definition for '%s': %s
- Error: [1240](#) SQLSTATE: [HY000](#) ([ER_KEY_REF_DO_NOT_MATCH_TABLE_REF](#))
Message: Key reference and table reference don't match
- Error: [1241](#) SQLSTATE: [21000](#) ([ER_OPERAND_COLUMNS](#))
Message: Operand should contain %d column(s)

- Error: [1242](#) SQLSTATE: [21000](#) ([ER_SUBQUERY_NO_1_ROW](#))
Message: Subquery returns more than 1 row
- Error: [1243](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_STMT_HANDLER](#))
Message: Unknown prepared statement handler (%.*s) given to %s
- Error: [1244](#) SQLSTATE: [HY000](#) ([ER_CORRUPT_HELP_DB](#))
Message: Help database is corrupt or does not exist
- Error: [1245](#) SQLSTATE: [HY000](#) ([ER_CYCLIC_REFERENCE](#))
Message: Cyclic reference on subqueries
- Error: [1246](#) SQLSTATE: [HY000](#) ([ER_AUTO_CONVERT](#))
Message: Converting column '%s' from %s to %s
- Error: [1247](#) SQLSTATE: [42S22](#) ([ER_ILLEGAL_REFERENCE](#))
Message: Reference '%s' not supported (%s)
- Error: [1248](#) SQLSTATE: [42000](#) ([ER_DERIVED_MUST_HAVE_ALIAS](#))
Message: Every derived table must have its own alias
- Error: [1249](#) SQLSTATE: [01000](#) ([ER_SELECT_REDUCED](#))
Message: Select %u was reduced during optimization
- Error: [1250](#) SQLSTATE: [42000](#) ([ER_TABLENAME_NOT_ALLOWED_HERE](#))
Message: Table '%s' from one of the SELECTs cannot be used in %s
- Error: [1251](#) SQLSTATE: [08004](#) ([ER_NOT_SUPPORTED_AUTH_MODE](#))
Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client
- Error: [1252](#) SQLSTATE: [42000](#) ([ER_SPATIAL_CANT_HAVE_NULL](#))
Message: All parts of a SPATIAL index must be NOT NULL
- Error: [1253](#) SQLSTATE: [42000](#) ([ER_COLLATION_CHARSET_MISMATCH](#))
Message: COLLATION '%s' is not valid for CHARACTER SET '%s'
- Error: [1254](#) SQLSTATE: [HY000](#) ([ER_SLAVE_WAS_RUNNING](#))
Message: Slave is already running
- Error: [1255](#) SQLSTATE: [HY000](#) ([ER_SLAVE_WAS_NOT_RUNNING](#))
Message: Slave already has been stopped
- Error: [1256](#) SQLSTATE: [HY000](#) ([ER_TOO_BIG_FOR_UNCOMPRESS](#))
Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)
- Error: [1257](#) SQLSTATE: [HY000](#) ([ER_ZLIB_Z_MEM_ERROR](#))
Message: ZLIB: Not enough memory
- Error: [1258](#) SQLSTATE: [HY000](#) ([ER_ZLIB_Z_BUF_ERROR](#))

Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: [1259](#) SQLSTATE: [HY000](#) ([ER_ZLIB_Z_DATA_ERROR](#))

Message: ZLIB: Input data corrupted

- Error: [1260](#) SQLSTATE: [HY000](#) ([ER_CUT_VALUE_GROUP_CONCAT](#))

Message: %d line(s) were cut by GROUP_CONCAT()

- Error: [1261](#) SQLSTATE: [01000](#) ([ER_WARN_TOO_FEW_RECORDS](#))

Message: Row %ld doesn't contain data for all columns

- Error: [1262](#) SQLSTATE: [01000](#) ([ER_WARN_TOO_MANY_RECORDS](#))

Message: Row %ld was truncated; it contained more data than there were input columns

- Error: [1263](#) SQLSTATE: [22004](#) ([ER_WARN_NULL_TO_NOTNULL](#))

Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld

- Error: [1264](#) SQLSTATE: [22003](#) ([ER_WARN_DATA_OUT_OF_RANGE](#))

Message: Out of range value for column '%s' at row %ld

- Error: [1265](#) SQLSTATE: [01000](#) ([WARN_DATA_TRUNCATED](#))

Message: Data truncated for column '%s' at row %ld

- Error: [1266](#) SQLSTATE: [HY000](#) ([ER_WARN_USING_OTHER_HANDLER](#))

Message: Using storage engine %s for table '%s'

- Error: [1267](#) SQLSTATE: [HY000](#) ([ER_CANT_AGGREGATE_2COLLATIONS](#))

Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: [1268](#) SQLSTATE: [HY000](#) ([ER_DROP_USER](#))

Message: Cannot drop one or more of the requested users

- Error: [1269](#) SQLSTATE: [HY000](#) ([ER_REVOKE_GRANTS](#))

Message: Can't revoke all privileges for one or more of the requested users

- Error: [1270](#) SQLSTATE: [HY000](#) ([ER_CANT_AGGREGATE_3COLLATIONS](#))

Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: [1271](#) SQLSTATE: [HY000](#) ([ER_CANT_AGGREGATE_NCOLLATIONS](#))

Message: Illegal mix of collations for operation '%s'

- Error: [1272](#) SQLSTATE: [HY000](#) ([ER_VARIABLE_IS_NOT_STRUCT](#))

Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: [1273](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_COLLATION](#))

Message: Unknown collation: '%s'

- Error: [1274](#) SQLSTATE: [HY000](#) ([ER_SLAVE_IGNORED_SSL_PARAMS](#))

Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)
 Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format
- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)
 Message: Field or reference '%s%s%s%s%s' of SELECT #d was resolved in SELECT #d
- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)
 Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL
- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)
 Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart
- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)
 Message: SQL thread is not to be started so UNTIL options are ignored
- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)
 Message: Incorrect index name '%s'
- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)
 Message: Incorrect catalog name '%s'
- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)
 Message: Query cache failed to set size %lu; new query cache size is %lu
- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)
 Message: Column '%s' cannot be part of FULLTEXT index
- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)
 Message: Unknown key cache '%s'
- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)
 Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work
- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)
 Message: Unknown table engine '%s'
- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)
 Message: '%s' is deprecated and will be removed in a future release. Please use %s instead
- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)
 Message: The target table %s of the %s is not updatable
- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)
 Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working
- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)
 Message: The MySQL server is running with the %s option so it cannot execute this statement
- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

Message: Column '%s' has duplicated value '%s' in %s

- Error: [1292](#) SQLSTATE: [22007](#) ([ER_TRUNCATED_WRONG_VALUE](#))

Message: Truncated incorrect %s value: '%s'

- Error: [1293](#) SQLSTATE: [HY000](#) ([ER_TOO_MUCH_AUTO_TIMESTAMP_COLS](#))

Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: [1294](#) SQLSTATE: [HY000](#) ([ER_INVALID_ON_UPDATE](#))

Message: Invalid ON UPDATE clause for '%s' column

- Error: [1295](#) SQLSTATE: [HY000](#) ([ER_UNSUPPORTED_PS](#))

Message: This command is not supported in the prepared statement protocol yet

- Error: [1296](#) SQLSTATE: [HY000](#) ([ER_GET_ERRMSG](#))

Message: Got error %d '%s' from %s

- Error: [1297](#) SQLSTATE: [HY000](#) ([ER_GET_TEMPORARY_ERRMSG](#))

Message: Got temporary error %d '%s' from %s

- Error: [1298](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_TIME_ZONE](#))

Message: Unknown or incorrect time zone: '%s'

- Error: [1299](#) SQLSTATE: [HY000](#) ([ER_WARN_INVALID_TIMESTAMP](#))

Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: [1300](#) SQLSTATE: [HY000](#) ([ER_INVALID_CHARACTER_STRING](#))

Message: Invalid %s character string: '%s'

- Error: [1301](#) SQLSTATE: [HY000](#) ([ER_WARN_ALLOWED_PACKET_OVERFLOWED](#))

Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: [1302](#) SQLSTATE: [HY000](#) ([ER_CONFLICTING_DECLARATIONS](#))

Message: Conflicting declarations: '%s%s' and '%s%s'

- Error: [1303](#) SQLSTATE: [2F003](#) ([ER_SP_NO_RECURSIVE_CREATE](#))

Message: Can't create a %s from within another stored routine

- Error: [1304](#) SQLSTATE: [42000](#) ([ER_SP_ALREADY_EXISTS](#))

Message: %s %s already exists

- Error: [1305](#) SQLSTATE: [42000](#) ([ER_SP_DOES_NOT_EXIST](#))

Message: %s %s does not exist

- Error: [1306](#) SQLSTATE: [HY000](#) ([ER_SP_DROP_FAILED](#))

Message: Failed to DROP %s %s

- Error: [1307](#) SQLSTATE: [HY000](#) ([ER_SP_STORE_FAILED](#))

Message: Failed to CREATE %s %s

- Error: [1308](#) SQLSTATE: [42000](#) ([ER_SP_LILABEL_MISMATCH](#))
 Message: %s with no matching label: %s
- Error: [1309](#) SQLSTATE: [42000](#) ([ER_SP_LABEL_REDEFINE](#))
 Message: Redefining label %s
- Error: [1310](#) SQLSTATE: [42000](#) ([ER_SP_LABEL_MISMATCH](#))
 Message: End-label %s without match
- Error: [1311](#) SQLSTATE: [01000](#) ([ER_SP_UNINIT_VAR](#))
 Message: Referring to uninitialized variable %s
- Error: [1312](#) SQLSTATE: [0A000](#) ([ER_SP_BADSELECT](#))
 Message: PROCEDURE %s can't return a result set in the given context
- Error: [1313](#) SQLSTATE: [42000](#) ([ER_SP_BADRETURN](#))
 Message: RETURN is only allowed in a FUNCTION
- Error: [1314](#) SQLSTATE: [0A000](#) ([ER_SP_BADSTATEMENT](#))
 Message: %s is not allowed in stored procedures
- Error: [1315](#) SQLSTATE: [42000](#) ([ER_UPDATE_LOG_DEPRECATED_IGNORED](#))
 Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored. This option will be removed in MySQL 5.6.
- Error: [1316](#) SQLSTATE: [42000](#) ([ER_UPDATE_LOG_DEPRECATED_TRANSLATED](#))
 Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN. This option will be removed in MySQL 5.6.
- Error: [1317](#) SQLSTATE: [70100](#) ([ER_QUERY_INTERRUPTED](#))
 Message: Query execution was interrupted
- Error: [1318](#) SQLSTATE: [42000](#) ([ER_SP_WRONG_NO_OF_ARGS](#))
 Message: Incorrect number of arguments for %s %s; expected %u, got %u
- Error: [1319](#) SQLSTATE: [42000](#) ([ER_SP_COND_MISMATCH](#))
 Message: Undefined CONDITION: %s
- Error: [1320](#) SQLSTATE: [42000](#) ([ER_SP_NORETURN](#))
 Message: No RETURN found in FUNCTION %s
- Error: [1321](#) SQLSTATE: [2F005](#) ([ER_SP_NORETURNEND](#))
 Message: FUNCTION %s ended without RETURN
- Error: [1322](#) SQLSTATE: [42000](#) ([ER_SP_BAD_CURSOR_QUERY](#))
 Message: Cursor statement must be a SELECT
- Error: [1323](#) SQLSTATE: [42000](#) ([ER_SP_BAD_CURSOR_SELECT](#))
 Message: Cursor SELECT must not have INTO
- Error: [1324](#) SQLSTATE: [42000](#) ([ER_SP_CURSOR_MISMATCH](#))

Message: Undefined CURSOR: %s

- Error: 1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN)

Message: Cursor is already open

- Error: 1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN)

Message: Cursor is not open

- Error: 1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR)

Message: Undeclared variable: %s

- Error: 1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS)

Message: Incorrect number of FETCH variables

- Error: 1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)

Message: No data - zero rows fetched, selected, or processed

- Error: 1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM)

Message: Duplicate parameter: %s

- Error: 1331 SQLSTATE: 42000 (ER_SP_DUP_VAR)

Message: Duplicate variable: %s

- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

Message: Duplicate condition: %s

- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

Message: Duplicate cursor: %s

- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

Message: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

Message: Subquery value not supported

- Error: 1336 SQLSTATE: 0A000 (ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG)

Message: %s is not allowed in stored function or trigger

- Error: 1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

Message: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

Message: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

Message: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

Message: Configuration file '%s' is too big

- Error: [1341](#) SQLSTATE: [HY000](#) ([ER_FPARSER_BAD_HEADER](#))
 Message: Malformed file type header in file '%s'
- Error: [1342](#) SQLSTATE: [HY000](#) ([ER_FPARSER_EOF_IN_COMMENT](#))
 Message: Unexpected end of file while parsing comment '%s'
- Error: [1343](#) SQLSTATE: [HY000](#) ([ER_FPARSER_ERROR_IN_PARAMETER](#))
 Message: Error while parsing parameter '%s' (line: '%s')
- Error: [1344](#) SQLSTATE: [HY000](#) ([ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER](#))
 Message: Unexpected end of file while skipping unknown parameter '%s'
- Error: [1345](#) SQLSTATE: [HY000](#) ([ER_VIEW_NO_EXPLAIN](#))
 Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table
- Error: [1346](#) SQLSTATE: [HY000](#) ([ER_FRM_UNKNOWN_TYPE](#))
 Message: File '%s' has unknown type '%s' in its header
- Error: [1347](#) SQLSTATE: [HY000](#) ([ER_WRONG_OBJECT](#))
 Message: '%s.%s' is not %s
- Error: [1348](#) SQLSTATE: [HY000](#) ([ER_NONUPDATEABLE_COLUMN](#))
 Message: Column '%s' is not updatable
- Error: [1349](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_DERIVED](#))
 Message: View's SELECT contains a subquery in the FROM clause
- Error: [1350](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_CLAUSE](#))
 Message: View's SELECT contains a '%s' clause
- Error: [1351](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_VARIABLE](#))
 Message: View's SELECT contains a variable or parameter
- Error: [1352](#) SQLSTATE: [HY000](#) ([ER_VIEW_SELECT_TMPTABLE](#))
 Message: View's SELECT refers to a temporary table '%s'
- Error: [1353](#) SQLSTATE: [HY000](#) ([ER_VIEW_WRONG_LIST](#))
 Message: View's SELECT and view's field list have different column counts
- Error: [1354](#) SQLSTATE: [HY000](#) ([ER_WARN_VIEW_MERGE](#))
 Message: View merge algorithm can't be used here for now (assumed undefined algorithm)
- Error: [1355](#) SQLSTATE: [HY000](#) ([ER_WARN_VIEW_WITHOUT_KEY](#))
 Message: View being updated does not have complete key of underlying table in it
- Error: [1356](#) SQLSTATE: [HY000](#) ([ER_VIEW_INVALID](#))
 Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer invoker of view lack rights to use them
- Error: [1357](#) SQLSTATE: [HY000](#) ([ER_SP_NO_DROP_SP](#))

Message: Can't drop or alter a %s from within another stored routine

- Error: [1358](#) SQLSTATE: [HY000](#) ([ER_SP_GOTO_IN_HNDLR](#))

Message: GOTO is not allowed in a stored procedure handler

- Error: [1359](#) SQLSTATE: [HY000](#) ([ER_TRG_ALREADY_EXISTS](#))

Message: Trigger already exists

- Error: [1360](#) SQLSTATE: [HY000](#) ([ER_TRG_DOES_NOT_EXIST](#))

Message: Trigger does not exist

- Error: [1361](#) SQLSTATE: [HY000](#) ([ER_TRG_ON_VIEW_OR_TEMP_TABLE](#))

Message: Trigger's '%s' is view or temporary table

- Error: [1362](#) SQLSTATE: [HY000](#) ([ER_TRG_CANT_CHANGE_ROW](#))

Message: Updating of %s row is not allowed in %strigger

- Error: [1363](#) SQLSTATE: [HY000](#) ([ER_TRG_NO_SUCH_ROW_IN_TRG](#))

Message: There is no %s row in %s trigger

- Error: [1364](#) SQLSTATE: [HY000](#) ([ER_NO_DEFAULT_FOR_FIELD](#))

Message: Field '%s' doesn't have a default value

- Error: [1365](#) SQLSTATE: [22012](#) ([ER_DIVISION_BY_ZERO](#))

Message: Division by 0

- Error: [1366](#) SQLSTATE: [HY000](#) ([ER_TRUNCATED_WRONG_VALUE_FOR_FIELD](#))

Message: Incorrect %s value: '%s' for column '%s' at row %ld

- Error: [1367](#) SQLSTATE: [22007](#) ([ER_ILLEGAL_VALUE_FOR_TYPE](#))

Message: Illegal %s '%s' value found during parsing

- Error: [1368](#) SQLSTATE: [HY000](#) ([ER_VIEW_NONUPD_CHECK](#))

Message: CHECK OPTION on non-updatable view '%s.%s'

- Error: [1369](#) SQLSTATE: [HY000](#) ([ER_VIEW_CHECK_FAILED](#))

Message: CHECK OPTION failed '%s.%s'

- Error: [1370](#) SQLSTATE: [42000](#) ([ER_PROCAccess_DENIED_ERROR](#))

Message: %s command denied to user '%s'@'%s' for routine '%s'

- Error: [1371](#) SQLSTATE: [HY000](#) ([ER_RELAY_LOG_FAIL](#))

Message: Failed purging old relay logs: %s

- Error: [1372](#) SQLSTATE: [HY000](#) ([ER_PASSWD_LENGTH](#))

Message: Password hash should be a %d-digit hexadecimal number

- Error: [1373](#) SQLSTATE: [HY000](#) ([ER_UNKNOWN_TARGET_BINLOG](#))

Message: Target log not found in binlog index

- Error: [1374](#) SQLSTATE: [HY000](#) ([ER_IO_ERR_LOG_INDEX_READ](#))
Message: I/O error reading log index file
- Error: [1375](#) SQLSTATE: [HY000](#) ([ER_BINLOG_PURGE_PROHIBITED](#))
Message: Server configuration does not permit binlog purge
- Error: [1376](#) SQLSTATE: [HY000](#) ([ER_FSEEK_FAIL](#))
Message: Failed on fseek()
- Error: [1377](#) SQLSTATE: [HY000](#) ([ER_BINLOG_PURGE_FATAL_ERR](#))
Message: Fatal error during log purge
- Error: [1378](#) SQLSTATE: [HY000](#) ([ER_LOG_IN_USE](#))
Message: A purgeable log is in use, will not purge
- Error: [1379](#) SQLSTATE: [HY000](#) ([ER_LOG_PURGE_UNKNOWN_ERR](#))
Message: Unknown error during log purge
- Error: [1380](#) SQLSTATE: [HY000](#) ([ER_RELAY_LOG_INIT](#))
Message: Failed initializing relay log position: %s
- Error: [1381](#) SQLSTATE: [HY000](#) ([ER_NO_BINARY_LOGGING](#))
Message: You are not using binary logging
- Error: [1382](#) SQLSTATE: [HY000](#) ([ER_RESERVED_SYNTAX](#))
Message: The '%s' syntax is reserved for purposes internal to the MySQL server
- Error: [1383](#) SQLSTATE: [HY000](#) ([ER_WSAS_FAILED](#))
Message: WSASStartup Failed
- Error: [1384](#) SQLSTATE: [HY000](#) ([ER_DIFF_GROUPS_PROC](#))
Message: Can't handle procedures with different groups yet
- Error: [1385](#) SQLSTATE: [HY000](#) ([ER_NO_GROUP_FOR_PROC](#))
Message: Select must have a group with this procedure
- Error: [1386](#) SQLSTATE: [HY000](#) ([ER_ORDER_WITH_PROC](#))
Message: Can't use ORDER clause with this procedure
- Error: [1387](#) SQLSTATE: [HY000](#) ([ER_LOGGING_PROHIBIT_CHANGING_OF](#))
Message: Binary logging and replication forbid changing the global server %s
- Error: [1388](#) SQLSTATE: [HY000](#) ([ER_NO_FILE_MAPPING](#))
Message: Can't map file: %s, errno: %d
- Error: [1389](#) SQLSTATE: [HY000](#) ([ER_WRONG_MAGIC](#))
Message: Wrong magic in %s
- Error: [1390](#) SQLSTATE: [HY000](#) ([ER_PS_MANY_PARAM](#))
Message: Prepared statement contains too many placeholders

- Error: [1391](#) SQLSTATE: [HY000](#) ([ER_KEY_PART_0](#))
Message: Key part '%s' length cannot be 0
- Error: [1392](#) SQLSTATE: [HY000](#) ([ER_VIEW_CHECKSUM](#))
Message: View text checksum failed
- Error: [1393](#) SQLSTATE: [HY000](#) ([ER_VIEW_MULTIUPDATE](#))
Message: Can not modify more than one base table through a join view '%s.%s'
- Error: [1394](#) SQLSTATE: [HY000](#) ([ER_VIEW_NO_INSERT_FIELD_LIST](#))
Message: Can not insert into join view '%s.%s' without fields list
- Error: [1395](#) SQLSTATE: [HY000](#) ([ER_VIEW_DELETE_MERGE_VIEW](#))
Message: Can not delete from join view '%s.%s'
- Error: [1396](#) SQLSTATE: [HY000](#) ([ER_CANNOT_USER](#))
Message: Operation %s failed for %s
- Error: [1397](#) SQLSTATE: [XAE04](#) ([ER_XAER_NOTA](#))
Message: XAER_NOTA: Unknown XID
- Error: [1398](#) SQLSTATE: [XAE05](#) ([ER_XAER_INVALID](#))
Message: XAER_INVALID: Invalid arguments (or unsupported command)
- Error: [1399](#) SQLSTATE: [XAE07](#) ([ER_XAER_RMFAIL](#))
Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state
- Error: [1400](#) SQLSTATE: [XAE09](#) ([ER_XAER_OUTSIDE](#))
Message: XAER_OUTSIDE: Some work is done outside global transaction
- Error: [1401](#) SQLSTATE: [XAE03](#) ([ER_XAER_RMERR](#))
Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency
- Error: [1402](#) SQLSTATE: [XA100](#) ([ER_XA_RBROLLBACK](#))
Message: XA_RBROLLBACK: Transaction branch was rolled back
- Error: [1403](#) SQLSTATE: [42000](#) ([ER_NONEXISTING_PROC_GRANT](#))
Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'
- Error: [1404](#) SQLSTATE: [HY000](#) ([ER_PROC_AUTO_GRANT_FAIL](#))
Message: Failed to grant EXECUTE and ALTER ROUTINE privileges
- Error: [1405](#) SQLSTATE: [HY000](#) ([ER_PROC_AUTO_REVOKE_FAIL](#))
Message: Failed to revoke all privileges to dropped routine
- Error: [1406](#) SQLSTATE: [22001](#) ([ER_DATA_TOO_LONG](#))
Message: Data too long for column '%s' at row %ld
- Error: [1407](#) SQLSTATE: [42000](#) ([ER_SP_BAD_SQLSTATE](#))
Message: Bad SQLSTATE: '%s'

- Error: [1408](#) SQLSTATE: [HY000](#) ([ER_STARTUP](#))
 Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s
- Error: [1409](#) SQLSTATE: [HY000](#) ([ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR](#))
 Message: Can't load value from file with fixed size rows to variable
- Error: [1410](#) SQLSTATE: [42000](#) ([ER_CANT_CREATE_USER_WITH_GRANT](#))
 Message: You are not allowed to create a user with GRANT
- Error: [1411](#) SQLSTATE: [HY000](#) ([ER_WRONG_VALUE_FOR_TYPE](#))
 Message: Incorrect %s value: '%s' for function %s
- Error: [1412](#) SQLSTATE: [HY000](#) ([ER_TABLE_DEF_CHANGED](#))
 Message: Table definition has changed, please retry transaction
- Error: [1413](#) SQLSTATE: [42000](#) ([ER_SP_DUP_HANDLER](#))
 Message: Duplicate handler declared in the same block
- Error: [1414](#) SQLSTATE: [42000](#) ([ER_SP_NOT_VAR_ARG](#))
 Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger
- Error: [1415](#) SQLSTATE: [0A000](#) ([ER_SP_NO_RETSER](#))
 Message: Not allowed to return a result set from a %s
- Error: [1416](#) SQLSTATE: [22003](#) ([ER_CANT_CREATE_GEOMETRY_OBJECT](#))
 Message: Cannot get geometry object from data you send to the GEOMETRY field
- Error: [1417](#) SQLSTATE: [HY000](#) ([ER_FAILED_ROUTINE_BREAK_BINLOG](#))
 Message: A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes
- Error: [1418](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_ROUTINE](#))
 Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
- Error: [1419](#) SQLSTATE: [HY000](#) ([ER_BINLOG_CREATE_ROUTINE_NEED_SUPER](#))
 Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
- Error: [1420](#) SQLSTATE: [HY000](#) ([ER_EXEC_STMT_WITH_OPEN_CURSOR](#))
 Message: You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.
- Error: [1421](#) SQLSTATE: [HY000](#) ([ER_STMT_HAS_NO_OPEN_CURSOR](#))
 Message: The statement (%lu) has no open cursor.
- Error: [1422](#) SQLSTATE: [HY000](#) ([ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG](#))
 Message: Explicit or implicit commit is not allowed in stored function or trigger.
- Error: [1423](#) SQLSTATE: [HY000](#) ([ER_NO_DEFAULT_FOR_VIEW_FIELD](#))

Message: Field of view '%s.%s' underlying table doesn't have a default value

- Error: [1424](#) SQLSTATE: [HY000](#) ([ER_SP_NO_RECURSION](#))

Message: Recursive stored functions and triggers are not allowed.

- Error: [1425](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_SCALE](#))

Message: Too big scale %d specified for column '%s'. Maximum is %lu.

- Error: [1426](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_PRECISION](#))

Message: Too big precision %d specified for column '%s'. Maximum is %lu.

- Error: [1427](#) SQLSTATE: [42000](#) ([ER_M_BIGGER_THAN_D](#))

Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

- Error: [1428](#) SQLSTATE: [HY000](#) ([ER_WRONG_LOCK_OF_SYSTEM_TABLE](#))

Message: You can't combine write-locking of system tables with other tables or lock types

- Error: [1429](#) SQLSTATE: [HY000](#) ([ER_CONNECT_TO_FOREIGN_DATA_SOURCE](#))

Message: Unable to connect to foreign data source: %s

- Error: [1430](#) SQLSTATE: [HY000](#) ([ER_QUERY_ON_FOREIGN_DATA_SOURCE](#))

Message: There was a problem processing the query on the foreign data source. Data source error: %s

- Error: [1431](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST](#))

Message: The foreign data source you are trying to reference does not exist. Data source error: %s

- Error: [1432](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE](#))

Message: Can't create federated table. The data source connection string '%s' is not in the correct format

- Error: [1433](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_DATA_STRING_INVALID](#))

Message: The data source connection string '%s' is not in the correct format

- Error: [1434](#) SQLSTATE: [HY000](#) ([ER_CANT_CREATE_FEDERATED_TABLE](#))

Message: Can't create federated table. Foreign data src error: %s

- Error: [1435](#) SQLSTATE: [HY000](#) ([ER_TRG_IN_WRONG_SCHEMA](#))

Message: Trigger in wrong schema

- Error: [1436](#) SQLSTATE: [HY000](#) ([ER_STACK_OVERRUN_NEED_MORE](#))

Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld -O thread_stack=#' to specify a bigger stack.

- Error: [1437](#) SQLSTATE: [42000](#) ([ER_TOO_LONG_BODY](#))

Message: Routine body for '%s' is too long

- Error: [1438](#) SQLSTATE: [HY000](#) ([ER_WARN_CANT_DROP_DEFAULT_KEYCACHE](#))

Message: Cannot drop default keycache

- Error: [1439](#) SQLSTATE: [42000](#) ([ER_TOO_BIG_DISPLAYWIDTH](#))

Message: Display width out of range for column '%s' (max = %lu)

- Error: [1440](#) SQLSTATE: [XAE08](#) ([ER_XAER_DUPID](#))
 Message: XAER_DUPID: The XID already exists
- Error: [1441](#) SQLSTATE: [22008](#) ([ER_DATETIME_FUNCTION_OVERFLOW](#))
 Message: Datetime function: %s field overflow
- Error: [1442](#) SQLSTATE: [HY000](#) ([ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG](#))
 Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.
- Error: [1443](#) SQLSTATE: [HY000](#) ([ER_VIEW_PREVENT_UPDATE](#))
 Message: The definition of table '%s' prevents operation %s on table '%s'.
- Error: [1444](#) SQLSTATE: [HY000](#) ([ER_PS_NO_RECURSION](#))
 Message: The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner
- Error: [1445](#) SQLSTATE: [HY000](#) ([ER_SP_CANT_SET_AUTOCOMMIT](#))
 Message: Not allowed to set autocommit from a stored function or trigger
- Error: [1446](#) SQLSTATE: [HY000](#) ([ER_MALFORMED_DEFINER](#))
 Message: Definer is not fully qualified
- Error: [1447](#) SQLSTATE: [HY000](#) ([ER_VIEW_FRM_NO_USER](#))
 Message: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!
- Error: [1448](#) SQLSTATE: [HY000](#) ([ER_VIEW_OTHER_USER](#))
 Message: You need the SUPER privilege for creation view with '%s'@'%s' definer
- Error: [1449](#) SQLSTATE: [HY000](#) ([ER_NO_SUCH_USER](#))
 Message: The user specified as a definer ('%s'@'%s') does not exist
- Error: [1450](#) SQLSTATE: [HY000](#) ([ER_FORBID_SCHEMA_CHANGE](#))
 Message: Changing schema from '%s' to '%s' is not allowed.
- Error: [1451](#) SQLSTATE: [23000](#) ([ER_ROW_IS_REFERENCED_2](#))
 Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)
- Error: [1452](#) SQLSTATE: [23000](#) ([ER_NO_REFERENCED_ROW_2](#))
 Message: Cannot add or update a child row: a foreign key constraint fails (%s)
- Error: [1453](#) SQLSTATE: [42000](#) ([ER_SP_BAD_VAR_SHADOW](#))
 Message: Variable '%s' must be quoted with `...`, or renamed
- Error: [1454](#) SQLSTATE: [HY000](#) ([ER_TRG_NO_DEFINER](#))
 Message: No definer attribute for trigger '%s'.'%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.
- Error: [1455](#) SQLSTATE: [HY000](#) ([ER_OLD_FILE_FORMAT](#))

Message: '%s' has an old format, you should re-create the '%s' object(s)

- Error: [1456](#) SQLSTATE: [HY000](#) ([ER_SP_RECURSION_LIMIT](#))

Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

- Error: [1457](#) SQLSTATE: [HY000](#) ([ER_SP_PROC_TABLE_CORRUPT](#))

Message: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)

- Error: [1458](#) SQLSTATE: [42000](#) ([ER_SP_WRONG_NAME](#))

Message: Incorrect routine name '%s'

- Error: [1459](#) SQLSTATE: [HY000](#) ([ER_TABLE_NEEDS_UPGRADE](#))

Message: Table upgrade required. Please do "REPAIR TABLE '%s'" or dump/reload to fix it!

- Error: [1460](#) SQLSTATE: [42000](#) ([ER_SP_NO_AGGREGATE](#))

Message: AGGREGATE is not supported for stored functions

- Error: [1461](#) SQLSTATE: [42000](#) ([ER_MAX_PREPARED_STMT_COUNT_REACHED](#))

Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error: [1462](#) SQLSTATE: [HY000](#) ([ER_VIEW_RECURSIVE](#))

Message: '%s`.`%s` contains view recursion

- Error: [1463](#) SQLSTATE: [42000](#) ([ER_NON_GROUPING_FIELD_USED](#))

Message: non-grouping field '%s' is used in %s clause

- Error: [1464](#) SQLSTATE: [HY000](#) ([ER_TABLE_CANT_HANDLE_SPKEYS](#))

Message: The used table type doesn't support SPATIAL indexes

- Error: [1465](#) SQLSTATE: [HY000](#) ([ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA](#))

Message: Triggers can not be created on system tables

- Error: [1466](#) SQLSTATE: [HY000](#) ([ER_REMOVED_SPACES](#))

Message: Leading spaces are removed from name '%s'

- Error: [1467](#) SQLSTATE: [HY000](#) ([ER_AUTOINC_READ_FAILED](#))

Message: Failed to read auto-increment value from storage engine

- Error: [1468](#) SQLSTATE: [HY000](#) ([ER_USERNAME](#))

Message: user name

- Error: [1469](#) SQLSTATE: [HY000](#) ([ER_HOSTNAME](#))

Message: host name

- Error: [1470](#) SQLSTATE: [HY000](#) ([ER_WRONG_STRING_LENGTH](#))

Message: String '%s' is too long for %s (should be no longer than %d)

- Error: [1471](#) SQLSTATE: [HY000](#) ([ER_NON_INSERTABLE_TABLE](#))

Message: The target table %s of the %s is not insertable-into

- Error: [1472](#) SQLSTATE: [HY000](#) ([ER_ADMIN_WRONG_MRG_TABLE](#))
 Message: Table '%s' is differently defined or of non-MyISAM type or doesn't exist
- Error: [1473](#) SQLSTATE: [HY000](#) ([ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT](#))
 Message: Too high level of nesting for select
- Error: [1474](#) SQLSTATE: [HY000](#) ([ER_NAME_BECOMES_EMPTY](#))
 Message: Name '%s' has become "
- Error: [1475](#) SQLSTATE: [HY000](#) ([ER_AMBIGUOUS_FIELD_TERM](#))
 Message: First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY
- Error: [1476](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_SERVER_EXISTS](#))
 Message: The foreign server, %s, you are trying to create already exists.
- Error: [1477](#) SQLSTATE: [HY000](#) ([ER_FOREIGN_SERVER_DOESNT_EXIST](#))
 Message: The foreign server name you are trying to reference does not exist. Data source error: %s
- Error: [1478](#) SQLSTATE: [HY000](#) ([ER_ILLEGAL_HA_CREATE_OPTION](#))
 Message: Table storage engine '%s' does not support the create option '%s'
- Error: [1479](#) SQLSTATE: [HY000](#) ([ER_PARTITION_REQUIRES_VALUES_ERROR](#))
 Message: Syntax error: %s PARTITIONING requires definition of VALUES %s for each partition
- Error: [1480](#) SQLSTATE: [HY000](#) ([ER_PARTITION_WRONG_VALUES_ERROR](#))
 Message: Only %s PARTITIONING can use VALUES %s in partition definition
- Error: [1481](#) SQLSTATE: [HY000](#) ([ER_PARTITION_MAXVALUE_ERROR](#))
 Message: MAXVALUE can only be used in last partition definition
- Error: [1482](#) SQLSTATE: [HY000](#) ([ER_PARTITION_SUBPARTITION_ERROR](#))
 Message: Subpartitions can only be hash partitions and by key
- Error: [1483](#) SQLSTATE: [HY000](#) ([ER_PARTITION_SUBPART_MIX_ERROR](#))
 Message: Must define subpartitions on all partitions if on one partition
- Error: [1484](#) SQLSTATE: [HY000](#) ([ER_PARTITION_WRONG_NO_PART_ERROR](#))
 Message: Wrong number of partitions defined, mismatch with previous setting
- Error: [1485](#) SQLSTATE: [HY000](#) ([ER_PARTITION_WRONG_NO_SUBPART_ERROR](#))
 Message: Wrong number of subpartitions defined, mismatch with previous setting
- Error: [1486](#) SQLSTATE: [HY000](#) ([ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR](#))
 Message: Constant, random or timezone-dependent expressions in (sub)partitioning function are not allowed
- Error: [1487](#) SQLSTATE: [HY000](#) ([ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR](#))
 Message: Expression in RANGE/LIST VALUES must be constant
- Error: [1488](#) SQLSTATE: [HY000](#) ([ER_FIELD_NOT_FOUND_PART_ERROR](#))

Message: Field in list of fields for partition function not found in table

- Error: 1489 SQLSTATE: HY000 (ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR)

Message: List of fields is only allowed in KEY partitions

- Error: 1490 SQLSTATE: HY000 (ER_INCONSISTENT_PARTITION_INFO_ERROR)

Message: The partition info in the frm file is not consistent with what can be written into the frm file

- Error: 1491 SQLSTATE: HY000 (ER_PARTITION_FUNC_NOT_ALLOWED_ERROR)

Message: The %s function returns the wrong type

- Error: 1492 SQLSTATE: HY000 (ER_PARTITIONS_MUST_BE_DEFINED_ERROR)

Message: For %s partitions each partition must be defined

- Error: 1493 SQLSTATE: HY000 (ER_RANGE_NOT_INCREASING_ERROR)

Message: VALUES LESS THAN value must be strictly increasing for each partition

- Error: 1494 SQLSTATE: HY000 (ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR)

Message: VALUES value must be of same type as partition function

- Error: 1495 SQLSTATE: HY000 (ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR)

Message: Multiple definition of same constant in list partitioning

- Error: 1496 SQLSTATE: HY000 (ER_PARTITION_ENTRY_ERROR)

Message: Partitioning can not be used stand-alone in query

- Error: 1497 SQLSTATE: HY000 (ER_MIX_HANDLER_ERROR)

Message: The mix of handlers in the partitions is not allowed in this version of MySQL

- Error: 1498 SQLSTATE: HY000 (ER_PARTITION_NOT_DEFINED_ERROR)

Message: For the partitioned engine it is necessary to define all %s

- Error: 1499 SQLSTATE: HY000 (ER_TOO_MANY_PARTITIONS_ERROR)

Message: Too many partitions (including subpartitions) were defined

- Error: 1500 SQLSTATE: HY000 (ER_SUBPARTITION_ERROR)

Message: It is only possible to mix RANGE/LIST partitioning with HASH/KEY partitioning for subpartitioning

- Error: 1501 SQLSTATE: HY000 (ER_CANT_CREATE_HANDLER_FILE)

Message: Failed to create specific handler file

- Error: 1502 SQLSTATE: HY000 (ER_BLOB_FIELD_IN_PART_FUNC_ERROR)

Message: A BLOB field is not allowed in partition function

- Error: 1503 SQLSTATE: HY000 (ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF)

Message: A %s must include all columns in the table's partitioning function

- Error: 1504 SQLSTATE: HY000 (ER_NO_PARTS_ERROR)

Message: Number of %s = 0 is not an allowed value

- Error: 1505 SQLSTATE: HY000 (ER_PARTITION_MGMT_ON_NONPARTITIONED)
Message: Partition management on a not partitioned table is not possible
- Error: 1506 SQLSTATE: HY000 (ER_FOREIGN_KEY_ON_PARTITIONED)
Message: Foreign key clause is not yet supported in conjunction with partitioning
- Error: 1507 SQLSTATE: HY000 (ER_DROP_PARTITION_NON_EXISTENT)
Message: Error in list of partitions to %s
- Error: 1508 SQLSTATE: HY000 (ER_DROP_LAST_PARTITION)
Message: Cannot remove all partitions, use DROP TABLE instead
- Error: 1509 SQLSTATE: HY000 (ER_COALESCE_ONLY_ON_HASH_PARTITION)
Message: COALESCE PARTITION can only be used on HASH/KEY partitions
- Error: 1510 SQLSTATE: HY000 (ER_REORG_HASH_ONLY_ON_SAME_NO)
Message: REORGANIZE PARTITION can only be used to reorganize partitions not to change their numbers
- Error: 1511 SQLSTATE: HY000 (ER_REORG_NO_PARAM_ERROR)
Message: REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH PARTITIONS
- Error: 1512 SQLSTATE: HY000 (ER_ONLY_ON_RANGE_LIST_PARTITION)
Message: %s PARTITION can only be used on RANGE/LIST partitions
- Error: 1513 SQLSTATE: HY000 (ER_ADD_PARTITION_SUBPART_ERROR)
Message: Trying to Add partition(s) with wrong number of subpartitions
- Error: 1514 SQLSTATE: HY000 (ER_ADD_PARTITION_NO_NEW_PARTITION)
Message: At least one partition must be added
- Error: 1515 SQLSTATE: HY000 (ER_COALESCE_PARTITION_NO_PARTITION)
Message: At least one partition must be coalesced
- Error: 1516 SQLSTATE: HY000 (ER_REORG_PARTITION_NOT_EXIST)
Message: More partitions to reorganize than there are partitions
- Error: 1517 SQLSTATE: HY000 (ER_SAME_NAME_PARTITION)
Message: Duplicate partition name %s
- Error: 1518 SQLSTATE: HY000 (ER_NO_BINLOG_ERROR)
Message: It is not allowed to shut off binlog on this command
- Error: 1519 SQLSTATE: HY000 (ER_CONSECUTIVE_REORG_PARTITIONS)
Message: When reorganizing a set of partitions they must be in consecutive order
- Error: 1520 SQLSTATE: HY000 (ER_REORG_OUTSIDE_RANGE)
Message: Reorganize of range partitions cannot change total ranges except for last partition where it can extend the range
- Error: 1521 SQLSTATE: HY000 (ER_PARTITION_FUNCTION_FAILURE)

Message: Partition function not supported in this version for this handler

- Error: [1522](#) SQLSTATE: [HY000](#) ([ER_PART_STATE_ERROR](#))

Message: Partition state cannot be defined from CREATE/ALTER TABLE

- Error: [1523](#) SQLSTATE: [HY000](#) ([ER_LIMITED_PART_RANGE](#))

Message: The %s handler only supports 32 bit integers in VALUES

- Error: [1524](#) SQLSTATE: [HY000](#) ([ER_PLUGIN_IS_NOT_LOADED](#))

Message: Plugin '%s' is not loaded

- Error: [1525](#) SQLSTATE: [HY000](#) ([ER_WRONG_VALUE](#))

Message: Incorrect %s value: '%s'

- Error: [1526](#) SQLSTATE: [HY000](#) ([ER_NO_PARTITION_FOR_GIVEN_VALUE](#))

Message: Table has no partition for value %s

- Error: [1527](#) SQLSTATE: [HY000](#) ([ER_FILEGROUP_OPTION_ONLY_ONCE](#))

Message: It is not allowed to specify %s more than once

- Error: [1528](#) SQLSTATE: [HY000](#) ([ER_CREATE_FILEGROUP_FAILED](#))

Message: Failed to create %s

- Error: [1529](#) SQLSTATE: [HY000](#) ([ER_DROP_FILEGROUP_FAILED](#))

Message: Failed to drop %s

- Error: [1530](#) SQLSTATE: [HY000](#) ([ER_TABLESPACE_AUTO_EXTEND_ERROR](#))

Message: The handler doesn't support autoextend of tablespaces

- Error: [1531](#) SQLSTATE: [HY000](#) ([ER_WRONG_SIZE_NUMBER](#))

Message: A size parameter was incorrectly specified, either number or on the form 10M

- Error: [1532](#) SQLSTATE: [HY000](#) ([ER_SIZE_OVERFLOW_ERROR](#))

Message: The size number was correct but we don't allow the digit part to be more than 2 billion

- Error: [1533](#) SQLSTATE: [HY000](#) ([ER_ALTER_FILEGROUP_FAILED](#))

Message: Failed to alter: %s

- Error: [1534](#) SQLSTATE: [HY000](#) ([ER_BINLOG_ROW_LOGGING_FAILED](#))

Message: Writing one row to the row-based binary log failed

- Error: [1535](#) SQLSTATE: [HY000](#) ([ER_BINLOG_ROW_WRONG_TABLE_DEF](#))

Message: Table definition on master and slave does not match: %s

- Error: [1536](#) SQLSTATE: [HY000](#) ([ER_BINLOG_ROW_RBR_TO_SBR](#))

Message: Slave running with --log-slave-updates must use row-based binary logging to be able to replicate row-based binary log events

- Error: [1537](#) SQLSTATE: [HY000](#) ([ER_EVENT_ALREADY_EXISTS](#))

Message: Event '%s' already exists

- Error: [1538](#) SQLSTATE: [HY000](#) ([ER_EVENT_STORE_FAILED](#))
 Message: Failed to store event %s. Error code %d from storage engine.
- Error: [1539](#) SQLSTATE: [HY000](#) ([ER_EVENT_DOES_NOT_EXIST](#))
 Message: Unknown event '%s'
- Error: [1540](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANT_ALTER](#))
 Message: Failed to alter event '%s'
- Error: [1541](#) SQLSTATE: [HY000](#) ([ER_EVENT_DROP_FAILED](#))
 Message: Failed to drop %s
- Error: [1542](#) SQLSTATE: [HY000](#) ([ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG](#))
 Message: INTERVAL is either not positive or too big
- Error: [1543](#) SQLSTATE: [HY000](#) ([ER_EVENT_ENDS_BEFORE_STARTS](#))
 Message: ENDS is either invalid or before STARTS
- Error: [1544](#) SQLSTATE: [HY000](#) ([ER_EVENT_EXEC_TIME_IN_THE_PAST](#))
 Message: Event execution time is in the past. Event has been disabled
- Error: [1545](#) SQLSTATE: [HY000](#) ([ER_EVENT_OPEN_TABLE_FAILED](#))
 Message: Failed to open mysql.event
- Error: [1546](#) SQLSTATE: [HY000](#) ([ER_EVENT_NEITHER_M_EXPR_NOR_M_AT](#))
 Message: No datetime expression provided
- Error: [1547](#) SQLSTATE: [HY000](#) ([ER_COL_COUNT_DOESNT_MATCH_CORRUPTED](#))
 Message: Column count of mysql.%s is wrong. Expected %d, found %d. The table is probably corrupted
- Error: [1548](#) SQLSTATE: [HY000](#) ([ER_CANNOT_LOAD_FROM_TABLE](#))
 Message: Cannot load from mysql.%s. The table is probably corrupted
- Error: [1549](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANNOT_DELETE](#))
 Message: Failed to delete the event from mysql.event
- Error: [1550](#) SQLSTATE: [HY000](#) ([ER_EVENT_COMPILE_ERROR](#))
 Message: Error during compilation of event's body
- Error: [1551](#) SQLSTATE: [HY000](#) ([ER_EVENT_SAME_NAME](#))
 Message: Same old and new event name
- Error: [1552](#) SQLSTATE: [HY000](#) ([ER_EVENT_DATA_TOO_LONG](#))
 Message: Data for column '%s' too long
- Error: [1553](#) SQLSTATE: [HY000](#) ([ER_DROP_INDEX_FK](#))
 Message: Cannot drop index '%s': needed in a foreign key constraint
- Error: [1554](#) SQLSTATE: [HY000](#) ([ER_WARN_DEPRECATED_SYNTAX_WITH_VER](#))
 Message: The syntax '%s' is deprecated and will be removed in MySQL %s. Please use %s instead

- Error: [1555](#) SQLSTATE: [HY000](#) ([ER_CANT_WRITE_LOCK_LOG_TABLE](#))
 Message: You can't write-lock a log table. Only read access is possible
- Error: [1556](#) SQLSTATE: [HY000](#) ([ER_CANT_LOCK_LOG_TABLE](#))
 Message: You can't use locks with log tables.
- Error: [1557](#) SQLSTATE: [23000](#) ([ER_FOREIGN_DUPLICATE_KEY](#))
 Message: Upholding foreign key constraints for table '%s', entry '%s', key %d would lead to a duplicate entry
- Error: [1558](#) SQLSTATE: [HY000](#) ([ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE](#))
 Message: Column count of mysql.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please use mysql_upgrade to fix this error.
- Error: [1559](#) SQLSTATE: [HY000](#) ([ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR](#))
 Message: Cannot switch out of the row-based binary log format when the session has open temporary tables
- Error: [1560](#) SQLSTATE: [HY000](#) ([ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT](#))
 Message: Cannot change the binary logging format inside a stored function or trigger
- Error: [1561](#) SQLSTATE: [HY000](#) ([ER_NDB_CANT_SWITCH_BINLOG_FORMAT](#))
 Message: The NDB cluster engine does not support changing the binlog format on the fly yet
- Error: [1562](#) SQLSTATE: [HY000](#) ([ER_PARTITION_NO_TEMPORARY](#))
 Message: Cannot create temporary table with partitions
- Error: [1563](#) SQLSTATE: [HY000](#) ([ER_PARTITION_CONST_DOMAIN_ERROR](#))
 Message: Partition constant is out of partition function domain
- Error: [1564](#) SQLSTATE: [HY000](#) ([ER_PARTITION_FUNCTION_IS_NOT_ALLOWED](#))
 Message: This partition function is not allowed
- Error: [1565](#) SQLSTATE: [HY000](#) ([ER_DDL_LOG_ERROR](#))
 Message: Error in DDL log
- Error: [1566](#) SQLSTATE: [HY000](#) ([ER_NULL_IN_VALUES_LESS_THAN](#))
 Message: Not allowed to use NULL value in VALUES LESS THAN
- Error: [1567](#) SQLSTATE: [HY000](#) ([ER_WRONG_PARTITION_NAME](#))
 Message: Incorrect partition name
- Error: [1568](#) SQLSTATE: [25001](#) ([ER_CANT_CHANGE_TX_ISOLATION](#))
 Message: Transaction isolation level can't be changed while a transaction is in progress
- Error: [1569](#) SQLSTATE: [HY000](#) ([ER_DUP_ENTRY_AUTOINCREMENT_CASE](#))
 Message: ALTER TABLE causes auto_increment resequencing, resulting in duplicate entry '%s' for key '%s'
- Error: [1570](#) SQLSTATE: [HY000](#) ([ER_EVENT_MODIFY_QUEUE_ERROR](#))
 Message: Internal scheduler error %d
- Error: [1571](#) SQLSTATE: [HY000](#) ([ER_EVENT_SET_VAR_ERROR](#))

Message: Error during starting/stopping of the scheduler. Error code %u

- Error: 1572 SQLSTATE: HY000 (ER_PARTITION_MERGE_ERROR)

Message: Engine cannot be used in partitioned tables

- Error: 1573 SQLSTATE: HY000 (ER_CANT_ACTIVATE_LOG)

Message: Cannot activate '%s' log

- Error: 1574 SQLSTATE: HY000 (ER_RBR_NOT_AVAILABLE)

Message: The server was not built with row-based replication

- Error: 1575 SQLSTATE: HY000 (ER_BASE64_DECODE_ERROR)

Message: Decoding of base64 string failed

- Error: 1576 SQLSTATE: HY000 (ER_EVENT_RECURSION_FORBIDDEN)

Message: Recursion of EVENT DDL statements is forbidden when body is present

- Error: 1577 SQLSTATE: HY000 (ER_EVENTS_DB_ERROR)

Message: Cannot proceed because system tables used by Event Scheduler were found damaged at server start

- Error: 1578 SQLSTATE: HY000 (ER_ONLY_INTEGERS_ALLOWED)

Message: Only integers allowed as number here

- Error: 1579 SQLSTATE: HY000 (ER_UNSUPPORTED_LOG_ENGINE)

Message: This storage engine cannot be used for log tables"

- Error: 1580 SQLSTATE: HY000 (ER_BAD_LOG_STATEMENT)

Message: You cannot '%s' a log table if logging is enabled

- Error: 1581 SQLSTATE: HY000 (ER_CANT_RENAME_LOG_TABLE)

Message: Cannot rename '%s'. When logging enabled, rename to/from log table must rename two tables: the log table to an archive table and another table back to '%s'

- Error: 1582 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT)

Message: Incorrect parameter count in the call to native function '%s'

- Error: 1583 SQLSTATE: 42000 (ER_WRONG_PARAMETERS_TO_NATIVE_FCT)

Message: Incorrect parameters in the call to native function '%s'

- Error: 1584 SQLSTATE: 42000 (ER_WRONG_PARAMETERS_TO_STORED_FCT)

Message: Incorrect parameters in the call to stored function '%s'

- Error: 1585 SQLSTATE: HY000 (ER_NATIVE_FCT_NAME_COLLISION)

Message: This function '%s' has the same name as a native function

- Error: 1586 SQLSTATE: 23000 (ER_DUP_ENTRY_WITH_KEY_NAME)

Message: '%s' は key '%s' において重複しています

- Error: 1587 SQLSTATE: HY000 (ER_BINLOG_PURGE_EMFILE)

Message: Too many files opened, please execute the command again

- Error: [1588](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANNOT_CREATE_IN_THE_PAST](#))
 Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.
- Error: [1589](#) SQLSTATE: [HY000](#) ([ER_EVENT_CANNOT_ALTER_IN_THE_PAST](#))
 Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.
- Error: [1590](#) SQLSTATE: [HY000](#) ([ER_SLAVE_INCIDENT](#))
 Message: The incident %s occurred on the master. Message: %s
- Error: [1591](#) SQLSTATE: [HY000](#) ([ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT](#))
 Message: Table has no partition for some existing values
- Error: [1592](#) SQLSTATE: [HY000](#) ([ER_BINLOG_UNSAFE_STATEMENT](#))
 Message: Statement may not be safe to log in statement format.
- Error: [1593](#) SQLSTATE: [HY000](#) ([ER_SLAVE_FATAL_ERROR](#))
 Message: Fatal error: %s
- Error: [1594](#) SQLSTATE: [HY000](#) ([ER_SLAVE_RELAY_LOG_READ_FAILURE](#))
 Message: Relay log read failure: %s
- Error: [1595](#) SQLSTATE: [HY000](#) ([ER_SLAVE_RELAY_LOG_WRITE_FAILURE](#))
 Message: Relay log write failure: %s
- Error: [1596](#) SQLSTATE: [HY000](#) ([ER_SLAVE_CREATE_EVENT_FAILURE](#))
 Message: Failed to create %s
- Error: [1597](#) SQLSTATE: [HY000](#) ([ER_SLAVE_MASTER_COM_FAILURE](#))
 Message: Master command %s failed: %s
- Error: [1598](#) SQLSTATE: [HY000](#) ([ER_BINLOG_LOGGING_IMPOSSIBLE](#))
 Message: Binary logging not possible. Message: %s
- Error: [1599](#) SQLSTATE: [HY000](#) ([ER_VIEW_NO_CREATION_CTX](#))
 Message: View `%s`.`%s` has no creation context
- Error: [1600](#) SQLSTATE: [HY000](#) ([ER_VIEW_INVALID_CREATION_CTX](#))
 Message: Creation context of view `%s`.`%s` is invalid
- Error: [1601](#) SQLSTATE: [HY000](#) ([ER_SR_INVALID_CREATION_CTX](#))
 Message: Creation context of stored routine `%s`.`%s` is invalid
- Error: [1602](#) SQLSTATE: [HY000](#) ([ER_TRG_CORRUPTED_FILE](#))
 Message: Corrupted TRG file for table `%s`.`%s`
- Error: [1603](#) SQLSTATE: [HY000](#) ([ER_TRG_NO_CREATION_CTX](#))
 Message: Triggers for table `%s`.`%s` have no creation context
- Error: [1604](#) SQLSTATE: [HY000](#) ([ER_TRG_INVALID_CREATION_CTX](#))

Message: Trigger creation context of table ``%s`.`%s` is invalid

- Error: 1605 SQLSTATE: HY000 (ER_EVENT_INVALID_CREATION_CTX)

Message: Creation context of event ``%s`.`%s` is invalid

- Error: 1606 SQLSTATE: HY000 (ER_TRG_CANT_OPEN_TABLE)

Message: Cannot open table for trigger ``%s`.`%s`

- Error: 1607 SQLSTATE: HY000 (ER_CANT_CREATE_SROUTINE)

Message: Cannot create stored routine ``%s`. Check warnings

- Error: 1608 SQLSTATE: HY000 (ER_SLAVE_AMBIGUOUS_EXEC_MODE)

Message: Ambiguous slave modes combination. %s

- Error: 1609 SQLSTATE: HY000 (ER_NO_FORMAT_DESCRIPTION_EVENT_BEFORE_BINLOG_STATEMENT)

Message: The BINLOG statement of type ``%s` was not preceded by a format description BINLOG statement.

- Error: 1610 SQLSTATE: HY000 (ER_SLAVE_CORRUPT_EVENT)

Message: Corrupted replication event was detected

- Error: 1611 SQLSTATE: HY000 (ER_LOAD_DATA_INVALID_COLUMN)

Message: Invalid column reference (%s) in LOAD DATA

- Error: 1612 SQLSTATE: HY000 (ER_LOG_PURGE_NO_FILE)

Message: Being purged log %s was not found

- Error: 1613 SQLSTATE: XA106 (ER_XA_RBTIMEOUT)

Message: XA_RBTIMEOUT: Transaction branch was rolled back: took too long

- Error: 1614 SQLSTATE: XA102 (ER_XA_RBDEADLOCK)

Message: XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected

- Error: 1615 SQLSTATE: HY000 (ER_NEED_REPREPARE)

Message: Prepared statement needs to be re-prepared

- Error: 1616 SQLSTATE: HY000 (ER_DELAYED_NOT_SUPPORTED)

Message: DELAYED option not supported for table '%s'

- Error: 1617 SQLSTATE: HY000 (WARN_NO_MASTER_INFO)

Message: The master info structure does not exist

- Error: 1618 SQLSTATE: HY000 (WARN_OPTION_IGNORED)

Message: <%s> option ignored

- Error: 1619 SQLSTATE: HY000 (WARN_PLUGIN_DELETE_BUILTIN)

Message: Built-in plugins cannot be deleted

- Error: 1620 SQLSTATE: HY000 (WARN_PLUGIN_BUSY)

Message: Plugin is busy and will be uninstalled on shutdown

- Error: [1621](#) SQLSTATE: [HY000](#) ([ER_VARIABLE_IS_READONLY](#))
 Message: %s variable '%s' is read-only. Use SET %s to assign the value
- Error: [1622](#) SQLSTATE: [HY000](#) ([ER_WARN_ENGINE_TRANSACTION_ROLLBACK](#))
 Message: Storage engine %s does not support rollback for this statement. Transaction rolled back and must be restarted
- Error: [1623](#) SQLSTATE: [HY000](#) ([ER_SLAVE_HEARTBEAT_FAILURE](#))
 Message: Unexpected master's heartbeat data: %s
- Error: [1624](#) SQLSTATE: [HY000](#) ([ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE](#))
 Message: The requested value for the heartbeat period %s %s
- Error: [1625](#) SQLSTATE: [HY000](#) ([ER_NDB_REPLICATION_SCHEMA_ERROR](#))
 Message: Bad schema for mysql.ndb_replication table. Message: %s
- Error: [1626](#) SQLSTATE: [HY000](#) ([ER_CONFLICT_FN_PARSE_ERROR](#))
 Message: Error in parsing conflict function. Message: %s
- Error: [1627](#) SQLSTATE: [HY000](#) ([ER_EXCEPTIONS_WRITE_ERROR](#))
 Message: Write to exceptions table failed. Message: %s"
- Error: [1628](#) SQLSTATE: [HY000](#) ([ER_TOO_LONG_TABLE_COMMENT](#))
 Message: Comment for table '%s' is too long (max = %lu)
- Error: [1629](#) SQLSTATE: [HY000](#) ([ER_TOO_LONG_FIELD_COMMENT](#))
 Message: Comment for field '%s' is too long (max = %lu)
- Error: [1630](#) SQLSTATE: [42000](#) ([ER_FUNC_INEXISTENT_NAME_COLLISION](#))
 Message: FUNCTION %s does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manual
- Error: [1631](#) SQLSTATE: [HY000](#) ([ER_DATABASE_NAME](#))
 Message: Database
- Error: [1632](#) SQLSTATE: [HY000](#) ([ER_TABLE_NAME](#))
 Message: Table
- Error: [1633](#) SQLSTATE: [HY000](#) ([ER_PARTITION_NAME](#))
 Message: Partition
- Error: [1634](#) SQLSTATE: [HY000](#) ([ER_SUBPARTITION_NAME](#))
 Message: Subpartition
- Error: [1635](#) SQLSTATE: [HY000](#) ([ER_TEMPORARY_NAME](#))
 Message: Temporary
- Error: [1636](#) SQLSTATE: [HY000](#) ([ER_RENAMED_NAME](#))
 Message: Renamed
- Error: [1637](#) SQLSTATE: [HY000](#) ([ER_TOO_MANY_CONCURRENT_TRXS](#))

Message: Too many active concurrent transactions

- Error: [1638](#) SQLSTATE: [HY000](#) ([WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED](#))

Message: Non-ASCII separator arguments are not fully supported

- Error: [1639](#) SQLSTATE: [HY000](#) ([ER_DEBUG_SYNC_TIMEOUT](#))

Message: debug sync point wait timed out

- Error: [1640](#) SQLSTATE: [HY000](#) ([ER_DEBUG_SYNC_HIT_LIMIT](#))

Message: debug sync point hit limit reached

Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the [include/errmsg.h](#) MySQL source file.
- The Message values correspond to the error messages that are listed in the [libmysql/errmsg.c](#) file. [%d](#) and [%s](#) represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: [2000](#) ([CR_UNKNOWN_ERROR](#))

Message: Unknown MySQL error

- Error: [2001](#) ([CR_SOCKET_CREATE_ERROR](#))

Message: Can't create UNIX socket (%d)

- Error: [2002](#) ([CR_CONNECTION_ERROR](#))

Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: [2003](#) ([CR_CONN_HOST_ERROR](#))

Message: Can't connect to MySQL server on '%s' (%d)

- Error: [2004](#) ([CR_IPSOCK_ERROR](#))

Message: Can't create TCP/IP socket (%d)

- Error: [2005](#) ([CR_UNKNOWN_HOST](#))

Message: Unknown MySQL server host '%s' (%d)

- Error: [2006](#) ([CR_SERVER_GONE_ERROR](#))

Message: MySQL server has gone away

- Error: [2007](#) ([CR_VERSION_ERROR](#))

Message: Protocol mismatch; server version = %d, client version = %d

- Error: [2008](#) ([CR_OUT_OF_MEMORY](#))

Message: MySQL client ran out of memory

- Error: [2009](#) ([CR_WRONG_HOST_INFO](#))

Message: Wrong host info

- Error: [2010 \(CR_LOCALHOST_CONNECTION\)](#)
Message: Localhost via UNIX socket
- Error: [2011 \(CR_TCP_CONNECTION\)](#)
Message: %s via TCP/IP
- Error: [2012 \(CR_SERVER_HANDSHAKE_ERR\)](#)
Message: Error in server handshake
- Error: [2013 \(CR_SERVER_LOST\)](#)
Message: Lost connection to MySQL server during query
- Error: [2014 \(CR_COMMANDS_OUT_OF_SYNC\)](#)
Message: Commands out of sync; you can't run this command now
- Error: [2015 \(CR_NAMEDPIPE_CONNECTION\)](#)
Message: Named pipe: %s
- Error: [2016 \(CR_NAMEDPIPEWAIT_ERROR\)](#)
Message: Can't wait for named pipe to host: %s pipe: %s (%lu)
- Error: [2017 \(CR_NAMEDPIPEOPEN_ERROR\)](#)
Message: Can't open named pipe to host: %s pipe: %s (%lu)
- Error: [2018 \(CR_NAMEDPIPESETSTATE_ERROR\)](#)
Message: Can't set state of named pipe to host: %s pipe: %s (%lu)
- Error: [2019 \(CR_CANT_READ_CHARSET\)](#)
Message: Can't initialize character set %s (path: %s)
- Error: [2020 \(CR_NET_PACKET_TOO_LARGE\)](#)
Message: Got packet bigger than 'max_allowed_packet' bytes
- Error: [2021 \(CR_EMBEDDED_CONNECTION\)](#)
Message: Embedded server
- Error: [2022 \(CR_PROBE_SLAVE_STATUS\)](#)
Message: Error on SHOW SLAVE STATUS:
- Error: [2023 \(CR_PROBE_SLAVE_HOSTS\)](#)
Message: Error on SHOW SLAVE HOSTS:
- Error: [2024 \(CR_PROBE_SLAVE_CONNECT\)](#)
Message: Error connecting to slave:
- Error: [2025 \(CR_PROBE_MASTER_CONNECT\)](#)
Message: Error connecting to master:
- Error: [2026 \(CR_SSL_CONNECTION_ERROR\)](#)
Message: SSL connection error

- Error: [2027 \(CR_MALFORMED_PACKET\)](#)
Message: Malformed packet
- Error: [2028 \(CR_WRONG_LICENSE\)](#)
Message: This client library is licensed only for use with MySQL servers having '%s' license
- Error: [2029 \(CR_NULL_POINTER\)](#)
Message: Invalid use of null pointer
- Error: [2030 \(CR_NO_PREPARE_STMT\)](#)
Message: Statement not prepared
- Error: [2031 \(CR_PARAMS_NOT_BOUND\)](#)
Message: No data supplied for parameters in prepared statement
- Error: [2032 \(CR_DATA_TRUNCATED\)](#)
Message: Data truncated
- Error: [2033 \(CR_NO_PARAMETERS_EXISTS\)](#)
Message: No parameters exist in the statement
- Error: [2034 \(CR_INVALID_PARAMETER_NO\)](#)
Message: Invalid parameter number
- Error: [2035 \(CR_INVALID_BUFFER_USE\)](#)
Message: Can't send long data for non-string/non-binary data types (parameter: %d)
- Error: [2036 \(CR_UNSUPPORTED_PARAM_TYPE\)](#)
Message: Using unsupported buffer type: %d (parameter: %d)
- Error: [2037 \(CR_SHARED_MEMORY_CONNECTION\)](#)
Message: Shared memory: %s
- Error: [2038 \(CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR\)](#)
Message: Can't open shared memory; client could not create request event (%lu)
- Error: [2039 \(CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR\)](#)
Message: Can't open shared memory; no answer event received from server (%lu)
- Error: [2040 \(CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR\)](#)
Message: Can't open shared memory; server could not allocate file mapping (%lu)
- Error: [2041 \(CR_SHARED_MEMORY_CONNECT_MAP_ERROR\)](#)
Message: Can't open shared memory; server could not get pointer to file mapping (%lu)
- Error: [2042 \(CR_SHARED_MEMORY_FILE_MAP_ERROR\)](#)
Message: Can't open shared memory; client could not allocate file mapping (%lu)
- Error: [2043 \(CR_SHARED_MEMORY_MAP_ERROR\)](#)
Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: [2044 \(CR_SHARED_MEMORY_EVENT_ERROR\)](#)
Message: Can't open shared memory; client could not create %s event (%lu)
- Error: [2045 \(CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR\)](#)
Message: Can't open shared memory; no answer from server (%lu)
- Error: [2046 \(CR_SHARED_MEMORY_CONNECT_SET_ERROR\)](#)
Message: Can't open shared memory; cannot send request event to server (%lu)
- Error: [2047 \(CR_CONN_UNKNOW_PROTOCOL\)](#)
Message: Wrong or unknown protocol
- Error: [2048 \(CR_INVALID_CONN_HANDLE\)](#)
Message: Invalid connection handle
- Error: [2049 \(CR_SECURE_AUTH\)](#)
Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)
- Error: [2050 \(CR_FETCH_CANCELED\)](#)
Message: Row retrieval was canceled by mysql_stmt_close() call
- Error: [2051 \(CR_NO_DATA\)](#)
Message: Attempt to read column without prior row fetch
- Error: [2052 \(CR_NO_STMT_METADATA\)](#)
Message: Prepared statement contains no metadata
- Error: [2053 \(CR_NO_RESULT_SET\)](#)
Message: Attempt to read a row while there is no result set associated with the statement
- Error: [2054 \(CR_NOT_IMPLEMENTED\)](#)
Message: This feature is not implemented yet
- Error: [2055 \(CR_SERVER_LOST_EXTENDED\)](#)
Message: Lost connection to MySQL server at '%s', system error: %d
- Error: [2056 \(CR_STMT_CLOSED\)](#)
Message: Statement closed indirectly because of a preceeding %s() call
- Error: [2057 \(CR_NEW_STMT_METADATA\)](#)
Message: The number of columns in the result set differs from the number of bound buffers. You must reset the statement, rebind the result set columns, and execute the statement again

付録C MySQL Change History

目次

Changes in release 5.1.x (Development)	1564
Changes in release 5.1.16 (Not yet released)	1564
Changes in release 5.1.15 (25 January 2007)	1568
Changes in release 5.1.14 (05 December 2006)	1575
Changes in release 5.1.13 (Not released)	1580
Changes in release 5.1.12 (24 October 2006)	1584
Changes in release 5.1.11 (26 May 2006)	1611
Changes in release 5.1.10 (Not released)	1614
Changes in release 5.1.9 (12 April 2006)	1620
Changes in release 5.1.8 (Not released)	1622
Changes in release 5.1.7 (27 February 2006)	1630
Changes in release 5.1.6 (01 February 2006)	1633
Changes in release 5.1.5 (10 January 2006)	1637
Changes in release 5.1.4 (21 December 2005)	1638
Changes in release 5.1.3 (29 November 2005)	1639
Changes in release 5.1.2 (Not released)	1640
Changes in release 5.1.1 (Not released)	1641
MySQL Connector/ODBC (MyODBC) Change History	1641
Changes in Connector/ODBC 5.0.11 (31 January 2007)	1641
Changes in Connector/ODBC 5.0.10 (14 December 2006)	1641
Changes in Connector/ODBC 5.0.9 (22 November 2006)	1642
Changes in Connector/ODBC 5.0.8 (17 November 2006)	1642
Changes in Connector/ODBC 5.0.7 (08 November 2006)	1643
Changes in Connector/ODBC 5.0.6 (03 November 2006)	1643
Changes in Connector/ODBC 5.0.5 (17 October 2006)	1643
Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)	1644
Changes in Connector/ODBC 5.0.2 (Never released)	1644
Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)	1644
Changes in Connector/ODBC 3.51.13 (Not yet released)	1645
Changes in Connector/ODBC 3.51.12	1645
Changes in Connector/ODBC 3.51.11	1646
Connector/NET Change History	1646
Changes in MySQL Connector/NET Version 5.0.4 (Not yet released)	1646
Changes in MySQL Connector/NET Version 5.0.3 (05 January 2007)	1646
Changes in MySQL Connector/NET Version 5.0.2 (06 November 2006)	1647
Changes in MySQL Connector/NET Version 5.0.1 (01 October 2006)	1648
Changes in MySQL Connector/NET Version 5.0.0 (08 August 2006)	1648
Changes in MySQL Connector/NET Version 1.0.10 (Not yet released)	1649
Changes in MySQL Connector/NET Version 1.0.9 (02 February 2007)	1649
Changes in MySQL Connector/NET Version 1.0.8 (20 October 2006)	1650
Changes in MySQL Connector/NET Version 1.0.7 (21 November 2005)	1651
Changes in MySQL Connector/NET Version 1.0.6 (03 October 2005)	1652
Changes in MySQL Connector/NET Version 1.0.5 (29 August 2005)	1652
Changes in MySQL Connector/NET Version 1.0.4 (20 January 2005)	1652
Changes in MySQL Connector/NET Version 1.0.3-gamma (12 October 2004)	1653
Changes in MySQL Connector/NET Version 1.0.2-gamma (15 November 2004)	1653
Changes in MySQL Connector/NET Version 1.0.1-beta2 (27 October 2004)	1654
Changes in MySQL Connector/NET Version 1.0.0 (01 September 2004)	1655
Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)	1655
Changes in MySQL Connector/NET Version 0.76	1658
Changes in MySQL Connector/NET Version 0.75	1659
Changes in MySQL Connector/NET Version 0.74	1660
Changes in MySQL Connector/NET Version 0.71	1661

Changes in MySQL Connector/NET Version 0.70	1662
Changes in MySQL Connector/NET Version 0.68	1663
Changes in MySQL Connector/NET Version 0.65	1664
Changes in MySQL Connector/NET Version 0.60	1664
Changes in MySQL Connector/NET Version 0.50	1664
MySQL Visual Studio Plugin Change History	1664
Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)	1664
Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)	1664
Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)	1665
MySQL Connector/J Change History	1665
Changes in MySQL Connector/J 5.1.x	1665
Changes in MySQL Connector/J 5.0.x	1665
Changes in MySQL Connector/J 3.1.x	1669
Changes in MySQL Connector/J 3.0.x	1684
Changes in MySQL Connector/J 2.0.x	1695
Changes in MySQL Connector/J 1.2b (04 July 1999)	1699
Changes in MySQL Connector/J 1.2.x and lower	1699

This appendix lists the changes from version to version in the MySQL source code through the latest version of MySQL 5.1, which is currently MySQL 5.1.15-beta. Starting with MySQL 5.0, we began offering a new version of the Manual for each new series of MySQL releases (5.0, 5.1, and so on). For information about changes in previous release series of the MySQL database software, see the corresponding version of this Manual. For information about legacy versions of the MySQL software through the 4.1 series, see MySQL 3.23, 4.0, 4.1 Reference Manual.

We update this section as we add new features in the 5.1 series, so that everybody can follow the development process.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you can't find on our download page (<http://dev.mysql.com/downloads/>), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last BitKeeper ChangeSet on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

The manual included in the source and binary distributions may not be fully accurate when it comes to the release changelog entries, because the integration of the manual happens at build time. For the most up-to-date release changelog, please refer to the online version instead.

Changes in release 5.1.x (Development)

An overview of which features were added in MySQL 5.1 can be found here: [MySQL 5.1 での新機能](#).

For a full list of changes, please refer to the changelog sections for each individual 5.1.x release.

Changes in release 5.1.16 (Not yet released)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on production level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise/>をご覧ください。

Functionality added or changed:

- Incompatible change: [TRUNCATE TABLE](#) now requires the [DROP](#) privilege rather than the [DELETE](#) privilege. (Bug #23556).
- [NDB Cluster](#) (Cluster APIs): A new [ndb_mgm_get_clusterlog_loglevel\(\)](#) function was added to the MGM API.
For more information, see [ndb_mgm_get_clusterlog_loglevel\(\)](#).
- [NDB Cluster](#) (Cluster APIs) / Incompatible change: The [AbortOption](#) type is now a member of the [NdbOperation](#) class; its values and behavior have also changed. [NdbTransaction::AbortOption](#) can no longer be used, and applications written against the NDB API may need to be rewritten and recompiled to accommodate these changes.
See [The NdbOperation::AbortOption Type](#), for more information.
- [NDB Cluster](#): Previously, when a data node failed more than 8 times in succession to start, this caused a forced shutdown of the cluster. Now, when a data node fails to start 7 consecutive times, the node will not start again until it is started with the [--initial](#) option, and a warning to this effect is written to the error log. (Bug #25984)
- [NDB Cluster](#): A number of new and more descriptive error messages covering transporter errors were added. (Bug #22025)
- [NDB Cluster](#): In the event that all cluster management and API nodes are configured with [ArbitrationRank=0](#), [ndb_mgmd](#) now issues the following warning when starting: [Cluster configuration warning: Neither MGM nor API nodes are configured with arbitrator, may cause complete cluster shutdown in case of host failure.](#) (Bug #23546)
- In the [INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS](#) table, the [UNIQUE_CONSTRAINT_NAME](#) column incorrectly named the referenced table. Now it names the referenced constraint, and a new column, [REFERENCED_TABLE_NAME](#), names the referenced table. (Bug #21713)
- [RAND\(\)](#) now allows non-constant initializers (such as a column name) as its argument. In this case, the seed is initialized with the value for each invocation of [RAND\(\)](#). (One implication of this is that for equal argument values, [RAND\(\)](#) will return the same value each time.) (Bug #6172)
- The bundled yaSSL library was upgraded to version 1.5.8.
- [CONNECTION](#) is no longer treated as a reserved word. (Bug #12204)

Bugs fixed:

- [NDB Cluster](#) (Cluster APIs): [libndbclient.so](#) was not versioned. (Bug #13522)
- [NDB Cluster](#): The [ndb_size.tmp](#) file (necessary for using the [ndb_size.pl](#) script) was missing from binary distributions. (Bug #24191)
- [NDB Cluster](#) (Cluster APIs / Disk Data): A delete and a read performed in the same operation could cause one or more of the cluster's data nodes to crash. This could occur when the operation affected more than 5 columns concurrently, or when one or more of the columns was of the [VARCHAR](#) type and was stored on disk. (Bug #25794)
- [NDB Cluster](#) (Replication): The error message [Last_Errno: 4294967295, Error in Write_rows event](#) now supplies a valid error code. (Bug #19896)
- [NDB Cluster](#) (Replication): Under some circumstances, the binlog thread could shut down while the slave SQL thread was still using it. (Bug #26015, Bug #26019)
- [NDB Cluster](#): A query with an [IN](#) clause against an [NDB](#) table employing explicit user-defined partitioning did not always return all matching rows. (Bug #25821)
- A memory leak could cause problems during a node or cluster shutdown or failure. (Bug #25997)
- [NDB Cluster](#): An appropriate error message was not provided when there was insufficient REDO log file space for the cluster to start. (Bug #25801)
- [NDB Cluster](#): An [UPDATE](#) using an [IN](#) clause on an [NDB](#) table on which there was a trigger caused [mysqld](#) to crash. (Bug #25522)
- [NDB Cluster](#): A memory allocation failure in the cluster Subscription Manager could cause the cluster to crash. (Bug #25239)

- **NDB Cluster**: In the event that cluster backup parameters such as **BackupWriteSize** were incorrectly set, no appropriate error was issued to indicate that this was the case. (Bug #19146)
- Using an **INFORMATION_SCHEMA** table with **ORDER BY** in a subquery could cause a server crash. (Bug #24630)
- Collation for **LEFT JOIN** comparisons could be evaluated incorrectly, leading to improper query results. (Bug #26017)
- For the **IF()** and **COALESCE()** function and **CASE** expressions, large unsigned integer values could be mishandled and result in warnings. (Bug #22026)
- The number of **setsockopt()** calls performed for reads and writes to the network socket was reduced to decrease system call overhead. (Bug #22943)
- A **WHERE** clause that used **BETWEEN** for **DATETIME** values could be treated differently for a **SELECT** and a view defined as that **SELECT**. (Bug #26124)
- **ORDER BY** on **DOUBLE** values could change the set of rows returned by a query. (Bug #19690)
- **LOAD DATA INFILE** did not work with pipes. (Bug #25807)
- **DISTINCT** queries that were executed using a loose scan for an **InnoDB** table that had been emptied caused a server crash. (Bug #26159)
- **ALTER TABLE** caused loss of **CASCADE** clauses for **InnoDB** tables. (Bug #24741)
- Type conversion errors during formation of index search conditions were not correctly checked, leading to incorrect query results. (Bug #22344)
- Within a stored routine, accessing a declared routine variable with **PROCEDURE ANALYSE()** caused a server crash. (Bug #23782)
- Use of already freed memory caused SSL connections to hang forever. (Bug #19209)
- **mysql.server stop** timed out too quickly (35 seconds) waiting for the server to exit. Now it waits up to 15 minutes, to ensure that the server exits. (Bug #25341)
- A yaSSL program named **test** was installed, causing conflicts with the **test** system utility. It is no longer installed. (Bug #25417)
- **perror** crashed on some platforms due to failure to handle a **NULL** pointer. (Bug #25344)
- **mysql_stmt_fetch()** did an invalid memory deallocation when used with the embedded server. (Bug #25492)
- **mysql_kill()** caused a server crash when used on an SSL connection. (Bug #25203)
- The **readline** library wrote to uninitialized memory, causing **mysql** to crash. (Bug #19474)
- yaSSL was sensitive to the presence of whitespace at the ends of lines in PEM-encoded certificates, causing a server crash. (Bug #25189)
- The **SEC_TO_TIME()** and **QUARTER()** functions sometimes did not handle **NULL** values correctly. (Bug #25643)
- The optimizer used a filesort rather than a **const** table read in some cases when the latter was possible. (Bug #16590)
- With **ONLY_FULL_GROUP_BY** enables, the server was too strict: Some expressions involving only aggregate values were rejected as non-aggregate (for example, **MAX(a) - MIN(a)**). (Bug #23417)
- Indexes disabled with **ALTER TABLE ... DISABLE KEYS** could in some cases be used by specifying **FORCE INDEX**. (Bug #20604)
- The arguments of the **ENCODE()** and the **DECODE()** functions were not printed correctly, causing problems in the output of **EXPLAIN EXTENDED** and in view definitions. (Bug #23409)
- An error in the name resolution of nested **JOIN ... USING** constructs was corrected. (Bug #25575)

- A return value of `-1` from user-defined handlers was not handled well and could result in conflicts with server code. (Bug #24987)
- The server might fail to use an appropriate index for `DELETE` when `ORDER BY`, `LIMIT`, and a non-restricting `WHERE` are present. (Bug #17711)
- Use of `ON DUPLICATE KEY UPDATE` defeated the usual restriction against inserting into a join-based view unless only one of the underlying tables is used. (Bug #25123)
- View definitions that used the `!` operator were treated as containing the `NOT` operator, which has a different precedence and can produce different results. (Bug #25580).
- Some queries against `INFORMATION_SCHEMA` that used subqueries failed. (Bug #23299).
- For a `UNIQUE` index containing many `NULL` values, the optimizer would prefer the index for `col IS NULL` conditions over other more selective indexes. (Bug #25407).
- `GROUP BY` and `DISTINCT` did not group `NULL` values for columns that have a `UNIQUE` index. (Bug #25551).
- `ALTER TABLE ... ENABLE KEYS` acquired a global lock, preventing concurrent execution of other statements that use tables. (Bug #25044).
- For an `InnoDB` table with any `ON DELETE` trigger, `TRUNCATE TABLE` mapped to `DELETE` and activated triggers. Now a fast truncation occurs and triggers are not activated. (Bug #23556).
- For `ALTER TABLE`, using `ORDER BY expression` could cause a server crash. Now the `ORDER BY` clause allows only column names to be specified as sort criteria (which was the only documented syntax, anyway). (Bug #24562)
- `readline` detection did not work correctly on NetBSD. (Bug #23293)
- The `--with-readline` option for `configure` does not work for commercial source packages, but no error message was printed to that effect. Now a message is printed. (Bug #25530)
- If an `ORDER BY` or `GROUP BY` list included a constant expression being optimized away and, at the same time, containing single-row subselects that return more than one row, no error was reported. If a query requires sorting by expressions containing single-row subselects that return more than one row, execution of the query may cause a server crash. (Bug #24653)
- To enable installation of MySQL RPMs on Linux systems running RHEL 4 (which includes SE-Linux) additional information was provided to specify some actions that are allowed to the MySQL binaries. (Bug #12676)
- Queries that evaluate `NULL IN (SELECT ... UNION SELECT ...)` could produce an incorrect result (`FALSE` instead of `NULL`). (Bug #24085)
- Within stored routines or prepared statements, inconsistent results occurred with multiple use of `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` when the `ON DUPLICATE KEY UPDATE` clause erroneously tried to assign a value to a column mentioned only in its `SELECT` part. (Bug #24491)
- Expressions of the form `(a, b) IN (SELECT a, MIN(b) FROM t GROUP BY a)` could produce incorrect results when column `a` of table `t` contained `NULL` values while column `b` did not. (Bug #24420)
- Expressions of the form `(a, b) IN (SELECT c, d ...)` could produce incorrect results if `a`, `b`, or both were `NULL`. (Bug #24127)
- An `AFTER UPDATE` trigger on an `InnoDB` table with a composite primary key caused the server to crash. (Bug #25398)
- A query that contained an `EXIST` subquery with a `UNION` over correlated and uncorrelated `SELECT` queries could cause the server to crash. (Bug #25219)
- A query with `ORDER BY` and `GROUP BY` clauses where the `ORDER BY` clause had more elements than the `GROUP BY` clause caused a memory overrun leading to a crash of the server. (Bug #25172)
- Certain joins using `Range checked for each record` in the query execution plan could cause the server to crash. (Bug #24776)

- If a prepared statement accessed a view, access to the tables listed in the query after that view was checked in the security context of the view. (Bug #24404)
- A nested query on a partitioned table returned fewer records than on the corresponding non-partitioned table, when the subquery affected more than one partition. (Bug #24186)
- Passing a `NULL` value to a user-defined function from within a stored procedure crashes the server. (Bug #25382)

Changes in release 5.1.15 (25 January 2007)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on production level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise>をご覧ください。

Functionality added or changed:

- Incompatible change: The following conditions apply to enabling the `read_only` system variable:
 - If you attempt to enable `read_only` while you have any explicit locks (acquired with `LOCK TABLES` or have a pending transaction, an error will occur.
 - If other clients hold explicit table locks or have pending transactions, the attempt to enable `read_only` blocks until the locks are released and the transactions end. While the attempt to enable `read_only` is pending, requests by other clients for table locks or to begin transactions also block until `read_only` has been set.
 - `read_only` can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.

Previously, the attempt to enable `read_only` would return immediately even if explicit locks or transactions were pending, so some data changes could occur for statements executing in the server at the same time. (Bug #11733, Bug #22009)

- Incompatible change: Previously, the `DATE_FORMAT()` function returned a binary string. Now it returns a string with a character set and collation given by `character_set_connection` and `collation_connection` so that it can return month and weekday names containing non-ASCII characters. (Bug #22646)
- Important: When using `MERGE` tables the definition of the `MERGE` table and the `MyISAM` tables are checked each time the tables are opened for access (including any `SELECT` or `INSERT` statement. Each table is compared for column order, types, sizes and associated. If there is a difference in any one of the tables then the statement will fail.
- Remote servers for use with the `FEDERATED` storage engine now can be managed with the new `CREATE/ALTER/DROP SERVER` syntax.
- The `--skip-thread-priority` option now is enabled by default for binary Mac OS X distributions. Use of thread priorities degrades performance on Mac OS X. (Bug #18526)
- Added the `--disable-grant-options` option to `configure`. If `configure` is run with this option, the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld` are disabled and cannot be used. For Windows, the `configure.js` script recognizes the `DISABLE_GRANT_OPTIONS` flag, which has the same effect.
- Partitioning of tables using the `FEDERATED` storage engine is no longer permitted. Attempting to create such a table or to modify an existing table so that it uses both partitioning and `FEDERATED` now fails with an error. (Bug #22451)
- In MySQL 5.1, `InnoDB` rolls back only the last statement on a transaction timeout. A new option, `--innodb_rollback_on_timeout`, causes `InnoDB` to abort and roll back the entire transaction if a transaction timeout occurs (the same behavior as in MySQL 4.1). (Bug #24200)

- The `Com_create_user` status variable was added (for counting `CREATE USER` statements). (Bug #22958)
- The `--memlock` option relies on system calls that are unreliable on some operating systems. If a crash occurs, the server now checks whether `--memlock` was specified and if so issues some information about possible workarounds. (Bug #22860)
- The default value of the `max_connections` variable has been increased to 151 in order that Websites running on Apache and using MySQL will not have more processes trying to access MySQL than the default number of connections available.

(The maximum number of Apache processes is determined by the Apache `MaxClient`, which defaults to 256, but is usually set to 150 in the `httpd.conf` commonly distributed with Apache. For more information about `MaxClient`, see http://httpd.apache.org/docs/2.2/mod/mpm_common.html#maxclients.)

(Bug #23883)

- The bundled yaSSL library was upgraded to version 1.5.0.
- Calling a non-deterministic stored routine when using statement-based replication now throws an error. Formerly, defining such a stored routine would cause an error to be thrown. (Bug #16456)
- The (undocumented) `UNIQUE_USERS()` and `GROUP_UNIQUE_USERS()` functions were removed. (Bug #22687)
- **NDB Cluster:** The `LockPagesInMainMemory` configuration parameter has changed its type and possible values. For more information, see [LockPagesInMainMemory \[906\]](#). (Bug #25686)

Important: The values `true` and `false` are no longer accepted for this parameter. If you were using this parameter and had it set to `false` in a previous release, you must change it to `0`. If you had this parameter set to `true`, you should instead use `1` to obtain the same behavior as previously, or `2` to take advantage of new functionality introduced with this release described in the section cited above.

Bugs fixed:

- `mysqld_multi` and `mysqlaccess` looked for option files in `/etc` even if the `--sysconfdir` option for `configure` had been given to specify a different directory. (Bug #24780)
- `SHOW COLUMNS` reported some `NOT NULL` columns as `NULL`. (Bug #22377)
- Attempts to access a `MyISAM` table with a corrupt column definition caused a server crash. (Bug #24401)
- `mysqltest_embedded` crashed at startup. (Bug #25890)
- Accessing a fixed record format table with a crashed key definition results in server/`myisamchk` segmentation fault. (Bug #24855)
- When opening a corrupted `.frm` file during a query, the server crashes. (Bug #24358)
- If there was insufficient memory to store or update a blob record in a `MyISAM` table then the table will marked as crashed. (Bug #23196)
- When updating a table that used a `JOIN` of the table itself (for example, when building trees) and the table was modified on one side of the expression, the table would either be reported as crashed or the wrong rows in the table would be updated. (Bug #21310)
- When `SET PASSWORD` was written to the binary log double quotes were included in the statement. If the slave was running in with the `sql_mode` set to `ANSI_QUOTES` the event would fail and halt the replication process. (Bug #24158)
- Using `CREATE TABLE ... SELECT` and rolling back the transaction would leave an empty table on the master, but the instructions would not be recorded in the binary log and therefore replicated to the slave. This would result in a difference between the master and slave databases. An implicit commit has been added to ensure consistency. (Bug #22865)
- When reading from the standard input on Windows, `mysqlbinlog` opened the input in text mode rather than binary mode and consequently misinterpreted some characters such as Control-Z. (Bug #23735)

- No warning was issued for use of the [DATA DIRECTORY](#) or [INDEX DIRECTORY](#) table options on a platform that does not support them. (Bug #17498)
- When a prepared statement failed during the prepare operation, the error code was not cleared when it was reused, even if the subsequent use was successful. (Bug #15518)
- [mysql_upgrade](#) failed when called with a [basedir](#) pathname containing spaces. (Bug #22801)
- Hebrew-to-Unicode conversion failed for some characters. Definitions for the following Hebrew characters (as specified by the ISO/IEC 8859-8:1999) were added: LEFT-TO-RIGHT MARK (LRM), RIGHT-TO-LEFT MARK (RLM) (Bug #24037)
- If there was insufficient memory available to [mysqld](#), this could sometimes cause the server to hang during startup. (Bug #24751)
- Using row-based replication to replicate to a table having at least one extra [BIT](#) column with a default value on the slave as compared to the master could cause the slave to fail. (Bug #24490)
- Optimizations that are legal only for subqueries without tables and [WHERE](#) conditions were applied for any subquery without tables. (Bug #24670)
- A query using [WHERE unsigned_column NOT IN \('negative_value'\)](#) could cause the server to crash. (Bug #24261)
- A [FETCH](#) statement using a cursor on a table which was not in the table cache could sometimes cause the server to crash. (Bug #24117)
- [CREATE TABLE ... SELECT](#) statements were not rolled back correctly. As part of the fix, such a statement now causes an implicit commit before and after it is executed. However, it does not cause a commit when used to create a temporary table. (Bug #22864)
- SSL connections could hang at connection shutdown. (Bug #24148)
- The [STDDEV\(\)](#) function returned a positive value for data sets consisting of a single value. (Bug #22555)
- [mysqltest](#) incorrectly tried to retrieve result sets for some queries where no result set was available. (Bug #19410)
- [mysqltest](#) crashed with a stack overflow. (Bug #24498)
- The server was built even when [configure](#) was run with the [--without-server](#) option. (Bug #23973)
- [mysqld_error.h](#) was not installed when only the client libraries were built. (Bug #21265)
- [mysql_install_db](#) did not create the [mysql.plugin](#) table if strict SQL mode was enabled. (Bug #24270)
- The row count for [MyISAM](#) tables was not updated properly, causing [SHOW TABLE STATUS](#) to report incorrect values. (Bug #23526)
- Using a view in combination with a [USING](#) clause caused column aliases to be ignored. (Bug #25106)
- Using [ALTER TABLE](#) to convert a [CSV](#) table containing [NULL](#) values to [MyISAM](#) resulted in warnings. (Bug #21328)
- A view was not handled correctly if the [SELECT](#) part contained ['Z'](#). (Bug #24293)
- Inserting a row into a table without specifying a value for a [BINARY\(N\) NOT NULL](#) column caused the column to be set to spaces, not zeroes. (Bug #14171)
- An assertion failed incorrectly for prepared statements that contained a single-row non-correlated subquery that was used as an argument of the IS NULL predicate. (Bug #25027)
- A table created with the [ROW_FORMAT = FIXED](#) table option loses the option if an index is added or dropped with [CREATE INDEX](#) or [DROP INDEX](#). (Bug #23404)
- Dropping a user-defined function sometimes did not remove the UDF entry from the [mysql.proc](#) table. (Bug #15439)
- The [BUILD/check-cpu](#) script did not recognize Celeron processors. (Bug #20061)
- Some problems uncovered by Valgrind were fixed. (Bug #25396)

- [mysql_fix_privilege_tables](#) did not handle a password containing embedded space or apostrophe characters. (Bug #17700)
 - On Windows, the [SLEEP\(\)](#) function could sleep too long, especially after a change to the system clock. (Bug #14094, Bug #17635, Bug #24686)
 - In the [INFORMATION_SCHEMA.KEY_COLUMN_USAGE](#) table, the value displayed for the [REFERENCED_TABLE_NAME](#) column was the table name as encoded for disk storage, not the actual table name. (Bug #25026)
 - Changing the value of [MI_KEY_BLOCK_LENGTH](#) in [myisam.h](#) and recompiling MySQL resulted in a [myisamchk](#) that saw existing [MyISAM](#) tables as corrupt. (Bug #22119)
 - A stored routine containing semicolon in its body could not be reloaded from a dump of a binary log. (Bug #20396)
 - For [SET](#), [SELECT](#), and [DO](#) statements that invoked a stored function from a database other than the default database, the function invocation could fail to be replicated. (Bug #19725)
 - For [ENUM](#) columns defined such that enumeration values contained commas, the commas were mapped to [0xff](#). (Bug #24660)
- Upgrade note: The fix for this problem affects tables containing [ENUM](#) columns that actually do have [0xff](#) in enumeration values. Such tables should be dumped using [mysqldump](#) with the current server before upgrading to MySQL 5.1.15 or higher and reloading the tables.
- [SET lc_time_names = value](#) allowed only exact literal values, not expression values. (Bug #22647)
 - Changes to the [lc_time_names](#) system variable were not replicated. (Bug #22645)
 - [SELECT ... FOR UPDATE](#), [SELECT ... LOCK IN SHARE MODE](#), [DELETE](#), and [UPDATE](#) statements executed using a full table scan were not releasing locks on rows that did not satisfy the [WHERE](#) condition. (Bug #20390)
 - A stored procedure, executed from a connection using a binary character set, and which wrote multibyte data, would write incorrectly escaped entries to the binary log. This caused syntax errors, and caused replication to fail. (Bug #23619, Bug #24492)
 - [mysqldump --order-by-primary](#) failed if the primary key name was an identifier that required quoting. (Bug #13926)
 - Re-execution of [CREATE DATABASE](#), [CREATE TABLE](#), and [ALTER TABLE](#) statements in stored routines or as prepared statements caused incorrect results. (Bug #22060)
 - The internal functions for table preparation, creation, and alteration were not re-execution friendly, causing problems in code that: repeatedly altered a table; repeatedly created and dropped a table; opened and closed a cursor on a table, altered the table, and then reopened the cursor. (Bug #4968, Bug #6895, Bug #19182, Bug #19733)
 - A workaround was implemented to avoid a race condition in the NPTL [pthread_exit\(\)](#) implementation. (Bug #24507)
 - The Instance Manager [DROP INSTANCE](#) command did not work. (Bug #23476)
 - The Instance Manager [STOP INSTANCE](#) command took too much time and caused Instance Manager to be unresponsive. (Bug #23215)
 - The Instance Manager [STOP INSTANCE](#) command could not be applied to instances in the [Crashed](#), [Failed](#), or [Abandoned](#) state. (Bug #22306)
 - Instance Manager could crash during shutdown. (Bug #19044)
 - A deadlock could occur, with the server hanging on [Closing tables](#), with a sufficient number of concurrent [INSERT DELAYED](#), [FLUSH TABLES](#), and [ALTER TABLE](#) operations. (Bug #23312)
 - A user-defined variable could be assigned an incorrect value if a temporary table was employed in obtaining the result of the query used to determine its value. (Bug #16861)
 - The optimizer removes expressions from [GROUP BY](#) and [DISTINCT](#) clauses if they happen to participate in [expression = constant](#) predicates of the [WHERE](#) clause, the idea being that, if the expression is equal to a constant, then it cannot take on multiple values. However, for predicates where the expression and the constant item are of different result types (for example, when a string column is compared to 0), this is not valid, and can lead to invalid

results in such cases. The optimizer now performs an additional check of the result types of the expression and the constant; if their types differ, then the expression is not removed from the [GROUP BY](#) list. (Bug #15881)

- Referencing an ambiguous column alias in an expression in the [ORDER BY](#) clause of a query caused the server to crash. (Bug #25427)
- Some [CASE](#) statements inside stored routines could lead to excessive resource usage or a crash of the server. (Bug #24854, Bug #19194)
- Some joins in which one of the joined tables was a view could return erroneous results or crash the server. (Bug #24345)
- [OPTIMIZE TABLE](#) tried to sort R-tree indexes such as spatial indexes, although this is not possible (see [OPTIMIZE TABLE](#) 構文). (Bug #23578)
- User-defined variables could consume excess memory, leading to a crash caused by the exhaustion of resources available to the [MEMORY](#) storage engine, due to the fact that this engine is used by MySQL for variable storage and intermediate results of [GROUP BY](#) queries. Where [SET](#) had been used, such a condition could instead give rise to the misleading error message [You may only use constant expressions with SET](#), rather than [Out of memory \(Needed NNNNNN bytes\)](#). (Bug #23443)
- [InnoDB](#): During a restart of the MySQL Server that followed the creation of a temporary table using the [InnoDB](#) storage engine, MySQL failed to clean up in such a way that [InnoDB](#) still attempted to find the files associated with such tables. (Bug #20867)
- Under some circumstances, a [REORGANIZE PARTITION](#) statement could crash [mysqld](#). (Bug #24502)
- A multi-table [DELETE QUICK](#) could sometimes cause one of the affected tables to become corrupted. (Bug #25048)
- A compressed [MyISAM](#) table that became corrupted could crash [myisamchk](#) and possibly the MySQL Server. (Bug #23139)
- Using [INSTALL PLUGIN](#) followed by a restart of the server caused an error due to memory not being properly initialized. (Bug #22694)
- A crash of the MySQL Server could occur when unpacking a [BLOB](#) column from a row in a corrupted [MyISAM](#) table. This could happen when trying to repair a table using either [REPAIR TABLE](#) or [myisamchk](#); it could also happen when trying to access such a 「broken」 row using statements like [SELECT](#) if the table was not marked as crashed. (Bug #22053)
- The [FEDERATED](#) storage engine did not support the [euckr](#) character set. (Bug #21556)
- The [FEDERATED](#) storage engine did not support the [utf8](#) character set. (Bug #17044)
- [mysql_upgrade](#) failed if the [--password](#) (or [-p](#)) option was given. (Bug #24896)
- For a nonexistent table, [DROP TEMPORARY TABLE](#) failed with an incorrect error message if [read_only](#) was enabled. (Bug #22077)
- The code for generating [USE](#) statements for binary logging of [CREATE PROCEDURE](#) statements resulted in confusing output from [mysqlbinlog](#) for [DROP PROCEDURE](#) statements. (Bug #22043)
- The [REPEAT\(\)](#) function could return [NULL](#) when passed a column for the count argument. (Bug #24947)
- Accuracy was improved for comparisons between [DECIMAL](#) columns and numbers represented as strings. (Bug #23260)
- [InnoDB](#) crashed while performing XA recovery of prepared transactions. (Bug #21468)
- [ROW_COUNT\(\)](#) did not work properly as an argument to a stored procedure. (Bug #23760)
- It was possible to use [DATETIME](#) values whose year, month, and day parts were all zeroes but whose hour, minute, and second parts contained nonzero values, an example of such an illegal [DATETIME](#) being '0000-00-00 11:23:45'. (Bug #21789)
- It was possible to set the backslash character (「 \ 」) as the delimiter character using [DELIMITER](#), but not actually possible to use it as the delimiter. (Bug #21412)

- [ALTER ENABLE KEYS](#) or [ALTER TABLE DISABLE KEYS](#) combined with another [ALTER TABLE](#) option other than [RENAME TO](#) did nothing. In addition, if [ALTER TABLE](#) was used on a table having disabled keys, the keys of the resulting table were enabled. (Bug #24395)
- An [ALTER TABLE](#) statement that used a [RENAME](#) clause in combination with a [MODIFY](#) or [CHANGE](#) that did not actually change the table (for example, when it changed a column's type from [INT](#) to [INT](#)). The behavior caused by this bug differed according to whether or not the storage engine used by the table was transactional or non-transactional. For transactional tables (such as those using the [InnoDB](#) storage engine), the statement simply failed; for non-transactional tables (such as those using the [MyISAM](#) storage engine), the [ALTER TABLE](#) statement succeeding renaming the table, but subsequent [SELECT](#) statements against the renamed table would fail. (Bug #22369)
- Queries of the form [SELECT ... WHERE string = ANY\(...\)](#) failed when the server used a single-byte character set and the client used a multi-byte character set. (Bug #20835)
- A partitioned table that used the [DATA DIRECTORY](#) option, where the data directory was the same as the directory in which the table definition file resided, became corrupted following [ALTER TABLE ENGINE=ARCHIVE](#). This was actually due to an issue with the [ARCHIVE](#) storage engine, and not with partitioned tables in general. (Bug #22634)
- [NDB Cluster](#) (Cluster APIs): Deletion of an [Ndb_cluster_connection](#) object took a very long time. (Bug #25487)
- [NDB Cluster](#) (Replication): Certain errors in replication setups could lead to subsequent node failures. (Bug #25755)
- [NDB Cluster](#) (Replication): Connecting a [mysqld](#) to a cluster where not all nodes were running, starting the remaining cluster nodes, and then disconnecting from the cluster caused the [mysqld](#) process to crash. (Bug #25387)
- [NDB Cluster](#) (Replication): Connecting an API node to the cluster during a node restart while performing database operations could cause the restarting node to fail. (Bug #25329)
- [NDB Cluster](#) (Disk Data): Following 3 or more missed local checkpoints by a cluster node, a restart of the node caused incorrect undo information to be used for Disk Data tables. (Bug #25636)
- [NDB Cluster](#) (Disk Data): Issuing a [TRUNCATE](#) statement on a Disk Data table caused the table to become an in-memory table. (Bug #25296)
- [NDB Cluster](#) (Disk Data): Changing a column specification on a Disk Data table caused the table to become an in-memory table. (Bug #24667)
- [NDB Cluster](#): It was not possible to create an [NDB](#) table with a key on two [VARCHAR](#) columns where both columns had a storage length in excess of 256. (Bug #25746)
- [NDB Cluster](#): Hosts in clusters with a large number of nodes could experience excessive CPU usage while obtaining configuration data. (Bug #25711)
- [NDB Cluster](#): In some circumstances, shutting down the cluster could cause connected [mysqld](#) processes to crash. (Bug #25668)
- [NDB Cluster](#): Non-32-bit, non-aligned columns were not handled correctly in explicitly partitioned [NDB](#) tables. (Bug #25587)
- [NDB Cluster](#): Some aggregate queries such as [SELECT COUNT\(*\)](#) performed a table scan on [NDB](#) tables rather than checking table statistics, causing such queries to perform much more slowly in MySQL Cluster 5.1 than in 5.0. (Bug #25567)
- [NDB Cluster](#): Memory allocations for [TEXT](#) columns were calculated incorrectly, resulting in space being wasted and other issues. (Bug #25562)
- [NDB Cluster](#): The failure of a master node during a node restart could lead to a resource leak, causing later node failures. (Bug #25554)
- [NDB Cluster](#): The failure of a node during a local checkpoint could lead to other node failures. (Bug #25468)
- [NDB Cluster](#): A node shutdown occurred if the master failed during a commit. (Bug #25364)
- [NDB Cluster](#): Creating a non-unique index with the [USING HASH](#) clause silently created an ordered index instead of issuing a warning. (Bug #24820)

- **NDB Cluster**: The management server did not handle logging of node shutdown events correctly in certain cases. (Bug #22013)
- **NDB Cluster**: A potential memory leak in the **NDB** storage engine's handling of file operations was uncovered. (Bug #21858)
- **NDB Cluster**: When stopping and restarting multiple data nodes, the last node to be restarted would sometimes hang in Phase 100. (Bug #19645)
- **NDB Cluster** (NDB API): Invoking the **NdbTransaction::execute()** method using execution type **Commit** and abort option **AO_IgnoreError** could lead to a crash of the transaction coordinator (**DBTC**). (Bug #25090)
- **NDB Cluster** (NDB API): A unique index lookup on a non-existent tuple could lead to a data node timeout (error 4012). (Bug #25059)
- **NDB Cluster** (NDB API): Due to an error in the computation of table fragment arrays, some transactions might not be executed from the correct starting point. (Bug #24914)
- **NDB Cluster** (Replication): Following a restart of the master cluster, the latest GCI was set to 0 upon reconnection to the slave. (Bug #21806)
- **NDB Cluster** (Disk Data): A **MEDIUMTEXT** column of a Disk Data table was stored in memory rather than on disk, even if the column was not indexed. (Bug #25001)
- **NDB Cluster** (Disk Data): Performing a node restart with a newly dropped Disk Data table could lead to failure of the node during the restart. (Bug #24917)
- **NDB Cluster** (Disk Data): When restoring from backup a cluster containing any Disk Data tables with hidden primary keys, a node failure resulted which could lead to a crash of the cluster. (Bug #24166)
- **NDB Cluster** (Disk Data): Repeated **CREATE**, **DROP**, or **TRUNCATE** in various combinations with system restarts between these operations could lead to the eventual failure of a system restart. (Bug #21948)
- **NDB Cluster** (Disk Data): Extents that should have been available for re-use following a **DROP TABLE** operation were not actually made available again until after the cluster performed a local checkpoint. (Bug #17605)
- **NDB Cluster**: Under certain rare circumstances, local checkpoints were not performed properly, leading to an inability to restart one or more data nodes. (Bug #24664)
- **NDB Cluster** (NDB API): When using the **NdbTransaction::execute()** method, a very long timeout (greater than 5 minutes) could result if the last data node being polled was disconnected from the cluster. (Bug #24949)
- **NDB Cluster**: When a data node was shut down using the management client **STOP** command, a connection event (**NDB_LE_Connected**) was logged instead of a disconnection event (**NDB_LE_Disconnected**). (Bug #22773)
- **NDB Cluster**: **SELECT** statements with a **BLOB** or **TEXT** column in the selected column list and a **WHERE** condition including a primary key lookup on a **VARCHAR** primary key produced empty result sets. (Bug #19956)
- **NDB Cluster**: **ndb_config** failed when trying to use 2 management servers and node IDs. (Bug #23887)
- **STR_TO_DATE()** returned **NULL** if the format string contained a space following a non-format character. (Bug #22029)
- **yaSSL** crashed on pre-Pentium Intel CPUs. (Bug #21765)
- Selecting into variables sometimes returned incorrect wrong results. (Bug #20836)
- Inserting **DEFAULT** into a column with no default value could result in garbage in the column. Now the same result occurs as when inserting **NULL** into a **NOT NULL** column. (Bug #20691)
- **mysql_fix_privilege_tables.sql** altered the **table_privs.table_priv** column to contain too few privileges, causing loss of the **CREATE VIEW** and **SHOW VIEW** privileges. (Bug #20589)
- A server crash occurred when using **LOAD DATA** to load a table containing a **NOT NULL** spatial column, when the statement did not load the spatial column. Now a **NULL supplied to NOT NULL column** error occurs. (Bug #22372)
- Unsigned **BIGINT** values treated as signed values by the **MOD()** function. (Bug #19955)

- Compiling PHP 5.1 with the MySQL static libraries failed on some versions of Linux. (Bug #19817)
- The `DELIMITER` statement did not work correctly when used in an SQL file run using the `SOURCE` statement. (Bug #19799)
- `VARBINARY` column values inserted on a MySQL 4.1 server had trailing zeroes following upgrade to MySQL 5.0 or later. (Bug #19371)
- Subqueries of the form `NULL IN (SELECT ...)` returned invalid results. (Bug #8804, Bug #23485)
- The `--extern` option for `mysql-test-run.pl` did not function correctly. (Bug #24354)
- `ALTER TABLE` statements that performed both `RENAME TO` and `{ENABLE|DISABLE} KEYS` operations caused a server crash. (Bug #24089)
- The MySQL 5.1.12 binaries for Windows were missing the `FEDERATED`, `EXAMPLE`, and `BLACKHOLE` storage engines. (Bug #23900)
- `myisampack` wrote to unallocated memory, causing a crash. (Bug #17951)
- Some small double precision numbers (such as `1.00000001e-300`) that should have been accepted were truncated to zero. (Bug #22129)
- The `mysql.server` script used the `source` command, which is less portable than the `.` command; it now uses `.` instead. (Bug #24294)
- `DATE_ADD()` requires complete dates with no 「zero」 parts, but sometimes did not return `NULL` when given such a date. (Bug #22229)
- Using `FLUSH TABLES` in one connection while another connection is using `HANDLER` statements caused a server crash. (Bug #21587)
- `FLUSH LOGS` or `mysqladmin flush-logs` caused a server crash if the binary log was not open. (Bug #17733)
- On HP-UX, `mysqltest` (non-thread-safe) crashed due to being linked against a thread-safe `libmysys` library. (Bug #23984)

Changes in release 5.1.14 (05 December 2006)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on production level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise/>をご覧ください。

Functionality added or changed:

- Incompatible change: Previously, you could create a user-defined function (UDF) or stored function with the same name as a built-in function, but could not invoke the UDF. Now an error occurs if you try to create such a UDF. The server also now generates a warning if you create a stored function with the same name as a built-in function. It is not considered an error to create a stored function with the same name as a built-in function because you can invoke the function using `db_name.func_name()` syntax. However, the server now generates a warning in this case. (Bug #22619, Bug #18239)

See [関数名の構文解析と名前解決](#), for the rules describing how the server interprets references to different kinds of functions.
- Incompatible change: The `prepared_stmt_count` system variable has been converted to the `Prepared_stmt_count` global status variable (viewable with the `SHOW GLOBAL STATUS` statement). (Bug #23159)

- **NDB Cluster:** Two major changes have taken place with regard to the MySQL Cluster system tables. These are:
 1. Incompatible change: The `cluster` database is no longer used. The tables formerly found in the `cluster` database are now in the `mysql` database, and have been renamed as `ndb_binlog_index`, `ndb_apply_status`, and `ndb_schema`.
 2. The `mysql.ndb_apply_status` and `mysql.ndb_schema` tables (formerly `cluster.apply_status` and `cluster.schema` are now created by `ndb_restore`, in the event that they do not already exist on the slave cluster. (Bug #14612)

Note: When upgrading from versions of MySQL previous to 5.1.14 to 5.1.14 or later, `mysql_fix_privilege_tables` merely creates a new `mysql.ndb_binlog_index` table, but does not remove the existing `cluster` database (or, if upgrading from MySQL 5.1.7 or earlier, the existing `cluster_replication` database), nor any of the tables in it.

For more information, see [レプリケーション スキーマおよびテーブル](#).

- **NDB Cluster:** It is now possible to create a unique hashed index on a column that is not defined as `NOT NULL`. Note that this change applies only to tables using the `NDB` storage engine.

Unique indexes on columns in `NDB` tables do not store null values because they are mapped to primary keys in an internal index table (and primary keys cannot contain nulls).

Normally, an additional ordered index is created when one creates unique indexes on `NDB` table columns; this can be used to search for `NULL` values. However, if `USING HASH` is specified when such an index is created, no ordered index is created.

The reason for permitting unique hash indexes with null values is that, in some cases, the user wants to save space if a large number of records are pre-allocated but not fully initialized. This also assumes that the user will not try to search for null values. Since MySQL does not support indexes that are not allowed to be searched in some cases, the `NDB` storage engine uses a full table scan with pushed conditions for the referenced index columns to return the correct result.

Note that a warning is returned if one creates a unique nullable hash index, since the query optimizer should be provided a hint not to use it with `NULL` values if this can be avoided.

(Bug #21507)

- **NDB Cluster:** Backup messages are now printed to the Cluster log. (Bug #24544)
- **NDB Cluster:** The error message `Management server closed connection`, when recorded in the MySQL error log, now includes a timestamp indicating when the error took place. (Bug #21519)
- **NDB Cluster (Disk Data):** The output of `mysqldump` now includes by default all tablespace and logfile group definitions used by any tables or databases that are dumped. (Bug #20839)

Note: The working of the `--all-tablespaces` or `-Y` option for `mysqldump` remains unaffected by this change.

- Direct and indirect usage of stored routines, user-defined functions, and table references is now prohibited in `CREATE EVENT` and `ALTER EVENT` statements. (Bug #22830)

See [CREATE EVENT Syntax](#), and [ALTER EVENT Syntax](#), for more specific information.

- `DROP TRIGGER` now supports an `IF EXISTS` clause. (Bug #23703)

Bugs fixed:

- **NDB Cluster:** If the value set for `MaxNoOfAttributes` is excessive, a suitable error message is now returned. (Bug #19352)
- **NDB Cluster:** Setting the configuration parameter `LockPagesInMainMemory` had no effect. (Bug #24461)
- **NDB Cluster:** Multiple occurrences of error conditions were logged with duplicated error messages rather than being being reported with a single error message stating that the error was encountered `N` times. (Bug #22313)
- **NDB Cluster:** Sudden disconnection of an SQL or data node could lead to shutdown of data nodes with the error `failed ndbrequire`. (Bug #24447)

- **NDB Cluster:** Different error messages were returned for similar cases involving failure to allocate memory for Cluster operations. (Bug #19203)
- **NDB Cluster:** Some values of `MaxNoOfTriggers` could cause the server to become inaccessible following startup of the data nodes. (Bug #19454)
- **NDB Cluster (Replication):** If errors occurred during purging of the binary logs, extraneous rows could remain left in the `binlog_index` table. (Bug #15021)
- **NDB Cluster (Disk Data):** `ndb_restore` could sometimes fail when attempting to restore Disk Data tables due to data node failure caused by accessing uninitialized memory. (Bug #24331)
- **NDB Cluster (Disk Data):** Excessive fragmentation of Disk Data files (including log files and data files) could occur during the course of normal use. (Bug #24143)
- **NDB Cluster (Disk Data):** It was possible to execute a statement for creating a Disk Data table that referred to a nonexistent tablespace, in which case the table was an in-memory **NDB** table. Such a statement instead now fails with an appropriate error message. (Bug #23576)
- **NDB Cluster (Disk Data):** Under some circumstances, a `DELETE` from a Disk Data table could cause `mysqld` to crash. (Bug #23542)
- **NDB Cluster (Cluster APIs):** Using `BIT` values with any of the comparison methods of the `NdbScanFilter` class caused the cluster's data nodes to fail. (Bug #24503)
- **NDB Cluster:** A value equal to or greater than the allowed maximum for `LongMessageBuffer` caused all data nodes to crash. (Bug #22547)
- **NDB Cluster:** The failure of a data node failure during a schema operation could lead to additional node failures. (Bug #24752)
- **NDB Cluster:** A committed read could be attempted before a data node had time to connect, causing a timeout error. (Bug #24717)
- **NDB Cluster:** The simultaneous shutdown of `mysqld` and `ndbd` processes caused unnecessary locking. (Bug #24655)
- **NDB Cluster:** The failure of the master node in a node group during the allocation of node IDs could cause `ndb_mgmd` to hang. (Bug #24543)
- **NDB Cluster:** In certain rare cases, a data node could crash due to a typographical error in the MySQL Cluster source code. (Bug #24476)
- **NDB Cluster:** Creating a new tables containing a `BLOB` column when the server was short of memory could cause the server to crash. (Bug #24470)
- **NDB Cluster:** Any statement following the execution of `CREATE TABLE ... LIKE ndb_table` (where `ndb_table` was a table using the **NDB** storage engine), would cause the `mysql` client to hang. (Bug #24301)
- **NDB Cluster:** When the management client command `ALL RESTART -i` was executed while one data node was not running, all data nodes in the cluster were shut down. (Bug #24105)
- **NDB Cluster:** A query using an index scan followed by a delete operation, and then a rollback could cause one or more data nodes to crash. (Bug #24039)
- **NDB Cluster (Cluster APIs):** Some MGM API function calls could yield incorrect return values in certain cases where the cluster was operating under a very high load, or experienced timeouts in inter-node communications. (Bug #24011)
- **NDB Cluster:** It was possible for the sum of the `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` configuration parameters, plus the number of system tables to exceed the maximum value for a `UInt32` number. In such a case, the cluster's data nodes failed to start, and no reason for this could easily be determined from the error messages provided. (Bug #22548)
- **NDB Cluster:** Given a table `mytbl` in a database `mydb` on a MySQL Server acting as an SQL node in a MySQL Cluster, then, following multiple `ALTER TABLE mytbl ENGINE=engine` statements — first, to change the storage

engine used for a table to **NDB**, and then again to change the table to use a non-**NDB** storage engine — a **DROP DATABASE mydb** statement executed on any SQL node in the cluster would cause **mydb** to be dropped on all SQL nodes in the cluster, even if **mydb** contained non-**NDB** tables. (Bug #21495)

- **NDB Cluster**: An incorrect error message was displayed in the event that the value of the **MaxNoOfOrderedIndexes** parameter was set too low. (Bug #20065)
- **NDB Cluster**: An incorrect error message was displayed in the event that the value of the **DataMemory** parameter was insufficient for the amount of data to be stored by the cluster. (Bug #19808)
- **NDB Cluster**: A unique constraint violation was not ignored by an **UPDATE IGNORE** statement when the constraint violation occurred on a non-primary key. (Bug #18487, Bug #24303)
- The stack size for NetWare binaries was increased to 128KB to prevent problems caused by insufficient stack size. (Bug #23504)
- Attempting to use a view containing **DEFINER** information for a non-existent user resulted in an error message that revealed the definer account. Now the definer is revealed only to superusers. Other users receive only an **access denied** message. (Bug #17254)
- The size of **MEMORY** tables and internal temporary tables was limited to 4GB on 64-bit Windows systems. (Bug #24052)
- For debug builds, **mysqladmin shutdown** displayed an extraneous **skipped 9 bytes from file: socket (3)** message. (Bug #21428)
- For queries that select from a view, the server was returning **MYSQL_FIELD** metadata inconsistently for view names and table names. For view columns, the server now returns the view name in the **table** field and, if the column selects from an underlying table, the table name in the **org_table** field. (Bug #20191)
- **CREATE FUNCTION X()** and **CREATE FUNCTION Y()** failed with a syntax error instead of warning the user that these function names are already used (for GIS functions). (Bug #21025)
- The loose index scan optimization for **GROUP BY** with **MIN** or **MAX** was not applied within other queries, such as **CREATE TABLE ... SELECT ...**, **INSERT ... SELECT ...**, or in the **FROM** clauses of subqueries. (Bug #24156)
- Queries using a column alias in an expression as part of an **ORDER BY** clause failed, an example of such a query being **SELECT mycol + 1 AS mynum FROM mytable ORDER BY 30 - mynum**. (Bug #22457)
- Trailing spaces were not removed from Unicode **CHAR** column values when used in indexes. This resulted in excessive usage of storage space, and could affect the results of some **ORDER BY** queries that made use of such indexes.

Note: When upgrading, it is necessary to re-create any existing indexes on Unicode **CHAR** columns in order to take advantage of the fix. This can be done by using a **REPAIR TABLE** statement on each affected table.

(Bug #22052)

- Warnings were generated when explicitly casting a character to a number (for example, **CAST('x' AS SIGNED)**), but not for implicit conversions in simple arithmetic operations (such as **'x' + 0**). Now warnings are generated in all cases. (Bug #11927)
- **BENCHMARK**, **ENCODE**, **DECODE**, and **FORMAT** could only accept a constant for some parameters, and could not be used in prepared statements. (Bug #22684)
- A query with a subquery that references columns of a view from the outer **SELECT** could return an incorrect result if used from a prepared statement. (Bug #20327)
- Constant expressions and some numeric constants used as input parameters to user-defined functions were not treated as constants. (Bug #18761)
- **INET_ATON()** returned a signed **BIGINT** value, not an unsigned value. (Bug #21466)
- In some cases, the parser failed to distinguish a user-defined function from a stored function. (Bug #21809)

- In some cases, a function that should be parsed as a user-defined function was parsed as a stored function. (Bug #24736)
- Subqueries for which a pushed-down condition did not produce exactly one key field could cause a server crash. (Bug #24056)
- `LAST_DAY('0000-00-00')` could cause a server crash. (Bug #23653)
- Through the C API, the member strings in `MYSQL_FIELD` for a query that contains expressions may return incorrect results. (Bug #21635)
- `mysql_affected_rows()` could return values different from `mysql_stmt_affected_rows()` for the same sequence of statements. (Bug #23383)
- `IN()` and `CHAR()` can return `NULL`, but did not signal that to the query processor, causing incorrect results for `IS NULL` operations. (Bug #17047)
- Instance Manager option-parsing code caused memory-allocation errors. (Bug #22242)
- A trigger that invoked a stored function could cause a server crash when activated by different client connections. (Bug #23651)
- `CONCURRENT` did not work correctly for `LOAD DATA INFILE`. (Bug #20637)
- The server source code had multiple exportable definitions of the `field_in_record_is_null()` function. These are now all declared `static`. (Bug #24190)
- Inserting a default or invalid value into a spatial column could fail with `Unknown error` rather than a more appropriate error. (Bug #21790)
- The server could send incorrect column count information to the client for queries that produce a larger number of columns than can fit in a two-byte number. (Bug #19216)
- Evaluation of subqueries that require the filesort algorithm were allocating and freeing the `sort_buffer_size` buffer many times, resulting in slow performance. Now the buffer is allocated once and reused. (Bug #21727)
- SQL statements close to the size of `max_allowed_packet` could produce binary log events larger than `max_allowed_packet` that could not be read by slave servers. (Bug #19402)
- View columns were always handled as having implicit derivation, leading to `illegal mix of collation errors` for some views in `UNION` operations. Now view column column derivation comes from the original expression given in the view definition. (Bug #21505)
- If elements in a non-top-level `IN` subquery were accessed by an index and the subquery result set included a `NULL` value, the quantified predicate that contained the subquery was evaluated to `NULL` when it should return a non-`NULL` value. (Bug #23478)
- Calculation of `COUNT(DISTINCT)`, `AVG(DISTINCT)`, or `SUM(DISTINCT)` when they are referenced more than once in a single query with `GROUP BY` could cause a server crash. (Bug #23184)
- For a cast of a `DATETIME` value containing microseconds to `DECIMAL`, the microseconds part was truncated without generating a warning. Now the microseconds part is preserved. (Bug #19491)
- Metadata for columns calculated from scalar subqueries was limited to integer, double, or string, even if the actual type of the column was different. (Bug #11032)
- The result for `CAST()` when casting a value to `UNSIGNED` was limited to the maximum signed `BIGINT` value, not the maximum unsigned value. (Bug #8663)
- Some unnecessary Valgrind warnings were removed from the server. (Bug #24488, Bug #24533).
- Using `EXPLAIN` caused a server crash for queries that selected from `INFORMATION_SCHEMA` in a subquery in the `FROM` clause. (Bug #22413)
- Invalidating the query cache caused a server crash for `INSERT INTO ... SELECT` statements that selected from a view. (Bug #20045)

- With row-based binary logging, replicated multiple-statement transaction deadlocks did not return the correct error code, causing the slave SQL thread to stop rather than roll back and re-execute. (Bug #23831)
- With row-based binary logging, for `CREATE TABLE IF NOT EXISTS LIKE temporary_table` statements, the `IF NOT EXISTS` clause was not logged. (Bug #22762)
- With row-based binary logging, `CREATE TABLE IF NOT EXISTS SELECT` statements were not logged properly. (Bug #22027)
- On slave servers, transactions that exceeded the lock wait timeout failed to roll back properly. (Bug #20697)
- Changes to character set variables prior to an action on a replication-ignored table were forgotten by slave servers. (Bug #22877)
- With `lower_case_table_names` set to 1, `SHOW CREATE TABLE` printed incorrect output for table names containing Turkish I (LATIN CAPITAL LETTER I WITH DOT ABOVE). (Bug #20404)
- When applying the `group_concat_max_len` limit, `GROUP_CONCAT()` could truncate multi-byte characters in the middle. (Bug #23451)
- For view renaming, the table name to filename encoding was not performed. (Bug #21370)
- For some problems relating to character set conversion or incorrect string values for `INSERT` or `UPDATE`, the server was reporting truncation or length errors instead. (Bug #18908)
- The XPath operators `<` and `>`, as implemented in the `ExtractValue()` function, operated in reverse. (Bug #22823)

Changes in release 5.1.13 (Not released)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on production level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise/>をご覧ください。

Functionality added or changed:

- Incompatible change: The number of function names affected by `IGNORE_SPACE` was reduced significantly in MySQL 5.1.13, from about 200 to about 30. (For details about `IGNORE_SPACE`, see [関数名の構文解析と名前解決](#).) This change improves the consistency of parser operation. However, it also introduces the possibility of incompatibility for old SQL code that relies on the following conditions:

- `IGNORE_SPACE` is disabled.
- The presence or absence of whitespace following a function name is used to distinguish between a built-in function and stored function that have the same name (for example, `PI()` versus `PI ()`).

For functions that are no longer affected by `IGNORE_SPACE` as of MySQL 5.1.13, that strategy no longer works. Either of the following approaches can be used if you have code that is subject to the preceding incompatibility:

- If a stored function has a name that conflicts with a built-in function, refer to the stored function with a schema name qualifier, regardless of whether whitespace is present. For example, write `schema_name.PI()` or `schema_name.PI ()`.
- Alternatively, rename the stored function to use a non-conflicting name and change invocations of the function to use the new name.

(Bug #21114)

- If the user specified the server options `--max-connections=N` or `--table-open-cache=M`, a warning would be given in some cases that some values were recalculated, with the result that `--table-open-cache` could be assigned greater value.

It should be noted that, in such cases, both the warning and the increase in the `--table-open-cache` value were completely harmless. Note also that it is not possible for the MySQL Server to predict or to control limitations on the maximum number of open files, since this is determined by the operating system.

The recalculation code has now been fixed to ensure that the value of `--table-open-cache` is no longer increased automatically, and that a warning is now given only if some values had to be decreased due to operating system limits.

(Bug #21915)

- Binary distributions of MySQL 5.1.12 were built without support for partitioning. This has been corrected except for NetWare. (Bug #23949)
- A change in the interfaces for the `INFORMATION_SCHEMA.FILES` table has made the table accessible to storage engines other than `NDB`. (Bug #23013)
- `mysqldump --single-transaction` now uses `START TRANSACTION /*!40100 WITH CONSISTENT SNAPSHOT */` rather than `BEGIN` to start a transaction, so that a consistent snapshot will be used on those servers that support it. (Bug #19660)
- `mysql_upgrade` now passes all the parameters specified on the command line to both `mysqlcheck` and `mysql` using the `upgrade_defaults` file. (Bug #20100)
- For the `CALL` statement, stored procedures that take no arguments now can be invoked without parentheses. That is, `CALL p()` and `CALL p` are equivalent. (Bug #21462)

Bugs fixed:

- MySQL 5.0.26 introduced an ABI incompatibility, which this release reverts. Programs compiled against 5.0.26 are not compatible with any other version and must be recompiled. (Bug #23427)
- If an `init_connect` SQL statement produced an error, the connection was silently terminated with no error message. Now the server writes a warning to the error log. (Bug #22158)
- If a table contains an `AUTO_INCREMENT` column, inserting into an insertable view on the table that does not include the `AUTO_INCREMENT` column should not change the value of `LAST_INSERT_ID()`, because the side effects of inserting default values into columns not part of the view should not be visible. MySQL was incorrectly setting `LAST_INSERT_ID()` to zero. (Bug #22584)
- `M % 0` returns `NULL`, but `(M % 0) IS NULL` evaluated to false. (Bug #23411)
- Within a stored routine, a view definition cannot refer to routine parameters or local variables. However, an error did not occur until the routine was called. Now it occurs during parsing of the routine creation statement. (Bug #20953)

Note: A side effect of this fix is that if you have already created such routines, an error will occur if you execute `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION`. You should drop these routines because they are erroneous.

- A client library crash was caused by executing a statement such as `SELECT * FROM t1 PROCEDURE ANALYSE()` using a server side cursor on a table `t1` that does not have the same number of columns as the output from `PROCEDURE ANALYSE()`. (Bug #17039)
- `mysql` did not check for errors when fetching data during result set printing. (Bug #22913)
- Adding a day, month, or year interval to a `DATE` value produced a `DATE`, but adding a week interval produced a `DATETIME` value. Now all produce a `DATE` value. (Bug #21811)
- The column default value in the output from `SHOW COLUMNS` or `SELECT FROM INFORMATION_SCHEMA.COLUMNS` was truncated to 64 characters. (Bug #23037)
- For not-yet-authenticated connections, the `Time` column in `SHOW PROCESSLIST` was a random value rather than `NULL`. (Bug #23379)

- The `Host` column in `SHOW PROCESSLIST` output was blank when the server was started with the `--skip-grant-tables` option. (Bug #22728)
- The `Handler_rollback` status variable sometimes was incremented when no rollback had taken place. (Bug #22728)
- Within a prepared statement, `SELECT (COUNT(*) = 1)` (or similar use of other aggregate functions) did not return the correct result for statement re-execution. (Bug #21354)
- Lack of validation for input and output `TIME` values resulted in several problems: `SEC_TO_TIME()` within subqueries incorrectly clipped large values; `SEC_TO_TIME()` treated `BIGINT UNSIGNED` values as signed; only truncation warnings were produced when both truncation and out-of-range `TIME` values occurred. (Bug #11655, Bug #20927)
- Range searches on columns with an index prefix could miss records. (Bug #20732)
- With `SQL_MODE=TRADITIONAL`, MySQL incorrectly aborted on warnings within stored routines and triggers. (Bug #20028)
- In `mysql`, invoking `connect` or `\r` with very long `db_name` or `host_name` parameters caused buffer overflow. (Bug #20894)
- `mysqldump --xml` produced invalid XML for `BLOB` data. (Bug #19745)
- For a debug server, a reference to an undefined user variable in a prepared statment executed with `EXECUTE` caused an assertion failure. (Bug #19356)
- Within a trigger for a base table, selecting from a view on that base table failed. (Bug #19111)
- `DELETE IGNORE` could hang for foreign key parent deletes. (Bug #18819)
- Transient errors in replication from master to slave may trigger multiple `Got fatal error 1236: 'binlog truncated in the middle of event'` errors on the slave. (Bug #4053)
- The value of the `warning_count` system variable was not being calculated correctly (also affecting `SHOW COUNT(*) WARNINGS`). (Bug #19024)
- `InnoDB` showed substandard performance with multiple queries running concurrently. (Bug #15815)
- There was a race condition in the `InnoDB fil_flush_file_spaces()` function. (Bug #24098)
- `FROM_UNIXTIME()` did not accept arguments up to `POWER(2,31)-1`, which it had previously. (Bug #9191)
- Some yaSSL-related memory leaks detected by Valgrind were fixed. (Bug #23981)
- If `COMPRESS()` returned `NULL`, subsequent invocations of `COMPRESS()` within a result set or within a trigger also returned `NULL`. (Bug #23254)
- `mysql` would lose its connection to the server if its standard output was not writable. (Bug #17583)
- `mysql-test-run` did not work correctly for RPM-based installations. (Bug #17194)
- The return value from `my_seek()` was ignored. (Bug #22828)
- Use of `PREPARE` with a `CREATE PROCEDURE` statement that contained a syntax error caused a server crash. (Bug #21868)
- Use of a DES-encrypted SSL certificate file caused a server crash. (Bug #21868)
- Column names were not quoted properly for replicated views. (Bug #19736)
- `InnoDB` used table locks (not row locks) within stored functions. (Bug #18077)
- `InnoDB` crashed when trying to display an error message about a foreign key constraint violation when the two tables are in different schemas. (Bug #23368)
- Statements such as `DROP PROCEDURE` and `DROP VIEW` were written to the binary log too late due to a race condition. (Bug #14262)
- At shutdown, Instance Manager told guarded server instances to stop, but did not wait until they actually stopped. (Bug #17486)

- It was not possible to do an atomic rename of the log tables without the possibility of losing rows. Now you can do this:

```
USE mysql;
CREATE TABLE IF NOT EXISTS general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

(Bug #17544, Bug #21785)

- **NDB Cluster** (Disk Data): In the event of an aborted multiple update, the space in the Disk Data log buffer to be freed as a result was actually freed twice, which could eventually lead to a crash. (Bug #23430)
- **NDB Cluster**: An **NDB** source file included a `memset()` call with reversed arguments. (Bug #23169)
- **NDB Cluster**: Under some conditions, the data distribution could become unbalanced in a MySQL Cluster with 2 or more node groups following the creation of a new table. (Bug #21690)

As part of the fix for this bug, two new NDB API methods were added to the `NdbDictionary::Object::Table` class. See the MySQL Cluster API documentation for details.
- Incorrect warnings occurred for use of `CREATE TABLE ... LIKE` or `REPAIR TABLE` with the log tables. (Bug #21966)
- MySQL failed to build on the Alpha platform. (Bug #23256)
- The optimizer failed to use equality propagation for `BETWEEN` and `IN` predicates with string arguments. (Bug #22753)
- The optimizer used the `ref` join type rather than `eq_ref` for a simple join on strings. (Bug #22367)
- The `WITH CHECK OPTION` for a view failed to prevent storing invalid column values for `UPDATE` statements. (Bug #16813)
- A literal string in a `GROUP BY` clause could be interpreted as a column name. (Bug #14019)
- Some queries that used `MAX()` and `GROUP BY` could incorrectly return an empty result. (Bug #22342)
- `WITH ROLLUP` could group unequal values. (Bug #20825)
- Use of a subquery that invoked a function in the column list of the outer query resulted in a memory leak. (Bug #21798)
- `LIKE` searches failed for indexed `utf8` character columns. (Bug #20471)
- `FLUSH INSTANCES` in Instance Manager triggered an assertion failure. (Bug #19368)
- `ALTER TABLE` was not able to rename a view. (Bug #14959)
- The optimizer sometimes mishandled R-tree indexes for `GEOMETRY` data types, resulting in a server crash. (Bug #21888)
- An unhandled `NULL` pointer caused a server crash. (Bug #22138)
- Use of `SQL_BIG_RESULT` did not influence the sort plan for query execution. (Bug #22781)
- The server did not allocate sufficient memory for some queries for which a `DISTINCT` to `GROUP BY` conversion is possible and an `ORDER BY` clause is present, resulting in a server crash. (Bug #20503)
- The range analysis optimizer did not take into account predicates for which an index could be used after reading `const` tables. In some cases this resulted in non-optimal execution plans. (Bug #19579)
- Entries in the slow query log could have an incorrect `Rows_examined` value. (Bug #12240)
- Insufficient memory (`myisam_sort_buffer_size`) could cause a server crash for several operations on **MyISAM** tables: repair table, create index by sort, repair by sort, parallel repair, bulk insert. (Bug #23175)
- `OPTIMIZE TABLE` with `myisam_repair_threads` > 1 could result in **MyISAM** table corruption. (Bug #8283)
- **NDB Cluster**: Restoring a cluster failed if there were any tables with 128 or more columns. (Bug #23494, Bug #23502)

- **NDB Cluster:** Cluster backups would fail when there were more than 2048 schema objects in the cluster. (Bug #23499)
- **NDB Cluster:** The management client command `ALL DUMP 1000` would cause the cluster to crash if data nodes were connected to the cluster but not yet fully started. (Bug #23203)
- **NDB Cluster:** `INSERT ... ON DUPLICATE KEY UPDATE` on an **NDB** table could lead to deadlocks and memory leaks. (Bug #23200)
- **NDB Cluster:** If a node restart could not be performed from the REDO log, no node takeover took place. This could cause partitions to be left empty during a system restart. (Bug #22893)
- **NDB Cluster:** Multiple node restarts in rapid succession could cause a system restart to fail (Bug #22892), or induce a race condition (Bug #23210).
- **NDB Cluster:** Attempting to create a unique constraint with `USING HASH` on an **NDB** table caused `mysqld` to crash. (Bug #21873)
- **NDB Cluster:** When inserting a row into an **NDB** table with a duplicate value for a non-primary unique key, the error issued would reference the wrong key. (Bug #21072)
- **NDB Cluster (NDB API):** When multiple processes or threads in parallel performed the same ordered scan with exclusive lock and updating the retrieved records, the scan could skip some records, which were not updated as the result. (Bug #20446)
- **NDB Cluster:** Aborting a cluster backup too soon after starting it caused a forced shutdown of the data nodes. (Bug #19148)

Changes in release 5.1.12 (24 October 2006)

コンパイル時の不手際のため、MySQL 5.1.12のバイナリ配布にはNDBクラスタやパーティショニングは含まれていませんでした。ご不便をお掛けし恐縮です。バージョン5.1.14へ更新してください。ソースからコンパイルする場合には、`--with-ndbcluster`、`--with-partition`オプションとともに`configure`を実行して下さい。

This is a new Beta development release, fixing recently discovered bugs.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise>をご覧ください。

Functionality added or changed:

- Incompatible change: Support for the BerkeleyDB (**BDB**) engine has been dropped from this release. Any existing tables that are in BDB format will not be readable from within MySQL from 5.1.12 or newer. You should convert your tables to another storage engine before upgrading to 5.1.12.
- Incompatible change: The namespace for scheduled events has changed, such that events are no longer unique to individual users. This also means that a user with the **EVENT** privilege on a given database can now view, alter, or drop any events defined on that database.

If you used scheduled events in an earlier MySQL 5.1 release, you should rename any of them having the same name and defined on the same database but belonging to different users — so that all events in a given database have unique names — before upgrading to 5.1.12 (or newer).

For additional information, see [The Event Scheduler and MySQL Privileges](#).

- Incompatible change: The permitted values for and behaviour of the `event_scheduler` system variable have changed. Permitted values are now **ON**, **OFF**, and **DISABLED**, with **OFF** being the default. It is not possible to change its value to or from **DISABLED** while the server is running.

For details, see [Event Scheduler Overview](#).

- Incompatible change: The plugin interface has changed: The `st_mysql_plugin` structure has a new `license` member to indicate the license type. (The allowable values are defined in `mysql/plugin.h`.) This change is not backward

compatible, so the API version ([MYSQL_PLUGIN_INTERFACE_VERSION](#)) has changed. For additional information, see [Writing Plugins](#).

- Incompatible change: The full-text parser plugin interface has changed in two ways:
 - The [MYSQL_FTPARSER_PARAM](#) structure has a new `flags` member. This is zero if there are no special flags, or [MYSQL_FTFLAGS_NEED_COPY](#), which means that `mysql_add_word()` must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.)

This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the [mysql_parse\(\)](#) function.

- The `mysql_parse()` and `mysql_add_word()` functions now take a [MYSQL_FTPARSER_PARAM](#) as their first argument, not a [MYSQL_FTPARSER_PARAM::mysql_ftparam](#) as before.

These changes are not backward compatible, so the API version ([MYSQL_FTPARSER_INTERFACE_VERSION](#)) has changed. For additional information, see [Writing Plugins](#).

- Incompatible change: In the [INFORMATION_SCHEMA.EVENTS](#) table, the [EVENT_DEFINITION](#) column now contains the SQL executed by a scheduled event.

The [EVENT_BODY](#) column now contains the language used for the statement or statements shown in [EVENT_DEFINITION](#). In MySQL 5.1, the value shown in [EVENT_BODY](#) is always `SQL`.

These changes were made to bring this table into line with the [INFORMATION_SCHEMA.ROUTINES](#) table, and that table's [ROUTINE_BODY](#) and [ROUTINE_DEFINITION](#) columns. (Bug #16992)

- Incompatible change: MySQL Cluster node and system restarts formerly required that all fragments use the same local checkpoint (LCP); beginning with this version, it is now possible for different fragments to use different LCPs during restarts. This means that data node filesystems must be rebuilt as part of any upgrade to this version by restarting all data nodes with the `--initial` option. (Bug #21271, Bug #21478)

See [クラスタのアップグレードおよびダウングレードの互換性](#), and related sections of the Manual before upgrading a MySQL Cluster to version 5.1.12 or later.

- Incompatible change: A number of MySQL constructs are now prohibited in partitioning expressions, beginning with this release. These include:

- A number of MySQL functions.

You can find [a complete list of these functions \[1025\]](#) under Partitioning Limitation.

- The bit operators `|`, `&`, `^`, `<<`, `>>`, and `~`.
- Nested function calls.
- Calls to stored routines, UDFs, or plugins.
- Character-to-integer conversions involving non-8-bit character sets or any of the [latin1_german2_ci](#), [latin2_czech_cs](#), or [cp1250_czech_cs](#) collations.

These restrictions were added in part as a result of Bug #18198 and related bug reports.

For more information about these and other restrictions on partitioned tables in MySQL, see [パーティショニングの制約と制限](#).

- Binary MySQL distributions no longer include a `mysqld-max` server. Instead, distributions contain a binary that includes the features previously included in the `mysqld-max` binary.
- The default binary log format (as used during replication) is now Mixed based, automatically using a combination of row-based and statement based log events as appropriate.

- The general query log and slow query logs now can be enabled or disabled at runtime with the [general_log](#) and [slow_query_log](#) system variables, and the name of the log files can be changed by setting the [general_log_file](#) and [slow_query_log_file](#) system variables. See [一般クエリ ログ](#), and [スロー クエリ ログ](#).
- The output generated by the server when using the `--xml` option has changed with regard to null values. It now matches the output from `mysqldump --xml`. That is, a column containing a `NULL` value is now reported as

```
<field name="column_name" xsi:nil="true" />
```

whereas a column containing the string value `'NULL'` is reported as

```
<field name="column_name">NULL</field>
```

and a column containing an empty string is reported as

```
<field name="column_name">>/field>
```

(Bug #21263)

- The following statements now can be executed as prepared statements (using [PREPARE](#) plus [EXECUTE](#)):

```
CACHE INDEX
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
{CREATE | RENAME | DROP} DATABASE
{CREATE | RENAME | DROP} USER
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
REVOKE
KILL
LOAD INDEX INTO CACHE
RESET {MASTER | SLAVE | QUERY CACHE}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
INSTALL PLUGIN
UNINSTALL PLUGIN
```

(Bug #20665)

- The Instance Manager `--passwd` option has been renamed to `--print-password-line`. Other options were added to enable management of the IM password file from the command line: `--add-user`, `--drop-user`, `--edit-user`, `--list-users`, `--check-password-file`, `--clean-password-file`, `--username`, and `--password`. The `--mysql-safe-compatible` option was added to cause the Instance Manager to act similarly to `mysqld_safe`.
- On Windows, typing Control-C caused the `mysql` client to crash. Now it causes `mysql` to attempt to kill the current statement. If this cannot be done, or Control-C is typed again before the statement is killed, `mysql` exits. (Bug#17926; see also Bug #1989)
- `TEXT` and `BLOB` columns do not support `DEFAULT` values. However, when a default of `"` was specified, the specification was silently ignored. This now results in a warning, or an error in strict mode. (Bug #19498)
- The `mysql` client now allows `\l` in the `prompt` command argument to insert the current delimiter into the prompt. (Bug #14448)
- Log table changes: By default, the log tables use the `CSV` storage engine, as before. But now the log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables. The use of `DROP TABLE` for log tables is similarly restricted: It cannot be used to drop a log table that is in use. The log must be disabled first. (These changes also correct a deadlock that occurred for an attempt to drop an in-use log table.) (Bug #18559)
- The `LEFT()` and `RIGHT()` functions return `NULL` if any argument is `NULL`. (Bug #11728)

- **EXPLAIN EXTENDED** now shows a **filtered** column that is an estimated percentage of the examined rows that will be joined with the previous tables. (Bug #14940)
 - For **mysqlshow**, if a database name argument contains wildcard characters (such as **'_'**) but matches a single database name exactly, treat the name as a literal name. This allows a command such as **mysqlshow information_schema** work without having to escape the wildcard character. (Bug #19147)
 - If a **DROP VIEW** statement named multiple views, it stopped with an error if a non-existent view was named and did not drop the remaining views. Now it continues on and reports an error at the end, similar to **DROP TABLE**. (Bug #16614)
 - **SHOW CREATE TABLE** now shows constraints for **InnoDB** tables. (Bug #16614)
 - Added the **--set-charset** option to **mysqlbinlog** to allow the character set to be specified for processing binary log files. (Bug #18351)
 - **NDB Cluster**: Backup messages are no longer printed to the Cluster log.
 - **NDB Cluster**: The **HELP** command in the Cluster management client now provides command-specific help. For example, **HELP RESTART** in **ndb_mgm** provides detailed information about the **START** command. (Bug #19620)
 - **NDB Cluster**: Inserting into an **NDB** table failed when the table had no primary key but had a unique key added after table was created on one or more **NOT NULL** columns. This occurred when the unique key had been adding using either **ALTER TABLE** or **CREATE UNIQUE KEY**. (Bug #22838)
 - **NDB Cluster**: The **ndb_config** utility now accepts **-c** as a short form of the **--ndb-connectstring** option. (Bug #22295)
 - **NDB Cluster**: Added the **--bind-address** option for **ndbd**. This allows a data node process to be bound to a specific network interface. (Bug #22195)
 - **NDB Cluster**: The **Ndb_number_of_storage_nodes** system variable was renamed to **Ndb_number_of_data_nodes**. (Bug #20848)
 - Added the **--ndb-use-copying-alter-table** option to **mysqld** to provide a fallback in case of problems with online **ALTER TABLE** operations on **NDBCluster** tables.
 - Added the **SHOW CONTRIBUTORS** statement.
 - It is no longer possible to create partitioned tables using the **CSV** storage engine.
 - **NDB Cluster**: The status variables **Ndb_connected_host** and **Ndb_connected_port** were renamed to **Ndb_config_from_host** and **Ndb_config_from_port**, respectively.
 - **NDB Cluster**: A number of erroneous, misleading, or missing error messages have been corrected. (Bug #17297 & Bug #19543)
 - **NDB Cluster**: It is no longer possible to create Cluster tables using any partitioning type other than **[LINEAR] KEY**. Attempting to do so now raises an error.
 - The **ExtractValue()** function now produces an error when passed an XML fragment that is not well-formed. (Bug #18201)
- (Previously, the function allowed invalid XML fragments to be used.)
- There were several issues regarding how **SHOW STATUS** affected some status variables and logging which could impact monitoring the MySQL Server. The behavior of this statement has been modified in two ways:
 - **SHOW STATUS** is no longer logged to the slow query log.
 - **SHOW STATUS** no longer updates any session status variables, except for **com_show_status**.
- However, **SHOW STATUS** continues to update global status variables to allow monitoring of what the server is actually doing. This is because **SHOW STATUS** creates temporary tables that may affect performance if it is called excessively often. (Bug #10210, Bug #19764)

- The [mysqldumpslow](#) script has been moved from client RPM packages to server RPM packages. This corrects a problem where [mysqldumpslow](#) could not be used with a client-only RPM install, because it depends on [my_print_defaults](#) which is in the server RPM. (Bug #20216)
- The bundled yaSSL library was upgraded to version 1.3.7.
- The bundled yaSSL library licensing has added a FLOSS exception similar to MySQL to resolve licensing incompatibilities with MySQL. (See the [extra/yassl/FLOSS-EXCEPTIONS](#) file in a MySQL source distribution for details.) (Bug #16755)
- [mysqslap](#) threads now try to connect up to 10 times if the initial connect attempt fails. (Bug #21297)
- For a successful dump, [mysqldump](#) now writes a SQL comment to the end of the dump file in the following format:

```
-- Dump completed on YYYY-MM-DD hh:mm:ss
```

(Bug #10877)

- The [mysqld](#) and [mysqlmanager](#) manpages have been reclassified from volume 1 to volume 8. (Bug #21220)
- In the [INFORMATION_SCHEMA.ROUTINES](#) table the [ROUTINE_DEFINITION](#) column now is defined as [NULL](#) rather than [NOT NULL](#). Also, [NULL](#) rather than the empty string is returned as the column value if the user does not have sufficient privileges to see the routine definition. (Bug #20230)
- [configure](#) now defines the symbol [DBUG_ON](#) in [config.h](#) to indicate whether the source tree is configured to be compiled with debugging support. (Bug #19517)
- The MySQL distribution now compiles on UnixWare 7.13. (Bug #20190)
- The [mysql](#) client used the default character set if it automatically reconnected to the server, which is incorrect if the character set had been changed. To enable the character set to remain synchronized on the client and server, the [mysql](#) command [charset](#) (or [\C](#)) that changes the default character set and now also issues a [SET NAMES](#) statement. The changed character set is used for reconnects. (Bug #11972)
- The [STATE](#) column of the [INFORMATION_SCHEMA.PROCESSLIST](#) table was increased from 30 to 64 characters to accommodate longer state values. (Bug #21652)
- [TIMESTAMP](#) columns that are [NOT NULL](#) now are reported that way by [SHOW COLUMNS](#) and [INFORMATION_SCHEMA](#). (Bug #20910)
- [INFORMATION_SCHEMA](#) contains new tables, [GLOBAL_STATUS](#), [SESSION_STATUS](#), [GLOBAL_VARIABLES](#), and [SESSION_VARIABLES](#), that correspond to the output from the [SHOW {GLOBAL|SESSION} STATUS](#) and [SHOW {GLOBAL|SESSION} VARIABLES](#) statements.
- The [BINARY](#) keyword now is forbidden as a data type attribute in stored routines (for example, [DECLARE v1 VARCHAR\(25\) BINARY](#)), because [DECLARE](#) does not support collations, and in this context [BINARY](#) specifies the binary collation of the variable's character set. (Bug #20701)
- The source distribution has been updated so that the UDF example can be compiled under Windows with CMake. See [Compiling and Installing User-Defined Functions](#). (Bug #19121)
- [LOAD DATA INFILE](#) no longer causes an implicit commit for all storage storage engines. It now causes an implicit commit only for tables using the [NDB](#) storage engine. (Bug #11151)
- The [LOAD DATA FROM MASTER](#) and [LOAD TABLE FROM MASTER](#) statements are deprecated. See [LOAD DATA FROM MASTER 構文](#), for recommended alternatives. (Bug #18822, Bug #9125, Bug #12187, Bug #14399, Bug #15025, Bug #20596)
- [mysqldump](#) now has a [--flush-privileges](#) option. It causes [mysqldump](#) to emit a [FLUSH PRIVILEGES](#) statement after dumping the [mysql](#) database. This option should be used any time the dump contains the [mysql](#) database and any other database that depends on the data in the [mysql](#) database for proper restoration. (Bug #21424)
- The default value of the [tmp_table_size](#) system variable was lowered from 32MB to 16MB because it is bounded by the value of [max_heap_table_size](#), which has a default of 16MB. (Bug #18875)

- The number of [InnoDB](#) threads is no longer limited to 1,000 on Windows. (Bug #22268)
- Memory consumption of the [InnoDB](#) data dictionary cache was roughly halved by cleaning up the data structures. (Bug #20877)
- Using `--with-debug` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.
- A new system variable, [lc_time_names](#), specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the [DATE_FORMAT\(\)](#), [DAYNAME\(\)](#) and [MONTHNAME\(\)](#) functions. See [MySQL サーバのローケル サポート](#).

Bugs fixed:

- Security fix: On Linux, and possibly other platforms using case-sensitive filesystems, it was possible for a user granted rights on a database to create or access a database whose name differed only from that of the first by the case of one or more letters. (CVE-2006-4226, Bug #17647)
- Security fix: If a user has access to [MyISAM](#) table [t](#), that user can create a [MERGE](#) table [m](#) that accesses [t](#). However, if the user's privileges on [t](#) are subsequently revoked, the user can continue to access [t](#) by doing so through [m](#). If this behavior is undesirable, you can start the server with the new `--skip-merge` option to disable the [MERGE](#) storage engine. (Bug #15195)
- Security fix: A stored routine created by one user and then made accessible to a different user using [GRANT EXECUTE](#) could be executed by that user with the privileges of the routine's definer. (CVE-2006-4227, Bug #18630)
- Incompatible change: For [utf8](#) columns, the full-text parser incorrectly considered several non-word punctuation and whitespace characters as word characters, causing some searches to return incorrect results. (Bug #19580)

The fix involves a change to the full-text parser, so any tables that have [FULLTEXT](#) indexes on [utf8](#) columns must be repaired with [REPAIR TABLE](#):

```
REPAIR TABLE tbl\_name QUICK;
```

- An invalid [GRANT](#) statement for which [Ok](#) was returned on a replication master caused an error on the slave and replication to fail. (Bug #6774)
- [InnoDB](#) locking was improved by removing a gap lock for the case that you try to delete the same row twice within a transaction. (Bug #13544)
- Deleting entries from a large [MyISAM](#) index could cause index corruption when it needed to shrink. Deletes from an index can happen when a record is deleted, when a key changes and must be moved, and when a key must be un-inserted because of a duplicate key. This can also happen in [REPAIR TABLE](#) when a duplicate key is found and in [myisamchk](#) when sorting the records by an index. (Bug #22384)
- Setting [myisam_repair_threads](#) caused any repair operation on a [MyISAM](#) table to fail to update the cardinality of indexes, instead making them always equal to 1. (Bug #18874)
- [ALTER EVENT](#) statements including only a [COMMENT](#) clause failed with a syntax error on two platforms: Linux for S/390, and OS X 10.4 for 64-bit PPC. (Bug #23423)
- For row-based replication, log rotation could occur at an improper time. (Bug #21474)
- For row-based replication, the [BINLOG](#) command did not lock tables properly, causing a crash for some table types. (Bug #19459)
- The parser rejected queries that selected from a table twice using a [UNION](#) within a subquery. The parser now supports arbitrary subquery, join, and parenthesis operations within [EXISTS](#) subqueries. A limitation still exists for scalar subqueries: If the subquery contains [UNION](#), the first [SELECT](#) of the [UNION](#) cannot be within parentheses. For example, `SELECT (SELECT a FROM t1 UNION SELECT b FROM t2)` will work, but `SELECT ((SELECT a FROM t1) UNION (SELECT b FROM t2))` will not. (Bug #14654)
- Incorrect type aggregation for [IN](#) and [CASE](#) expressions could lead to an incorrect result. (Bug #18360)

- When using row based replication, a `CREATE TABLE...SELECT` statement would be replicated, even if the table creation failed on the master (for example, due to a duplicate key failure). (Bug #20265)
- The optimizer did not take advantage of indexes on columns used for the second or third arguments of `BETWEEN`. (Bug #18165)
- Subqueries with aggregate functions but no `FROM` clause could return incorrect results. (Bug #21540)
- The `CSV` storage engine failed to detect some table corruption. (Bug #22080)
- For multiple-table `UPDATE` statements, storage engines were not notified of duplicate-key errors. (Bug #21381)
- Successive invocations of a `COUNT(*)` query containing a join on two `MyISAM` tables and a `WHERE` clause of the form `WHERE (table1.column1 = table2.column2) OR table2.column2 IS NULL` yielded different results. (Bug #21019)
- It was possible to provide the `ExtractValue()` function with input containing 「tags」 that were not valid XML; for example, it was possible to use tag names beginning with a digit, which are disallowed by the W3C's XML 1.0 specification. Such cases caused the function to return 「junk」 output rather than an error message signalling the user as to the true nature of the problem. (Bug #20854)
- The presence of a subquery in the `ON` clause of a join in a view definition prevented the `MERGE` algorithm from being used for the view in cases where it should be allowed. (Bug #21646)
- `BIT` columns were not replicated properly under row-based replication. (Bug #22550)
- Conversion of values inserted into a `BIT` column could affect adjacent columns. (Bug #22271)
- The URL into the online manual that is printed in the stack trace message by the server was out of date. (Bug #21449)
- Incorrect results could be obtained from re-execution of a parametrized prepared statement or a stored routine with a `SELECT` that uses `LEFT JOIN` with a second table having only one row. (Bug #21081)
- `PROCEDURE ANALYSE()` returned incorrect values of `M FLOAT(M, D)` and `DOUBLE(M, D)`. (Bug #20305)
- Join conditions using partial indexes on `utf8` columns of `InnoDB` tables incorrectly ignored rows where the length of the actual value was greater than the length of the partial index. (Bug #19960)
- On an `INSERT` into an updatable but non-insertable view, an error message was issued stating that the view was not updatable. Now the message says the view is not insertable-into. (Bug #5505)
- `INSERT DELAYED` did not honor `SET INSERT_ID` or the `auto_increment_*` system variables. (Bug #20627, Bug #20830)
- For character sets having a `mbmaxlen` value of 2, any `ALTER TABLE` statement changed `TEXT` columns to `MEDIUMTEXT`. (Bug #21620)
- A query that used `GROUP BY` and an `ALL` or `ANY` quantified subquery in a `HAVING` clause could trigger an assertion failure. (Bug #21853)
- For an `ENUM` column that used the `ucs2` character set, using `ALTER TABLE` to modify the column definition caused the default value to be lost. (Bug #20108)
- An `UPDATE` that referred to a key column in the `WHERE` clause and activated a trigger that modified the column resulted in a loop. (Bug #20670)
- A loaded storage engine plugin did not load after a server restart. (Bug #21610)
- Creating a `TEMPORARY` table with the same name as an existing table that was locked by another client could result in a lock conflict for `DROP TEMPORARY TABLE` because the server unnecessarily tried to acquire a name lock. (Bug #21096)
- After `FLUSH TABLES WITH READ LOCK` followed by `UNLOCK TABLES`, attempts to drop or alter a stored routine failed with an error that the routine did not exist, and attempts to execute the routine failed with a lock conflict error. (Bug #21414)

- [mysql_com.h](#) unnecessarily referred to the [ulong](#) type. (Bug #22227)
- Incorporated some portability fixes into the definition of `__attribute__` in [my_global.h](#). (Bug #2717)
- Linking the [pthreads](#) library to single-threaded MySQL libraries caused [dlopen\(\)](#) to fail at runtime on HP-UX. (Bug #18267)
- Loading a plugin caused any an existing plugin with the same name to be lost. (Bug #20615)
- In the package of pre-built time zone tables that is available for download at <http://dev.mysql.com/downloads/timezones.html>, the tables now explicitly use the [utf8](#) character set so that they work the same way regardless of the system character set value. (Bug #21208)
- The build process incorrectly tried to overwrite [sql/lex_hash.h](#). This caused the build to fail when using a shadow link tree pointing to original sources that were owned by another account. (Bug #18888)
- [mysql_ftdump](#) produced bad counts for common words. (Bug #22326)
- The optimizer could make an incorrect index choice for indexes with a skewed key distribution. (Bug #22393)
- When records are merged from the insert buffer and the page needs to be reorganized, [InnoDB](#) used incorrect column length information when interpreting the records of the page. This caused a server crash due to apparent corruption of secondary indexes in [ROW_FORMAT=COMPACT](#) that contain prefix indexes of fixed-length columns. Data files should not be corrupted, but the crash was likely to repeat every time the server was restarted. (Bug #21638)
- Instance Manager didn't close the client socket file when starting a new [mysqld](#) instance. [mysqld](#) inherited the socket, causing clients connected to Instance Manager to hang. (Bug #12751)
- Using [GROUP_CONCAT\(\)](#) on the result of a subquery in the [FROM](#) clause that itself used [GROUP_CONCAT\(\)](#) could cause a server crash. (Bug #22015)
- Instance Manager had a race condition involving [mysqld](#) PID file removal. (Bug #22379)
- Execution of a prepared statement that uses an [IN](#) subquery with aggregate functions in the [HAVING](#) clause could cause a server crash. (Bug #22085)
- Selecting from a [MERGE](#) table could result in a server crash if the underlying tables had fewer indexes than the [MERGE](#) table itself. (Bug #21617, Bug #22937)
- A locking safety check in [InnoDB](#) reported a spurious error `stored_select_lock_type is 0 inside ::start_stmt()` for [INSERT ... SELECT](#) statements in [innodb_locks_unsafe_for_binlog](#) mode. The safety check was removed. (Bug #10746)
- [make install](#) tried to build files that should already have been built by [make all](#), causing a failure if installation was performed using a different account than the one used for the initial build. (Bug #19738)
- The source distribution would not build on Windows due to a spurious dependency on [ib_config.h](#). (Bug #22224)
- It was possible for a stored routine with a non-[latin1](#) name to cause a stack overrun. (Bug #21311)
- The server returns a more informative error message when it attempts to open a [MERGE](#) table that has been defined to use non-[MyISAM](#) tables. (Bug #10974)
- Within stored routines, some error messages were printed incorrectly. A non-null-terminated string was passed to a message-printing routine that expected a null-terminated string. (Bug #20778)
- [SUBSTR\(\)](#) results sometimes were stored improperly into a temporary table when multi-byte character sets were used. (Bug #20204)
- Certain malformed [INSERT](#) statements could crash the [mysql](#) client. (Bug #21142)
- [EXPLAIN EXTENDED](#) now shows a [filtered](#) column that is an estimated percentage of the examined rows that will be joined with the previous tables. This was added while dealing with a problem of MySQL choosing the wrong index for some queries. (Bug #14940)

- On Mac OS X, zero-byte [read\(\)](#) or [write\(\)](#) calls to an SMB-mounted filesystem could return a non-standard return value, leading to data corruption. Now such calls are avoided. (Bug #12620)
- With [TRADITIONAL](#) SQL mode, assignment of out-of-bound values and rounding of assigned values was done correctly, but assignment of the same numbers represented as strings sometimes was handled differently. (Bug #6147)
- The source distribution failed to compile when configured with the [--without-geometry](#) option. (Bug #12991)
- The source distribution failed to compile when configured with the [--with-libwrap](#) option. (Bug #18246)
- The feature of being able to recover a temporary table named [#sql_id](#) in InnoDB by creating a table named [rsq_id_recover_innodb_tmp_table](#) was broken by the introduction of the new identifier encoding in MySQL 5.1.6 (Bug #21313)
- [ALTER EVENT](#) in the body of a stored procedure led to a crash when the procedure was called. This affected only those [ALTER EVENT](#) statements which changed the interval of the event. (Bug #22397)
- A [DATE](#) can be represented as an integer (such as [20060101](#)) or as a string (such as ['2006.01.01'](#)). When a [DATE](#) (or [TIME](#)) column is compared in one [SELECT](#) against both representations, constant propagation by the optimizer led to comparison of [DATE](#) as a string against [DATE](#) as an integer. This could result in integer comparisons such as [2006](#) against [20060101](#), erroneously producing a false result. (Bug #21475)
- When a statement used a stored function that inserted into an [AUTO_INCREMENT](#) column, the generated [AUTO_INCREMENT](#) value was not written into the binary log, so a different value could in some cases be inserted on the slave. (Bug #20341)
- A stored procedure that used [LAST_INSERT_ID\(\)](#) did not replicate properly using statement-based binary logging. (Bug #20339)
- Use of the [--no-pager](#) option caused [mysql](#) to crash. (Bug #19363)
- For [INSERT ... ON DUPLICATE KEY UPDATE](#), use of [VALUES\(col_name\)](#) within the [UPDATE](#) clause sometimes was handled incorrectly. (Bug #21555)
- When [event_scheduler](#) was set to [DISABLED](#), its value was not displayed correctly by [SHOW VARIABLES](#) or [SELECT @@global.event_scheduler](#). (Bug #22662)
- Row equalities (such as [WHERE \(a,b\) = \(c,d\)](#)) were not taken into account by the optimizer, resulting in slow query execution. Now they are treated as conjunctions of equalities between row elements. (Bug #16081)
- If the binary logging format was changed between the times when a locked table was modified and when it was unlocked, the binary log contents were incorrect. (Bug #20863)
- Column names supplied for a view created on a master server could be lost on a slave server. (Bug #19419)
- For a [MyISAM](#) table locked with [LOCK TABLES ...WRITE](#), queries optimized using the [index_merge](#) method did not show rows inserted with the lock in place. (Bug #20256)
- Table aliases in multiple-table [DELETE](#) statements sometimes were not resolved. (Bug #21392)
- A query result could be sorted improperly when using [ORDER BY](#) for the second table in a join. (Bug #21302)
- The [--collation-server](#) server option was being ignored. With the fix for this problem, if you choose a non-default character set with [--character-set-server](#), you should also use [--collation-server](#) to specify the collation. (Bug #15276)
- A function result in a comparison was replaced with a constant by the optimizer under some circumstances when this optimization was invalid. (Bug #21698)
- A subquery that uses an index for both the [WHERE](#) and [ORDER BY](#) clauses produced an empty result. (Bug #21180)
- If the [auto_increment_offset](#) setting causes MySQL to generate a value larger than the column's maximum possible value, the [INSERT](#) statement is accepted in strict SQL mode, whereas but should fail with an error. (Bug #20573)
- Queries containing a subquery that used aggregate functions could return incorrect results. (Bug #16792)

- Row-based replication failed when the query cache was enabled on the slave. (Bug #17620)
 - The `index_merge/Intersection` optimizer could have a memory overrun when the number of table columns covered by an index is sufficiently large, possibly resulting in a server crash. (Bug #16201)
 - The `MD5()`, `SHA1()`, and `ENCRYPT()` functions should return a binary string, but the result sometimes was converted to the character set of the argument. `MAKE_SET()` and `EXPORT_SET()` now use the correct character set for their default separators, resulting in consistent result strings which can be coerced according to normal character set rules. (Bug #20536)
 - `EXPLAIN` sometimes returned an incorrect `select_type` for a `SELECT` from a view, compared to the `select_type` for the equivalent `SELECT` from the base table. (Bug #5500)
 - An InnoDB mutex was not acquired and released under the same condition, leading to deadlock in some rare situations involving XA transactions. (Bug #21833)
 - With row-based replication, replicating a statement to a slave where the table had additional columns relative to the master table did not work. (Bug #19069)
 - For a `MyISAM` table with a `FULLTEXT` index, compression with `myisampack` or a check with `myisamchk` after compression resulted in table corruption. (Bug #19702)
 - With `max_sp_recursion` set to 0, a stored procedure that executed a `SHOW CREATE PROCEDURE` statement for itself triggered a recursion limit exceeded error, though the statement involves no recursion. (Bug #21416)
 - `mysqldump` did not add version-specific comments around `WITH PARSE` and `TABLESPACE ... STORAGE DISK` clauses for `CREATE TABLE` statements, causing the dump file to fail when loaded into older servers. (Bug #20841)
 - `BIN()`, `OCT()`, and `CONV()` did not work with BIT values. (Bug #15583)
 - The optimizer could produce an incorrect result after `AND` with collations such as `latin1_german2_ci`, `utf8_czech_ci`, and `utf8_lithianian_ci`. (Bug #9509)
 - The server could crash for the second execution of a function containing a `SELECT` statement that uses an aggregating `IN` subquery. (Bug #21493)
 - `UPGRADE` was treated as a reserved word, although it is not. (Bug #21772)
 - Database and table names have a maximum length of 64 characters (even if they contain multi-byte characters), but were being truncated to 64 bytes. (Bug #21432)
 - Usernames have a maximum length of 16 characters (even if they contain multi-byte characters), but were being truncated to 16 bytes. (Bug #20393)
 - A query could produce different results with and without an index, if the `WHERE` clause contained a range condition that used an invalid `DATETIME` constant. (Bug #16249)
 - `COUNT(*)` queries with `ORDER BY` and `LIMIT` could return the wrong result. (Bug #21787)
- Note: This problem was introduced by the fix for Bug #9676, which limited the rows stored in a temporary table to the `LIMIT` clause. This optimization is not applicable to non-group queries with aggregate functions. The current fix disables the optimization in such cases.
- Memory overruns could occur for certain kinds of subqueries. (Bug #21477)
 - Adding `ORDER BY` to a `SELECT DISTINCT(expr)` query could produce incorrect results. (Bug #21456)
 - Memory used by scheduled events was not freed when the events were dropped. (Bug #18683)
 - A scheduled event that took longer to execute than the length of time scheduled between successive executions could "skip" executions. For example, an event defined with `EVERY 1 SECOND` — but which required longer than 1 second to complete — might be executed only once every 2 seconds. (Bug #16417)
 - When used in the `DO` clause of a `CREATE EVENT` statement, the statements `CREATE EVENT`, `CREATE FUNCTION`, and `CREATE PROCEDURE` caused the server to crash. (These statements are not permitted inside `CREATE EVENT`.) (Bug #16409, Bug #18896)

- A subselect used in the **ON SCHEDULE** clause of a **CREATE EVENT** or **ALTER EVENT** statement caused the server to crash, rather than producing an error as expected. (Bug #16394)
- **mysql** displayed an empty string for **NULL** values. (Bug #21618)
- **mysql_upgrade** produced a malformed **upgrade_defaults** file by overwriting the **[client]** group header with a **password** option. This prevented **mysqlcheck** from running successfully when invoked by **mysql_upgrade**. (Bug #21011)
- **mysql_config --libmysqld-libs** did not produce any SSL options necessary for linking **libmysqld** with SSL support enabled. (Bug #21239)
- **yaSSL** had a conflicting definition for **socklen_t** on **hurd-i386** systems. (Bug #22326)
- On Windows, inserting into a **MERGE** table after renaming an underlying **MyISAM** table caused a server crash. (Bug #20789)
- **character_set_results** can be **NULL** to signify 「no conversion,」 but some code did not check for **NULL**, resulting in a server crash. (Bug #21913)
- If a partitioned **InnoDB** table contained an **AUTO_INCREMENT** column, a **SHOW** statement could cause an assertion failure with more than one connection. (Bug #20493)
- Running **InnoDB** with many concurrent threads could cause memory corruption and a seg fault due to a bug introduced in MySQL 5.1.11. (Bug #20213)
- Using **DROP TABLE** with concurrent queries causes **mysqld** to crash. (Bug #21784)
- The **ExtractValue()** function did not accept XML tag names containing a period (.) character. (Bug #20795)
- For table-format output, **mysql** did not always calculate columns widths correctly for columns containing multi-byte characters in the column name or contents. (Bug #17939)
- Identifiers with embedded escape characters were not handled correctly by some **SHOW** statements due to some old code that was doing some extra unescaping. (Bug #19874)
- **InnoDB** was slow with more than 100,000 **.idb** files. (Bug #21112)
- **SHOW INNODB STATUS** contained some duplicate output. (Bug #21113)
- After an **INSERT ... ON DUPLICATE KEY UPDATE** statement that updated an existing row, **LAST_INSERT_ID()** could return a value not in the table. (Bug #11460)
- Selecting from **INFORMATION_SCHEMA.FILES** could crash the server. (Bug #21676)
- Using cursors with **READ COMMITTED** isolation level could cause **InnoDB** to crash. (Bug #19834)
- A server or network failure with an open client connection would cause the client to hang even though the server was no longer available. (Bug #9678)
- Using **ALTER TABLE ... REORGANIZE PARTITIONS** to reduce the number of subpartitions to 1 caused the server to crash. (Bug #21210)
- **NDB Cluster**: Data nodes added while the cluster was running in single user mode were all assigned node ID 0, which could later cause multiple node failures. Adding of nodes in single user mode is no longer possible. (Bug #20395)
- **NDB Cluster**: Attempting to create an **NDB** table on a MySQL with an existing non-Cluster table with the same name in the same database could result in data loss or corruption. Now, if such a table is encountered during autodiscovery, a warning is written to the error log of the affected **mysqld**, and the local table is overwritten. (Bug #21378)
- **NDB Cluster** (NDB API): Inactivity timeouts for scans were not correctly handled. (Bug #23107)
- **NDB Cluster** (NDB API): Attempting to read a nonexistent tuple using **Commit** mode for **NdbTransaction::execute()** caused node failures. (Bug #22672)

- **NDB Cluster** (NDB API): The inclusion of `my_config.h` in `NdbApi.h` required anyone wishing to write NDB API applications against MySQL 5.1 to have a complete copy of the 5.1 sources. (Bug #21253)
- **NDB Cluster** (NDB API): The `NdbOperation::getBlobHandle()` method, when called with the name of a nonexistent column, caused a segmentation fault. (Bug #21036)
- **NDB Cluster**: The `--help` output from **NDB** binaries did not include file-related options. (Bug #21994)
- **NDB Cluster**: The node recovery algorithm was missing a version check for tables in the `ALTER_TABLE_COMMITTED` state (as opposed to the `TABLE_ADD_COMMITTED` state, which has the version check). This could cause inconsistent schemas across nodes following node recovery. (Bug #21756)
- **NDB Cluster**: The output for the `--help` option used with **NDB** executable programs (`ndbd`, `ndb_mgm`, `ndb_restore`, `ndb_config`, and so on) referred to the `Ndb.cfg` file, instead of `my.cnf`. (Bug #21585)
- **NDB Cluster**: Partition distribution keys were updated only for the primary and starting replicas during node recovery. This could lead to node failure recovery for clusters having an odd number of replicas. (Bug #21535)

Note: We recommend values for `NumberOfReplicas` that are even powers of 2, for best results.

- **NDB Cluster**: The `ndb_mgm` management client did not set the exit status on errors, always returning 0 instead. (Bug #21530)
- **NDB Cluster**: Cluster logs were not rotated following the first rotation cycle. (Bug #21345)
- **NDB Cluster**: Condition pushdown did not work correctly with `DATETIME` columns. (Bug #21056)
- **NDB Cluster**: Under some circumstances, local checkpointing would hang, keeping any unstarted nodes from being started. (Bug #20895)
- **NDB Cluster**: Using an invalid node ID with the management client `STOP` command could cause `ndb_mgm` to hang. (Bug #20575)
- **NDB Cluster**: In some cases where `SELECT COUNT(*)` from an **NDB** table should have yielded an error, `MAX_INT` was returned instead. (Bug #19914)
- **NDB Cluster**: Following the restart of an MGM node, the Cluster management client did not automatically reconnect. (Bug #19873)
- **NDB Cluster**: Error messages given when trying to make online changes parameters such as `NoOfReplicas` that can only be changed via a complete shutdown and restart of the cluster did not indicate the true nature of the problem. (Bug #19787)
- **NDB Cluster**: `ndb_restore` did not always make clear that it had recovered successfully from temporary errors while restoring a cluster backup. (Bug #19651)
- **NDB Cluster**: In rare situations with resource shortages, a crash could result from insufficient `IndexScanOperations`. (Bug #19198)
- **NDB Cluster**: `ndb_mgm -e show | head` would hang after displaying the first 10 lines of output. (Bug #19047)
- **NDB Cluster**: The error returned by the cluster when too many nodes were defined did not make clear the nature of the problem. (Bug #19045)
- **NDB Cluster**: A problem with takeover during a system restart caused ordered indexes to be rebuilt incorrectly. This also adversely affected Cluster Replication. (Bug #15303)
- **NDB Cluster**: Some queries involving joins on very large **NDB** tables could crash the MySQL server. (Bug #21059)
- **NDB Cluster** (Disk Data): `mysqldump` did not back up tablespace or log file group information for Disk Data tables correctly. (Specifically, `UNDO_BUFFER_SIZE` and `INITIAL_SIZE` values were misreported.) Trying to restore from such a backup would produce error 1296 (`Got error 1504 'Out of logbuffer memory' from NDB`). (Bug #20809)
- **NDB Cluster** (Disk Data): The `INFORMATION_SCHEMA.FILES` table showed incorrect values in the `EXTENT_SIZE`, `FREE_EXTENTS`, and `TOTAL_EXTENTS` columns for UNDO log files. (Bug #20073)

- **NDB Cluster** (Disk Data): Deletes from Disk Data tables used a non-optimal scan to find the rows to be deleted, resulting in poor performance. The fix causes disk order rather than memory order to be used, and can improve performance of Disk Data deletes by up to ~300% in some cases. (Bug #17929)
- **NDB Cluster**: A scan timeout returned Error 4028 (Node failure caused abort of transaction) instead of Error 4008 (Node failure caused abort of transaction...). (Bug #21799)
- **NDB Cluster**: The message Error 0 in readAutoIncrementValue(): no Error was written to the error log whenever SHOW TABLE STATUS was performed on a Cluster table that did not have an AUTO_INCREMENT column. (Bug #21033)
- **NDB Cluster** (Direct APIs): The storage/ndb directory was missing from the server binary distribution, making it impossible to compile NDB API and MGM API applications. This directory can be found as /usr/include/storage/ndb after installing that distribution. (Bug #21955)
- **NDB Cluster**: ndb_size.pl and ndb_error_reporter were missing from RPM packages. (Bug #20426)
- The ndb_mgm program was included in both the MySQL-ndb-tools and MySQL-ndb-management RPM packages, resulting in a conflict if both were installed. Now ndb_mgm is included only in MySQL-ndb-tools. (Bug #21058)
- libmysqld produced some warnings to stderr which could not be silenced. These warnings now are suppressed. (Bug #13717)
- If a column definition contained a character set declaration, but a DEFAULT value began with an introducer, the introducer character set was used as the column character set. (Bug #20695)
- If a query had a condition of the form tableX.key = tableY.key, which participated in equality propagation and also was used for ref access, then early ref-access NULL filtering was not performed for the condition. This could make query execution slower. (Bug #19649)
- The optimizer sometimes produced an incorrect row-count estimate after elimination of const tables. This resulted in choosing extremely inefficient execution plans in some cases when distribution of data in joins were skewed. (Bug #21390)
- Query results could be incorrect if the WHERE clause contained t.key_part NOT IN (val_list), where val_list is a list of more than 1000 constants. (Bug #21282)
- STR_TO_DATE() sometimes would return NULL if the %D format specifier was not the last specifier in the format string. (Bug #20987)
- On Windows, a definition for mysql_set_server_option() was missing from the C client library. (Bug #16513)
- The myisam_stats_method variable was mishandled when set from an option file or on the command line. (Bug #21054)
- The optimizer assumed that if (a=x AND b=x) is true, (a=x AND b=x) AND a=b is also true. But that is not always so if a and b have different data types. (Bug #21159)
- InnoDB did not honor IGNORE INDEX, which prevented using IGNORE INDEX in cases where an index sort would be slower than a filesort. (Bug #21174)
- For connections that required a SUBJECT value, a check was performed to verify that the value was correct, but the connection was not refused if not. (Bug #20411)
- Some Linux-x86_64-icc packages (of previous releases) mistakenly contained 32-bit binaries. Only ICC builds are affected, not gcc builds. Solaris and FreeBSD x86_64 builds are not affected. (Bug #22238)
- INSERT ... SELECT sometimes generated a spurious Column count doesn't match value count error. (Bug #21774)
- Some user-level level errors were being written to the server's error log, which is for server errors. (Bug #20402)
- to the client could have an empty column name. When using tables created under MySQL 4.1 with a 5.0 server, if the tables contained VARCHAR columns, for some queries the metadata sent to the client could have an empty column name. (Bug #14897)

- On 64-bit systems, use of the [cp1250](#) character set with a primary key column in a [LIKE](#) clause caused a server crash for patterns having letters in the range 128..255. (Bug #19741)
- [ORDER BY RAND\(\) LIMIT 1](#) always set a user variable to the last possible value from the table. (Bug #16861)
- [N'xxx'](#) and [_utf8'xxx'](#) were not treated as equivalent because [N'xxx'](#) failed to unescape backslashes (`\`) and doubled apostrophe/single quote characters (`"`). (Bug #17313)
- A subquery in the [WHERE](#) clause of the outer query and using [IN](#) and [GROUP BY](#) returned an incorrect result. (Bug #16255)
- When [NOW\(\)](#) was used in a [BETWEEN](#) clause of the definition for a view, it was replaced with a constant in the view. (Bug #15950)
- A stored procedure with a [CONTINUE](#) handler that encountered an error continued to execute a statement that caused an error, rather with the next statement following the the one that caused the error. (Bug #8153)
- [libmysqlclient](#) defined a symbol [BN_bin2bn](#) which belongs to OpenSSL. This could break applications that also linked against OpenSSL's [libcrypto](#) library. The fix required correcting an error in a build script that was failing to add rename macros for some functions. (Bug #21930)
- Queries that used the [index_merge](#) and [sort_union](#) methods to access an [InnoDB](#) table could produce inaccurate results. This issue was introduced in MySQL 5.1.10 when a new handler and bitmap interface was implemented. (Bug #21277)
- The [SELECT](#) privilege was required for an insert on a view, instead of the [INSERT](#) privilege. (Bug #21261)

Note: This fixes a regression that was introduced by the fix for Bug #20989.

- Running [SHOW MASTER LOGS](#) at the same time as binary log files were being switched would cause [mysqld](#) to hang. (Bug #21965)
- Building [mysql](#) on Windows with CMake 2.4 would fail to create [libmysqld](#) correctly. (Bug #20907)
- The server's handling of the number of partitions or subpartitions specified in a [PARTITIONS](#) or [SUBPARTITIONS](#) clause was changed. Beginning with this release, the number of partitions must:
 - be a positive, non-zero integer
 - not have any leading zeroes
 - not be an expression

Also beginning with this version, no attempt is made to convert, truncate, or evaluate a [PARTITIONS](#) or [SUBPARTITIONS](#) value; instead, the [CREATE TABLE](#) or [ALTER TABLE](#) statement containing the [PARTITIONS](#) or [SUBPARTITIONS](#) clause now fails with an appropriate error message. (Bug #15890)

- A misleading error message was displayed when attempting to define a unique key that was not valid for a partitioned table. (Bug #21862)
- Errors could be generated during the execution of certain prepared statements that ran queries on partitioned tables. (Bug #21658)
- Using relative paths for [DATA DIRECTORY](#) or [INDEX DIRECTORY](#) with a partitioned table generated a warning rather than an error, and caused 「junk」 files to be created in the server's data directory. (Bug #21350)
- Using [EXPLAIN PARTITIONS](#) with a query on a table whose partitioning expression was based on the value of a [DATE](#) column could sometimes cause the server to crash. (Bug #21339)
- Running [SHOW TABLE STATUS](#) on any [InnoDB](#) table having at least one record could crash the server. Note that this was not due to any issue in the [InnoDB](#) storage engine, but rather with [AUTO_INCREMENT](#) handling in the partitioning code — however, the table did not have to have an [AUTO_INCREMENT](#) column for the bug to manifest. (Bug #21173)
- Some [ALTER TABLE](#) statements affecting a table's subpartitioning could hang. (Bug #21143)

- Scheduled events that invoked stored procedures executing DDL operations on partitioned tables could crash the server. (Bug #20548)
- The yaSSL library bundled with `libmysqlclient` had some conflicts with OpenSSL. Now macros are used to rename the conflicting symbols to have a prefix of `ya`. (Bug #19810)
- It is possible to create `MERGE` tables into which data cannot be inserted (by not specifying a `UNION` clause. However, when an insert was attempted, the error message was confusing. Now an error occurs indicating that the table is read-only. (Bug #17766)
- User-created tables having a name beginning with `#sql` were not visible to `SHOW TABLES` and could collide with internal temporary table names. Now they are not hidden and do not collide. (Bug #1405)
- A `NUL` byte within a prepared statement string caused the rest of the string not to be written to the query log, allowing logging to be bypassed. (Bug #21813)
- `mysql_upgrade` created temporary files in a possibly insecure way. (Bug #21224)
- Some prepared statements caused a server crash when executed a second time. (Bug #21166)
- With `query_cache_type` set to 0, `RESET QUERY CACHE` was very slow and other threads were blocked during the operation. Now a cache reset is faster and non-blocking. (Bug #21051)
- When `DROP DATABASE` or `SHOW OPEN TABLES` was issued while concurrently issuing `DROP TABLE` (or `RENAME TABLE`, `CREATE TABLE LIKE` or any other statement that required a name lock) in another connection, the server crashed. (Bug #19403, Bug #21216)
- Use of zero-length variable names caused a server crash. (Bug #20908)
- Prepared statements caused general log and server memory corruption. (Bug #14346)
- `mysqldump` incorrectly tried to use `LOCK TABLES` for tables in the `INFORMATION_SCHEMA` database. (Bug #21527)
- Use of the `--prompt` option or `prompt` command caused `mysql` to be unable to connect to the Instance Manager. (Bug #17485)
- The server crashed if it tried to access a `CSV` table for which the data file had been removed. (Bug #15205)
- `CREATE USER` did not respect the 16-character username limit. (Bug #10668)
- Creating a partitioned table that used the `InnoDB` storage engine and then restarting `mysqld` with `--skip-innodb` caused MySQL to crash. (Bug #20871)
- In mixed-format binary logging mode, stored functions, triggers, and views that use functions in their body that require row-based logging did not replicate reliably because the logging did not switch from statement-based to row-based format. For example, `INSERT INTO t SELECT FROM v`, where `v` is a view that selects `UUID()` could cause problems. This limitation has been removed. (Bug #20930)
- For user-defined functions created with `CREATE FUNCTION`, the `DEFINER` clause is not legal, but no error was generated. (Bug #21269)
- `mysqld --flush` failed to flush `MyISAM` table changes to disk following an `UPDATE` statement for which no updated column had an index. (Bug #20060)
- In mixed-format binary logging mode, stored functions, triggers, and views that use functions in their body that require row-based logging did not replicate reliably because the logging did not switch from statement-based to row-based format. For example, `INSERT INTO t SELECT FROM v`, where `v` is a view that selects `UUID()` could cause problems. This limitation has been removed. (Bug #20930)
- When not running in strict mode, the server failed to convert the invalid years portion of a `DATE` or `DATETIME` value to '0000' when inserting it into a table. (Bug #19370)
- The dropping of a temporary table whose name contained a backtick (``) character was not correctly written to the binary log, which also caused it not to be replicated correctly. (Bug #19188)

- Intermediate tables created during the execution of an `ALTER TABLE` statement were visible in the output of `SHOW TABLES`. (Bug #18775)
- When setting a column to its implicit default value as the result of inserting a `NULL` into a `NOT NULL` column as part of a multi-row insert or `LOAD DATA` operation, the server returned a misleading warning message. (Bug #14770)
- The `--with-collation` option was not honored for client connections. (Bug #7192)
- Users who had the `SHOW VIEW` privilege for a view and privileges on one of the view's base table could not see records in `INFORMATION_SCHEMA` tables relating to the base table. (Bug #20543)
- An issue with yaSSL prevented Connector/J clients from connecting to the server using a certificate. (Bug #19705)
- Some server errors were not reported to the client, causing both to try to read from the connection until a hang or crash resulted. (Bug #16581)
- `FEDERATED` tables raised invalid duplicate key errors when attempting on one server to insert rows having the same primary key values as rows that had been deleted from the linked table on the other server. (Bug #18764)
- The C API failed to return a status message when invoking a stored procedure. (Bug #15752)
- `AUTHORS` and `CONTRIBUTORS` were not treated as reserved words. (Bug #19939)
- Stored procedures did not use the character set defined for the database in which they were created. (Bug #16676)
- `CREATE PROCEDURE`, `CREATE FUNTION`, `CREATE TRIGGER`, and `CREATE VIEW` statements containing multi-line comments (`/* ... */`) could not be replicated. (Bug #20438)
- The final parenthesis of a `CREATE INDEX` statement occurring in a stored procedure was omitted from the binary log when the stored procedure was called. (Bug #19207)
- Attempting to insert a string of greater than 4096 bytes into a `FEDERATED` table resulted in the error `ERROR 1296 (HY000) at line 2: Got error 10000 'Error on remote system: 1054: Unknown column 'string-value' from FEDERATED`. This error was raised regardless of the type of column involved (`VARCHAR`, `TEXT`, and so on.) (Bug #17608)
- Performance during an import on a table with a trigger that called a stored procedure was severely degraded. (Bug #21013)
- Repeated `DROP TABLE` statements in a stored procedure could sometimes cause the server to crash. (Bug #19399)
- On 64-bit Windows, a missing table generated error 1017, not the correct value of 1146. (Bug #21396)
- The same trigger error message was produced under two conditions: The trigger duplicated an existing trigger name, or the trigger duplicated an existing combination of action and event. Now different messages are produced for the two conditions so as to be more informative. (Bug #10946)
- The value returned by a stored function returning a string value was not of the declared character set. (Bug #16211)
- `FLUSH TABLES` followed by a `LOCK TABLES` statement to lock a log table and a non-log table caused an infinite loop and high CPU use. Now `FLUSH TABLES` ignores log tables. To flush the log tables, use `FLUSH LOGS` instead. (Bug #20139)
- If a filename was specified for the `--log` or `--log-slow_queries` options but the server was logging to tables and not files, the server produced no error message. (Bug #17599)
- `mysqlcheck` tried to check views instead of ignoring them. (Bug #16502)
- Long multiple-row `INSERT` statements could take a very long time for some multi-byte character sets. (Bug #15811)
- For `mysql`, escaping with backslash sometimes did not work. (Bug #20103)
- Under certain circumstances, `AVG(key_val)` returned a value but `MAX(key_val)` returned an empty set due to incorrect application of `MIN()/MAX()` optimization. (Bug #20954)

- Using aggregate functions in subqueries yielded incorrect results under certain circumstances due to incorrect application of `MIN()/MAX()` optimization. (Bug #20792)
- A query using `WHERE column = constant OR column IS NULL` did not return consistent results on successive invocations. The `column` in each part of the `WHERE` clause could be either the same column, or two different columns, for the effect to be observed. (Bug #21019)
- The `PASSWORD()` function returned invalid results when used in some `UNION` queries. (bug #16881)
- `USE` did not refresh database privileges when employed to re-select the current database. (Bug #10979)
- A query using `WHERE NOT (column < ANY (subquery))` yielded a different result from the same query using the same `column` and `subquery` with `WHERE (column > ANY (subquery))`. (Bug #20975)
- A user variable set to a value selected from an unsigned column was stored as a signed value. (Bug #7498)
- `SELECT` statements using `GROUP BY` against a view could have missing columns in the output when there was a trigger defined on one of the base tables for the view. (Bug #20466)
- A `SELECT` with a subquery that was bound to the outer query over multiple columns returned different results when a constant was used instead of one of the dependant columns. (Bug #18925)
- **InnoDB**: Quoted Unicode identifiers were not handled correctly. This included names of tables, columns, and foreign keys. (Bug #18800)
- A stored procedure that created and invoked a prepared statement was not executed when called in a `mysqld` init-file. (Bug #17843)
- Using the extended syntax for `TRIM()` — that is, `TRIM(... FROM ...)` — in a `SELECT` statement defining a view caused an invalid syntax error when selecting from the view. (Bug #17526)
- Assignments of values to variables of type `TEXT` were handled incorrectly in stored routines. (Bug #17225)
- When performing a `GROUP_CONCAT()`, the server transformed `BLOB` columns `VARCHAR` columns, which could cause erroneous results when using Connector/J and possibly other MySQL APIs. (Bug #16712)
- The type of the value returned by the `VARIANCE()` function varied according to the type of the input value. The function should always return a `DOUBLE` value. (Bug #10966)
- Performing an `INSERT` on a view that was defined using a `SELECT` that specified a collation and a column alias caused the server to crash (Bug #21086).
- A query of the form shown here caused the server to crash:

```
SELECT * FROM t1 NATURAL JOIN (
  t2 JOIN (
    t3 NATURAL JOIN t4,
    t5 NATURAL JOIN t6
  )
  ON (t3.id3 = t2.id3 AND t5.id5 = t2.id5)
);
```

(Bug #21007)

- `myisam_ftdump` would fail when trying to open a MyISAM index file that you did not have write permissions to access, even though the command would only be reading from the file. (Bug #17122)
- `REPLACE ... SELECT` for a view required the `INSERT` privilege for tables other than the table being modified. (Bug #20989)
- `mysqldump` sometimes did not select the correct database before trying to dump views from it, resulting in an empty result set that caused `mysqldump` to die with a segmentation fault. (Bug #21014)
- With mixed-format binary logging, `INSERT DELAYED` statements were logged using statement-based logging, and they did not replicate properly for statements that used values such as `UUID()`, `RAND()`, or user-defined variables

that require row-based logging. To correct this, the `DELAYED` handler thread now switches to row-based logging if the logging format is mixed. (Bug #20633, Bug #20649)

- Using `EXPLAIN PARTITIONS` with a `UNION` query could crash the server. This could occur whether or not the query actually used any partitioned tables. (Bug #20484)
- Partition pruning could cause incorrect results from queries, such as missing rows, when the partitioning expression relied on a `BIGINT UNSIGNED` column. (Bug #20257)
- The implementation for `UNCOMPRESS()` did not indicate that it could return `NULL`, causing the optimizer to do the wrong thing. (Bug #18539)
- `TIMESTAMPDIFF()` examined only the date and ignored the time when the requested difference unit was months or quarters. (Bug #16226)
- `pererror` did not properly report `NDB` error codes. (Bug #16561)
- `mysqlimport` sends a `set @@character_set_database=binary` statement to the server, but this is not understood by pre-4.1 servers. Now `mysqlimport` encloses the statement within a `/*!40101 ... */` comment so that old servers will ignore it. (Bug #15690)
- The character set was not being properly initialized for `CAST()` with a type like `CHAR(2) BINARY`, which resulted in incorrect results or even a server crash. (Bug #17903)
- For ODBC compatibility, MySQL supports use of `WHERE col_name IS NULL` for `DATE` or `DATETIME` columns that are `NOT NULL`, to allow column values of `'0000-00-00'` or `'0000-00-00 00:00:00'` to be selected. However, this was not working for `WHERE` clauses in `DELETE` statements. (Bug #8143)
- The `--master-data` option for `mysqldump` requires certain privileges, but `mysqldump` generated a truncated dump file without producing an appropriate error message or exit status if the invoking user did not have those privileges. (Bug #21215)
- `ALTER VIEW` did not retain existing values of attributes that had been originally specified but were not changed in the `ALTER VIEW` statement. (Bug #21080)
- `mysql` crashed for very long arguments to the `connect` command. (Bug #21042)
- `pererror` crashed on Solaris due to `NULL` return value of `strerror()` system call. (Bug #20145)
- The `query` command for `mysqltest` did not work. (Bug #19890)
- For certain queries, the server incorrectly resolved a reference to an aggregate function and crashed. (Bug #20868)
- When executing a `SELECT` with `ORDER BY` on a view that is constructed from a `SELECT` statement containing a stored function, the stored function was evaluated too many times. (Bug #19862)
- A `SELECT` that used a subquery in the `FROM` clause that did not select from a table failed when the subquery was used in a join. (Bug #21002)
- Subqueries on `INFORMATION_SCHEMA` tables could erroneously return an empty result. (Bug #21231)
- Issuing a `SHOW CREATE FUNCTION` or `SHOW CREATE PROCEDURE` statement without sufficient privileges could crash the `mysql` client. (Bug #20664)
- In a view defined with `SQL SECURITY DEFINER`, the `CURRENT_USER()` function returned the invoker, not the definer. (Bug #20570)
- `DATE_ADD()` and `DATE_SUB()` returned `NULL` when the result date was on the day `'9999-12-31'`. (Bug #12356)
- For a `DATE` parameter sent via a `MYSQL_TIME` data structure, `mysql_stmt_execute()` zeroed the hour, minute, and second members of the structure rather than treating them as read-only. (Bug #20152)
- The `DATA DIRECTORY` table option did not work for `TEMPORARY` tables. (Bug #8706)
- If the files for an open table were removed at the OS level (external to the server), the server exited with an assertion failure. (Bug #16532)

- Some memory leaks in the [libmysqld](#) embedded server were corrected. (Bug #16017)
- With the [auto_increment_increment](#) system variable set larger than 1, if the next generated [AUTO_INCREMENT](#) value would be larger than the column's maximum value, the value would be clipped down to that maximum value and inserted, even if the resulting value would not be in the generated sequence. This could cause problems for master-master replication. Now the server clips the value down to the previous value in the sequence, which correctly produces a duplicate-key error if that value already exists in the column. (Bug #20524)
- If a table on a slave server had a higher [AUTO_INCREMENT](#) counter than the corresponding master table (even though all rows of the two tables were identical), in some cases [REPLACE](#) or [INSERT ... ON DUPLICATE KEY UPDATE](#) would not replicate properly using statement-based logging. (Different values would be inserted on the master and slave.) (Bug #20188)
- [mysqslap](#) did not enable the [CLIENT_MULTI_RESULTS](#) flag when connecting, which is necessary for executing stored procedures. (Bug #20365)
- When creating a table using [CREATE...SELECT](#) and a stored procedure, there would be a mismatch between the binary log and transaction cache which would cause a server crash. (Bug #21039)
- When run with the [--use-threads](#) option, [mysqlimport](#) returned a random exit code. (Bug #21188)
- The effect of a stored function or trigger that caused [AUTO_INCREMENT](#) values to be generated for multiple tables was not logged properly if statement-based logging was used. Only the first table's value was logged, causing replication to fail. Under mixed logging format, this is dealt with by switching to row-based logging for the function or trigger. For statement-based logging, this remains a problem. (Bug #19630)
- Changing the definition of a [DECIMAL](#) column with [ALTER TABLE](#) caused loss of column values. (Bug #18014)
- Under heavy load (executing more than 1024 simultaneous complex queries), a problem in the code that handles internal temporary tables could lead to writing beyond allocated space and memory corruption. Use of more than 1024 simultaneous cursors server wide also could lead to memory corruption. (This applies both to stored procedure and C API cursors.) (Bug #21206)
- A race condition during slave server shutdown caused an assert failure. (Bug #20850)
- [mysqldump](#) produced a malformed dump file when dumping multiple databases that contained views. (Bug #20221)
- Partitions were represented internally as the wrong data type, which led in some cases to failures of queries such as [SELECT COUNT\(*\) FROM INFORMATION_SCHEMA.PARTITIONS WHERE PARTITION_NAME = 'partition_name'](#). (Bug #20340)
- Searches against a [ZEROFILL](#) column of a partitioned table could fail when the [ZEROFILL](#) column was part of the table's partitioning key. (Bug #20733)
- In mixed binary logging mode, a temporary switch from statement-based logging to row-based logging occurs when storing a row that uses a function such as [UUID\(\)](#) into a temporary table. However, temporary table changes are not written to the binary log under row-based logging, so the row does not exist on the slave. A subsequent select from the temporary table to a non-temporary table using statement-based logging works correctly on the master, but not on the slave where the row does not exist. The fix for this is that replication does not switch back from row-based logging to statement-based logging until there are no temporary tables for the session. (Bug #20499)
- Re-executing a stored procedure with a complex stored procedure cursor query could lead to a server crash. (Bug #15217)
- Views created from prepared statements inside of stored procedures were created with a definition that included both [SQL_CACHE](#) and [SQL_NO_CACHE](#). (Bug #17203)
- Updating a column of a [FEDERATED](#) table to [NULL](#) sometimes failed. (Bug #16494)
- Performing [INSERT ... SELECT ... JOIN ... USING](#) without qualifying the column names caused [ERROR 1052 "column 'x' in field list is ambiguous"](#) even in cases where the column references were unambiguous. (Bug #18080)
- Closing of temporary tables failed if binary logging was not enabled. (Bug #20919)

- For statements that have a [DEFINER](#) clause such as [CREATE TRIGGER](#) or [CREATE VIEW](#), long usernames or hostnames could cause a buffer overflow. (Bug #16899)
- [mysqldump](#) would not dump views that had become invalid because a table named in the view definition had been dropped. Instead, it quit with an error message. Now you can specify the [--force](#) option to cause [mysqldump](#) to keep going and write a SQL comment containing the view definition to the dump output. (Bug #17371)
- [InnoDB](#) (Partitioning): Updating an [InnoDB](#) table using [HASH](#) partitioning with a composite primary key would cause the server to hang. (Bug #20852)
- Old partition and subpartition files were not always removed following [ALTER TABLE ... REORGANIZE PARTITION](#) statements. (Bug #20770)
- Merging multiple partitions having subpartitions into a single partition with subpartitions, or splitting a single partition having subpartitions into multiple partitions with subpartitions, could sometimes crash the server. These issues were associated with a failure reported in the [partition_range](#) test. (Bug #20766, Bug #20767, Bug #20893, Bug #21357)
- Some queries using [ORDER BY ... DESC](#) on subpartitioned tables could crash the server. (Bug #20389)
- Referring to a stored function qualified with the name of one database and tables in another database caused a `「table doesn't exist」` error. (Bug #18444)
- For [NDB](#) and possibly [InnoDB](#) tables, a [BEFORE UPDATE](#) trigger could insert incorrect values. (Bug #18437)
- For multiple [INSERT DELAYED](#) statements executed in a batch by the delayed-insert handler thread, not all rows were written to the binary log. (Bug #20821)
- Triggers on tables in the [mysql](#) database caused a server crash. Triggers for tables in this database now are disallowed. (Bug #18005, Bug #18361)
- The length of the pattern string prefix for [LIKE](#) operations was calculated incorrectly for multi-byte character sets. As a result, the scanned range was wider than necessary if the prefix contained any multi-byte characters, and rows could be missing from the result set. (Bug #16674, Bug #18359)
- Very complex [SELECT](#) statements could create temporary tables that were too big, but for which the temporary files did not get removed, causing subsequent queries to fail. (Bug #11824)
- Multiple-table updates with [FEDERATED](#) tables could cause a server crash. (Bug #19773)
- On Windows, terminating [mysqld](#) with Control-C could result in a crash during shutdown. (Bug #18235)
- Renaming a database to itself caused a server crash. (Bug #19392)
- For spatial data types, the server formerly returned these as [VARSTRING](#) values with a binary collation. Now the server returns spatial values as [BLOB](#) values. (Bug #10166)
- Using tables from MySQL 4.x in MySQL 5.x, in particular those with [VARCHAR](#) fields and using [INSERT DELAYED](#) to update data in the table would result in either data corruption or a server crash. (Bug #16611, Bug #16218, Bug #17294)
- Using [SELECT](#) and a table join while running a concurrent [INSERT](#) operation would join incorrect rows. (Bug #14400)
- Using [SELECT](#) on a corrupt [MyISAM](#) table using the dynamic record format could cause a server crash. (Bug #19835)
- Checking a [MyISAM](#) table (using [CHECK TABLE](#)) having a spatial index and only one row would wrongly indicate that the table was corrupted. (Bug #17877)
- A `Table ... doesn't exist` error could occur for statements that called a function defined in another database. (Bug #17199)
- [SHOW GRANTS FOR CURRENT_USER](#) did not return definer grants when executed in [DEFINER](#) context (such as within a stored procedure defined with [SQL SECURITY DEFINER](#)), it returned the invoker grants. (Bug #15298)

- Portions of statements related to partitioning were not surrounded by version-specific comments by `mysqldump`, breaking backwards compatibility for dump files. (Bug #19488)
- A `DELETE FROM table` with no `WHERE` clause (deleting all rows) running concurrently with `INSERT` statements on a storage engine with row-level locking (such as `NDB`) could produce inconsistent results when using statement-based replication. (Bug #19066)
- Concatenating the results of multiple constant subselects produced incorrect results. (Bug #16716)
- The use of `MIN()` and `MAX()` on columns with a partial index produced incorrect results in some queries. (Bug #18206)
- The `WITH CHECK OPTION` was not enforced when a `REPLACE` statement was executed against a view. (Bug #19789)
- For `SELECT ... FOR UPDATE` statements that used `DISTINCT` or `GROUP BY` over all key parts of a unique index (or primary key), the optimizer unnecessarily created a temporary table, thus losing the linkage to the underlying unique index values. This caused a `Result set not updatable` error. (The temporary table is unnecessary because under these circumstances the distinct or grouped columns must also be unique.) (Bug #16458)
- A buffer overwrite error in Instance Manager caused a crash. (Bug #20622)
- Re-execution of a prepared multiple-table `DELETE` statement that involves a trigger or stored function can result in a server crash. (Bug #19634)
- On Windows, corrected a crash stemming from differences in Visual C runtime library routines from POSIX behavior regarding invalid file descriptors. (Bug #18275)
- Creation of a view as a join of views or tables could fail if the views or tables are in different databases. (Bug #20482)
- Use of `MIN()` or `MAX()` with `GROUP BY` on a `ucs2` column could cause a server crash. (Bug #20076)
- `INSERT INTO ... SELECT ... LIMIT 1` could be slow because the `LIMIT` was ignored when selecting candidate rows. (Bug #9676)
- Queries on tables that were partitioned by `KEY` and had a `VARCHAR` column as the partitioning key produced an empty result set. (Bug #20086)
- The omission of leading zeros in dates could lead to erroneous results when these were compared with the output of certain date and time functions. (Bug #16377)
- An invalid comparison between keys in partial indexes over multi-byte character fields could lead to incorrect result sets if the selected query execution plan used a range scan by a partial index over a `UTF8` character field. This also caused incorrect results under similar circumstances with many other character sets. (Bug #14896)
- A prepared statement that altered partitioned table within a stored procedure failed with the error `Unknown prepared statement handler`. (Bug #17138)
- A query selecting records from a single partition of a partitioned table and using `ORDER BY ic DESC` (where `ic` represents an indexed column) could cause errors or crash the server. (Bug #20583)
- **NDB Cluster** (Disk Data): On some platforms, `ndbd` compiled with `gcc 4` would crash when attempting to run `CREATE LOGFILE GROUP`. (Bug #21981)
- **NDB Cluster**: Setting `TransactionDeadlockDetectionTimeout` to a value greater than 12000 would cause scans to deadlock, time out, fail to release scan records, until the cluster ran out of scan records and stopped processing. (Bug #21800)
- **NDB Cluster**: Data was stored unevenly between partitions due to all `BLOB` data being placed in partition 0. (Bug #21690)
- **NDB Cluster**: The server provided a non-descriptive error message when encountering a fatally corrupted REDO log. (Bug #21615)

- **NDB Cluster:** A partial rollback could lead to node restart failures. (Bug #21536)
- **NDB Cluster:** The failure of a unique index read due to an invalid schema version could be handled incorrectly in some cases, leading to unpredictable results. (Bug #21384)
- **NDB Cluster:** In a cluster with more than 2 replicas, a manual restart of one of the data nodes could fail and cause the other nodes in its nodegroup to shut down. (Bug #21213)
- **NDB Cluster:** When the redo buffer ran out of space, a **Pointer too large** error was raised and the cluster could become unusable until restarted with `--initial`. (Bug #20892)
- **NDB Cluster:** In some situations with a high disk-load, writing of the redo log could hang, causing a crash with the error message **GCP STOP detected**. (Bug #20904)
- **NDB Cluster:** A vague error message was returned when reading of both schema files occurred during a restart of the cluster. (Bug #20860)
- **NDB Cluster:** The server did not honor the value set for `ndb_cache_check_time` in the `my.cnf` file. (Bug #20708)
- **NDB Cluster:** The server failed with a non-descriptive error message when out of data memory. (Bug #18475)
- **NDB Cluster** (Direct APIs): Invoking the MGM API function `ndb_mgm_listen_event()` caused a memory leak. (Bug #21671)
- **NDB Cluster:** The management client **ALL STATUS** command could sometimes report the status of some data nodes incorrectly. (Bug #13985)
- **NDB Cluster** (Disk Data): Trying to create a Disk Data table using a nonexistent tablespace or trying to drop a nonexistent data file from a tablespace produced an uninformative error message. (Bug #21751)
- **NDB Cluster** (Disk Data): Errors could occur when dropping a data file during a node local checkpoint. (Bug #21710)
- **NDB Cluster** (Disk Data): Creating a tablespace and log file group, then attempting to restart the cluster without using the `--initial` option and without having created any Disk Data tables could cause a forced shutdown of the cluster and raise a configuration error. (Bug #21172)
- **NDB Cluster:** Responses to the **ALL DUMP 1000** management client command were printed multiple times in the cluster log for each cluster node. (Bug #21044)
- A memory leak was found when running `ndb_mgm -e "SHOW"`. (Bug #21670)
- **NDB Cluster** (Direct APIs): The MGM API function `ndb_logevent_get_fd()` was not actually implemented. (Bug #21129)
- **NDB Cluster:** Restarting a data node while DDL operations were in progress on the cluster could cause other data nodes to fail. This could also lead to `mysqld` hanging or crashing under some circumstances. (Bug #21017, Bug #21050)
- **NDB Cluster:** A cluster data node could crash when an ordered index became full before the table containing the index was full. (Bug #14935)
- **NDB Cluster:** The repeated creating and dropping of a table would eventually lead to **NDB Error 826, Too many tables and attributes ... Insufficient space**. (Bug #20847)
- **NDB Cluster:** **REPLACE** statements did not work correctly on an **NDB** table having both a primary key and a unique key. In such cases, proper values were not set for columns which were not explicitly referenced in the statement. (Bug #20728)
- **NDB Cluster:** Trying to create or drop a table while a node was restarting caused the node to crash. This is now handled by raising an error. (Bug #18781)
- **NDB Cluster:** A race condition could in some circumstances following a **DROP TABLE**. (Bug #20897)
- **NDB Cluster:** Running `ndbd --nowait-nodes=id` where `id` was the node ID of a node that was already running would fail with an invalid error message. (Bug #20419)

- **NDB Cluster:** When stopping and restarting multiple data nodes, the last node to be restarted would sometimes hang in Phase 100. (Bug #19645)
- **NDB Cluster:** The `DATA_LENGTH` and `AVG_ROW_LENGTH` columns of the `INFORMATION_SCHEMA.TABLES` table did not report the size of variable-width column values correctly. (Bug #18413)

See [INFORMATION_SCHEMA TABLES テーブル](#), for more information.

- **NDB Cluster:** When attempting to restart the cluster following a data import, the cluster would fail during Phase 4 of the restart with **Error 2334: Job buffer congestion**. (Bug #20774)
- **NDB Cluster:** A node failure during a scan could sometime cause the node to crash when restarting too quickly following the failure. (Bug #20197)
- **NDB Cluster:** It was possible to use port numbers greater than 65535 for `ServerPort` in the `config.ini` file. (Bug #19164)
- **NDB Cluster (Replication):** In some cases, a large number of MySQL servers sending requests to the cluster simultaneously could cause the cluster to crash. This could also be triggered by many NDB API clients making simultaneous event subscriptions or unsubscriptions. (Bug #20683)
- **NDB Cluster (Direct APIs):** `NdbScanOperation::readTuples()` and `NdbIndexScanOperation::readTuples()` ignored the `batch` parameter. (Bug #20252)
- **NDB Cluster:** Cluster system status variables were not updated. (Bug #11459)
- **NDB Cluster (Disk Data):** Trying to create Disk Data tables when running the cluster in diskless mode would crash the cluster's data nodes. (Bug #20008)

Note: Disk Data tables are now disabled when running in diskless mode.

- **NDB Cluster (Disk Data):** A data file created on one tablespace could be dropped using `ALTER TABLESPACE ... DROP DATAFILE` on a different tablespace. (Bug #20053)
- **NDB Cluster:** Truncating a table on one `mysqld` caused other `mysqld` processes in the cluster to return **ERROR 1412 (HY000): Table definition has changed, please retry transaction** on subsequent queries. (Bug #20705)
- **NDB Cluster:** The cluster's data nodes would fail while trying to load data when `NoOfFrangmentLogFiles` was equal to 1. (Bug #19894)
- **NDB Cluster:** A problem with error handling when `ndb_use_exact_count` was enabled could lead to incorrect values returned from queries using `COUNT()`. A warning is now returned in such cases. (Bug #19202)
- **NDB Cluster:** Restarting a failed node could crash the cluster. (Bug #18782)
- **NDB Cluster (Disk Data):** The failure of a `CREATE TABLESPACE` or `CREATE LOGFILE GROUP` statement did not revert all changes made prior to the point of failure. (Bug #16341)
- **NDB Cluster:** Creating tables with variable-size columns caused `DataMemory` to be used but not freed when the tables were dropped. (Bug #20007)
- **NDB Cluster:** Restoring a backup made using `ndb_restore` failed when the backup had been taken from a cluster whose data memory was full. (Bug #19852)
- **NDB Cluster:** An excessive number of `ALTER TABLE` operations could cause the cluster to fail with NDB error code 773 (**Out of string memory, please modify StringMemory**). (Bug #19275)
- **NDB Cluster:** `TEXT` columns in Cluster tables having both an explicit primary key and a unique key were not correctly updated by `REPLACE` statements. (Bug #19906)
- **NDB Cluster:** Running out of `DataMemory` could sometimes crash `ndbd` and `mysqld` processes. (Bug #19185)
- **NDB Cluster (Replication):** A node failure could send duplicate events, causing a `mysqld` replicating tables containing `BLOBs` to crash.

- **NDB Cluster** (Disk Data): **INFORMATION_SCHEMA.FILES** records for **UNDO** files showed incorrect values in the **EXTENT_SIZE**, **FREE_EXTENTS**, and **TOTAL_EXTENTS** columns. (Bug #20073)
- **NDB Cluster**: An internal formatting error caused some management client error messages to be unreadable. (Bug #20016)
- **NDB Cluster**: Running management client commands while **mgmd** was in the process of disconnecting could cause the management server to fail. (Bug #19932)
- **NDB Cluster** (NDBAPI): Update operations on blobs were not checked for illegal operations.
Note: Read locks with blob update operations are now upgraded from read committed to read shared.
- **NDB Cluster**: The management client **ALL STOP** command shut down **mgmd** processes (as well as **ndbd** processes). (Bug #18966)
- **NDB Cluster**: Renaming of table columns was not supported as fast a **ALTER TABLE** for NDB tables. (Bug #20456)
- **NDB Cluster**: Under some circumstances, repeated DDL operations on one **mysqld** could cause failure of a second **mysqld** attached to the same cluster. (Bug #19770)
- **NDB Cluster** (Replication): One or more of the **mysqld** processes could fail when subjecting a Cluster replication setup with multiple **mysqld** processes on both the master and slave clusters to high loads. (Bug #19768)
- **NDB Cluster**: **LOAD DATA LOCAL** failed to ignore duplicate keys in Cluster tables. (Bug #19496)
- **NDB Cluster**: Repeated **CREATE - INSERT - DROP** operations tables could in some circumstances cause the MySQL table definition cache to become corrupt, so that some **mysqld** processes could access table information but others could not. (Bug #18595)
- **NDB Cluster** (Disk Data): Running a large number of scans on Disk Data could cause subsequent scans to perform poorly. (Bug #20334)
- **NDB Cluster** (Disk Data): An issue with disk allocation could sometimes cause a forced shutdown of the cluster when running a mix of memory and Disk Data tables. (Bug #18780)
- **NDB Cluster**: The **mgm** client command **ALL CLUSTERLOG STATISTICS=15**; had no effect. (Bug #20336)
- **NDB Cluster**: Under certain conditions, a starting node could miss transactions, leading to inconsistencies between the primary and backup replicas. (Bug #19929)
- **NDB Cluster**: An uncommitted row could sometimes be checkpointed and thus incorrectly included in a backup. (Bug #19928)
- **NDB Cluster**: A **DELETE** of many rows immediately followed by an **INSERT** on the same table could cause the **ndbd** process on the backup replica to crash. (Bug #19293)
- **NDB Cluster**: **TRUNCATE TABLE** failed to reset the **AUTO_INCREMENT** counter. (Bug #18864)
- **NDB Cluster**: **SELECT ... FOR UPDATE** failed to lock the selected rows. (Bug #18184)
- **NDB Cluster**: New **mysqld** processes were allowed to connect without a restart of the cluster, causing the cluster to crash. (Bug #13266)
- **NDB Cluster**: The failure of a data node when preparing to commit a transaction (that is, while the node's status was **CS_PREPARE_TO_COMMIT**) could cause the failure of other cluster data nodes. (Bug #20185)
- **NDB Cluster**: Renaming a table in such a way as to move it to a different database failed to move the table's indexes. (Bug #19967)
- **NDB Cluster**: **SHOW ENGINE NDB STATUS** could sometimes return an incorrect value of **0** for the latest epoch, which could cause problems with synchronising the binlog. (Bug #20142)
- **NDB Cluster**: A **CREATE TABLE** statement involving foreign key constraints raised an error rather than being silently ignored (see **CREATE TABLE 構文**). (Bug #18483)

This bug affected Cluster in MySQL 5.1 only.

- **NDB Cluster** (Replication): Data definition and data manipulation statements on different tables were not serialised correctly in the binlog. For example, there was no guarantee that a **CREATE TABLE** statement and an update on a different table would occur in the same order in the binlog as they did on the cluster being replicated. (Bug #18947)
- **NDB Cluster**: Resources for unique indexes on Cluster table columns were incorrectly allocated, so that only one-fourth as many unique indexes as indicated by the value of **UniqueHashIndexes** could be created. (Bug #19623)
- A cast problem caused incorrect results for prepared statements that returned float values when MySQL was compiled with **gcc 4.0**. (Bug #19694)
- Some queries that used **ORDER BY** and **LIMIT** performed quickly in MySQL 3.23, but slowly in MySQL 4.x/5.x due to an optimizer problem. (Bug #4981)
- **mysql_upgrade** was missing from binary MySQL distributions. (Bug #18516, Bug #20403, Bug #20556)
- It was possible using **ALTER EVENT ... RENAME ...** to move an event to a database on which the user did not have the **EVENT** privilege. (Bug #18897)
- A number of dependency issues in the RPM **bench** and **test** packages caused installation of these packages to fail. (Bug #20078)
- Queries using an indexed column as the argument for the **MIN()** and **MAX()** functions following an **ALTER TABLE ... DISABLE KEYS** statement returned **Got error 124 from storage engine** until **ALTER TABLE ... ENABLE KEYS** was run on the table. (Bug #20357)
- A redundant table map event could be generated in the binary log when there were no actual changes to a table being replicated. In addition, a slave failed to stop when attempting to replicate a table that did not exist on the slave. (Bug #18948)
- Multiple calls to a stored procedure that altered a partitioned **MyISAM** table would cause the server to crash. (Bug #19309)
- Adding an index to a partitioned table that had been created using **AUTO_INCREMENT = value** caused the **AUTO_INCREMENT** value to be reset. (Bug #19281)
- A **CREATE TABLE** that produced a **The PARTITION function returns the wrong type** error also caused an **Incorrect information in file** to be printed to **STDERR**, and a junk file to be left in the database directory. (Bug #16000)
- Nested natural joins worked executed correctly when executed as a non-prepared statement could fail with an **Unknown column 'col_name' in 'field list'** error when executed as a prepared statement, due to a name resolution problem. (Bug #15355)
- The **max_length** metadata value for columns created from **CONCAT()** could be incorrect when the collation of an argument differed from the collation of the **CONCAT()** itself. In some contexts such as **UNION**, this could lead to truncation of the column contents. (Bug #15962)
- The **MD5()** and **SHA()** functions treat their arguments as case-sensitive strings. But when they are compared, their arguments were compared as case-insensitive strings, which leads to two function calls with different arguments (and thus different results) compared as being identical. This can lead to a wrong decision made in the range optimizer and thus to an incorrect result set. (Bug #15351)
- For **BOOLEAN** mode full-text searches on non-indexed columns, **NULL** rows generated by a **LEFT JOIN** caused incorrect query results. (Bug #14708)
- If the general log table reached a large enough file size (27GB), **SELECT COUNT(*)** on the table caused a server crash. (Bug #17589)
- Identifiers could not contain bytes with a value of 255, though that should be allowed as of the identifier-encoding changes made in MySQL 5.1.6. (Bug #12982)
- **BIT** columns in a table could cause joins that use the table to fail. (Bug #18895)

- A [UNION](#) over more than 128 [SELECT](#) statements that use an aggregate function failed. (Bug #18175)
- [InnoDB](#) unlocked its data directory before committing a transaction, potentially resulting in non-recoverable tables if a server crash occurred before the commit. (Bug #19727)
- Multiple-table [DELETE](#) statements containing a subquery that selected from one of the tables being modified caused a server crash. (Bug #19225)
- With settings of [read_buffer_size](#) >= 2G and [read_rnd_buffer_size](#) >=2G, [LOAD DATA INFILE](#) failed with no error message or caused a server crash for files larger than 2GB. (Bug #12982)
- [REPLACE](#) statements caused activation of [UPDATE](#) triggers, not [DELETE](#) and [INSERT](#) triggers. (Bug #13479)
- The thread for [INSERT DELAYED](#) rows was maintaining a separate [AUTO_INCREMENT](#) counter, resulting in incorrect values being assigned if [DELAYED](#) and non-[DELAYED](#) inserts were mixed. (Bug #20195)
- [mysqldump](#) wrote an extra pair of [DROP DATABASE](#) and [CREATE DATABASE](#) statements if run with the [--add-drop-database](#) option and the database contained views. (Bug #17201)
- Shutting down a slave in a replication scenario where temporary tables are in use would cause the slave to produce a core dump. (Bug #19881)
- When a statement is executed that does not generate any rows, an extra table map event and associated binrows event would be generated and written to the binary log. (Bug #19995)
- File size specifications for [InnoDB](#) data files were case sensitive. (Bug #19609)
- Compilation on Windows would fail if row based replication was disabled using [--without-row-based-replication](#). (Bug #16837)
- [InnoDB](#) did not increment the [handler_read_prev](#) counter. (Bug #19542)
- In the [INFORMATION_SCHEMA.FILES](#) table, the [INITIAL_SIZE](#), [MAXIMUM_SIZE](#), and [AUTOEXTEND_SIZE](#) columns incorrectly were being stored as [VARCHAR](#) rather than [BIGINT](#). (Bug #19544).
- For [mysqld](#), Valgrind revealed problems that were corrected: Possible uninitialized data in a string comparison (Bug #20783); memory corruption in replication slaves when switching databases (Bug #19022); syscall write parameter pointing to uninitialized byte (Bug #20579); uninitialized doublewrite memory in [InnoDB](#) (Bug#20791).
- For [ndb_mgmd](#), Valgrind revealed problems that were corrected: A memory leak (Bug #19318); a dependency on an uninitialized variable (Bug #20333).
- An update that used a join of a table to itself and modified the table on both sides of the join reported the table as crashed. (Bug #18036)
- SSL connections using yaSSL on OpenBSD could fail. (Bug #19191)
- Following a failed attempt to add an index to an [ARCHIVE](#) table, it was no longer possible to drop the database in which the table had been created. (Bug #17310)
- Using [ALTER TABLE ... ENGINE = x](#), where [x](#) was not a storage engine supported by the server, would cause [mysqld](#) to crash. (Bug #20397)
- Defining a table partitioned by [LIST](#) with a single [PARTITION ... VALUES IN \(NULL\)](#) clause could lead to server crashes, particularly with queries having [WHERE](#) conditions comparing the partitioning key with a constant. (Bug #19801 / Bug #20268)
- Values greater than 2 gigabytes used in the VALUES LESS THAN clause of a table partitioned by [RANGE](#) were treated as negative numbers. (Bug #16002)
- The [fill_help_tables.sql](#) file did not load properly if the [ANSI_QUOTES](#) SQL mode was enabled. (Bug #20542)
- The [fill_help_tables.sql](#) file did not contain a [SET NAMES 'utf8'](#) statement to indicate its encoding. This caused problems for some settings of the MySQL character set such as [big5](#). (Bug #20551)

- The `--default-storage-engine` server option did not work. (Bug #20168)
- The MySQL server startup script `/etc/init.d/mysql` (created from `mysql.server`) is now marked to ensure that the system services `ypbind`, `nscd`, `ldap`, and `NTP` are started first (if these are configured on the machine). (Bug #18810)
- For a reference to a non-existent index in `FORCE INDEX`, the error message referred to a column, not an index. (Bug #17873)
- The `ENGINE` clause was displayed in the output of `SHOW CREATE TABLE` for partitioned tables when the SQL mode included `no_table_options`. (Bug #19695)
- `ALTER TABLE ... COALESCE PARTITION` did not delete the files associated with the partitions that were removed. (Bug #19305)
- `ALTER TABLE ... REBUILD PARTITION` could cause the server to hang or crash. (Bug #19122)
- Some yaSSL public function names conflicted with those from OpenSSL, causing conflicts for applications that linked against both OpenSSL and a version of `libmysqlclient` that was built with yaSSL support. The yaSSL public functions now are renamed to avoid this conflict. (Bug #19575)
- `CHECK TABLE` on a `MyISAM` table briefly cleared its `AUTO_INCREMENT` value, while holding only a read lock. Concurrent inserts to that table could use the wrong `AUTO_INCREMENT` value. `CHECK TABLE` no longer modifies the `AUTO_INCREMENT` value. (Bug #19604)
- If there is a global read lock, `CREATE DATABASE`, `RENAME DATABASE`, and `DROP DATABASE` could deadlock. (Bug #19815)
- `EXPLAIN PARTITIONS` would produce illegible output in the `partitions` column if the length of text to be displayed in that column was too long. This could occur when very many partitions were defined for the table, partitions were given very long names, or due to a combination of the two. (Bug #19684)
- Trying to execute a query having a `WHERE` clause using `int_col = "string_value" OR int_col IS NULL` on a partitioned table whose partitioning or subpartitioning function used the integer column `int_col` would crash the server. (Bug #19055)
- On Linux, `libmysqlclient` when compiled with yaSSL using the `icc` compiler had a spurious dependency on C++ libraries. (Bug #20119)
- In MySQL 5.1.11, the `--with-openssl` and `--with-yassl` options were replaced by `--with-ssl`. But no message was issued if the old options were given. Now `configure` produces a message indicating that the new option should be used and exits. (Bug #20002)
- Grant table modifications sometimes did not refresh the in-memory tables if the hostname was `"` or not specified. (Bug #16297)
- Invalid escape sequences in option files caused MySQL programs that read them to abort. (Bug #15328)
- Using `ALTER TABLE` on a subpartitioned table caused the server to crash. (Bug #19067)
- For a table having `LINEAR HASH` subpartitions, the `LINEAR` keyword did not appear in the `SUBPARTITION_METHOD` column of the `INFORMATION_SCHEMA.PARTITIONS` table. (Bug #20161)
- Race conditions on certain platforms could cause the Instance Manager to fail to initialize. (Bug #19391)
- `ALTER TABLE` on a table created prior to 5.0.3 would cause table corruption if the `ALTER TABLE` did one of the following:
 - Change the default value of a column.
 - Change the table comment.
 - Change the table password.
 (Bug #17001)

- An [ALTER TABLE](#) operation that does not need to copy data, when executed on a table created prior to MySQL 4.0.25, could result in a server crash for subsequent accesses to the table. (Bug #19192)
- The binary log lacked character set information for table name when dropping temporary tables. (Bug #14157)
- A [B-TREE](#) index on a [MEMORY](#) table erroneously reported duplicate entry error for multiple [NULL](#) values. (Bug #12873)
- Race conditions on certain platforms could cause the Instance Manager to try to restart the same instance multiple times. (Bug #18023)
- [OPTIMIZE TABLE](#) and [REPAIR TABLE](#) yielded incorrect messages or warnings when used on partitioned tables. (Bug #17455)
- Selecting data from a [MEMORY](#) table with a [VARCHAR](#) column and a [HASH](#) index over it returned only the first row matched. (Bug #18233)
- RPM packages had spurious dependencies on Perl modules and other programs. (Bug #13634)

Changes in release 5.1.11 (26 May 2006)

This is a new Beta development release, fixing recently discovered bugs.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise>をご覧ください。

Functionality added or changed:

- Incompatible change: The Event Scheduler can now be in one of three states (on, off, or the new suspended state). In addition, due to the fact that [SET GLOBAL event_scheduler](#); now acts in a synchronous rather than asynchronous manner, the Event Scheduler thread can no longer be activated or deactivated at run time. (Bug #17619)

For more information regarding these changes, see [Event Scheduler Overview](#).

- Previously, to build MySQL from source with SSL support enabled, you would invoke [configure](#) with either the [--with-openssl](#) or [--with-yassl](#) option. Those options both have been replaced by the [--with-ssl](#) option. By default, [--with-ssl](#) causes the bundled yaSSL library to be used. To select OpenSSL instead, give the option as [--with-ssl=path](#), where [path](#) is the directory where the OpenSSL header files and libraries are located.
- Added the [--ssl-verify-server-cert](#) option to MySQL client programs. This option causes the server's Common Name value in its certificate to be verified against the hostname used when connecting to the server, and the connection is rejected if there is a mismatch. Added [MYSQL_OPT_SSL_VERIFY_SERVER_CERT](#) option for the [mysql_options\(\)](#) C API function to enable this verification. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default. (Bug #17208)
- Added the [ssl_ca](#), [ssl_capath](#), [ssl_cert](#), [ssl_cipher](#), and [ssl_key](#) system variables, which display the values given via the corresponding command options. See [SSL コマンド オプション](#). (Bug#19606)
- [NDB Cluster](#): The limit of 2048 ordered indexes per cluster has been lifted. There is now no upper limit on the number of ordered indexes (including [AUTO_INCREMENT](#) columns) that may be used. (Bug #14509)
- Added the [log_queries_not_using_indexes](#) system variable. (Bug#19616)
- Added the [--angel-pid-file](#) option to [mysqlmanager](#) for specifying the file in which the angel process records its process ID when [mysqlmanager](#) runs in daemon mode. (Bug #14106)
- The [ENABLE KEYS](#) and [DISABLE KEYS](#) clauses for the [ALTER TABLE](#) statement are now supported for partitioned tables. (Bug #19502)
- It is now possible to use [NEW.var_name](#) values within triggers as [INOUT](#) parameters to stored procedures. (Bug #14635)
- The default for the [innodb_thread_concurrency](#) system variable was changed to [8](#). (Bug #15868)

- [mysql_explain_log](#) (a third-party program) is no longer included in MySQL distributions.

Bugs fixed:

- Security fix: An SQL-injection security hole has been found in multi-byte encoding processing. The bug was in the server, incorrectly parsing the string escaped with the [mysql_real_escape_string\(\)](#) C API function. (CVE-2006-2753, Bug #8378)

This vulnerability was discovered and reported by Josh Berkus <josh@postgresql.org> and Tom Lane <tgl@sss.pgh.pa.us> as part of the inter-project security collaboration of the OSDB consortium. For more information about SQL injection, please see the following text.

Discussion: An SQL-injection security hole has been found in multi-byte encoding processing. An SQL-injection security hole can include a situation whereby when a user supplied data to be inserted into a database, the user might inject SQL statements into the data that the server will execute. With regards to this vulnerability, when character set unaware-escaping is used (for example, [addslashes\(\)](#) in PHP), it is possible to bypass the escaping in some multi-byte character sets (for example, SJIS, BIG5 and GBK). As a result, a function such as [addslashes\(\)](#) is not able to prevent SQL-injection attacks. It is impossible to fix this on the server side. The best solution is for applications to use character set-aware escaping offered by a function such as [mysql_real_escape_string\(\)](#).

However, a bug was detected in how the MySQL server parses the output of [mysql_real_escape_string\(\)](#). As a result, even when the character set-aware function [mysql_real_escape_string\(\)](#) was used, SQL injection was possible. This bug has been fixed.

Workarounds: If you are unable to upgrade MySQL to a version that includes the fix for the bug in [mysql_real_escape_string\(\)](#) parsing, but run MySQL 5.0.1 or higher, you can use the [NO_BACKSLASH_ESCAPES](#) SQL mode as a workaround. (This mode was introduced in MySQL 5.0.1.) [NO_BACKSLASH_ESCAPES](#) enables an SQL standard compatibility mode, where backslash is not considered a special character. The result will be that queries will fail.

To set this mode for the current connection, enter the following SQL statement:

```
SET sql_mode='NO_BACKSLASH_ESCAPES';
```

You can also set the mode globally for all clients:

```
SET GLOBAL sql_mode='NO_BACKSLASH_ESCAPES';
```

This SQL mode also can be enabled automatically when the server starts by using the command-line option [--sql-mode=NO_BACKSLASH_ESCAPES](#) or by setting [sql-mode=NO_BACKSLASH_ESCAPES](#) in the server option file (for example, [my.cnf](#) or [my.ini](#), depending on your system).

- The patch for Bug #8303 broke the fix for Bug #8378 and was undone. (In string literals with an escape character (\) followed by a multi-byte character that has a second byte of (\), the literal was not interpreted correctly. The next byte now is escaped, not the entire multi-byte character. This means it a strict reverse of the [mysql_real_escape_string\(\)](#) function.)
- The client libraries had not been compiled for position-independent code on Solaris-SPARC and AMD x86_64 platforms. (Bug #13159, Bug #14202, Bug #18091)
- Altering a [VARCHAR](#) column in a [MyISAM](#) table to make it longer could cause corruption of the following column. (Bug #19386)
- A [CREATE TABLE](#) statement that created a table from a materialized view did not inherit default values from the underlying table. (Bug #19089)
- [NDB Cluster](#): A Cluster whose storage nodes were installed from the [MySQL-ndb-storage-*](#) RPMs could not perform [CREATE](#) or [ALTER](#) operations that made use of non-default character sets or collations. (Bug #14918)
- [NDB Cluster](#): mysqld processes did not always detect cluster shutdown, leading to issues with Cluster replication and schema distribution. (Bug #19395)
- [NDB Cluster](#): [SELECT MIN\(unique_column\)](#) from a Cluster table with user-defined partitioning crashed the server. (Bug #18730)

- Premature optimization of nested subqueries in the **FROM** clause that refer to aggregate functions could lead to incorrect results. (Bug #19077)
- For dates with 4-digit year parts less than 200, an implicit conversion to add a century was applied for date arithmetic performed with **DATE_ADD()**, **DATE_SUB()**, **+ INTERVAL**, and **- INTERVAL**. (For example, **DATE_ADD('0050-01-01 00:00:00', INTERVAL 0 SECOND)** became '2050-01-01 00:00:00'.) Now these operations return **NULL** rather than an incorrect non-**NULL** value. (Bug #18997)
- **BLOB** or **TEXT** arguments to or values returned from stored functions were not copied properly if too long and could become garbled. (Bug #18587)
- Simultaneous scheduled events whose actions conflicted with one another could crash the server. (Bug #16428)
- It was not possible to invoke a stored routine containing dynamic SQL from a scheduled event. (Bug #19264)
- **NDB Cluster**: Running **ALL START** in the **NDB** management client or restarting multiple nodes simultaneously could under some circumstances cause the cluster to crash. (Bug #19930)
- The result from **CONV()** is a string, but was not always treated the same way as a string when converted to a real value for an arithmetic operation. (Bug #13975)
- **CREATE TABLE ... SELECT ...** statements that used a stored function explicitly or implicitly (through a view) resulted in a **Table not locked** error. (Bug #12472, Bug #15137)
- Within a trigger, **SET** used the SQL mode of the invoking statement, not the mode in effect at trigger creation time. (Bug #6951)
- The server no longer uses a signal handler for signal 0 because it could cause a crash on some platforms. (Bug #15869)
- The embedded server crashed with row-based replication enabled. (Bug #18518)
- Display better error message for **ALTER TABLE** operations that will result in duplicate keys due to **AUTO_INCREMENT** resequencing. (Bug #14573)
- The **Data_free** column in the output of **SHOW TABLE STATUS** always displayed 0 for partitioned tables. (Bug #19501)
- Adding an index to a table created using partitioning by **KEY** and the **MEMORY** storage engine caused the server to crash. (Bug #19140)
- When creating a table using **CREATE TABLE ... PARTITION BY ... SELECT ...**, the partitioning clause was ignored. (Bug #19062)
- **ALTER TABLE ENGINE=...** failed when used to change a MySQL Cluster table having no explicit primary key to use a different storage engine. (Bug #19010)
Note: As a consequence of this fix, **SHOW CREATE TABLE** no longer displays auto-partitioning information for **NDBCluster** tables.
- **NDB Cluster** (NDBAPI): On big-endian platforms, **NdbOperation::write_attr()** did not update 32-bit fields correctly. (Bug #19537)
- **NDB Cluster**: Using 「stale」 **mysqld .FRM** files could cause a newly-restored cluster to fail. This situation could arise when restarting a MySQL Cluster using the **--initial** option while leaving connected **mysqld** processes running. (Bug #16875)
- **NDB Cluster** (Replication): Memory was not freed after some **ALTER TABLE** operations, which could cause **mysqld** processes to crash. (Bug #19885)
- **NDB Cluster** (NDBAPI): The **Ndb::dropEventOperation()** method failed to clean up all objects used, which could cause memory leaks to occur. (Bug #17610)
- **NDB Cluster**: Data node failures could cause excessive CPU usage by **ndb_mgmd**. (Bug #13987)
- **NDB Cluster**: **TRUNCATE** failed on tables having **BLOB** or **TEXT** columns with the error **Lock wait timeout exceeded**. This affected both in-memory and Disk Data tables. (Bug #19201)

- Revised memory allocation for local objects within stored functions and triggers to avoid memory leak for repeated function or trigger invocation. (Bug #17260)
- `EXPLAIN ... SELECT INTO` caused the client to hang. (Bug #15463)
- Symlinking `.mysql_history` to `/dev/null` to suppress statement history saving by `mysql` did not work. (`mysql` deleted the symlink and recreated `.mysql_history` as a regular file, and then wrote history to it.) (Bug #16803)
- The `basedir` and `tmpdir` system variables could not be accessed via `@@var_name` syntax. (Bug #1039)
- Corrected several problems with the treatment of the `--log-error` option by `mysqld_safe`. These problems were manifest as differences from `mysqld` in error log handling.
 - If a filename was given for `--log-error`, `mysqld_safe` ignored it and did not pass it to `mysqld`, which then wrote error information to `stderr` and resulted in incorrect log rotation when `FLUSH LOGS` was used.
 - `mysqld_safe` now adds `.err` to the end of the filename if no extension is present (the same as `mysqld`).
 - `mysqld_safe` treated a relative pathname as relative to its own current working directory. Now it treats a relative pathname as relative to the data directory (the same as `mysqld`).

In addition, some argument quoting problems were corrected. (Bug #6061)

- Returning the value of a system variable from a stored function caused a server crash. (Bug #18037)
- Use of uninitialized user variables in a subquery in the `FROM` clause resulted in bad entries in the binary log. (Bug #19136)
- `IS_USED_LOCK()` could return an incorrect connection identifier. (Bug #16501)
- Concurrent reading and writing of privilege structures could crash the server. (Bug #16372)

Changes in release 5.1.10 (Not released)

Note: This was an internal release only, and no binaries were published.

MySQL 5.1.10 includes the patches for recently reported security vulnerabilities in the MySQL client-server protocol. We would like to thank Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting these to us.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise>をご覧ください。

Functionality added or changed:

- Security enhancement: Added the global `max_prepared_stmt_count` system variable to limit the total number of prepared statements in the server. This limits the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. The current number of prepared statements is available through the `prepared_stmt_count` system variable. (Bug #16365)
- The `mysql_upgrade` command has been converted from a shell script to a C program, so it is available on non-Unix systems such as Windows. This program should be run for each MySQL upgrade. See [mysql_upgrade — MySQL アップグレードのテーブル チェック](#).
- Binary distributions that include SSL support now are built using yaSSL when possible.
- The `MySQL-shared-compat-5.1.X-i386.rpm` shared compatibility RPMs no longer contain libraries for MySQL 5.0. This avoids a conflict because the 5.0 and 5.1 libraries share the same soname number. It contains libraries for 3.23, 4.0, 4.1, and 5.1. (Bug #19288)
- The `ONLY_FULL_GROUP_BY` SQL mode now also applies to the `HAVING` clause. That is, columns not named in the `GROUP BY` clause cannot be used in the `HAVING` clause if not used in an aggregate function. (Bug #18739)
- SQL syntax for prepared statements now supports `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE`. (Bug #19308)

- XPath expressions passed to the [ExtractValue\(\)](#) and [UpdateXML\(\)](#) functions can now include the colon character (`:`). This enables use of these functions with XML which employs namespaces. (Bug #18170)
- The bundled yaSSL library was upgraded to version 1.3.5. This improves handling of certain problems with SSL-related command options. (Bug #17737)
- For the [mysql](#) client, typing Control-C causes [mysql](#) to attempt to kill the current statement. If this cannot be done, or Control-C is typed again before the statement is killed, [mysql](#) exits. Previously, Control-C caused [mysql](#) to exit in all cases. (Bug #1989)
- Creating a table in an InnoDB database with a column name that matched the name of an internal InnoDB column (including [DB_ROW_ID](#), [DB_TRX_ID](#), [DB_ROLL_PTR](#) and [DB_MIX_ID](#)) would cause a crash. MySQL now returns error 1005 (cannot create table) with [errno](#) set to -1. (Bug #18934)
- On Windows, some names such as [nul](#), [prn](#), and [aux](#) could not be used as filenames because they are reserved as device names. These are now allowable names in MySQL. They are encoded by appending [@@@](#) to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms. (Bug #17870)
- Added the [REFERENTIAL_CONSTRAINTS](#) table to [INFORMATION_SCHEMA](#). It provides information about foreign keys.
- Added [--debug](#) option to Instance Manager.
- Added the [have_dynamic_loading](#) system variable that indicates whether the server supports dynamic loading of plugins.
- Added the [sql_big_selects](#) system variable to the output of [SHOW VARIABLES](#).
- You must now have the [DROP](#) privilege to drop table partitions. (Bug #17139)
- [NDB Cluster](#): It is now possible to perform a partial start of a cluster. That is, it is now possible to bring up the cluster without running [nbd](#) [--initial](#) on all configured data nodes first. (Bug #18606)
- [NDB Cluster](#): It is now possible to restore a Cluster backup between big-endian and little-endian machines. (Bug #19255)
- [NDB Cluster](#): It is now possible to install MySQL with Cluster support to a non-default location and change the search path for font description files using either the [--basedir](#) or [--character-sets-dir](#) options. (Previously in MySQL 5.1, [nbd](#) searched only the default path for character sets.)
- In result set metadata, the [MYSQL_FIELD.length](#) value for [BIT](#) columns now is reported in number of bits. For example, the value for a [BIT\(9\)](#) column is 9. (Formerly, the value was related to number of bytes.) (Bug #13601)
- Added the [KEY_BLOCK_SIZE](#) table option and index option. This can be used in [CREATE TABLE](#), [ALTER TABLE](#), and [CREATE INDEX](#) statements to provide a hint to the storage engine about the size to use for index key blocks. The engine is allowed to change the value if necessary.

Bugs fixed:

- Security fix: A malicious client, using specially crafted invalid login or [COM_TABLE_DUMP](#) packets was able to read uninitialized memory, which potentially, though unlikely in MySQL, could have led to an information disclosure. (CVE-2006-1516, CVE-2006-1517) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.
- Security fix: A malicious client, using specially crafted invalid [COM_TABLE_DUMP](#) packets was able to trigger an exploitable buffer overflow on the server. (CVE-2006-1518) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.
- [NDB Cluster](#) (Replication): Using the [--binlog-do-db](#) option caused problems with [CREATE TABLE](#) on the cluster acting as the replication master. (Bug #19492)
- [NDB Cluster](#): Some queries having a [WHERE](#) clause of the form [c1=val1 OR c2 LIKE 'val2'](#) were not evaluated correctly. (Bug # 17421)

- **NDB Cluster:** Repeated use of the `SHOW` and `ALL STATUS` commands in the `ndb_mgm` client could cause the `mgmd` process to crash. (Bug #18591)
- **NDB Cluster:** An issue with `ndb_mgmd` prevented more than 27 `mysqld` processes from connecting to a single cluster at one time. (Bug #17150)
- Running `myisampack` followed by `myisamchk` with the `--unpack` option would corrupt the `auto_increment` key. (Bug #12633)
- A view definition that referred to an alias in the `HAVING` clause could be saved in the `.frm` file with the alias replaced by the expression that it referred to, causing failure of subsequent `SELECT * FROM view_name` statements. (Bug #19573)
- A compatibility issue with NPTL (Native POSIX Thread Library) on Linux could result in a deadlock with `FLUSH TABLES WITH READ LOCK` under some conditions. (Bug #20048)
- **MyISAM** table deadlock was possible if one thread issued a `LOCK TABLES` request for write locks and then an administrative statement such as `OPTIMIZE TABLE`, if between the two statements another client meanwhile issued a multiple-table `SELECT` for some of the locked tables. (Bug #16986)
- The patch for Bug #17164 introduced the problem that some outer joins were incorrectly converted to inner joins. (Bug #19816)
- A `NUL` byte within a comment in a statement string caused the rest of the string not to be written to the query log, allowing logging to be bypassed. (CVE-2006-0903) (Bug #17667)
- **NDB Cluster:** `mysqld` could crash when attempting an update if the cluster had failed previously. (Bug #18798)
- `mysql-test-run.pl` started **NDB** even for test cases that didn't need it. (Bug #19083)
- Selecting from a view that used `GROUP BY` on a non-constant temporal interval (such as `DATE(col) + INTERVAL TIME_TO_SEC(col) SECOND`) could cause a server crash. (Bug #19490)
- An outer join of two views that was written using `{ OJ ... }` syntax could cause a server crash. (Bug #19396)
- `mysql` displayed `NULL` for strings that are empty or contain only spaces. (Bug #19564)
- A range access optimizer heuristic was invalid, causing some queries to be much slower in MySQL 5.0 than in 4.0. (Bug #17379, Bug #18940)
- `SELECT DISTINCT` queries sometimes returned only the last row. (Bug #18068)
- Eliminated some memory corruption problems that resulted in `double free or corruption` errors and a server crash. (Bug #19154)
- Use of `CONVERT_TZ()` in a stored function or trigger (or in a stored procedure called from a stored function or trigger) caused an error. (Bug #11081)
- Some queries were slower in 5.0 than in 4.1 because some 4.1 cost-evaluation code had not been merged into 5.0. (Bug #14292)
- `MySQL-shared-compat-5.1.9-0.i386.rpm` incorrectly depended on `glibc` 2.3 and could not be installed on a `glibc` 2.2 system. (Bug #16539)
- Updates to a `MEMORY` table caused the size of `BTREE` indexes for the table to increase. (Bug #18160)
- `REPAIR TABLE` did not restore the length for packed keys in tables created under MySQL 4.x. (Bug #17810)
- The parser leaked memory when its stack needed to be extended. (Bug #18930)
- When `myisamchk` needed to rebuild a table, `AUTO_INCREMENT` information was lost. (Bug #10405)
- Use of `--default-storage-engine=innodb` resulted in an error with the server reporting that `InnoDB` was an unknown table type. (Bug #16691)
- Executing a `CREATE EVENT` statement could cause 100% CPU usage. (Bug #19170)

- After calling `FLUSH STATUS`, the `max_used_connections` variable did not increment for existing connections and connections which use the thread cache. (Bug #15933)
- MySQL would not compile on Linux distributions that use the tinfo library. (Bug #18912)
- An issue with file handling in the partitioning code could cause `mysqld` to crash when started and then stopped within a very short period of time. (Bug #19313)
- **NDB Cluster** (Replication): When taking part in Cluster replication of tables containing `BLOB` columns, `mysqld` falsely reported a large memory leak in the replication buffers when there was none. (Bug #19247)
- **NDB Cluster**: Stopping multiple nodes could cause node failure handling not to be completed. (Bug #19039)
- **NDB Cluster**: A simultaneous `DROP TABLE` and table update operation utilising a table scan could trigger a node failure. (Bug #18597)
- **NDB Cluster**: `ndbd` could sometimes fail to start with the error `Node failure handling not completed` following a graceful restart. (Bug #18550)
- **NDB Cluster**: Backups could fail for large clusters with many tables, where the number of tables approached `MaxNoOfTables`. (Bug #17607)
- **NDB Cluster**: A 5.1.6 or newer server did not read local checkpoints recorded by any other 5.1 version, thus preventing a system restart following an upgrade. (Bug #19333)
- **NDB Cluster**: Trying to restore the `apply_status` table from a 5.0 cluster backup failed on a 5.1 server. (Bug #18935)
- **NDB Cluster** (Disk Data): `CREATE LOGFILE GROUP` accepted values other than `NDB` or `NDBCLUSTER` in the `ENGINE` clause. (Bug #18604)
- **NDB Cluster** (Disk Data): Omitting the required `ENGINE` clause from a `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statement caused the server to crash. An appropriate error message is now returned instead. (Bug #18603)
- **NDB Cluster**: `mysqldump` included in its output data from the internal `cluster` database. (Bug #17840)
- Event-creation statements enclosed in multi-line comments using `/*!version_number ... */` syntax were not parsed correctly. (Bug #18078)
- Within a trigger, `CONNECTION_ID()` did not return the connection ID of the thread that caused the trigger to be activated. (Bug #16461)
- `mysqltest` incorrectly interpreted some `ER_xxx` error names given in the `error` command. (Bug #18495)
- For single-`SELECT` union constructs of the form `(SELECT ... ORDER BY order_list1 [LIMIT n]) ORDER BY order_list2`, the `ORDER BY` lists were concatenated and the `LIMIT` clause was ignored. (Bug #18767)
- Logging to the `mysql.general_log` and `mysql.slow_log` tables did not work for Windows builds because the `CSV` storage engine was unavailable. The `CSV` engine now is enabled in Windows builds. (Bug #17368)
- `LOAD DATA FROM MASTER` would fail when trying to load the `INFORMATION_SCHEMA` database from the master, because the `INFORMATION_SCHEMA` system database would already exist on the slave. (Bug #18607)
- The binary log would create an incorrect `DROP` query when creating temporary tables during replication. (Bug #17263)
- `CREATE VIEW` statements would not be replicated to the slave if the `--replicate-wild-ignore-table` rule was enabled. (Bug #18715)
- In `mysqltest`, `--sleep=0` had no effect. Now it correctly causes `sleep` commands in test case files to sleep for 0 seconds. (Bug #18312)
- Attempting to set the default value of an `ENUM` or `SET` column to `NULL` caused a server crash. (Bug #19145)
- Index corruption could occur in cases when `key_cache_block_size` was not a multiple of `myisam_block_size` (for example, with `key_cache_block_size=1536` and `myisam_block_size=1024`). (Bug #19079)

- The `sql_big_selects` system variable was not displayed by `SHOW VARIABLES`. (Bug #17849)
- The `sql_notes` and `sql_warnings` system variables were not always displayed correctly by `SHOW VARIABLES` (for example, they were displayed as `ON` after being set to `OFF`). (Bug #16195)
- `LAST_INSERT_ID()` in a stored function or trigger returned zero. . (Bug #15728)
- Use of `CONVERT_TZ()` in a view definition could result in spurious syntax or access errors. (Bug #15153)
- The `system_time_zone` and `version_*` system variables could not be accessed via `SELECT @@var_name` syntax. (Bug #12792, Bug #15684)
- Conversion of a number to a `CHAR UNICODE` string returned an invalid result. (Bug #18691)
- Some fast `ALTER TABLE` operations (requiring no temporary table) did not work for all tables. (Bug #19011)
- `DELETE` and `UPDATE` statements that used large `NOT IN (value_list)` clauses could use large amounts of memory. (Bug #15872)
- Prevent recursive views caused by using `RENAME TABLE` on a view after creating it. (Bug #14308)
- A `LOCK TABLES` statement that failed could cause `MyISAM` not to update table statistics properly, causing a subsequent `CHECK TABLE` to report table corruption. (Bug #18544)
- A failed `ALTER TABLE` operation could fail to clean up a temporary `.frm` file. (Bug #18129)
- For a reference to a non-existent stored function in a stored routine that had a `CONTINUE` handler, the server continued as though a useful result had been returned, possibly resulting in a server crash. (Bug #18787)
- `InnoDB` did not use a consistent read for `CREATE ... SELECT` when `innodb_locks_unsafe_for_binlog` was set. (Bug #18350)
- `InnoDB` could read a delete mark from its system tables incorrectly. (Bug #19217)
- `myisamchk` and `myisam_ftdump` should allow either table names or `.MYI` filenames as arguments, but allowed only table names. (Bug #19220)
- `DROP DATABASE` did not drop stored routines associated with the database if the database name was longer than 21 characters. (Bug #18344)
- Avoid trying to include `<asm/atomic.h>` when it doesn't work in C++ code. (Bug #13621)
- Executing `SELECT` on a large table that had been compressed within `mysampack` could cause a crash. (Bug #17917)
- `NDB Cluster` (NDBAPI): Passing a nonexistent index name to `NdbIndexScanOperation::setBound()` caused a segmentation fault. (Bug #19088)
- `ALTER TABLE ... REBUILD PARTITION` returned an inaccurate error message. (Bug #16819)
- `InnoDB`: A `DELETE` followed by an `INSERT` and then by an `UPDATE` on a partitioned `InnoDB` table caused subsequent queries to return incorrect results. (Bug #17992)
- `NDB Cluster`: A table insert or update of more than 128 bytes of data in a 4-replica Cluster could cause a node to crash. (Bug #18622)
- `NDB Cluster` (Disk Data): Concurrent table schema operations and operations on log files groups, tablespaces, data files, or undofiles could lead to Cluster node failures. (Bug #18575)
- `NDB Cluster`: An issue with replication caused a `mysqld` connected to a replicated cluster to crash when entering single user mode. (Bug #18535)
- Successive `ALTER TABLE ... DROP PARTITION` statements on the same subpartitioned table could eventually cause the server to crash. (Bug #18962)
- `NDB Cluster`: When attempting to create an index on a `BIT` or `BLOB` column, `Error 743: Unsupported character set in table or index` was returned instead of `Error 906: Unsupported attribute type in index`.

- **NDB Cluster**: The Cluster binlog `mysqld` accepted updates even though the binary log was not set up, which could lead to updates missing from the binary log. (Bug #18932)
- **NDB Cluster**: Concurrent `INSERT` and `ROLLBACK` statements from different connections could cause node failures. (Bug #19245)
- **NDB Cluster** (Disk Data): Running an `INSERT` and a `DELETE` on a Disk Data table in the same transaction could cause a deadlock. (Bug #19244)
- **NDB Cluster** (Disk Data): Issuing a `CREATE LOGFILE GROUP` statement during the drop of an `NDB` table would cause database corruption. (Bug #19141)
- **NDB Cluster**: `ndb_restore` failed to restore a backup made from a 5.0 cluster to a 5.1 cluster. (Bug #18210)
- **NDB Cluster**: Adding an index to an unsigned integer column did not work correctly. (Bug #18133)
- **NDB Cluster**: A `SELECT` from an `NDB` table with `ORDER BY indexed_column` and a `LIMIT` clause would fail following `ALTER TABLE`. (Bug #18094)
- **NDB Cluster**: Performing multiple `ALTER TABLE` operations on the same `NDB` table from different `mysqld` processes in the same cluster led to schema versioning errors when trying to access the table again following the restart of one of the `mysqld` processes. (Bug #16445)
- **NDB Cluster**: Starting `mysqld` without `--log-bin` caused DDL statements on `NDB` tables to time out. (Bug #19214)
- **NDB Cluster**: Fragment IDs were not being logged correctly, causing `ndb_restore_log` to fail. (Bug #18594)
- Casting a string to `DECIMAL` worked, but casting a trimmed string (using `LTRIM()` or `RTRIM()`) resulted in loss of decimal digits. (Bug #17043)
- **NDB Cluster** (Replication): Delete and update of rows in a table without a primary key failed on the slave. (Bug #17400)
- **NDB Cluster**: On slow networks or CPUs, the management client `SHOW` command could sometimes erroneously show all data nodes as being master nodes belonging to nodegroup 0. (Bug #15530)
- The XPath `string-length()` function was not implemented for use with `ExtractValue()`. (Bug #16319)
- Fix the way that Instance Manager finds the version number of instances, so that it works properly when the executable name isn't the same as what the Instance Manager launched (such as when wrapping a `libtool`-wrapped executable from the source tree). (Bug #19059)
- The server attempted to flush uninitialized log tables during `SIGHUP` processing, causing a crash. (Bug #18848)
- If the second or third argument to `BETWEEN` was a constant expression such as `'2005-09-01 - INTERVAL 6 MONTH'` and the other two arguments were columns, `BETWEEN` was evaluated incorrectly. (Bug #18618)
- If the first argument to `BETWEEN` was a `DATE` or `TIME` column of a view and the other arguments were constants, `BETWEEN` did not perform conversion of the constants to the appropriate temporary type, resulting in incorrect evaluation. (Bug #16069)
- `ExtractValue` function did not return character data within `<![CDATA[]]>` as expected. (Bug #18285)
- Server and clients ignored the `--sysconfdir` option that was passed to `configure`. (Bug #15069)
- It was possible to create a `RANGE`-partitioned table with a partition defined using the clause `VALUES LESS THAN (NULL)`, even though such a partition could never contain any values whatsoever. (Bug #18752)
- Running an `ALTER TABLE` on a partitioned table simultaneously experiencing a high number of concurrent DML statements could crash the server. (Bug #18572)
- It was possible to use trailing spaces in the names of partitions and subpartitions. Attempting to do so now raises the error `Incorrect partition name`. (Bug #17973)
- `LIKE` searches failed on a `CHAR` column used as the partitioning column of a table partitioned by `KEY`. (Bug #17946)

- If the **WHERE** condition of a query contained an **OR**-ed **FALSE** term, the set of tables whose rows cannot serve for null-complements in outer joins was determined incorrectly. This resulted in blocking possible conversions of outer joins into joins by the optimizer for such queries. (Bug #17164)
- The **ExtractValue()** function failed with a syntax error when the XPath expression used special characters such as **Ñ** (「N-tilde」). (Bug #16233)
- Inserts failed with duplicate key errors on a table partitioned using an **AUTO_INCREMENT** column for the partitioning key. (Bug #18552, Bug #18753)
- Delimited identifiers for partitions were not being treated the same as delimited identifiers for other database objects (such as tables and columns) with regard to allowed characters. (Bug #18750)
- A query on a table partitioned or subpartitioned by **HASH** did not display all results when using a **WHERE** condition involving a column used in the hashing expression. (Bug #18423, Bug #18329)
- If the server were built without partition support, it was possible to run partitioning-related statements with no errors or warnings, even though these statements would have no effect. Now such statements are disallowed unless the server has been compiled using the **--with-partition** option. (Bug #15561)
- **NDB Cluster**: In a 2-node cluster with a node failure, restarting the node with a low value for **StartPartialTimeout** could cause the cluster to come up partitioned (「split-brain」 issue). (Bug #16447)

A similar issue could occur when the cluster was first started with a sufficiently low value for this parameter. (Bug #18612)
- **NDB Cluster**: On systems with multiple network interfaces, data nodes would get 「stuck」 in startup phase 2 if the interface connecting them to the management server was working on node startup while the interface interconnecting the data nodes experienced a temporary outage. (Bug #15695)
- **NDB Cluster**: Unused open handlers for tables in which the metadata had changed were not properly closed. This could result in stale results from Cluster tables following an **ALTER TABLE**. (Bug #13228)
- **NDB Cluster**: Uninitialized internal variables could lead to unexpected results. (Bug #11033, Bug #11034)
- The presence of multiple equalities in a condition after reading a constant table could cause the optimizer not to use an index. This resulted in certain queries being much slower than in MySQL 4.1. (Bug #16504)
- A recent change caused the **mysql** client not to display **NULL** values correctly and to display numeric columns left-justified rather than right-justified. The problems have been corrected. (Bug #18265)
- **InnoDB** failure to release an adaptive hash index latch could cause a server crash if the query cache was enabled. (Bug #15758)
- **InnoDB**: **ALTER TABLE** to add or drop a foreign key for an **InnoDB** table had no effect. (Bug #18477)
- **NDB Cluster**: Attempting to create an index using multiple columns on an explicitly partitioned table in a replicated Cluster database could cause the master **mysqld** process to crash. (Bug #18284)
- **NDB Cluster**: Queries using **ORDER BY pkN** failed against a **LIST**-partitioned Cluster table having a multi-column primary key, where **pkN** represents one of the columns making up the primary key. (Bug #18598)
- Updating a field value when also requesting a lock with **GET_LOCK()** would cause slave servers in a replication environment to terminate. (Bug #17284)

Changes in release 5.1.9 (12 April 2006)

This is a new Beta development release, fixing recently discovered bugs.

NOTE: This Beta release, as any other pre-production release, should not be installed on production-level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

この項目は前回のMySQL公式リリース以降に適用されたすべての変更とバグ修正を説明します。更に頻繁でありご使用のバージョンと機能に合わせた更新情報を希望される場合には、MySQLエンタープライズ(商用版MySQL)への登録をお考えください。詳細は、<http://www.mysql.com/products/enterprise>をご覧ください。

Functionality added or changed:

- `SHOW PLUGIN` was renamed to `SHOW PLUGINS`. `SHOW PLUGIN` now is deprecated and generates a warning. (Bug #17112)
- Binary MySQL distributions now include a `mysqld-max` server, in addition to the usual `mysqld` optimized server and the `mysqld-debug` debugging server.
- `mysqld_safe` no longer checks for a `mysqld-max` binary. Instead, `mysqld_safe` nows checks only for the standard `mysqld` server unless another server binary is specified explicitly via `--mysqld` or `--mysqld-version`. If you previously relied on the implicit invocation of `mysqld-max`, you should use an appropriate option now. (Bug #17861)
- For partitioned tables, the output of `SHOW TABLE STATUS` now shows in the `Engine` column the name of the storage engine used by all partitions for the table; in the `Create_options` column, the output now shows `partitioned` for a partitioned table. This change also affects the values shown in the corresponding columns of the `INFORMATION_SCHEMA.TABLES` table. (Bug #17631)
- **NDB Cluster:** A new `--nowait-nodes` startup option for `ndbd` makes it possible to 「skip」 specific nodes without waiting for them to start when starting the cluster. See [ndbdのコマンド オプション](#).
- The **NDBCluster** storage engine now supports `CREATE TABLE` statements of arbitrary length. (Previously, `CREATE TABLE` statements for MySQL Cluster tables could contain a maximum of 4096 characters only.) (Bug #17813)
- Large file support was re-enabled for the MySQL server binary for the AIX 5.2 platform. (Bug #13571)
- The `mysql_get_ssl_cipher()` C API function was added.

Bugs fixed:

- Security fix: Invalid arguments to `DATE_FORMAT()` caused a server crash. (CVE-2006-3469, Bug #20729) Thanks to Jean-David Maillefer for discovering and reporting this problem to the Debian project and to Christian Hammers from the Debian Team for notifying us of it.
- **NDB Cluster:** `BLOB` columns did not work correctly with user-partitioned **NDB** tables. (Bug #16796)
- `mysql_config` returned incorrect libraries on `x86_64` systems. (Bug #13158)
- `mysql_reconnect()` sent a `SET NAMES` statement to the server, even for pre-4.1 servers that do not understand the statement. (Bug #18830)
- `COUNT(*)` on a **MyISAM** table could return different results for the base table and a view on the base table. (Bug #18237)
- For `mysql.server`, if the `basedir` option was specified after `datadir` in an option file, the setting for `datadir` was ignored and assumed to be located under `basedir`. (Bug #16240)
- For full-text searches in boolean mode, and when a full-text parser plugin was used, a `MYSQL_FTPARSER_PARAM::ftparser_state` could have been corrupted by recursive calls to the plugin. (Bug #18836)
- `EXTRACT(QUARTER FROM date)` returned unexpected results. (Bug #18100)
- `TRUNCATE` did not reset the `AUTO_INCREMENT` counter for **MyISAM** tables when issued inside a stored procedure. (Bug #14945)

Note: This bug did not affect **InnoDB** tables. Also, `TRUNCATE` does not reset the `AUTO_INCREMENT` counter for **NDBCluster** tables regardless of when it is called (see Bug #18864).

- The server was always built as though `--with-extra-charsets=complex` had been specified. (Bug #12076)
- Partition pruning did not work properly for some kinds of partitioning and subpartitioning, with certain `WHERE` clauses. (Partitions and subpartitions that should have been marked as used were not so marked.) The error could

manifest as incorrect content in [EXPLAIN PARTITIONS](#) output as well as missing rows in the results of affected queries. (Bug #18558)

- [NDB Cluster](#): An uninitialized internal variable could lead to unexpected results. (Bug #18831)
- For tables created in a MySQL 4.1 installation upgraded to MySQL 5.0 and up, multiple-table updates could update only the first matching row. (Bug #16281)
- Complex queries with nested joins could cause a server crash. (Bug #18279)
- A query against a partitioned table using [WHERE col IS NULL](#) could produce incorrect results given the following conditions:
 - The table had partitions and subpartitions
 - The partitioning function depended on a single column [col](#) of one of the MySQL integer types
 - The partitioning function was not monotonically increasing

The same issue could cause the server to crash when run in debug mode. (Bug #18659)

- [CAST\(double AS SIGNED INT\)](#) for large [double](#) values outside the signed integer range truncates the result to be within range, but the result sometimes had the wrong sign, and no warning was generated. (Bug #15098)
- [MEDIUMINT](#) columns were not handled in the same way as other column types by partition pruning.

Partition pruning would sometimes use inappropriate columns in performing queries.

Both of these issues were rectified as part of the same bugfix. (Bug #18025)

- Quoted values could not be used for partition option values. (Bug #13520)
- Delimited identifiers could not be used in defining partitions. (Bug #13433)
- Building the server using [--with-example-storage-engine](#) failed to enable the [EXAMPLE](#) storage engine in the server. (Bug #18464)
- Triggers created in one version of the server could not be dropped after upgrading to a newer version. (Bug #15921)
- Queries using [WHERE ... IS NULL](#) returned incorrect results from partitioned tables. (Bug #18070)
- Partition pruning did not perform correctly with partitions on [NULL](#), and could potentially crash the server. (Bug #18053)
- If [InnoDB](#) encountered a [HA_ERR_LOCK_TABLE_FULL](#) error and rolled back a transaction, the transaction was still written to the binary log. (Bug #18283)

Changes in release 5.1.8 (Not released)

Note: This was an internal release only, and no binaries were published.

Functionality added or changed:

- In order not to break legacy applications, support for [TYPE = engine_name](#) has been restored, but now generates a warning.

Important: This option has been deprecated since MySQL 4.0. Beginning with MySQL 5.2, [TYPE = engine_name](#) will no longer be available and will produce a syntax error. You should not use [TYPE](#) in any new applications, and you should immediately begin conversion of existing applications to use the [ENGINE = engine_name](#) syntax instead. (Bug #17501)

- The deprecated constructs in the following table now generate warnings. You should not employ them in new applications, as they are likely to be removed in a future version of MySQL. Use the equivalents shown in the table's second column instead. For the same reason, existing applications depending on the deprecated constructs should be converted to make use of the current equivalents as soon as possible. (Bug #17501)

Deprecated / Obsolete:	Current / Preferred:
<code>@@table_type</code>	<code>@@storage_engine</code>
<code>@@log_bin</code>	<code>@@log_bin_trust_function_creators</code>
<code>TIMESTAMP</code>	See 日付時刻関数 .
<code>TYPE=</code>	<code>ENGINE=</code>
<code>BACKUP TABLE</code>	<code>mysqldump</code> , <code>mysqlhotcopy</code> , or MySQL Administrator
<code>RESTORE TABLE, LOAD TABLE FROM MASTER</code>	<code>mysqldump</code> , <code>mysql</code> , or MySQL Administrator
<code>SHOW TABLE TYPES</code>	<code>SHOW [STORAGE] ENGINES</code>
<code>SHOW INNODB STATUS</code>	<code>SHOW ENGINE INNODB STATUS</code>
<code>SHOW MUTEX STATUS</code>	<code>SHOW ENGINE INNODB MUTEX</code>
<code>SHOW BDB LOGS, SHOW LOGS</code>	<code>SHOW ENGINE BDB LOGS</code>

The deprecated items shown in the table are not guaranteed to be available in MySQL 5.2 or later.

- Incompatible change: For purposes of determining placement, [RANGE](#) partitioning now treats [NULL](#) as less than any other value. (Formerly, [NULL](#) was treated as equal to zero.) See [MySQLパーティショニングの NULL 値の取り扱い](#). (Bug #15447)
- Incompatible Change: The semantics of `ALTER TABLE t ENGINE=X`; for partitioned tables is changed, and now means that the storage engine used for table `t` is changed to `X`.

The previous statement formerly (prior to MySQL 5.1.8) meant that all partitioning was removed from the table. In order to remove the partitioning of a table, the syntax `ALTER TABLE t REMOVE PARTITIONING`; is introduced. The `REMOVE PARTITIONING` option can be used in combination with existing `ALTER TABLE` options such as those employed for adding or dropping columns or indexes. (Bug #17754)

- Added the `--sysdate-is-now` option to `mysqld` to enable `SYSDATE()` to be treated as an alias for `NOW()`. See [日付時刻関数](#). (Bug #15101)
- The [NDBCluster](#) storage engine now supports `INSERT IGNORE` and `REPLACE` statements. Previously, these statements failed with an error. (Bug #17431)
- Events no longer support times past the end of the Unix epoch. (Formerly, such dates were interpreted as being at the beginning of the Unix epoch.) (Bug #16396)

- Event names are now case-insensitive. That is (for example), you cannot have events with the names [Myevent](#) and [MyEvent](#) belonging to the same database and definer. (Bug #16415)
- Builds for Windows, Linux, and Unix (except AIX) platforms now have SSL support enabled, in the server as well as in the client libraries. Because part of the SSL code is written in C++, this does introduce dependencies on the system's C++ runtime libraries in several cases, depending on compiler specifics. (Bug #18195)
- Temporary tables may no longer be partitioned. (Bug #17497)
- Added the `--events` option to [mysqldump](#) to enable events to be included in the dump output. (Bug #16853)
- [NDB Cluster](#) (Disk Data): You can now have only one log file group at any one time. See [CREATE LOGFILE GROUP 構文](#). (Bug #16386)
- The syntax for [CREATE PROCEDURE](#) and [CREATE FUNCTION](#) statements now includes a [DEFINER](#) clause. The [DEFINER](#) value specifies the security context to be used when checking access privileges at routine invocation time if the routine has the [SQL SECURITY DEFINER](#) characteristic. See [CREATE PROCEDUREおよびCREATE FUNCTION 構文](#), for more information.

When [mysqldump](#) is invoked with the `--routines` option, it now dumps the [DEFINER](#) value for stored routines.

- The output from [SHOW CREATE TABLE](#) is more consistent about using uppercase for keywords. Data types still are in lowercase. (Bug #10460)
- The [ExtractValue\(\)](#) function with [contains\(\)](#) now uses the SQL collation in making comparisons. Previously, comparisons were always binary (that is, case-sensitive). (Bug #16316)
- The [cluster_replication](#) database has been renamed to [cluster](#). This will effect replication between MySQL Clusters where one cluster is running MySQL 5.1.8 or later, and the other is running MySQL 5.1.7 or earlier. See [MySQL Cluster レプリケーション](#), and especially [レプリケーション スキーマおよびテーブル](#).
- The stability of [CREATE](#) and [DROP](#) operations on [NDB](#) tables containing [BLOB](#) columns has been improved. (Bug #17761)
- More specific error messages are now given when attempting to create an excessive number of partitions or subpartitions. (Previously, no distinction was made between an excessive number of partitions and an excessive number of subpartitions.) (Bug #17393)
- The [mysqltest](#) utility now converts all [CR/LF](#) combinations to [LF](#) to allow test cases intended for Windows to work properly on UNIX-like systems. (Bug #13809)
- The [mysql_ping](#) function will now retry if the [reconnect](#) flag is set and error [CR_SERVER_LOST](#) is encountered during the first attempt to ping the server. (Bug #14057)
- [mysqldump](#) now surrounds the [DEFINER](#), [SQL SECURITY DEFINER](#) and [WITH CHECK OPTION](#) clauses of a [CREATE VIEW](#) statement with "not in version" comments to prevent errors in earlier versions of MySQL. (Bug #14871)
- For an event having no [STARTS](#) time specified when it was created, the [mysql.event](#) table's [start](#) column now displays the creation time rather than [NULL](#). (Bug #16537)

In addition, both the [SHOW EVENTS](#) statement's [Starts](#) column and the [STARTS](#) column of the [INFORMATION_SCHEMA.EVENTS](#) table are now empty rather than [NULL](#) when [STARTS](#) was not used in the [CREATE EVENT](#) statement.

- [MICROSECOND](#) intervals are no longer allowed for events. (Bug #16411)
- Description of the [EVENT](#) privilege has been changed to [To create, alter, drop, and execute events](#). (Bug #16412)
- The [binlog_format](#) system variable now is dynamic and can be changed at runtime, as described in [レプリケーション フォーマット](#).
- The [binlog_format](#) system variable now can be set to a third format, [MIXED](#), as described in [レプリケーション フォーマット](#).

- A slave server may switch the format automatically now. This happens when the server is running in either [STATEMENT](#) or [MIXED](#) format and encounters a row in the binary log that is written in [ROW](#) logging format. In that case, the slave switches to row-based replication temporarily for that event, and switches back to the previous format afterwards.
- Partition pruning was made more stable, particularly in cases involving queries using tests for [NULL](#) values in the [WHERE](#) clause against subpartitioned tables which were partitioned by [LIST\(some_function\(col1, ... ,colN\) \)](#). (Bug #17891)
- Names of subpartitions must now be unique for an entire table, and not merely within the same partition. (Bug #15408)
- The output of [SHOW CREATE EVENT](#) no longer qualifies the event name with the name of the schem to which the event belongs. (Bug #17714)
- The client API will now attempt to reconnect on TCP/IP if the [reconnect](#) flag is set, as is the case with sockets. (Bug #2845)
- The XPath [last\(\)](#) function is now implemented for use with [ExtractValue\(\)](#). (Bug #16318)

Bugs fixed:

- Stored routine names longer than 64 characters were silently truncated. Now the limit is properly enforced and an error occurs. (Bug #17015)
- During conversion from one character set to [ucs2](#), multi-byte characters with no [ucs2](#) equivalent were converted to multiple characters, rather than to [0x003F QUESTION MARK](#). (Bug #15375)
- Slave servers would retry the execution of a SQL statement an infinite number of times, ignoring the value [SLAVE_TRANSACTION_RETRIES](#) when using the NDB engine. (Bug #16228)
- Replication of data stored in a partitioned table would cause slave servers to issue a assertion and terminate. (Bug #18436)
- The [mysql_close\(\)](#) C API function leaked handles for shared-memory connections on Windows. (Bug #15846)
- Checks for permissions on database operations could be performed in a case-insensitive manner (a user with permissions on database [MYDATABASE](#) could by accident get permissions on database [myDataBase](#)), if the privilege data were still cached from a previous check. (Bug #17279)
- The server would crash when [SHOW STATUS](#) was called on a server linked with [yaSSL](#). (Bug #18310)
- [SELECT ... WHERE column LIKE 'A%'](#), when [column](#) had a key and used the [latin2_czech_cs](#) collation, caused the wrong number of rows to be returned. (Bug #17374)
- Using [ALTER TABLE](#) to increase the length of a [BINARY\(M\)](#) column caused column values to be padded with spaces rather than [0x00](#) bytes. (Bug #16857)
- Execution of a stored function or trigger which inserted data into a table while running concurrent selects on the same table could result in storing incorrect data in the query cache. (Bug #14767)
- [NDB Cluster](#): Attempting to restart a node with dropped events still pending would fail. (Bug #18491)
- [NDB Cluster](#): In asynchronous replication scenarios, binary log events could be lost on the remote [mysqld](#). (Bug #18472)
- [NDB Cluster](#): Two [mysqld](#) processes starting at the same time could cause a race condition. (Bug #18472)
- [NDB Cluster](#): Two [mysqld](#) processes did not synchronise [DROP TABLE](#) binary log events correctly. (Bug #18395)
- [NDB Cluster](#): When multiple node restarts were attempted without allowing each restart to complete, the error message returned was [Array index out of bounds](#) rather than [Too many crashed replicas](#). (Bug #18349)
- A [SELECT ... ORDER BY ...](#) from a view defined using a function could crash the server. An example of such a view might be [CREATE VIEW v1 AS SELECT SQRT\(c1\) FROM t1](#). (Bug #18386)

- [REPAIR TABLE](#), [OPTIMIZE TABLE](#), and [ALTER TABLE](#) operations on transactional tables (or on tables of any type on Windows) could corrupt triggers associated with those tables. (Bug #18153)
- [MyISAM](#): Performing a bulk insert on a table referenced by a trigger would crash the table. (Bug #17764)
- Using [ORDER BY intvar](#) within a stored procedure (where [intvar](#) is an integer variable or expression) would crash the server. (Bug #16474)

Note: The use of an integer *i* in an [ORDER BY i](#) clause for sorting the result by the *i*th column is deprecated (and non-standard). It should not be used in new applications. See [SELECT 構文](#).
- A [SELECT](#) using a function against a nested view would crash the server. (Bug #15683)
- [ALTER TABLE ... ADD COLUMN ... AFTER ...](#) failed when used on partitioned tables. (Bug #16806)
- [NDB Cluster](#): A timeout in the handling of an [ABORT](#) condition with more that 32 operations could yield a node failure. (Bug #18414)
- [NDB Cluster](#): A node restart immediately following a [CREATE TABLE](#) would fail. Important: This fix supports 2-node Clusters only. (Bug #18385)
- [NDB Cluster](#): In event of a node failure during a rollback, a 「false」 lock could be established on the backup for that node, which lock could not be removed without restarting the node. (Bug #18352)
- [NDB Cluster](#): The cluster created a crashed replica of a table having an ordered index — or when logging was not enabled, of a table having a table or unique index — leading to a crash of the cluster following 8 successive restarts. (Bug #18298)
- [NDB Cluster](#): When replacing a failed master node, the replacement node could cause the cluster to crash from a buffer overflow if it had an excessively large amount of data to write to the cluster log. (Bug #18118)
- [NDB Cluster](#): Restarting nodes were allowed to start and join the cluster too early. (Bug #16772)
- [NDB Cluster](#): Issuing a [DROP LOGFILE GROUP](#) statement would cause [ndbd](#) processes to crash if MySQL had been compiled with [gcc4](#). (Bug #18295)
- Using triggers with partitioned [InnoDB](#) tables led to incorrect results. (Bug #17744)
- Calling [CREATE TABLE](#) or [ALTER TABLE](#) twice on a partitioned table in a stored procedure or a prepared statement resulted in errors and sometimes server crashes. (Bug #17290)
- A problem with [NULLs](#) and interval mapping sometimes caused incorrect results or crashes when trying to use less-than searches on partitioned tables. (Bug #17173)
- [CREATE TABLE ... PARTITION ... AS SELECT ...](#) would cause the server to crash. (Bug #15336)
- Creating a partition which depends on an expression containing a column using the UTF8 character set would cause the server to crash. (Bug #14367)
- Invoking more than once a prepared statement that creates a partitioned table would crash the server. (Bug #14350)
- [NDB Cluster](#): A [SELECT ... ORDER BY](#) query on an explicitly partitioned Cluster table with no explicit indexes would crash the server. (Bug #17899)
- The [ExtractValue\(\)](#) function did not return an error when passed an invalid XPath string. (Bug #18172)
- Stored procedures that call UDFs and pass local string variables caused server crashes. (Bug #17261)
- Connecting to a server with a UCS2 default character set with a client using a non-UCS2 character set crashed the server. (Bug #18004)
- Loading of UDFs in a statically linked MySQL caused a server crash. UDF loading is now blocked if the MySQL server is statically linked. (Bug #11835)
- A security enhancement in Visual Studio 8 could cause a MySQL debug server compiled with it to hang when running [SELECT](#) queries against partitioned tables. (Bug #17722)

- **NDB Cluster:** `auto_increment` values were not propagated correctly in statement-based replication. (Bug #18208)
- Repeated invocations of a stored procedure containing a `SHOW CREATE EVENT` statement would result in the error `Packets out of order`. (Bug #17403)
- Repeated invocations of a stored procedure containing a `CREATE EVENT` or `ALTER EVENT` statement would crash the server. (Bug #16408)
- Renaming and adding a new column to a partitioned table in the same `ALTER TABLE` statement caused the server to crash. (Bug #17772)
- `ALTER TABLE ... REBUILD PARTITION` with no partition name specified would crash the server. (Bug #17940)
- Trying to add a partition to a table having subpartitions could crash the server. (Bug #17140)
- Attempting to use a conflicting `VALUES` clause in `ALTER TABLE ... ADD PARTITION` caused the server to crash. An example of such a conflicting clause would be that uses `VALUES LESS THAN (constant)` (which indicates a range) with a table that is partitioned by `LIST`. (Bug #17127)
- `ALTER TABLE ... COALESCE PARTITION` failed with an `Out of Memory` error. (Bug #16810)
- Names of subpartitions were not displayed in the output of `SHOW CREATE TABLE`. (Bug #16370)
- Setting up subpartitions on at least one but not all the partitions of a partitioned table caused the server to crash. (Bug #15407)
- Using the `position()` function in the XPath argument to `ExtractValue()` crashed the server. (Bug #18171)
- A query with a `WHERE date_column > date_value` condition failed on a table partitioned by `RANGE`. (Bug #17894)
- Cursors in stored routines could cause a server crash. (Bug #16887)
- Replication slaves could not replicate triggers from older servers that included no `DEFINER` clause in the trigger definition. Now the trigger executes with the privileges of the invoker (which on the slave is the slave SQL thread). (Bug #16266)
- Character set conversion of string constants for `UNION` of constant and table column was not done when it was safe to do so. (Bug #15949)
- `NULL` values were written to the `mysql.slow_log` table incorrectly. (Bug #17600)
- A query with a `WHERE date_column > date_value` condition failed on a table partitioned by `RANGE`. (Bug #17894)
- A failed `ALTER TABLE ... ADD PRIMARY KEY` on a partitioned table would result in bad table metadata and could possibly crash the server. (Bug #17097)
- No error was reported when subpartitions were defined for a non-subpartitioned table. (Bug #15961)
- Searches on indexed columns of partitioned tables failed to find all matching rows following updates of the indexed columns. (Bug #14526)
- The `DEFINER` value for stored routines was not replicated. (Bug #15963)
- Use of `TRUNCATE TABLE` for a `TEMPORARY` table on a master server was propagated to slaves properly, but slaves did not decrement the `Slave_open_temp_tables` counter properly. (Bug #17137)
- `SELECT COUNT(*)` for a `MyISAM` table could return different results depending on whether an index was used. (Bug #14980)
- Updating a view that filters certain rows to set a filtered out row to be included in the table caused infinite loop. For example, if the view has a `WHERE` clause of `salary > 100` then issuing an `UPDATE` statement of `SET salary = 200 WHERE id = 10`, caused an infinite loop. (Bug #17726)
- Creating a table with the same name as the mapped name of another table caused a server crash. For example, if MySQL maps the table name `txu#P#p1` to `txu@0023P@0023p1` on disk, creating another table named `txu@0023P@0023p1` crashed the server. (Bug #17142)

- **NDB Cluster:** Adding an index together with replication could cause `mysqld` to crash. (Bug #18106)
- **NDB Cluster:** Insufficient `StringBuffer` memory when attempting to create a trigger caused the server to crash. (Bug #18101)
- **NDB Cluster:** Variable-length columns used as primary keys were not handled correctly. (Bug #18075)
- **NDB Cluster:** Row-based replication could fail with tables using `VARCHAR` columns for primary keys and having `BLOB` columns. (Bug #18067)
- **NDB Cluster:** `CREATE UNIQUE INDEX` on a column containing non-unique data could cause one or more `ndbd` nodes to hang or crash. (Bug #18040)
- **NDB Cluster (Disk Data):** `CREATE UNIQUE INDEX` failed with `Error 4243: Index not found`. (Bug #18039)
- **NDB Cluster:** Node recovery of tables with `VARCHAR` columns using character sets was inconsistent, which could cause a number of issues, including the data nodes failing to restart and `ALTER TABLE` statements to hang. (Bug #18026)
- **NDB Cluster:** In some cases, a single `ndbd` node would fail following a system restart. (Bug #17854)
- **NDB Cluster (Replication):** The binary log on the secondary master was not being set up correctly following a table rename. (Bug #17838)
- **NDB Cluster:** With a single replica, transactions waiting in the log synchronisation queue were not being restarted, causing them to be aborted. (Bug #17536)
- **NDB Cluster (Disk Data):** It was not possible to create more than 9 tablespaces. (Bug #16913)
- **NDB Cluster:** Inserting and deleting `BLOB` column values while a backup was in process could cause the loss of an `ndbd` node. (Bug #14028)
- **NDB Cluster:** `UNDO_BUFFER_SIZE` was limited to 17 MB. (Bug #16657, Bug #17890)
- Using `ALTER TABLE ... REBUILD PARTITION` without specifying the name of the partition caused the server to crash, rather than reporting a syntax error. (Bug #17947)
- When attempting to insert a `0` into a `LIST`-partitioned table that had no value-list containing `0`, no error was reported. (Bug #15253)
- If the server was started with the `--skip-grant-tables` option, it was impossible to create a trigger or a view without explicitly specifying a `DEFINER` clause. (Bug #16777)
- The server would execute stored routines that had a non-existent definer. (Bug #13198)
- **NDB Cluster:** A simultaneous `RENAME` of several tables was logged multiple times. (Bug #17827)
- **NDB Cluster:** Trying to perform a `DELETE` from a Cluster table following a `LOCK TABLES` would cause the `ndbd` processes to hang. (Bug #17812)
- `ALTER TABLE ... REORGANIZE PARTITION` failed with `Error on rename of filename ...` on Windows. (Bug #17720)
- **NDB Cluster:** `ALTER TABLE ... ADD INDEX` failed with `ERROR 756: Index on disk column is not supported` when run against a Disk Data table having a primary key. (Bug #17888)
- **NDB Cluster:** `DELETE` operations on `NDB` tables could cause memory leaks. (Bug #16874)
- **NDB Cluster:** Some query cache statistics were not always correctly reported for Cluster tables. (Bug #16795)
- The `EXAMPLE` storage engine did not work on Windows. (Bug #17721)
- For `FEDERATED` tables, a `SELECT` statement with an `ORDER BY` clause did not return rows in the proper order. (Bug #17377)
- Setting the `myisam_repair_threads` system variable to a value larger than 1 could cause corruption of large `MyISAM` tables. (Bug #11527)

- The length of a [VARCHAR\(\)](#) column that used the [utf8](#) character set would increase each time the table was re-created in a stored procedure or prepared statement, eventually causing the [CREATE TABLE](#) statement to fail. (Bug #13134)
- The MySQL server could crash with out of memory errors when performing aggregate functions on a [DECIMAL](#) column. (Bug #17602)
- [INSERT](#) statements executed by scheduled events were not written to the general log. (Bug #16413)
- A memory leak caused warnings on slaves for certain statements that executed without warning on the master. (Bug #16175)
- Naming a partition using the characters Ç or ç (「c-cedilla」; Unicode [00C7](#) or [00E7](#)) made unreadable the table containing the partition. (Bug #14527)
- The [self\(\)](#) XPath function was not handled correctly by [ExtractValue\(\)](#). (Bug #16315)
- The [ExtractValue\(\)](#) function would not accept expressions which matched element names containing an underscore character. (Bug #16320)
- [NDB Cluster](#): Trying to update very large partitioned tables using the [NDB](#) storage engine sometimes caused the server to crash. (Bug #16385, Bug #17806)
- Slow queries executed by scheduled events were not being written to the slow query log. (Bug #16426)
- On Linux, creation of table partitions failed within a stored procedure. (Bug #14363)
- [mysql_fix_privilege_tables](#) didn't create the [mysql.plugin](#) table. (Bug #17568)
- Improper checking of binary log statements could result in a server crash. (Bug #17457)
- The [ExtractValue\(\)](#) function allowed the use of the [!](#) character in identifiers by ignoring the illegal character. This is now correctly reported as a syntax error. (Bug #16313)
- [NDB Cluster](#): Trying to insert a value into a nonexistent [LIST](#) partition of an [NDB](#) table would cause the server to crash. (Bug #17763)
- [NDB Cluster](#): [ALTER TABLE](#) on a partitioned [NDB](#) table could cause the server to crash. (Bug #17499)
- [NDB Cluster](#): A repeated [SELECT](#) on a partitioned table that used the [NDB](#) storage engine could cause the server to crash. (Bug #17390)
- [NDB Cluster](#): Using [ALTER TABLE ... ADD PARTITION](#) on a table partitioned by [LIST](#) would cause the client to hang. (Bug #17701)
- Triggers created without [BEGIN](#) and [END](#) clauses resulted in 「You have an error in your SQL syntax」 errors when dumping and replaying a binary log. (Bug #16878)
- The [RENAME TABLE](#) statement did not move triggers to the new table. (Bug #13525)
- Clients compiled from source with the [--without-readline](#) did not save command history from session to session. (Bug #16557)
- Stored routines that contained only a single statement were not written properly to the dumpfile when using [mysqldump](#). (Bug #14857)
- Issuing [GRANT EXECUTE](#) on a procedure would display any warnings related to the creation of the procedure. (Bug #7787)
- Attempting to add a new partition to a table partitioned by a unique key would cause an [Out of memory](#) error. (Bug #17169)
- In a highly concurrent environment, a server crash or deadlock could result from execution of a statement that used stored functions or activated triggers coincident with alteration of the tables used by these functions or triggers. (Bug #16593)

Changes in release 5.1.7 (27 February 2006)

Functionality added or changed:

- Incompatible change: `TYPE = engine_name` is no longer accepted as a synonym for the `ENGINE = engine_name` table option. (`TYPE` has been deprecated since MySQL 4.0.)
- Several changes were made to make upgrades easier:
 - Added the `mysql_upgrade` program that checks all tables for incompatibilities with the current version of MySQL Server and repairs them if necessary. This program should be run for each MySQL upgrade (rather than `mysql_fix_privilege_tables`). See [mysql_upgrade — MySQL アップグレードのテーブル チェック](#).
 - Added the `FOR UPGRADE` option for the `CHECK TABLE` statement. This option checks whether tables are incompatible with the current version of MySQL Server.
 - Added the `--check-upgrade` to `mysqlcheck` that invokes `CHECK TABLE` with the `FOR UPGRADE` option. Added the `--fix-db-names` and `--fix-table-names` options to `mysqlcheck`.
- Incompatible change: The `mysql_stmt_attr_get()` C API function now returns a boolean rather than an unsigned int for `STMT_ATTR_UPDATE_MAX_LENGTH`. (Bug #16144)
- Added the `--wait-timeout` option to `mysqlmanager` to allow configuration of the timeout for dropping an inactive connection, and increased the default timeout from 30 seconds to 28,800 seconds (8 hours). (Bug #12674, Bug#15980)
- All subpartitions within a given partitioned table are now guaranteed to have unique names. (Bug #15408)
- Added the `RENAME DATABASE` statement.
- The SQL mode in effect at the time an event is created or altered is recorded and used during event execution. (Bug #16407)
- Added the `PROCESSLIST` table to `INFORMATION_SCHEMA`.
- Attempting to read pre-5.1.6 partitioned tables with a MySQL 5.1.7 (or later) server now generates a suitable warning message. (Bug #16695)

For additional information about this issue, see [Changes in release 5.1.6 \(01 February 2006\)](#).

- **NDB Cluster:** Attempting to `SELECT ... FROM INFORMATION_SCHEMA.FILES` now raises a warning in the event that the cluster has crashed. (Bug #17087)
- **NDB Cluster:** It is now possible to replicate NDB tables having no explicit primary key. See [MySQL Cluster レプリケーション](#).
- Removed the `have_isam` and `have_raid` system variables.
- Status messages added to `ndb_restore` to allow users to know that data files for Disk Data are being created. (Bug #16873)
- Added the `IN NATURAL LANGUAGE MODE` and `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` modifiers for full-text searches. See [全文検索関数](#).
- Creator privileges are now checked for all events before execution. (Bug #17289)
- `CREATE EVENT`, `DROP EVENT`, and `ALTER EVENT` statements are not allowed in triggers. (Bug #16410)
- New `charset` command added to `mysql` command-line client. By typing `charset name` or `\C name` (such as `\C UTF8`), the client character set can be changed without reconnecting. (Bug #16217)
- In row-based replication, when executing a `Rows_log_event`, the associated table was locked, the rows applied and the lock released. This did not work since there are storage engines that count locks and perform an autocommit when the number of locks reach zero. Now we ensure that all table maps come before all `ROWS` events in a statement.

Bugs fixed:

- **NDB Cluster:** `ndbd` restart could sometimes fail due to incorrect memory access. (Bug #17417)
- Execution times for scheduled events were not calculated correctly: the last execution time was used as a base rather than the actual start time. (Bug #17494)
- Using an XPath expression containing `=` with `ExtractValue()` caused the server to crash. (Bug #16242)
- **NDB Cluster:** An unhandled resources issue could cause node failure with a `DELETE FROM TABLE` affecting thousands of rows. (Bug #16492)
- **NDB Cluster:** Cluster tables not having an explicit primary key could not be replicated. (Bug #14541)
- For a transaction that used `MyISAM` and `InnoDB` tables, interruption of the transaction due to a dropped connection on a master server caused slaves to lose synchrony. (Bug #16559)
- `SELECT` with `GROUP BY` on a view could cause a server crash. (Bug #16382)
- `SET TRANSACTION ISOLATION LEVEL` acted like `SET SESSION TRANSACTION ISOLATION LEVEL`. That is, it set the isolation level for longer than the next transaction. (Bug #7955)
- `SHOW CREATE TABLE` produced extraneous spaces after the `PRIMARY KEY` keywords. (Bug #13883)
- If the query optimizer transformed a `GROUP BY` clause in a subquery, it did not also transform the `HAVING` clause if there was one, producing incorrect results. (Bug #16603)
- `SUBSTRING_INDEX()` could yield inconsistent results when applied with the same arguments to consecutive rows in a query. (Bug #14676)
- `myisam_ftdump` did not work for `FULLTEXT` indexes associated with a parser plugin. (Bug #17116)
- `BIT` fields were not properly handled when using row-based replication. (Bug #13418)
- Column counts were encoded incorrectly in the binary log for row-based logging format. (Bug #17678)
- Data truncations on non-`UNIQUE` indexes could crash `InnoDB` when using multi-byte character sets. (Bug #17530)
- An `ALTER DATABASE` statement on a replication master crashed the slaves. (Bug #17521)
- Partitioning with certain `SUBPARTITION BY HASH` clauses caused an error when querying for a partitioned column using an `IS NULL` comparison. (Bug #17430, Bug #17432)
- The `mysql_fix_privilege_tables.sql` script did not properly initialize the `Event_priv` column to 'Y' for those accounts that should have the `EVENT` privilege. (Bug #16400)
- **NDB Cluster:** Inserting the output of `REPEAT('some_string', some_int)` into a `BLOB` column resulted in the error `Invalid blob attributes or invalid blob parts table`. (Bug #17505)
- **NDB Cluster:** Row-based replication was not being set up correctly if a backup was already in progress. For example, connecting a `mysqld` instance to a cluster which was being backed up would result in the message `NDB: skipping setup table test.t1` being written to the error log. (Bug #17459)
- **NDB Cluster:** `CREATE TEMPORARY TABLE` of a Cluster table would fail with an `Unsupported` error or crash the server. (Bug #17210, Bug #16552)
- **NDB Cluster:** `UNIQUE` keys in Cluster tables were limited to 225 bytes in length. (Bug #15918)
- **NDB Cluster:** `REPLACE` failed when attempting to update a primary key value in a Cluster table. (Bug #14007)
- **NDB Cluster:** Creating `NDB` tables containing `BLOB` columns but no primary key caused unpredictable behavior. (Bug #17559)
- Creating an event and using a whitespace character other than space following the `DO` keyword caused a server crash. (Bug #17453)

- Previously, a stored function invocation was written to the binary log as `DO func_name()` if the invocation changes data and occurs within a non-logged statement, or if the function invokes a stored procedure that produces an error. These invocations now are logged as `SELECT func_name()` instead for better control over error code checking (slave servers could stop due to detecting a different error than occurred on the master). (Bug #14769)
- `CHECKSUM TABLE` returned different values on MyISAM table depending on whether the `QUICK` or `EXTENDED` options were used. (Bug #8841)
- Querying the `INFORMATION_SCHEMA.PARTITIONS` table on a non-max server caused a server crash. This also happened following the creation of a table with a very large number (hundreds) of partitions. (Bug #16591, Bug #17141)
- Attempting to add a new partition to a table partitioned by a unique key would cause an `Out of memory` error. (Bug #17169)
- MySQL server dropped client connection for certain `SELECT` statements against views defined that used `MERGE` algorithm. (Bug #16260)
- A statement containing `GROUP BY` and `HAVING` clauses could return incorrect results when the `HAVING` clause contained logic that returned `FALSE` for every row. (Bug #14927)
- Using `GROUP BY` on column used in `WHERE` clause could cause empty set to be returned. (Bug #16203)
- `DROP DATABASE` did not drop events for the database. (Bug #16406)
- Race conditions between event creation, dropping, and execution could result in a server crash or hang. (Bug #17373)
- `SET sql_mode = N`, where `N > 31`, did not work properly. (Bug #13897)
- Repeated invocation of `my_init()` and `my_end()` caused corruption of character set data and connection failure. (Bug #6536)
- When used with the `ExtractValue()` function, an XPath expression having no leading `「 / 」` character would crash the server. (Bug #16234)
- A `SELECT` from the last partition of a subpartitioned table having a `UNIQUE KEY` could crash the MySQL Server. (Bug #16907)
- A `SELECT` on a subpartitioned table having a multiple-column `PRIMARY` or `UNIQUE KEY`, and whose partitioning function used only the first column of the key, could cause `mysqld` to crash. (Bug #16901)
- Using `REPLACE INTO` on a partitioned table having a primary key would crash the server in the event of a duplicate key error. (Bug #16782)
- `DROP TABLE` would sometimes fail on a table having subpartitions that used the default storage engine. (Bug #16775)
- **NDB Cluster:** Sharing of table names containing special characters between multiple SQL nodes was not handled correctly when binary logging was enabled (a timeout error resulted). (Bug #17415)
- **NDB Cluster:** Table definitions were not shared between multiple SQL nodes in a cluster without binary logging being enabled. (Bug #17414)
- **NDB Cluster:** Cluster log file paths were truncated to 128 characters. They may now be as long as `MAX_PATH` (the maximum path length permitted by the operating system). (Bug #17411)
- `SHOW CREATE EVENT` displayed no output. (Bug #16423)
- Statements that contained Unicode characters were not logged to the log tables correctly. (Bug #16905)
- On Windows platforms, some attempts to create partitioned tables from the command line would cause the `mysql` client to hang. (Bug #17082)
- **NDB Cluster:** `SHOW CREATE TABLE` would fail when run against a table created in a different session. (Bug #17340)

- **NDB Cluster**: Following multiple forced shutdowns and restarts of data nodes, **DROP DATABASE** could fail. (Bug #17325)
- **NDB Cluster**: An **UPDATE** with an inner join failed to match any records if both tables in the join did not have a primary key. (Bug #17257)
- **NDB Cluster**: A **DELETE** with a join in the **WHERE** clause failed to retrieve any records if both tables in the join did not have a primary key. (Bug #17249)
- The **NDB Cluster** storage engine did not allow views to be updated. (Bug #17206)
- **NDB Cluster**: Row-based replication of a cluster failed to take **--binlog_ignore_db** settings into account. (Bug #17188)
- Trying to create a partitioned table with more than 32 attributes failed. (Bug #17179)
- **NDB Cluster**: When attempting to import data into an **NDB** table using **LOAD DATA INFILE**, the server would hang in the event of a duplicate key error. (Bug #17154)
- **NDB Cluster**: In some cases, **LOAD DATA INFILE** did not load all data into **NDB** tables. (Bug #17081)
- **NDB Cluster**: Performing large numbers of data manipulation statements on cluster tables using Disk Data could lead to a server crash. ()
- **NDB Cluster**: In some cases, a cluster using Disk Data tables could not be restarted following a normal shutdown. (Bug #16872)
- **NDB Cluster**: The **REDO** log would become corrupted (and thus unreadable) in some circumstances, due to a failure in the query handler. (Bug #17295)
- **NDB Cluster**: **CREATE TABLE new_tbl LIKE old_tbl**; failed when **old_tbl** used the **NDB** storage engine. (Bug #17005)
- **NDB Cluster**: No error message was generated for setting **NoOfFragmentLogFiles** too low. (Bug #13966)
- **NDB Cluster**: No error message was generated for setting **MaxNoOfAttributes** too low. (Bug #13965)
- The **SELECT** privilege was required for triggers that performed no selects. (Bug #15196)
- The **UPDATE** privilege was required for triggers that performed no updates. (Bug #15166)
- **CAST(... AS TIME)** operations returned different results when using versus not using prepared-statement protocol. (Bug #15805)
- Killing a long-running query containing a subquery could cause a server crash. (Bug #14851)
- **InnoDB** could display an incorrect error message for a cascading update. (Bug #9680)
- A **RETURN** statement within a trigger caused a server crash. **RETURN** now is disallowed within triggers. To exit immediately, use **LEAVE**. (Bug #16829)

Changes in release 5.1.6 (01 February 2006)

Functionality added or changed:

- Packaging changes: MySQL 5.1.6 introduces some changes to distribution packaging:
 - Distributions include both a **mysqld** optimized server and **mysqld-debug** debugging server. There is no separate debug distribution.
 - There is no longer a **mysqld-max** server. (Note: This changed in MySQL 5.1.9: The **mysqld-max** server also is included in binary distributions.)
 - Server binaries no longer are stripped, except for RPM distributions.
 - Binary distributions for Unix and Unix-like systems no longer include **safe_mysqld** as a link to **mysqld_safe**. **safe_mysqld** has been deprecated since MySQL 4.0 and now is removed.

- Incompatible change: This release introduces the [TRIGGER](#) privilege. Previously, the [SUPER](#) privilege was needed to create or drop triggers. Now those operations require the [TRIGGER](#) privilege. This is a security improvement because you no longer need to grant users the [SUPER](#) privilege to enable them to create triggers. However, the requirement that the account named in a trigger's [DEFINER](#) clause must have the [SUPER](#) privilege has changed to a requirement for the [TRIGGER](#) privilege. After upgrading, be sure to update your grant tables as described in [mysql_fix_privilege_tables — MySQL システム テーブルのアップグレード](#). This process assigns the [TRIGGER](#) privilege to all accounts that had the [SUPER](#) privilege. (After updating, you might also consider whether any of those accounts no longer need [SUPER](#).) If you fail to update the grant tables, triggers may fail when activated. (Bug #9142)
- Incompatible change: Before MySQL 5.1.6, the server writes general query log and slow query log entries to log files. As of MySQL 5.1.6, the server's logging capabilities for these logs are more flexible. Log entries can be written to log files (as before) or to the [general_log](#) and [slow_log](#) tables in the [mysql](#) database. If logging is enabled, either or both destinations can be selected. The [--log-output](#) option controls the destination or destinations of log output. See [一般クエリとスロー クエリのログ出力先の選択](#).

If logging is enabled, the default destination now is to log to tables, which differs from earlier versions. If you had the server configured for logging to log files formerly, use [--log-output=FILE](#) to preserve this behavior after an upgrade to MySQL 5.1.6 or higher.

- Incompatible change: Due to a change in the naming scheme for partitioning and subpartitioning files, it is not possible for the server to read partitioned tables created in previous MySQL versions. A suggested workaround is (1) to create a non-partitioned table with the same table schema using a standard [CREATE TABLE](#) statement (that is, with no partitioning clauses) and then (2) to issue a [SELECT INTO](#) to copy the data into the non-partitioned table before the upgrade; following the upgrade, you can partition the new table using [ALTER TABLE ... PARTITION BY ...](#). Alternatively, you can dump the table using [mysqldump](#) prior to upgrading and reload it afterwards with [LOAD DATA](#). In either case, you should drop the pre-5.1.6 partitioned tables before upgrading to 5.1.6 or later. (Bug #13437)

Important: If any partitioned tables that were created prior to MySQL 5.1.6 are present following an upgrade to MySQL 5.1.6 or later, it is also not possible to read from the [INFORMATION_SCHEMA.PARTITIONS](#) table, nor will you be able to drop those tables or the database or databases in which they are located. In this event, you must: (1) shut down [mysqld](#); (2) manually delete the table, partition, and (if any) subpartition files; and then (3) restart the MySQL Server. (Bug #16695)

- Incompatible change: Words with apostrophes are now matched in a [FULLTEXT](#) search against non-apostrophe words (for example, a search for [Jerry](#) will match against the term [Jerry's](#)). Users upgrading to this version must issue [REPAIR TABLE](#) statements for tables containing [FULLTEXT](#) indexes. (Bug #14194)
- MySQL 5.1.6 introduces the Event Scheduler which allows one to schedule statements for execution at predetermined times. Events can be transient (one-time-only) or recurrent at regular intervals, and may execute queries and statements permitted in stored routines, including compound statements.

Events can be altered after creation, and dropped when no longer needed.

Information about scheduled events can be obtained using the statements [SHOW EVENTS](#) and [SHOW CREATE EVENT](#), or by querying the [INFORMATION_SCHEMA.EVENTS](#) table. All of these are available beginning in MySQL 5.1.6.

Users must have the [EVENT](#) privilege (also added in 5.1.6) to create events.

For more information, see [19章Event Scheduler](#).

- Replication between MySQL Clusters is now supported. It is now also possible to replicate between a MySQL Cluster and a non-cluster database. See [MySQL Cluster レプリケーション](#).
- Special characters in database and table identifiers now are encoded when creating the corresponding directory names and filenames. This relaxes the restrictions on the characters that can appear in identifiers. See [ファイル名への識別子のマッピング](#).
- Queries against partitioned tables can now take advantage of partition pruning. In some cases, this can result in query execution that is an order of magnitude faster than the same query against a non-partitioned version of the same table.

- The `mysqldump` utility now supports an option for dumping tablespaces. Use `-Y` or `--all-tablespaces` to enable this functionality. (Bug #16753)
- Partition support is not an 「engine」, but it was included in the output of `SHOW ENGINES`. Now it is not. (Bug #14355) The `have_partition_engine` variable was renamed to `have_partitioning`. (Bug #16718)
- `ANALYZE TABLE` is now supported for partitioned tables. (Bug #13441)
- Added the `event_scheduler` system variable.
- Added the `ndb_extra_logging` system variable.
- Added the `FILES` table to `INFORMATION_SCHEMA`.
- Added the `EVENTS` table to `INFORMATION_SCHEMA`.
- Added the `PARTITIONS` table to `INFORMATION_SCHEMA`.
- The `ARCHIVE` storage engine now supports the `AUTO_INCREMENT` column attribute and the `AUTO_INCREMENT` table option. `ARCHIVE ストレージエンジン`.
- Added support for the `CREATE INDEX` and `DROP INDEX` statements to the `NDB Cluster` storage engine.
- Server plugins can register their own status variables to be displayed by the `SHOW STATUS` statement.

Bugs fixed:

- An indexing error sometimes caused values to be assigned to the wrong `RANGE` partition. (Bug #16684)
- `NDB Cluster`: `ndb_delete_all` would run out of memory on tables containing `BLOB` columns. (Bug #16693)
- Using the `TRUNCATE()` function with a negative number for the second argument on a `BIGINT` column returned incorrect results. (Bug #8461)
- When the fulltext search parser plugin returned more words than half of the length (in bytes) of the query string, the server would crash. (Bug #16722)
- Improper memory handling for stored routine variables could cause memory overruns and binary log corruption. (Bug #15588)
- A `FULLTEXT` query in a prepared statement could result in unexpected behavior. (Bug #14496)
- `STR_TO_DATE(1,NULL)` caused a server crash. (CVE-2006-3081, Bug #15828)
- An `INSERT` statement in a stored procedure corrupted the binary log. (Bug #16621)
- The `mysql_real_connect()` C API function incorrectly reset the `MYSQL_OPT_RECONNECT` option to its default value. (Bug #15719)
- Specifying a value for `--tmpdir` without a trailing slash had unpredictable results. (Bug #15904)
- Attempting to insert data into a partitioned table that used the `BLACKHOLE` storage engine caused `mysqld` to crash. (Bug #14524)
- Using `RANGE` partitioning with a `CASE` statement as the partitioning function would cause records to be placed in the wrong partition. (Bug #15393)
- `ALTER TABLE ... ADD PARTITIONS` on a table with one partition crashed the server. (Bug #15820)
- `NDB Cluster` returned incorrect `Can't find file` error for OS error 24, changed to `Too many open files`. (Bug #15020)
- Multi-byte path names for `LOAD DATA` and `SELECT ... INTO OUTFILE` caused errors. Added the `character_set_filesystem` system variable, which controls the interpretation of string literals that refer to filenames. (Bug #12448)
- Certain subqueries where the inner query is the result of an aggregate function would return different results on MySQL 5.0 than on MySQL 4.1. (Bug #15347)

- The error message for specifying values for which no partition exists returned wrong values on certain platforms. (Bug #15910)
- Certain Japanese table names were not properly saved during a `CREATE TABLE` statement. (Bug #3906)
- NDB Cluster leaked disk space when performing INSERTS/DELETES in a loop. (Bug #16771)
- NDB Cluster returned wrong error when tablespace on disk was full. (Bug #16738)
- The `DATA DIRECTORY` and `INDEX DIRECTORY` clauses of a `CREATE TABLE` statement involving partitions did not work. (Bug #14354)
- Subselect could return wrong results when records cache and grouping was involved. (Bug #15347)
- In some cases the query optimizer did not properly perform multiple joins where inner joins followed left joins, resulting in corrupted result sets. (Bug #15633)
- The absence of a table in the left part of a left or right join was not checked prior to name resolution, which resulted in a server crash. (Bug #15538)
- **NDB Cluster:** Trying to import too many dumped tables requiring resources beyond those allocated in the cluster configuration would cause the server to crash instead of reporting an insufficient resources error. (Bug #16455)
- **NDB Cluster** (Disk Data): Tablespaces created using parameters with relatively low values (< 10 MB) produced filesizes much smaller than expected. (Bug #16742)
- **NDB Cluster:** `CREATE TABLESPACE` statements were incorrectly parsed on 64-bit platforms. (`INITIAL SIZE size` worked, but `INITIAL SIZE = size` failed.) (Bug #13556)
- Trying to add more than one partition in a single `ALTER TABLE ... ADD PARTITION` statement caused the server to crash. (Bug #16534)
- Creating a partitioned table using a storage engine other than the session default storage engine caused the server to crash. (Bug #15966)
- An `ALTER TABLE ... PARTITION BY ...` statement did not have any effect. (Bug #15523)
- **NDBCluster** (Disk Data): The error message generated by a failed `ADD UNDOFILE` did not provide any reasons for the failure. (Bug #16267)
- **NDBCluster** (Disk Data): `DROP LOGFILE GROUP` corrupted the cluster file system and caused `ndbd` to fail when running more than one node on the same system. (Bug #16193)
- **NDBCluster:** A bitfield whose offset and length totaled 32 would crash the cluster. (Bug #16125)
- **NDBCluster:** Upon the completion of a scan where a key request remained outstanding on the primary replica and a starting node died, the scan did not terminate. This caused incompleted error handling of the failed node. (Bug #15908)
- **NDBCluster:** The `ndb_autodiscover` test failed sporadically due to a node not being permitted to connect to the cluster. (Bug #15619)
- Using `mysqldump` to obtain a dump of a partitioned table employing the **NDB** storage engine produced a non-functional table creation statement. (Bug #13155)
- `SHOW CREATE TABLE` did not display the `PARTITIONS` clause for tables partitioned by `HASH` or `KEY`. (Bug #14327)
- Inserting a negative value into an integer column used as the partitioning key for a table partitioned by `HASH` could cause the server to crash. (Bug #15968)
- With a table partitioned by `LIST`, inserting a value which was smaller than any value shown in the partitioning value-lists could cause the server to crash. (Bug #14365)
- `ALTER TABLE ... DROP PARTITION` would truncate all `DATE` column values in the table's remaining partitions to `NULL`. (Bug #13644)

- [ALTER TABLE ... ADD PARTITION](#) could crash the server or cause an [Out of memory](#) error in some circumstances. (Bug #13447)
- The server would allow foreign keys to be declared in the definition of a partitioned table despite the fact that partitioned tables do not support foreign keys (see [パーティショニングの制約と制限](#)). (Bug #13446)
- A [SELECT](#) from a key-partitioned table with a multi-column key could cause the server to crash. (Bug #13445)
- Issuing a [TRUNCATE](#) statement twice in succession on the same partitioned table would cause the server to crash. (Bug #13442)
- Using a [REPLACE](#) statement on a partitioned table caused the server to crash. (Bug #13440)
- Using an identifier rather than a literal integer value in the [LESS THAN](#) clause of a range-partitioned table could cause the server to crash and corruption of tables. (Bug #13439)
- Using [ENGINE=...](#) within a [PARTITION](#) clause could cause the server to crash. (Bug #13438)
- [CREATE TABLE ... LIKE](#) did not work if the table whose schema was to be copied was a partitioned table. (Bug #13435)
- [SHOW CREATE TABLE](#) did not display the [PARTITIONS](#) clause for tables partitioned by [HASH](#) or [KEY](#). (Bug #14327)
- Certain permission management statements could create a [NULL](#) hostname for a user, resulting in a server crash. (Bug #15598)
- Temporary table aliasing did not work inside stored functions. (Bug #12198)
- Parallel builds occasionally failed on Solaris. (Bug #16282)

Changes in release 5.1.5 (10 January 2006)

Functionality added or changed:

- Added the [INFORMATION_SCHEMA ENGINES](#) table.
- Added the [INFORMATION_SCHEMA PLUGINS](#) table and the [SHOW PLUGIN](#) statement.
- Added the [binlog_format](#) system variable that controls whether to use row-based or statement-based binary logging. Added the [--binlog-format](#) and [--binlog-row-event-max-size](#) server options for binary logging control. See [レプリケーション フォーマット](#).
- Plugins now can have status variables that are displayed in the output from [SHOW STATUS](#). See [Writing Plugins](#).
- Added the XML functions [ExtractValue\(\)](#) and [UpdateXML\(\)](#). [ExtractValue\(\)](#) returns the content of a fragment of XML matching a given XPath expression. [UpdateXML\(\)](#) replaces the element selected from a fragment of XML by an XPath expression supplied by the user with a second XML fragment (also user-supplied), and returns the modified XML. See [XML 関数](#).
- Added the [--base64-output](#) option to [mysqlbinlog](#) to print all binary log entries using base64 encoding. This is for debugging only. Logs produced using this option should not be applied on production systems.
- Added the [--port-open-timeout](#) option to [mysqld](#) to control how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. (Bug #15591)
- Two new Hungarian collations are included: [utf8_hungarian_ci](#) and [ucs2_hungarian_ci](#). These support the correct sort order for Hungarian vowels. However, they do not support the correct order for sorting Hungarian consonant contractions; this issue will be fixed in a future release.

Bugs fixed:

- [InnoDB](#): An [UPDATE](#) statement with no index column in the [WHERE](#) condition locked all the rows in the table. (Bug #3300)
- [INSERT DELAYED](#) caused [mysqld](#) to crash. (Bug #16095)

- The output of `mysqldump --triggers` did not contain the `DEFINER` clause in dumped trigger definitions. (Bug #15110)
- The output of `SHOW TRIGGERS` contained extraneous whitespace. (Bug #15103)
- An `INSERT ... SELECT` statement between tables in a `MERGE` set can return errors when statement involves insert into child table from merge table or vice-versa. (Bug #5390)
- A `COMMIT` statement followed by a `ALTER TABLE` statement on a BDB table caused server crash. (Bug #14212)
- `InnoDB`: Comparison of indexed `VARCHAR CHARACTER SET ucs2 COLLATE ucs2_bin` columns using `LIKE` could fail. (Bug #14583)
- Creating a trigger caused a server crash if the table or trigger database was not known because no default database had been selected. (Bug #14863)
- Issuing a `DROP USER` command could cause some users to encounter a `hostname is not allowed to connect to this MySQL server` error. (Bug #15775)
- The `--plugin_dir` option was not working. Also fix error with specifying parser name for fulltext. (Bug #16068)
- Attempting to insert into a table partitioned by `LIST` a value less than any specified in one of the table's partition definitions resulted in a server crash. In such cases, `mysqld` now returns `ERROR 1500 (HY000): Table has no partition for value v`, where `v` is the out-of-range value. (Bug #15819)

Changes in release 5.1.4 (21 December 2005)

Functionality added or changed:

- Added the `mysqlslap` program, which is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server.
- Added the `--server-id` option to `mysqlbinlog` to enable only those events created by the server having the given server ID to be extracted. (Bug #15485)
- It is now possible to build the server such that `MyISAM` tables can support up to 128 keys rather than the standard 64. This can be done by configuring the build using the option `--with-max-indexes=N`, where $N \leq 128$ is the maximum number of indexes to permit per table. (Bug #10932)
- The bundled `BDB` library was upgraded to version 4.4.16.
- Added the `cp1250_polish_ci` collation for the `cp1250` character set.
- Added the `myisam_use_mmap` system variable.
- Added the `--bdb-data-direct` and `--bdb-log-direct` server options.

Bugs fixed:

- `SHOW ENGINES` output showed the `FEDERATED` engine as `DISABLED` even for builds with `FEDERATED` support. (Bug #15559)
- Server could not be built on default Debian systems with BDB enabled. (Bug #15734)
- `BDB`: A `DELETE`, `INSERT`, or `UPDATE` of a `BDB` table could cause the server to crash where the query contained a subquery using an index read. (Bug #15536)
- A left join on a column that having a `NULL` value could cause the server to crash. (Bug #15268)
- It was not possible to reorganize a partition reusing a discarded partition name.

Now, for example, you can create a table such as this one:

```
CREATE TABLE t1 (a INT)
PARTITION BY RANGE (a) (
  PARTITION p0 VALUES LESS THAN (10),
  PARTITION p1 VALUES LESS THAN (20),
  PARTITION p2 VALUES LESS THAN MAXVALUE
```

```
);
```

and then repartition it as shown here:

```
ALTER TABLE t1 REORGANIZE PARTITION p2 INTO (
PARTITION p2 VALUES LESS THAN (30)
);
```

Previously, attempting to do so would produce the error **All partitions must have unique names in the table**. (Bug #15521)

- **NDB Cluster**: The `--ndb` option for `pererror` did not function. (Bug #15486)
- The **BLACKHOLE** storage engine did not handle transactions properly: Rolled-back transactions were written to the binary log. Now they are not. (Bug #15406)
- **NDB Cluster**: Using `ORDER BY primary_key_column` when selecting from a table having the primary key on a **VARCHAR** column caused a forced shutdown of the cluster. (Bug #14828, Bug #15240, Bug #15682, Bug #15517)
- **ANALYZE TABLE** did not properly update table statistics for a **MyISAM** table with a **FULLTEXT** index containing stopwords, so a subsequent **ANALYZE TABLE** would not recognize the table as having already been analyzed. (Bug #14902)
- The maximum value of **MAX_ROWS** was handled incorrectly on 64-bit systems. (Bug #14155)
- Multiple-table update operations were counting updates and not updated rows. As a result, if a row had several updates it was counted several times for the `「rows matched」` value but updated only once. (Bug #15028)
- **SELECT** queries that began with an opening parenthesis were not being placed in the query cache. (Bug #14652)
- Space truncation was being ignored when inserting into **BINARY** or **VARBINARY** columns. Now space truncation results in a warning, or an error in strict mode. (Bug #14299)
- Selecting from a view processed with the temptable algorithm caused a server crash if the query cache was enabled. (Bug #15119)
- Creating a view that referenced a stored function that selected from a view caused a crash upon selection from the view. (Bug #15096)
- Creating a view within a stored procedure could result in an out of memory error or a server crash. (Bug #14885)
- **SHOW CREATE DATABASE** was sometimes refused when the client had privileges for the database. (Bug #9785)
- **mysql** ignored the **MYSQL_TCP_PORT** environment variable. (Bug #5792)
- **ROW_COUNT()** returned an incorrect result after **EXECUTE** of a prepared statement. (Bug #14956)
- Invalid casts to **DATE** values now result in a message of **Incorrect datetime value**, rather than **Truncated incorrect datetime value**. (Bug #8294)
- Attempts to assign **NULL** to a **NOT NULL** column in strict mode now result in a message of **Column 'col_name' cannot be null**, rather than **Column set to default value; NULL supplied to NOT NULL column 'col_name' at row n**. (Bug #11491)
- For binary string data types, **mysqldump --hex-blob** produced an illegal output value of **0x** rather than **"**. (Bug #13318)
- Some comparisons for the **IN()** operator were inconsistent with equivalent comparisons for the **=** operator. (Bug #12612)

Changes in release 5.1.3 (29 November 2005)

Functionality added or changed:

This is the first public alpha release of the current MySQL 5.1 development branch, providing an insight to upcoming features. Although some of these are still under heavy development, this release includes the following new features and changes (in comparison to the current MySQL 5.0 production release):

- Partitioning: allows distributing portions of individual tables across a filesystem, according to rules which can be set when the table is created. In effect, different portions of a table are stored as separate tables in different locations, but from the user point of view, the partitioned table is still a single table. See [15章パーティショニング](#), for further information on this functionality. (Author: Mikael Ronström)
- Plugin API: MySQL 5.1 adds support for a very flexible plugin API that enables loading and unloading of various components at runtime, without restarting the server. Although the work on this is not finished yet, plugin full-text parsers are a first step in this direction. This allows users to implement their own input filter on the indexed text, enabling full-text search capability on arbitrary data such as PDF files or other document formats. A pre-parser full-text plugin performs the actual parsing and extraction of the text and hands it over to the built-in MySQL full-text search. (Author: Sergey Vojtovich)

The plugin API requires the `mysql.plugin` table. When upgrading from an older version of MySQL, you should run the `mysql_fix_privilege_tables` command to create this table. See [mysql_fix_privilege_tables — MySQL システム テーブルのアップグレード](#).

Incompatible change: Plugins are installed in the directory named by the `plugin_dir` system variable. This variable also controls the location from which the server loads user-defined functions (UDFs), which is a change from earlier versions of MySQL. That is, all UDF library files now must be installed in the plugin directory. When upgrading from an older version of MySQL, you must migrate your UDF files to the plugin directory.

- The Instance Manager (IM) now has some additional functionality:
 - `SHOW instance_name LOG FILES` provides a listing of all log files used by the instance. (Author: Petr Chardin)
 - `SHOW instance_name LOG {ERROR | SLOW | GENERAL} size` retrieves a part of the specified log file. (Author: Petr Chardin)
 - `SET instance_name.option_name=option_value` sets an option to the specified value and writes it to the config file. See [mysqlmanager — MySQL Instance Manager](#), for more details on these new commands. (Author: Petr Chardin)
- The performance of boolean full-text searches (using the 「+」 Operator) has been improved. See [全文検索関数](#), for more details about full-text searching. (Author: Sergey Vojtovich)
- `VARCHAR` fields used in MySQL Cluster tables are now variable-sized; that is, they now only allocate as much space as required to store the data. Previously, a `VARCHAR(n)` column allocated $n+2$ bytes (aligned to 4 bytes), regardless if the actual inserted value required that much space. (In other words, a `VARCHAR` column always required the same, fixed, amount of storage as a `CHAR` column of the same size.)
- Renamed the `table_cache` system variable to `table_open_cache`. Any scripts that refer to `table_cache` should be updated to use the new name.
- Added the `table_definition_cache` system variable. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache.
- Added the `SHOW AUTHORS` statement.
- Added the `SHOW FUNCTION CODE` and `SHOW PROCEDURE CODE` statements (available only for servers that have been built with debugging support). See [SHOW PROCEDURE CODE と SHOW FUNCTION CODE 構文](#).
- `RAND()` no longer allows non-constant initializers. (Prior to MySQL 5.0.13, the effect of non-constant initializers is undefined.) (Bug #6172)

Bugs fixed:

- Set functions could not be aggregated in outer subqueries. (Bug #12762)

Changes in release 5.1.2 (Not released)

Functionality added or changed:

- Added [MAXLOCKS](#), [MINLOCKS](#), [MAXWRITE](#), and [MINWRITE](#) as allowable values of the [--bdb-lock-detect](#) option. (Bug #14876)
- Added the [bdb_cache_parts](#) and [bdb_region_size](#) system variables, and allowed [bdb_cache_size](#) to be larger than 4GB on systems that support it. (Bug #14895)
- Added [Transactions](#), [XA](#), and [Savepoints](#) columns to [SHOW ENGINES](#) output.
- Added [--replace](#) to [mysqldump](#). This option uses [REPLACE INTO](#), rather than [INSERT INTO](#), when writing the dumpfile.

Bugs fixed:

- Foreign keys were not properly enforced in [TEMPORARY](#) tables. Foreign keys now are disallowed in [TEMPORARY](#) tables. (Bug #12084)

Changes in release 5.1.1 (Not released)

Bugs fixed:

- Performing a [CREATE TABLE](#) statement with a [PARTITION BY](#) clause in a prepared statement could crash a server running in debug mode. (Bug #12097)
- [NDB](#): Specifying the wrong nodegroup in a [CREATE TABLE](#) using partitioning would lead to the table name being locked after the [CREATE TABLE](#) statement failed (that is, the table name could not be re-used). (Bug #12114)
- Using [ORDER BY](#) in a query with a partitioned table on a 64-bit operating system could crash the server. (Bug #12116)
- When two threads competed for the same table, a deadlock could occur if one thread also had a lock on another table through [LOCK TABLES](#) and the thread was attempting to remove the table in some manner while the other thread tried to place locks on both tables. (Bug #10600)

MySQL Connector/ODBC (MyODBC) Change History

Changes in Connector/ODBC 5.0.11 (31 January 2007)

Functionality added or changed:

- Driver now builds and is partially tested under Linux with the iODBC driver manager.
- Added support for ODBC v2 statement options using attributes.

Bugs fixed:

- Updates of [MEMO](#) or [TEXT](#) columns from within Microsoft Access would fail. (Bug #25263)
- Transaction support has been added and tested. (Bug #25045)
- Connection string parsing for DSN-less connections could fail to identify some parameters. (Bug #25316)
- Fixed occasional mis-handling of the [SQL_NUMERIC_C](#) type.
- Fixed the binding of certain integer types.

Changes in Connector/ODBC 5.0.10 (14 December 2006)

Connector/ODBC 5.0.10 is the sixth BETA release.

Functionality added or changed:

- Added wide-string type info for [SQLGetTypeInfo\(\)](#).

- Added loose handling of retrieving some diagnostic data. (Bug #15782)
- Added initial support for removing braces when calling stored procedures and retrieving result sets from procedure calls. (Bug #24485)
- Added initial unicode support in data and metadata. (Bug #24837)
- Significant performance improvement when retrieving large text fields in pieces using [SQLGetData\(\)](#) with a buffer smaller than the whole data. Mainly used in Access when fetching very large text fields. (Bug #24876)

Bugs fixed:

- String query parameters are now escaped correctly. (Bug #19078)
- Editing DSN no longer crashes ODBC data source administrator. (Bug #24675)

Changes in Connector/ODBC 5.0.9 (22 November 2006)

Connector/ODBC 5.0.9 is the fifth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Added recognition of [SQL_C_SHORT](#) and [SQL_C_TINYINT](#) as C types.
- Added support for column binding as [SQL_NUMERIC_STRUCT](#).

Bugs fixed:

- Fixed wildcard handling of and listing of catalogs and tables in [SQLTables](#).
- Catch use of [SQL_ATTR_PARAMSET_SIZE](#) and report error until we fully support.
- Corrected retrieval multiple field types bit and blob/text.
- Fixed buffer length return for [SQLDriverConnect](#).
- Added limit of display size when requested via [SQLColAttribute/SQL_DESC_DISPLAY_SIZE](#).
- Fixed statistics to fail if it couldn't be completed.
- Fixed [SQLGetData](#) to clear the NULL indicator correctly during multiple calls.
- ODBC v2 behaviour in driver now supports ODBC v3 date/time types (since [DriverManager](#) maps them).

Changes in Connector/ODBC 5.0.8 (17 November 2006)

Connector/ODBC 5.0.8 is the fourth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Made distinction between [CHAR/BINARY](#) (and VAR versions).
- Wildcards now support escaped chars and underscore matching (needed to link tables with underscores in access).
- Also made [SQL_DESC_NAME](#) only fill in the name if there was a data pointer given, otherwise just the length.
- Fixed display size to be length if max length isn't available.

Bugs fixed:

- Length now used when handling bind parameter (needed in particular for [SQL_WCHAR](#)) - this enables updating char data in MS Access.

- Fixed string length to chars, not bytes, returned by `SQLGetDiagRec`.
- Fixed using wrong pointer for `SQL_MAX_DRIVER_CONNECTIONS` in `SQLGetInfo`.
- Fixed binding using `SQL_C_LONG`.
- Allow `SQLDescribeCol` to be called to retrieve the length of the column name, but not the name itself.
- Fix size return from `SQLDescribeCol`.
- Fixed handling of numeric pointers in `SQLColAttribute`.
- Fixed type returned for `MYSQL_TYPE_LONG` to `SQL_INTEGER` instead of `SQL_TINYINT`.
- Fixed `MDiagnostic` to use correct v2/v3 error codes.
- Set default return to `SQL_SUCCESS` if nothing is done for `SQLSpecialColumns`.
- Updated retrieval of descriptor fields to use the right pointer types.

Changes in Connector/ODBC 5.0.7 (08 November 2006)

Connector/ODBC 5.0.7 is the third BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Improved trace/log.
- Added support for `SQLStatistics` to `MYODBCShell`.

Bugs fixed:

- Fixed `SQLDescribeCol` returning column name length in bytes rather than chars.
- `SQLBindParameter` now handles `SQL_C_DEFAULT`.
- Corrected incorrect column index within `SQLStatistics`. Many more tables can now be linked into MS Access.

Changes in Connector/ODBC 5.0.6 (03 November 2006)

Connector/ODBC 5.0.6 is the second BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.
- Installation is provided in the form of a standard Microsoft System Installer (MSI).
- Connector/ODBC supports both `User` and `System` DSNs.

Changes in Connector/ODBC 5.0.5 (17 October 2006)

Connector/ODBC 5.0.5 is the first BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.

Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- The following ODBC API functions have been added in this release:
 - [SQLBindParameter](#)
 - [SQLBindCol](#)

Changes in Connector/ODBC 5.0.2 (Never released)

Connector/ODBC 5.0.2 was an internal implementation and testing release.

Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)

Features, limitations and notes on this release:

- Connector/ODBC 5.0 is Unicode aware.
- Connector/ODBC is currently limited to basic applications. ADO applications and Microsoft Office are not supported.
- Connector/ODBC must be used with a Driver Manager.
- The following ODBC API functions are implemented:
 - [SQLAllocHandle](#)
 - [SQLCloseCursor](#)
 - [SQLColAttribute](#)
 - [SQLColumns](#)
 - [SQLConnect](#)
 - [SQLCopyDesc](#)
 - [SQLDisconnect](#)
 - [SQLExecDirect](#)
 - [SQLExecute](#)
 - [SQLFetch](#)
 - [SQLFreeHandle](#)
 - [SQLFreeStmt](#)
 - [SQLGetConnectAttr](#)
 - [SQLGetData](#)
 - [SQLGetDescField](#)
 - [SQLGetDescRec](#)
 - [SQLGetDiagField](#)
 - [SQLGetDiagRec](#)

- [SQLGetEnvAttr](#)
- [SQLGetFunctions](#)
- [SQLGetStmtAttr](#)
- [SQLGetTypeInfo](#)
- [SQLNumResultCols](#)
- [SQLPrepare](#)
- [SQLRowcount](#)
- [SQLTables](#)

The following ODBC API function are implemented, but not yet support all the available attributes/options:

- [SQLSetConnectAttr](#)
- [SQLSetDescField](#)
- [SQLSetDescRec](#)
- [SQLSetEnvAttr](#)
- [SQLSetStmtAttr](#)

Changes in Connector/ODBC 3.51.13 (Not yet released)

Functionality added or changed:

- Added support for the [HENV](#) handlers in [SQLEndTran\(\)](#).
- Added auto-reconnect option to Connector/ODBC option parameters.
- Added auto is null option to Connector/ODBC option parameters. (Bug #10910)

Bugs fixed:

- On 64-bit systems, some types would be incorrectly returned. (Bug #26024)
- Using the [SQL_ATTR_CONNECTION_TIMEOUT](#) attributed did not update the timeout. (Bug #19823)
- Using [DataAdapter](#), Connector/ODBC may continually consume memory when reading the same records within a loop (Windows Server 2003 SP1/SP2 only). (Bug #20459)
- Connector/ODBC may insert the wrong parameter values when using prepared statements under 64-bit Linux. (Bug #22446)
- Using Connector/ODBC, with [SQLBindCol](#) and binding the length to the return value from [SQL_LEN_DATA_AT_EXEC](#) fails with a memory allocation error. (Bug #20547)
- The [SQLDriverConnect\(\)](#) ODBC method did not work with recent Connector/ODBC releases. (Bug #12393)

Changes in Connector/ODBC 3.51.12

Functionality added or changed:

- N/A

Bugs fixed:

- File DSNs could not be saved. (Bug #12019)

- [SQLColumns\(\)](#) returned no information for tables that had a column named using a reserved word. (Bug #9539)

Changes in Connector/ODBC 3.51.11

Functionality added or changed: No changes.

Bugs fixed:

- [mysql_list_dbcolumns\(\)](#) and [insert_fields\(\)](#) were retrieving all rows from a table. Fixed the queries generated by these functions to return no rows. (Bug #8198)
- [SQLGetTypeInfo\(\)](#) returned [tinyblob](#) for [SQL_VARBINARY](#) and nothing for [SQL_BINARY](#). Fixed to return [varbinary](#) for [SQL_VARBINARY](#), [binary](#) for [SQL_BINARY](#), and [longblob](#) for [SQL_LONGVARBINARY](#). (Bug #8138)

Connector/NET Change History

Changes in MySQL Connector/NET Version 5.0.4 (Not yet released)

Bugs fixed:

- [BINARY](#) and [VARBINARY](#) columns would be returned as a string, not binary, datatype. (Bug #25605)
- Connector/NET would not compile properly when used with Mono 1.2. (Bug #24263)
- Opening a connection would be slow due to hostname lookup. (Bug #26152).
- Connector/NET would fail to install under Windows Vista. (Bug #26430)
- Incorrect values/formats would be applied when the [OldSyntax](#) connection string option was used. (Bug #25950)
- Registry would be incorrectly populated with installation locations. (Bug #25928)
- Filling a table schema through a stored procedure triggers a runtime error. (Bug #25609)
- [MySQLConnection](#) throws an exception when connecting to MySQL v4.1.7. (Bug #25726)
- Returned data types of a [DataTypes](#) collection do not contain the right correct CLR Datatype. (Bug #25907)
- [GetSchema](#) and [DataTypes](#) would throw an exception due to an incorrect table name. (Bug #25906)
- Times with negative values would be returned incorrectly. (Bug #25912)
- [SELECT](#) did not work correctly when using a [WHERE](#) clause containing a UTF-8 string. (Bug #25651)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug #25614)
- When connecting to a MySQL Server earlier than version 4.1, the connection would hang when reading data. (Bug #25458)
- When connecting to a server, the return code from the connection could be zero, even though the hostname was incorrect. (Bug #24802)
- Using [ExecuteScalar\(\)](#) with more than one query, where one query fails, will hang the connection. (Bug #25443)

Changes in MySQL Connector/NET Version 5.0.3 (05 January 2007)

Functionality added or changed:

- SSL support has been updated.
- The [ViewColumns](#) [GetSchema](#) collection has been updated.
- The [CommandBuilder.DeriveParameters](#) function has been updated to the procedure cache.

- Improved speed and performance by re-architecting certain sections of the code.
- Usage Advisor has been implemented. The Usage Advisor checks your queries and will report if you are using the connection inefficiently.
- The [MySqlCommand](#) object now supports asynchronous query methods. This is implemented using the [BeginExecuteNonQuery](#) and [EndExecuteNonQuery](#) methods.
- Metadata from stored procedures and stored function execution are cached.
- PerfMon hooks have been added to monitor the stored procedure cache hits and misses.
- The ShapZipLib library has been replaced with the deflate support provided within .NET 2.0.
- Support for the embedded server and client library have been removed from this release. Support will be added back to a later release.

Bugs fixed:

- An exception would be raised, or the process would hang, if [SELECT](#) privileges on a database were not granted and a stored procedure was used. (Bug #25033)
- Nested transactions (which are unsupported) do not raise an error or warning. (Bug #22400)
- When adding parameter objects to a command object, if the parameter direction is set to [ReturnValue](#) before the parameter is added to the command object then when the command is executed it throws an error. (Bug #25013)
- Additional text added to error message (Bug #25178)
- Stored procedure executions are not thread safe. (Bug #23905)
- Using [Driver.IsTooOld\(\)](#) would return the wrong value. (Bug #24661)
- Deleting a connection to a disconnected server when using the Visual Studio Plugin would cause an assertion failure. (Bug #23687)
- When using a [DBNull.Value](#) as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the [DBNull.Value](#) . (Bug #24565)

Changes in MySQL Connector/NET Version 5.0.2 (06 November 2006)

Functionality added or changed:

- Important change: Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the number of procedures that are cached by using the [procedure cache](#) connection string.
- An [Ignore Prepare](#) option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.

Bugs fixed:

- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. (Bug #23071)

- Within Mono, using the [PreparedStatement](#) interface could result in an error due to a [BitArray](#) copying error. (Bug #18186)
- Using Windows Vista (RC2) as a non-privileged user would raise a [Registry key 'Global' access denied](#). (Bug #22882)
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve hostnames. (Bug #23758)
- Column names with accented characters were not parsed properly causing malformed column names in result sets. (Bug #23657)
- Connector/NET did not work as a data source for the [SqlDataSource](#) object used by ASP.NET 2.0. (Bug #16126)
- A [System.FormatException](#) exception would be raised when invoking a stored procedure with an [ENUM](#) input parameter. (Bug #23268)
- During installation, an antivirus error message would be raised (indicating a malicious script problem). (Bug #23245)
- An exception would be thrown when calling [GetSchemaTable](#) and [fields](#) was null. (Bug #23538)

Changes in MySQL Connector/NET Version 5.0.1 (01 October 2006)

Bugs fixed:

- Using [ExecuteScalar](#) with a datetime field, where the value of the field is "0000-00-00 00:00:00", a [MySQLConversionException](#) exception would be raised. (Bug #11991)
- An [MySQL.Data.Types.MySqlConversionException](#) would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug #9619)
- Incorrect field/data lengths could be returned for [VARCHAR](#) UTF8 columns. Bug (#14592)
- Submitting an empty string to a command object through [prepare](#) raises an [System.IndexOutOfRangeException](#), rather than a Connector/Net exception. (Bug #18391)
- Starting a transaction on a connection created by [MySQL.Data.MySqlClient.MySqlClientFactory](#), using [BeginTransaction](#) without specifying an isolation level, causes the SQL statement to fail with a syntax error. (Bug #22042)
- Connector/NET on a Turkish operating system, may fail to execute certain SQL statements correctly. (Bug #22452)
- You can now install the Connector/NET MSI package from the command line using the [/passive](#), [/quiet](#), [/q](#) options. (Bug #19994)
- The [MySqlException](#) class is now derived from the [DbException](#) class. (Bug #21874)
- Executing multiple queries as part of a transaction returns [There is already an openDataReader associated with this Connection which must be closed first](#). (Bug #7248)
- The <#> would not be accepted within column/table names, even though it was valid. (Bug #21521)

Changes in MySQL Connector/NET Version 5.0.0 (08 August 2006)

This is a new Alpha development release, fixing recently discovered bugs.

NOTE: This Alpha release, as any other pre-production release, should not be installed on production level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL has worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release. Please refer to our bug database at <http://bugs.mysql.com/> for more details about the individual bugs fixed in this version.

Bugs fixed:

- CommandText: Question mark in comment line is being parsed as a parameter. (Bug #6214)

Functionality added or changed:

- Implemented Usage Advisor.
- Added Async query methods.
- Reimplemented PacketReader/PacketWriter support into [MySQLStream](#) class.
- Added internal implementation of SHA1 so we don't have to distribute the OpenNetCF on mobile devices.
- Added usage advisor warnings for requesting column values by the wrong type.
- Reworked connection string classes to be simpler and faster.
- Added procedure metadata caching.
- Added perfmon hooks for stored procedure cache hits and misses.
- Implemented [MySQLConnectionBuilder](#) class.
- Implemented [MySQLClientFactory](#) class.
- Implemented classes and interfaces for ADO.Net 2.0 support.
- Replaced use of ICSHarpCode with .NET 2.0 internal deflate support.
- Refactored test suite to test all protocols in a single pass.
- Completely refactored how column values are handled to avoid boxing in some cases.

Changes in MySQL Connector/NET Version 1.0.10 (Not yet released)

Bugs fixed:

- [BINARY](#) and [VARBINARY](#) columns would be returned as a string, not binary, datatype. (Bug #25605)

Changes in MySQL Connector/NET Version 1.0.9 (02 February 2007)

Functionality added or changed:

- Important change: Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

- Important change: Binaries for .NET 1.0 are no longer supplied with this release. If you need support for .NET 1.0, you must build from source.
- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the number of procedures that are cached by using the [procedure cache](#) connection string.
- An [Ignore Prepare](#) option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.
- The ICSHarpCode ZipLib is no longer used by the Connector, and is no longer distributed with it.
- Improved [CommandBuilder.DeriveParameters](#) to first try and use the procedure cache before querying for the stored procedure metadata. Return parameters created with [DeriveParameters](#) now have the name [RETURN_VALUE](#).

Bugs fixed:

- Trying to fill a table schema through a stored procedure triggers a runtime error. (Bug #25609)
- [MySQLConnection](#) throws an exception when connecting to MySQL v4.1.7. (Bug #25726)
- [SELECT](#) did not work correctly when using a [WHERE](#) clause containing a UTF-8 string. (Bug #25651)
- Times with negative values would be returned incorrectly. (Bug #25912)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug #25614)
- When connecting to a server, the return code from the connection could be zero, even though the hostname was incorrect. (Bug #24802)
- Using [ExecuteScalar\(\)](#) with more than one query, where one query fails, will hang the connection. (Bug #25443)
- Nested transactions do not raise an error or warning. (Bug #22400)
- When adding parameter objects to a command object, if the parameter direction is set to [ReturnValue](#) before the parameter is added to the command object then when the command is executed it throws an error. (Bug #25013)
- Additional text added to error message. (Bug #25178)
- The [CommandBuilder](#) would mistakenly add insert parameters for a table column with auto incrementation enabled. (Bug #23862)
- Stored procedure executions are not thread safe. (Bug #23905)
- Using [Driver.IsTooOld\(\)](#) would return the wrong value. (Bug #24661)
- When using a [DBNull.Value](#) as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the [DBNull.Value](#). (Bug #24565)
- Within Mono, using the [PreparedStatement](#) interface could result in an error due to a [BitArray](#) copying error. (Bug 18186)
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve hostnames. (Bug #23758)
- An [System.OverflowException](#) would be raised when accessing a varchar field over 255 bytes. Bug (#23749)

Changes in MySQL Connector/NET Version 1.0.8 (20 October 2006)

Bugs fixed:

- Using [ExecuteScalar](#) with a datetime field, where the value of the field is "0000-00-00 00:00:00", a [MySQLConversionException](#) exception would be raised. (Bug #11991)
- The [MySQLDateTime](#) class did not contain constructors. (Bug #15112)
- [DataReader](#) would show the value of the previous row (or last row with non-null data) if the current row contained a [datetime](#) field with a null value. (Bug #16884)
- An [MySQL.Data.Types.MySQLConversionException](#) would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug #9619)
- When using [MySQLDataAdapter](#), connections to a MySQL server may remain open and active, even though the use of the connection has been completed and the data received. (Bug #8131)
- Incorrect field/data lengths could be returned for [VARCHAR](#) UTF8 columns. Bug (#14592)
- Submitting an empty string to a command object through [prepare](#) raises an [System.IndexOutOfRangeException](#), rather than a Connector/Net exception. (Bug #18391)
- Connector/NET on a Turkish operating system, may fail to execute certain SQL statements correctly. (Bug #22452)

- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug #19994)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug #7248)
- Calling `MySqlCommandBuilder.DeriveParameters` for a stored procedure that has no parameters would cause an application crash. (Bug #15077)
- A `SELECT` query on a table with a date with a value of '0000-00-00' would hang the application. (Bug #17736)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug #21521)
- Calling `Close` on a connection after calling a stored procedure would trigger a `NullReferenceException`. (Bug #20581)
- `IDataRecord.GetString` would raise `NullPointerException` for null values in returned rows. Method now throws `SqlNullValueException`. (Bug #19294)
- An exception would be raised when using an output parameter to a `System.String` value. (Bug #17814)
- The `DiscoverParameters` function would fail when a stored procedure used a `NUMERIC` parameter type. (Bug #19515)
- When running a query that included a date comparison, a `DateReader` error would be raised. (Bug #19481)
- Parameter substitution in queries where the order of parameters and table fields did not match would substitute incorrect values. (Bug #19261)
- When working with multiple threads, character set initialization would generate errors. (Bug #17106)
- When using an unsigned 64-bit integer in a stored procedure, the unsigned bit would be lost stored. (Bug #16934)
- The connection string parser did not allow single or double quotes in the password. (Bug #16659)
- The `CommandBuilder` ignored `Unsigned` flag at `Parameter` creation. (Bug #17375)
- `CHAR` type added to `MySqlDbType`. (Bug #17749)
- Unsigned data types were not properly supported. (Bug #16788)

Functionality added or changed:

- Stored procedures are now cached.
- The method for retrieving stored procedure metadata has been changed so that users without `SELECT` privileges on the `mysql.proc` table can use a stored procedure.

Changes in MySQL Connector/NET Version 1.0.7 (21 November 2005)

- Unsigned `tinyint` (NET byte) would lead to an incorrectly determined parameter type from the parameter value. (Bug #18570)
- The parameter collection object's `Add()` method added parameters to the list without first checking to see whether they already existed. Now it updates the value of the existing parameter object if it exists. (Bug #13927)
- A `#42000Query was empty` exception occurred when executing a query built with `MySqlCommandBuilder`, if the query string ended with a semicolon. (Bug #14631)
- Implemented the `MySqlCommandBuilder.DeriveParameters` method that is used to discover the parameters for a stored procedure. (Bug #13632)
- Added support for the `cp932` character set. (Bug #13806)
- Calling a stored procedure where a parameter contained special characters (such as '@') would produce an exception. Note that `ANSI_QUOTES` had to be enabled to make this possible. (Bug #13753)

- A statement that contained multiple references to the same parameter could not be prepared. (Bug #13541)
- The [Ping\(\)](#) method did not update the [State](#) property of the [Connection](#) object. (Bug #13658)

Changes in MySQL Connector/NET Version 1.0.6 (03 October 2005)

- The [nant](#) build sequence had problems. (Bug #12978)
- Serializing a parameter failed if the first value passed in was [NULL](#). (Bug #13276)
- Field names that contained the following characters caused errors: [\(\)%<>/](#) (Bug #13036)
- The Connector/NET 1.0.5 installer would not install alongside Connector/NET 1.0.4. (Bug #12835)
- Connector/NET 1.0.5 could not connect on Mono. (Bug #13345)

Changes in MySQL Connector/NET Version 1.0.5 (29 August 2005)

- With multiple hosts in the connection string, Connector/NET would not connect to the last host in the list. (Bug #12628)
- Connector/NET interpreted the new decimal data type as a byte array. (Bug #11294)
- The [cp1250](#) character set was not supported. (Bug #11621)
- Connection could fail when .NET thread pool had no available worker threads. (Bug #10637)
- Decimal parameters caused syntax errors. (Bug #11550, Bug #10486, Bug #10152)
- A call to a stored procedure caused an exception if the stored procedure had no parameters. (Bug #11542)
- Certain malformed queries would trigger a [Connection must be valid and open](#) error message. (Bug #11490)
- The [MySqlCommandBuilder](#) class could not handle queries that referenced tables in a database other than the default database. (Bug #8382)
- Connector/NET could not work properly with certain regional settings. (WL#8228)
- Trying to use a stored procedure when [Connection.Database](#) was not populated generated an exception. (Bug #11450)
- Trying to read a [TIMESTAMP](#) column generated an exception. (Bug #7951)
- Parameters were not recognized when they were separated by linefeeds. (Bug #9722)
- Calling [MySqlConnection.clone](#) when a connection string had not yet been set on the original connection would generate an error. (Bug #10281)
- Added support to call a stored function from Connector/NET. (Bug #10644)
- Connector/NET could not connect to MySQL 4.1.14. (Bug #12771)
- The [ConnectionString](#) property could not be set when a [MySqlConnection](#) object was added with the designer. (Bug #12551, Bug #8724)

Changes in MySQL Connector/NET Version 1.0.4 (20 January 2005)

- Calling prepare causing exception. (Bug #7243)
- Fixed another small problem with prepared statements.
- [MySqlCommand.Connection](#) returns an [IDbConnection](#). (Bug #7258)
- [MySqlAdapter.Fill](#) method throws error message [Non-negative number required](#). (Bug #7345)

- Clone method bug in [MySqlCommand](#). (Bug #7478)
- [MySqlDataReader.GetString\(index\)](#) returns non-Null value when field is [Null](#). (Bug #7612)
- [MySqlReader.GetInt32](#) throws exception if column is unsigned. (Bug #7755)
- [GetBytes](#) is working no more. (Bug #7704).
- Quote character \222 not quoted in [EscapeString](#). (Bug #7724)
- Fixed problem that causes named pipes to not work with some blob functionality.
- Fixed problem with shared memory connections.
- Problem with Multiple resultsets. (Bug #7436)
- Added or filled out several more topics in the API reference documentation.

Changes in MySQL Connector/NET Version 1.0.3-gamma (12 October 2004)

- Made MySQL the default named pipe name.
- Now [SHOW COLLATION](#) is used upon connection to retrieve the full list of charset ids.
- Fixed Invalid character set index: 200. (Bug #6547)
- Installer now includes options to install into GAC and create [Start Menu](#) items.
- Int64 Support in [MySqlCommand](#) Parameters. (Bug #6863)
- Connections now do not have to give a database on the connection string.
- [MySqlDataReader.GetChar\(int i\)](#) throws [IndexOutOfRangeException](#) exception. (Bug #6770)
- Fixed problem where multiple resultsets having different numbers of columns would cause a problem.
- Exception stack trace lost when re-throwing exceptions. (Bug #6983)
- Fixed major problem with detecting null values when using prepared statements.
- Errors in parsing stored procedure parameters. (Bug #6902)
- Integer "out" parameter from stored procedure returned as string. (Bug #6668)
- [MySqlDateTime](#) in Datatables sorting by Text, not Date. (Bug #7032)
- Invalid query string when using inout parameters (Bug #7133)
- Test suite fails with MySQL 4.0 because of case sensitivity of table names. (Bug #6831)
- Inserting [DateTime](#) causes [System.InvalidCastException](#) to be thrown. (Bug #7132)
- InvalidCast when using [DATE_ADD](#)-function. (Bug #6879)
- An Open Connection has been Closed by the Host System. (Bug #6634)
- Added [ServerThread](#) property to [MySqlConnection](#) to expose server thread id.
- Added Ping method to [MySqlConnection](#).
- Changed the name of the test suite to [MySql.Data.Tests.dll](#).

Changes in MySQL Connector/NET Version 1.0.2-gamma (15 November 2004)

- Fixed problem with [MySqlBinary](#) where string values could not be used to update extended text columns

- Fixed Installation directory ignored using custom installation (Bug #6329)
- Fixed problem where setting command text leaves the command in a prepared state
- Fixed double type handling in MySqlParameter(string parameterName, object value) (Bug #6428)
- Fixed Zero date "0000-00-00" is returned wrong when filling Dataset (Bug #6429)
- Fixed problem where calling stored procedures might cause an "Illegal mix of collations" problem.
- Added charset connection string option
- Fixed #HY000 Illegal mix of collations (latin1_swedish_ci,IMPLICIT) and (utf8_general_ (Bug #6322)
- Added the TableEditor CS and VB sample
- Fixed Charset-map for UCS-2 (Bug #6541)
- Updated the installer to include the new samples
- Fixed Long inserts take very long time (Bu #5453)
- Fixed Objects not being disposed (Bug #6649)
- Provider is now using character set specified by server as default

Changes in MySQL Connector/NET Version 1.0.1-beta2 (27 October 2004)

- Fixed BUG #5602 Possible bug in MySqlParameter(string, object) constructor
- Fixed BUG #5458 Calling GetChars on a longtext column throws an exception
- Fixed BUG #5474 cannot run a stored procedure populating mysqlcommand.parameters
- Fixed BUG #5469 Setting DbType throws NullReferenceException
- Fixed problem where connector was not issuing a CMD_QUIT before closing the socket
- Fixed BUG #5392 MySqlCommand sees "?" as parameters in string literals
- Fixed problem with ConnectionInternal where a key might be added more than once
- CP1252 is now used for Latin1 only when the server is 4.1.2 and later
- Fixed BUG #5388 DataReader reports all rows as NULL if one row is NULL
- Virtualized driver subsystem so future releases could easily support client or embedded server support
- Field buffers being reused to decrease memory allocations and increase speed
- Fixed problem where using old syntax while using the interfaces caused problems
- Using PacketWriter instead of Packet for writing to streams
- Refactored compression code into CompressedStream to clean up NativeDriver
- Added test case for resetting the command text on a prepared command
- Fixed problem where MySqlParameterCollection.Add() would throw unclear exception when given a null value (Bug #5621)
- Fixed constructor initialize problems in MySqlCommand() (Bug #5613)
- Fixed Parsing the ';' char (Bug #5876)
- Fixed missing Reference in DbType setter (Bug #5897)

- Fixed System.OverflowException when using YEAR datatype (Bug #6036)
- Added Aggregate function test (wasn't really a bug)
- Fixed serializing of floating point parameters (double, numeric, single, decimal) (Bug #5900)
- IsNullable error (Bug #5796)
- Fixed problem where connection lifetime on the connect string was not being respected
- Fixed problem where Min Pool Size was not being respected
- Fixed MySqlDataReader and 'show tables from ...' behavior (Bug #5256)
- Implemented SequentialAccess
- Fixed MySqlDateTime sets IsZero property on all subseq.records after first zero found (Bug #6006)
- Fixed Can't display Chinese correctly (Bug #5288)
- Fixed Russian character support as well
- Fixed Method TokenizeSql() uses only a limited set of valid characters for parameters (Bug #6217)
- Fixed NET Connector source missing resx files (Bug #6216)
- Fixed DBNull Values causing problems with retrieving/updating queries. (Bug #5798)
- Fixed Yet Another "object reference not set to an instance of an object" (Bug #5496)
- Fixed problem in PacketReader where it could try to allocate the wrong buffer size in EnsureCapacity
- Fixed GetBoolean returns wrong values (Bug #6227)
- Fixed IndexOutOfBounds when reading BLOB with DataReader with GetString(index) (Bug #6230)

Changes in MySQL Connector/NET Version 1.0.0 (01 September 2004)

- Thai encoding not correctly supported. (Bug #3889)
- Updated many of the test cases.
- Fixed problem with using compression.
- Bumped version number to 1.0.0 for beta 1 release.
- Added [COPYING.rtf](#) file for use in installer.
- Removed all of the XML comment warnings.
- Removed some last references to ByteFX.

Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)

- Added test fixture for prepared statements.
- All type classes now implement a [SerializeBinary](#) method for sending their data to a [PacketWriter](#).
- Added [PacketWriter](#) class that will enable future low-memory large object handling.
- Fixed many small bugs in running prepared statements and stored procedures.
- Changed command so that an exception will not be throw in executing a stored procedure with parameters in old syntax mode.
- [SingleRow](#) behavior now working right even with limit.

- [GetBytes](#) now only works on binary columns.
- Logger now truncates long sql commands so blob columns don't blow out our log.
- host and database now have a default value of "" unless otherwise set.
- Connection Timeout seems to be ignored. (Bug #5214)
- Added test case for bug# 5051: GetSchema not working correctly.
- Fixed problem where [GetSchema](#) would return false for [IsUnique](#) when the column is key.
- [MySqlDataReader](#) [GetXXX](#) methods now using the field level [MySqlValue](#) object and not performing conversions.
- [DataReader](#) returning NULL for time column. (Bug #5097)
- Added test case for [LOAD DATA LOCAL INFILE](#).
- Added replacetext custom nant task.
- Added [CommandBuilderTest](#) fixture.
- Added Last One Wins feature to [CommandBuilder](#).
- Fixed persist security info case problem.
- Fixed [GetBool](#) so that 1, true, "true", and "yes" all count as true.
- Make parameter mark configurable.
- Added the "old syntax" connection string parameter to allow use of @ parameter marker.
- [MySqlCommandBuilder](#). (Bug #4658)
- [ByteFX.MySqlClient](#) caches passwords if [Persist Security Info](#) is false. (Bug #4864)
- Updated license banner in all source files to include FLOSS exception.
- Added new .Types namespace and implementations for most current MySql types.
- Added [MySqlField41](#) as a subclass of [MySqlField](#).
- Changed many classes to now use the new .Types types.
- Changed type [enum int](#) to [Int32](#), [short](#) to [Int16](#), and [bigint](#) to [Int64](#).
- Added dummy types [UInt16](#), [UInt32](#), and [UInt64](#) to allow an unsigned parameter to be made.
- Connections are now reset when they are pulled from the connection pool.
- Refactored auth code in driver so it can be used for both auth and reset.
- Added [UserReset](#) test in [PoolingTests.cs](#).
- Connections are now reset using [COM_CHANGE_USER](#) when pulled from the pool.
- Implemented [SingleResultSet](#) behavior.
- Implemented support of unicode.
- Added char set mappings for utf-8 and ucs-2.
- Time fields overflow using bytefx .net mysql driver (Bug #4520)
- Modified time test in data type test fixture to check for time spans where hours > 24.
- Wrong string with backslash escaping in [ByteFx.Data.MySqlClient.MySqlParameter](#). (Bug #4505)

- Added code to Parameter test case TestQuoting to test for backslashes.
- [MySQLCommandBuilder](#) fails with multi-word column names. (Bug #4486)
- Fixed bug in [TokenizeSql](#) where underscore would terminate character capture in parameter name.
- Added test case for spaces in column names.
- [MySQLDataReader.GetBytes](#) don't works correctly. (Bug #4324)
- Added [GetBytes\(\)](#) test case to [DataReader](#) test fixture.
- Now reading all server variables in [InternalConnection.Configure](#) into [Hashtable](#).
- Now using [string\[\]](#) for index map in [CharSetMap](#).
- Added CRInSQL test case for carriage returns in SQL.
- Setting [maxPacketSize](#) to default value in [Driver.ctor](#).
- Setting [MySQLDbType](#) on a parameter doesn't set generic type. (Bug #4442)
- Removed obsolete data types [Long](#) and [LongLong](#).
- Overflow exception thrown when using "use pipe" on connection string. (Bug #4071)
- Changed "use pipe" keyword to "pipe name" or just "pipe".
- Allow reading multiple resultsets from a single query.
- Added flags attribute to [ServerStatusFlags](#) enum.
- Changed name of [ServerStatus](#) enum to [ServerStatusFlags](#).
- Inserted data row doesn't update properly.
- Error processing show create table. (Bug #4074)
- Change [Packet.ReadLenInteger](#) to [ReadPackedLong](#) and added [packet.ReadPackedInteger](#) that always reads integers packed with 2,3,4.
- Added [syntax.cs](#) test fixture to test various SQL syntax bugs.
- Improper handling of time values. Now time value of 00:00:00 is not treated as null. (Bug #4149)
- Moved all test suite files into [TestSuite](#) folder.
- Fixed bug where null column would move the result packet pointer backward.
- Added new nant build script.
- Clear tablename so it will be regen'ed properly during the next [GenerateSchema](#). (Bug #3917)
- [GetValues](#) was always returning zero and was also always trying to copy all fields rather than respecting the size of the array passed in. (Bug #3915)
- Implemented shared memory access protocol.
- Implemented prepared statements for MySQL 4.1.
- Implemented stored procedures for MySQL 5.0.
- Renamed [MySQLInternalConnection](#) to [InternalConnection](#).
- SQL is now parsed as chars, fixes problems with other languages.
- Added logging and allow batch connection string options.

- [RowUpdating](#) event not set when setting the [DataAdapter](#) property. (Bug #3888)
- Fixed bug in char set mapping.
- Implemented 4.1 authentication.
- Improved open/auth code in driver.
- Improved how connection bits are set during connection.
- Database name is now passed to server during initial handshake.
- Changed namespace for client to [MySql.Data.MySqlClient](#).
- Changed assembly name of client to [MySql.Data.dll](#).
- Changed license text in all source files to GPL.
- Added the [MySqlClient.build](#) Nant file.
- Removed the mono batch files.
- Moved some of the unused files into notused folder so nant build file can use wildcards.
- Implemented shared memory accesss.
- Major revamp in code structure.
- Prepared statements now working for MySql 4.1.1 and later.
- Finished implementing auth for 4.0, 4.1.0, and 4.1.1.
- Changed namespace from [MySQL.Data.MySQLClient](#) back to [MySql.Data.MySqlClient](#).
- Fixed bug in [CharSetMapping](#) where it was trying to use text names as ints.
- Changed namespace to [MySQL.Data.MySQLClient](#).
- Integrated auth changes from UC2004.
- Fixed bug where calling any of the GetXXX methods on a datareader before or after reading data would not throw the appropriate exception (thanks Luca Morelli).
- Added [TimeSpan](#) code in parameter.cs to properly serialize a timespan object to mysql time format (thanks Gianluca Colombo).
- Added [TimeStamp](#) to parameter serialization code. Prevented [DataAdatper](#) updates from working right (thanks Michael King).
- Fixed a misspelling in [MySqlHelper.cs](#) (thanks Patrick Kristiansen).

Changes in MySQL Connector/NET Version 0.76

- Driver now using charset number given in handshake to create encoding.
- Changed command editor to point to [MySqlClient.Design](#).
- Fixed bug in [Version.isAtLeast](#).
- Changed [DBConnectionString](#) to support changes done to [MySqlConnectionString](#).
- Removed [SqlCommandEditor](#) and [DataAdapterPreviewDialog](#).
- Using new long return values in many places.
- Integrated new [CompressedStream](#) class.

- Changed [ConnectionString](#) and added attributes to allow it to be used in [MySqlClient.Design](#).
- Changed [packet.cs](#) to support newer lengths in [ReadLenInteger](#).
- Changed other classes to use new properties and fields of [MySqlConnection](#).
- [ConnectionInternal](#) is now using PING to see whether the server is alive.
- Moved toolbox bitmaps into resource folder.
- Changed [field.cs](#) to allow values to come directly from row buffer.
- Changed to use the new driver.Send syntax.
- Using a new packet queueing system.
- Started work handling the "broken" compression packet handling.
- Fixed bug in [StreamCreator](#) where failure to connect to a host would continue to loop infinitely (thanks Kevin Casella).
- Improved connectstring handling.
- Moved designers into Pro product.
- Removed some old commented out code from [command.cs](#).
- Fixed a problem with compression.
- Fixed connection object where an exception throw prior to the connection opening would not leave the connection in the connecting state (thanks Chris Cline).
- Added GUID support.
- Fixed sequence out of order bug (thanks Mark Reay).

Changes in MySQL Connector/NET Version 0.75

- Enum values now supported as parameter values (thanks Philipp Sumi).
- Year datatype now supported.
- Fixed compression.
- Fixed bug where a parameter with a [TimeSpan](#) as the value would not serialize properly.
- Fixed bug where default constructor would not set default connection string values.
- Added some XML comments to some members.
- Work to fix/improve compression handling.
- Improved [ConnectionString](#) handling so that it better matches the standard set by [SqlClient](#).
- A [MySqlException](#) is now thrown if a username is not included in the connection string.
- Localhost is now used as the default if not specified on the connection string.
- An exception is now thrown if an attempt is made to set the connection string while the connection is open.
- Small changes to [ConnectionString](#) docs.
- Removed [MultiHostStream](#) and [MySqlStream](#). Replaced it with [Common/StreamCreator](#).
- Added support for Use Pipe connection string value.
- Added Platform class for easier access to platform utility functions.

- Fixed small pooling bug where new connection was not getting created after [IsAlive](#) fails.
- Added [Platform.cs](#) and [StreamCreator.cs](#).
- Fixed [Field.cs](#) to properly handle 4.1 style timestamps.
- Changed [Common.Version](#) to [Common.DBVersion](#) to avoid name conflict.
- Fixed [field.cs](#) so that text columns return the right field type.
- Added [MySqlError](#) class to provide some reference for error codes (thanks Geert Veenstra).

Changes in MySQL Connector/NET Version 0.74

- Added Unix socket support (thanks Mohammad DAMT).
- Only calling [Thread.Sleep](#) when no data is available.
- Improved escaping of quote characters in parameter data.
- Removed misleading comments from [parameter.cs](#).
- Fixed pooling bug.
- Fixed [ConnectionString](#) editor dialog (thanks marco p (pomarc)).
- [UserId](#) now supported in connection strings (thanks Jeff Neeley).
- Attempting to create a parameter that is not input throws an exception (thanks Ryan Gregg).
- Added much documentation.
- Checked in new [MultiHostStream](#) capability. Big thanks to Dan Guisinger for this. he originally submitted the code and idea of supporting multiple machines on the connect string.
- Added a lot of documentation.
- Fixed speed issue with 0.73.
- Changed to [Thread.Sleep\(0\)](#) in [MySqlDataStream](#) to help optimize the case where it doesn't need to wait (thanks Todd German).
- Prepopulating the idlepools to [MinPoolSize](#).
- Fixed [MySqlPool](#) deadlock condition as well as stupid bug where [CreateNewPooledConnection](#) was not ever adding new connections to the pool. Also fixed [MySqlStream.ReadBytes](#) and [ReadByte](#) to not use [TicksPerSecond](#) which does not appear to always be right. (thanks Matthew J. Peddlesden)
- Fix for precision and scale (thanks Matthew J. Peddlesden).
- Added [Thread.Sleep\(1\)](#) to stream reading methods to be more cpu friendly (thanks Sean McGinnis).
- Fixed problem where [ExecuteReader](#) would sometime return null (thanks Lloyd Dupont).
- Fixed major bug with null field handling (thanks Naucki).
- Enclosed queries for [max_allowed_packet](#) and [characteraset](#) inside try catch (and set defaults).
- Fixed problem where socket was not getting closed properly (thanks Steve!).
- Fixed problem where [ExecuteNonQuery](#) was not always returning the right value.
- Fixed [InternalConnection](#) to not use [@@session.max_allowed_packet](#) but use [@@max_allowed_packet](#). (Thanks Miguel)
- Added many new XML doc lines.

- Fixed sql parsing to not send empty queries (thanks Rory).
- Fixed problem where the reader was not unpeeking the packet on close.
- Fixed problem where user variables were not being handled (thanks Sami Vaaraniemi).
- Fixed loop checking in the MySqlPool (thanks Steve M. Brown)
- Fixed [ParameterCollection.Add](#) method to match [SqlCommand](#) (thanks Joshua Mouch).
- Fixed [ConnectionString](#) parsing to handle no and yes for boolean and not lowercase values (thanks Naucki).
- Added [InternalConnection](#) class, changes to pooling.
- Implemented Persist Security Info.
- Added [security.cs](#) and [version.cs](#) to project
- Fixed [DateTime](#) handling in [Parameter.cs](#) (thanks Burkhard Perens-Golomb).
- Fixed parameter serialization where some types would throw a cast exception.
- Fixed [DataReader](#) to convert all returned values to prevent casting errors (thanks Keith Murray).
- Added code to [Command.ExecuteReader](#) to return null if the initial SQL command throws an exception (thanks Burkhard Perens-Golomb).
- Fixed [ExecuteScalar](#) bug introduced with restructure.
- Restructure to allow for [LOCAL DATA INFILE](#) and better sequencing of packets.
- Fixed several bugs related to restructure.
- Early work done to support more secure passwords in Mysql 4.1. Old passwords in 4.1 not supported yet.
- Parameters appearing after system parameters are now handled correctly (Adam M. (adamml)).
- Strings can now be assigned directly to blob fields (Adam M.).
- Fixed float parameters (thanks Pent).
- Improved Parameter constructor and [ParameterCollection.Add](#) methods to better match [SqlCommand](#) (thanks Joshua Mouch).
- Corrected [Connection.CreateCommand](#) to return a [MySqlCommand](#) type.
- Fixed connection string designer dialog box problem (thanks Abraham Guyt).
- Fixed problem with sending commands not always reading the response packet (thanks Joshua Mouch).
- Fixed parameter serialization where some blobs types were not being handled (thanks Sean McGinnis).
- Removed spurious [MessageBox.show](#) from [DataReader](#) code (thanks Joshua Mouch).
- Fixed a nasty bug in the split sql code (thanks everyone!).

Changes in MySQL Connector/NET Version 0.71

- Fixed bug in [MySqlStream](#) where too much data could attempt to be read (thanks Peter Belbin)
- Implemented [HasRows](#) (thanks Nash Pherson).
- Fixed bug where tables with more than 252 columns cause an exception (thanks Joshua Kessler).
- Fixed bug where SQL statements ending in ; would cause a problem (thanks Shane Krueger).
- Fixed bug in driver where error messages were getting truncated by 1 character (thanks Shane Krueger).

- Made [MySqlException](#) serializable (thanks Mathias Hasselmann).

Changes in MySQL Connector/NET Version 0.70

- Updated some of the character code pages to be more accurate.
- Fixed problem where readers could be opened on connections that had readers open.
- Moved test to separate assembly [MySqlClientTests](#).
- Fixed stupid problem in driver with sequence out of order (Thanks Peter Belbin).
- Added some pipe tests.
- Increased default max pool size to 50.
- Compiles with Mono 0-24.
- Fixed connection and data reader dispose problems.
- Added [String](#) datatype handling to parameter serialization.
- Fixed sequence problem in driver that occurred after thrown exception (thanks Burkhard Perken-Golomb).
- Added support for [CommandBehavior.SingleRow](#) to [DataReader](#).
- Fixed command sql processing so quotes are better handled (thanks Theo Spears).
- Fixed parsing of double, single, and decimal values to account for non-English separators. You still have to use the right syntax if you using hard coded sql, but if you use parameters the code will convert floating point types to use '.' appropriately internal both into the server and out.
- Added [MySqlStream](#) class to simplify timeouts and driver coding.
- Fixed [DataReader](#) so that it is closed properly when the associated connection is closed. [thanks smishra]
- Made client more [SqlClient](#) compliant so that [DataReaders](#) have to be closed before the connection can be used to run another command.
- Improved [DBNull.Value](#) handling in the fields.
- Added several unit tests.
- Fixed [MySqlException](#) base class.
- Improved driver coding
- Fixed bug where [NextResult](#) was returning false on the last resultset.
- Added more tests for MySQL.
- Improved casting problems by equating unsigned 32bit values to [Int64](#) and unsigned 16bit values to [Int32](#), and so forth.
- Added new constructor for [MySqlParameter](#) for (name, type, size, srccol)
- Fixed bug in [MySqlDataReader](#) where it didn't check for null fieldlist before returning field count.
- Started adding [MySqlClient](#) unit tests (added [MySqlClient/Tests](#) folder and some test cases).
- Fixed some things in Connection String handling.
- Moved [INIT_DB](#) to [MySqlPool](#). I may move it again, this is in preparation of the conference.
- Fixed bug inside [CommandBuilder](#) that prevented inserts from happening properly.

- Reworked some of the internals so that all three execute methods of `Command` worked properly.
- Fixed many small bugs found during benchmarking.
- The first cut of `CoconnectionPooling` is working. "min pool size" and "max pool size" are respected.
- Work to enable multiple resultsets to be returned.
- Character sets are handled much more intelligently now. The driver queries MySQL at startup for the default character set. That character set is then used for conversions if that code page can be loaded. If not, then the default code page for the current OS is used.
- Added code to save the inferred type in the name,value constructor of `Parameter`.
- Also, inferred type if value of null parameter is changed using `Value` property.
- Converted all files to use proper Camel case. MySQL is now `MySql` in all files. `PgSQL` is now `PgSql`.
- Added attribute to `PgSql` code to prevent designer from trying to show.
- Added `MySQLDbType` property to `Parameter` object and added proper conversion code to convert from `DbType` to `MySQLDbType`.
- Removed unused `ObjectToString` method from `MySQLParameter.cs`.
- Fixed `Add(..)` method in `ParameterCollection` so that it doesn't use `Add(name, value)` instead.
- Fixed `IndexOf` and `Contains` in `ParameterCollection` to be aware that parameter names are now stored without `@`.
- Fixed `Command.ConvertSQLToBytes` so it only allows characters that can be in MySQL variable names.
- Fixed `DataReader` and `Field` so that blob fields read their data from `Field.cs` and `GetBytes` works right.
- Added simple query builder editor to `CommandText` property of `MySQLCommand`.
- Fixed `CommandBuilder` and `Parameter` serialization to account for Parameters not storing `@` in their names.
- Removed `MySQLFieldType` enum from `Field.cs`. Now using `MySQLDbType` enum.
- Added `Designer` attribute to several classes to prevent designer view when using VS.Net.
- Fixed Initial catalog typo in `ConnectionString` designer.
- Removed 3 parameter constructor for `MySQLParameter` that conflicted with (name, type, value).
- Changed `MySQLParameter` so `paramName` is now stored without leading `@` (this fixed null inserts when using designer).
- Changed `TypeConverter` for `MySQLParameter` to use the constructor with all properties.

Changes in MySQL Connector/NET Version 0.68

- Fixed sequence issue in driver.
- Added `DbParametersEditor` to make parameter editing more like `SqlClient`.
- Fixed `Command` class so that parameters can be edited using the designer
- Update connection string designer to support `Use Compression` flag.
- Fixed string encoding so that European characters like ä will work correctly.
- Creating base classes to aid in building new data providers.
- Added support for UID key in connection string.

- Field, parameter, command now using `DBNull.Value` instead of null.
- `CommandBuilder` using `DBNull.Value`.
- `CommandBuilder` now builds insert command correctly when an `auto_insert` field is not present.
- Field now uses `typeof` keyword to return `System.Types` (performance).

Changes in MySQL Connector/NET Version 0.65

- `MySQLCommandBuilder` now implemented.
- Transaction support now implemented (not all table types support this).
- `GetSchemaTable` fixed to not use `xsd` (for Mono).
- Driver is now Mono-compatible.
- `TIME` data type now supported.
- More work to improve Timestamp data type handling.
- Changed signatures of all classes to match corresponding `SqlClient` classes.

Changes in MySQL Connector/NET Version 0.60

- Protocol compression using `SharpZipLib` (www.icsharpcode.net).
- Named pipes on Windows now working properly.
- Work done to improve `Timestamp` data type handling.
- Implemented `IEnumerable` on `DataReader` so `DataGrid` would work.

Changes in MySQL Connector/NET Version 0.50

- Speed increased dramatically by removing bugging network sync code.
- Driver no longer buffers rows of data (more ADO.Net compliant).
- Conversion bugs related to `TIMESTAMP` and `DATETIME` fields fixed.

MySQL Visual Studio Plugin Change History

Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)

Bugs fixed:

- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. (Bug #23071)
- The Add Connection dialog of the Server Explorer would freeze when accessing databases with capitalized characters in their name. (Bug #24875)

Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)

This is a bug fix release to resolve an incompatibility issue with Connector/NET 5.0.1.

It is critical that this release only be used with Connector/NET 5.0.1. After installing Connector/NET 5.0.1, you will need to make a small change in your `machine.config` file. This file should be located at `%win%\Microsoft.Net\Framework\v2.0.50727\CONFIG\machine.config` (`%win%` should be the location of your Windows folder). Near the bottom of the file you will see a line like this:

```
<add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data"/>
```

It needs to be changed to be like this:

```
<add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data,
Version=5.0.1.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
```

Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)

This is the first beta release.

Features in this release:

- DDEX (Data Designer Extensibility) compatibility.
- Ability to work with MySQL objects (tables, views, stored procedures, etc) from within Server Explorer.

MySQL Connector/J Change History

Changes in MySQL Connector/J 5.1.x

Changes in MySQL Connector/J 5.1.0 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is false (i.e. Connector/J does not use server-side prepared statements).

Changes in MySQL Connector/J 5.0.x

Changes in MySQL Connector/J 5.0.5 (Not yet released)

Functionality added or changed:

- Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is false (i.e. Connector/J does not use server-side prepared statements).

- Performance enhancement of initial character set configuration, driver will only send commands required to configure connection character set session variables if the current values on the server do not match what is required.

Bugs fixed:

- Comments automatically generated by the Hibernate framework could cause problems in the parser when using batched inserts. (Bug #25025)

- When using a JDBC connection URL that is malformed, the [NonRegisteringDriver.getPropertyInfo](#) method will throw a Null Pointer Exception (NPE) error. (Bug #22628)
- Using [DatabaseMetaData.getSQLKeywords\(\)](#) does not return a consistent list of reserved keywords. (Bug #24794)
- Specifying [US-ASCII](#) as the character set in a connection to a MySQL 4.1 or newer server does not map correctly. (Bug #24840)
- Storing a [java.util.Date](#) objection in a [BLOB](#) column would not be serialized correctly during [setObject](#). (Bug #25787)
- A query execution which timed out did not always throw a [MySQLTimeoutException](#). (Bug #25836)
- A connection error would occur when connecting to a MySQL server with certain character sets. Some collations/character sets reported as "unknown" (specifically as variants of existing character sets), and inability to override the detected server character set. (Bug #23645)
- Using [setFetchSize\(\)](#) breaks prepared [SHOW](#) and other commands. (Bug #24360)
- Using [DATETIME](#) column would result in time shifts when [useServerPrepStmts](#) was true. (Bug #24344)
- Inconsistency between [getSchemas](#) and [INFORMATION_SCHEMA](#). (Bug #23304)
- When using the [rewriteBatchedStatements](#) connection option with [PreparedState.executeBatch\(\)](#) an internal memory leak would occur. (Bug #25073)
- Fixed issue where field-level for metadata from [DatabaseMetaData](#) when using [INFORMATION_SCHEMA](#) didn't have references to current connections, sometimes leading to [NullPointerExceptions](#) when introspecting them via [ResultSetMetaData](#).
- Connector/J now returns a better error message when server doesn't return enough information to determine stored procedure/function parameter types. (Bug #24065)
- When using server-side prepared statements and timestamp columns, value would be incorrectly populated (with nanoseconds, not microseconds). (Bug #21438)
- Timer instance used for [Statement.setQueryTimeout\(\)](#) created per-connection, rather than per-VM, causing memory leak. (Bug #25514)
- Results sets from [UPDATE](#) statements with multi-statement queries would cause an [SQLException](#) error. (Bug #25009)
- [StringUtil.indexOfIgnoreCaseRespectQuotes\(\)](#) isn't case-insensitive on the first character of the target. This bug also affected [rewriteBatchedStatements](#) functionality when prepared statements did not use uppercase for the [VALUES](#) clause. (Bug #25047)
- Some exceptions thrown out of [StandardSocketFactory](#) were needlessly wrapped, obscuring their true cause, especially when using socket timeouts. (Bug #21480)
- [DatabaseMetaData.getSchemas\(\)](#) doesn't return a [TABLE_CATALOG](#) column. (Bug #23303)
- [EscapeProcessor](#) gets confused by multiple backslashes. We now push the responsibility of syntax errors back on to the server for most escape sequences. (Bug #25399)
- [INOUT](#) parameters in [CallableStatements](#) get doubly-escaped. (Bug #25379)
- Connector/J would ignore the [socketFactory](#) property, making it difficult to use the [NamedPipeSocketFactory](#). (Bug #26326)

Changes in MySQL Connector/J 5.0.4 (20 October 2006)

Bugs fixed:

- Column names don't match metadata in cases where server doesn't return original column names (column functions) thus breaking compatibility with applications that expect 1-1 mappings between [findColumn\(\)](#) and [rsmd.getColumnName\(\)](#), usually manifests itself as "Can't find column ()" exceptions. (Bug #21379)

- When using `information_schema` for metadata, `COLUMN_SIZE` for `getColumns()` is not clamped to range of `java.lang.Integer` as is the case when not using `information_schema`, thus leading to a truncation exception that isn't present when not using `information_schema`. (Bug #21544)
- Newlines causing whitespace to span confuse procedure parser when getting parameter metadata for stored procedures. (Bug #22024)
- Driver was using milliseconds for `Statement.setQueryTimeout()` when specification says argument is to be in seconds. (Bug #22359)
- Workaround for server crash when calling stored procedures via a server-side prepared statement (driver now detects `prepare(stored procedure)` and substitutes client-side prepared statement). (Bug #22297)
- Added new `_ci` collations to `CharsetMapping` - `utf8_unicode_ci` not working. (Bug #22456)
- Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement). (Bug #22290)
- `DBMD.getColumns()` does not return expected `COLUMN_SIZE` for the SET type, now returns length of largest possible set disregarding whitespace or the "," delimiters to be consistent with the ODBC driver. (Bug #22613)

Other changes:

- Fixed configuration property `jdbcCompliantTruncation` was not being used for reads of result set values.
- Driver now supports `{call sp}` (without "()") if procedure has no arguments).
- Driver now sends numeric 1 or 0 for client-prepared statement `setBoolean()` calls instead of '1' or '0'.
- `DatabaseMetaData` correctly reports true for `supportsCatalog*()` methods.

Changes in MySQL Connector/J 5.0.3 (26 July 2006)

- Fixed `Statement.cancel()` causes `NullPointerException` if underlying connection has been closed due to server failure. (Bug #20650)
- Added configuration option `noAccessToProcedureBodies` which will cause the driver to create basic parameter metadata for `CallableStatements` when the user does not have access to procedure bodies via `SHOW CREATE PROCEDURE` or selecting from `mysql.proc` instead of throwing an exception. The default value for this option is `false`

Bugs fixed:

- If the connection to the server has been closed due to a server failure, then the cleanup process will call `Statement.cancel()`, triggering a `NullPointerException`, even though there is no active connection. (Bug #20650)

Changes in MySQL Connector/J 5.0.2-beta (11 July 2006)

- Fixed can't use `XAConnection` for local transactions when no global transaction is in progress. (Bug #17401)
- Fixed driver fails on non-ASCII platforms. The driver was assuming that the platform character set would be a superset of MySQL's `latin1` when doing the handshake for authentication, and when reading error messages. We now use Cp1252 for all strings sent to the server during the handshake phase, and a hard-coded mapping of the `language` system variable to the character set that is used for error messages. (Bug #18086)
- Fixed `ConnectionProperties` (and thus some subclasses) are not serializable, even though some J2EE containers expect them to be. (Bug #19169)
- Fixed `MySQLValidConnectionChecker` for JBoss doesn't work with `MySQLXADataSources`. (Bug #20242)
- Better caching of character set converters (per-connection) to remove a bottleneck for multibyte character sets.
- Added connection/datasource property `pinGlobalTxToPhysicalConnection` (defaults to `false`). When set to `true`, when using `XAConnections`, the driver ensures that operations on a given XID are always routed to the same physical connection. This allows the `XAConnection` to support `XA START ... JOIN` after `XA END` has been called, and is also

a workaround for transaction managers that don't maintain thread affinity for a global transaction (most either always maintain thread affinity, or have it as a configuration option).

- [MysqlXaConnection.recover\(int flags\)](#) now allows combinations of [XAResource.TMSTARTRSCAN](#) and [TMENDRSCAN](#). To simulate the 「scanning」 nature of the interface, we return all prepared XIDs for [TMSTARTRSCAN](#), and no new XIDs for calls with [TMNOFLAGS](#), or [TMENDRSCAN](#) when not in combination with [TMSTARTRSCAN](#). This change was made for API compliance, as well as integration with IBM WebSphere's transaction manager.

Changes in MySQL Connector/J 5.0.1-beta (Not Released)

Not released due to a packaging error

Changes in MySQL Connector/J 5.0.0-beta (22 December 2005)

- [XADataSource](#) implemented (ported from 3.2 branch which won't be released as a product). Use [com.mysql.jdbc.jdbc2.optional.MysqlXADataSource](#) as your datasource class name in your application server to utilize XA transactions in MySQL-5.0.10 and newer.
- [PreparedStatement.setString\(\)](#) didn't work correctly when [sql_mode](#) on server contained [NO_BACKSLASH_ESCAPES](#) and no characters that needed escaping were present in the string.
- Attempt detection of the MySQL type [BINARY](#) (it's an alias, so this isn't always reliable), and use the [java.sql.Types.BINARY](#) type mapping for it.
- Moved [-bin-g.jar](#) file into separate [debug](#) subdirectory to avoid confusion.
- Don't allow [.setAutoCommit\(true\)](#), or [.commit\(\)](#) or [.rollback\(\)](#) on an XA-managed connection as per the JDBC specification.
- If the connection [useTimezone](#) is set to [true](#), then also respect time zone conversions in escape-processed string literals (for example, ["{ts ...}"](#) and ["{t ...}"](#)).
- Return original column name for [RSMD.getColumnNames\(\)](#) if the column was aliased, alias name for [.getColumnLabel\(\)](#) (if aliased), and original table name for [.getTableName\(\)](#). Note this only works for MySQL-4.1 and newer, as older servers don't make this information available to clients.
- Setting [useJDBCCompliantTimezoneShift=true](#) (it's not the default) causes the driver to use GMT for all [TIMESTAMP/DATETIME](#) time zones, and the current VM time zone for any other type that refers to time zones. This feature can not be used when [useTimezone=true](#) to convert between server and client time zones.
- Add one level of indirection of internal representation of [CallableStatement](#) parameter metadata to avoid class not found issues on JDK-1.3 for [ParameterMetadata](#) interface (which doesn't exist prior to JDBC-3.0).
- Added unit tests for [XADataSource](#), as well as friendlier exceptions for XA failures compared to the "stock" [XAException](#) (which has no messages).
- Idle timeouts cause [XAConnections](#) to whine about rolling themselves back. (Bug #14729)
- Added support for Connector/MXJ integration via url subprotocol [jdbc:mysql:mxj://...](#)
- Moved all [SQLException](#) constructor usage to a factory in [SQLException](#) (ground-work for JDBC-4.0 [SQLState](#)-based exception classes).
- Removed Java5-specific calls to [BigDecimal](#) constructor (when result set value is ["](#), [\(int\)0](#) was being used as an argument indirectly via method return value. This signature doesn't exist prior to Java5.)
- Added service-provider entry to [META-INF/services/java.sql.Driver](#) for JDBC-4.0 support.
- Return ["\[VAR\]BINARY"](#) for [RSMD.getColumnTypeName\(\)](#) when that is actually the type, and it can be distinguished (MySQL-4.1 and newer).
- When fix for Bug #14562 was merged from 3.1.12, added functionality for [CallableStatement](#)'s parameter metadata to return correct information for [.getParameterClassName\(\)](#).

- Fuller synchronization of [Connection](#) to avoid deadlocks when using multithreaded frameworks that multithread a single connection (usually not recommended, but the JDBC spec allows it anyways), part of fix to Bug #14972).
- Implementation of [Statement.cancel\(\)](#) and [Statement.setQueryTimeout\(\)](#). Both require MySQL-5.0.0 or newer server, require a separate connection to issue the KILL QUERY statement, and in the case of [setQueryTimeout\(\)](#) creates an additional thread to handle the timeout functionality.

Note: Failures to cancel the statement for [setQueryTimeout\(\)](#) may manifest themselves as [RuntimeExceptions](#) rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

Changes in MySQL Connector/J 3.1.x

Changes in MySQL Connector/J 3.1.15 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is false (i.e. Connector/J does not use server-side prepared statements).

Bugs fixed:

- Specifying [US-ASCII](#) as the character set in a connection to a MySQL 4.1 or newer server does not map correctly. (Bug #24840)

Changes in MySQL Connector/J 3.1.14 (10-19-2006)

- Fixed updatable result set throws ClassCastException when there is row data and moveToInsertRow() is called. (Fixes Bug#20479)
- Fixed Updatable result set that contains a BIT column fails when server-side prepared statements are used. (Fixes Bug#20485)
- Fixed memory leak with profileSQL=true. (Fixes Bug#16987)
- Connection fails to localhost when using timeout and IPv6 is configured. (Fixes Bug#19726)
- Fixed NullPointerException in MysqlDataSourceFactory due to Reference containing RefAddr with null content. (Fixes Bug#16791)
- Fixed ResultSet.getShort() for UNSIGNED TINYINT returns incorrect values when using server-side prepared statements. (Fixes Bug#20306)
- Fixed can't pool server-side prepared statements, exception raised when re-using them. (Fixes Bug#20687)
- Fixed BUG#21062 - ResultSet.getSomeInteger() doesn't work for BIT(>1).
- Fixed BUG#18880 - ResultSet.getFloatFromString() can't retrieve values near Float.MIN/MAX_VALUE.
- Fixed BUG#20888 - escape of quotes in client-side prepared statements parsing not respected. Patch covers more than bug report, including NO_BACKSLASH_ESCAPES being set, and stacked quote characters forms of escaping (i.e. " or "").
- Fixed BUG#19993 - ReplicationDriver does not always round-robin load balance depending on URL used for slaves list.
- Fixed calling toString() on ResultSetMetaData for driver-generated (i.e. from DatabaseMetaData method calls, or from getGeneratedKeys()) result sets would raise a NullPointerException.
- Fixed Bug#21207 - Driver throws NPE when tracing prepared statements that have been closed (in asSQL()).

- Removed logger autodetection altogether, must now specify logger explicitly if you want to use a logger other than one that logs to STDERR.
- Fixed BUG#22290 - Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement).
- Driver now sends numeric 1 or 0 for client-prepared statement `setBoolean()` calls instead of '1' or '0'.
- Fixed bug where driver would not advance to next host if `roundRobinLoadBalance=true` and the last host in the list is down.
- Fixed BUG#18258 - `DatabaseMetaData.getTables()`, `columns()` with bad catalog parameter threw exception rather than return empty result set (as required by spec).
- Check and store value for `continueBatchOnError` property in constructor of `Statements`, rather than when executing batches, so that `Connections` closed out from underneath statements don't cause `NullPointerExceptions` when it's required to check this property.
- Fixed bug when calling stored functions, where parameters weren't numbered correctly (first parameter is now the return value, subsequent parameters if specified start at index "2").

Changes in MySQL Connector/J 3.1.13 (26 May 2006)

- `INOUT` parameter does not store `IN` value. (Bug #15464)
- Exception thrown for new decimal type when using updatable result sets. (Bug #14609)
- No "dos" character set in MySQL > 4.1.0. (Bug #15544)
- `PreparedStatement.setObject()` serializes `BigInteger` as object, rather than sending as numeric value (and is thus not complementary to `.getObject()` on an `UNSIGNED LONG` type). (Bug #15383)
- `ResultSet.getShort()` for `UNSIGNED TINYINT` returned wrong values. (Bug #11874)
- `lib-nodist` directory missing from package breaks out-of-box build. (Bug #15676)
- `DBMD.getColumns()` returns wrong type for `BIT`. (Bug #15854)
- Fixed issue where driver was unable to initialize character set mapping tables. Removed reliance on `.properties` files to hold this information, as it turns out to be too problematic to code around class loader hierarchies that change depending on how an application is deployed. Moved information back into the `CharsetMapping` class. (Bug #14938)
- Fixed updatable result set doesn't return `AUTO_INCREMENT` values for `insertRow()` when multiple column primary keys are used. (the driver was checking for the existence of single-column primary keys and an autoincrement value > 0 instead of a straightforward `isAutoIncrement()` check). (Bug #16841)
- Fixed `Statement.getGeneratedKeys()` throws `NullPointerException` when no query has been processed. (Bug #17099)
- Fixed driver trying to call methods that don't exist on older and newer versions of Log4j. The fix is not trying to auto-detect presence of log4j, too many different incompatible versions out there in the wild to do this reliably. (Bug #13469)

If you relied on autodetection before, you will need to add "logger=com.mysql.jdbc.log.Log4JLogger" to your JDBC URL to enable Log4J usage, or alternatively use the new "CommonsLogger" class to take care of this.

- Added support for Apache Commons logging, use "com.mysql.jdbc.log.CommonsLogger" as the value for the "logger" configuration property.
- `LogFactory` now prepends "com.mysql.jdbc.log" to log class name if it can't be found as-specified. This allows you to use "short names" for the built-in log factories, for example "logger=CommonsLogger" instead of "logger=com.mysql.jdbc.log.CommonsLogger".
- Fixed issue with `ReplicationConnection` incorrectly copying state, doesn't transfer connection context correctly when transitioning between the same read-only states. (Bug #15570)

- Fixed issue where server-side prepared statements don't cause truncation exceptions to be thrown when truncation happens. (Bug #18041)
- Added performance feature, re-writing of batched executes for [Statement.executeBatch\(\)](#) (for all DML statements) and [PreparedStatement.executeBatch\(\)](#) (for INSERTs with VALUE clauses only). Enable by using "rewriteBatchedStatements=true" in your JDBC URL.
- Fixed [CallableStatement.registerOutParameter\(\)](#) not working when some parameters pre-populated. Still waiting for feedback from JDBC experts group to determine what correct parameter count from [getMetaData\(\)](#) should be, however. (Bug #17898)
- Fixed calling [clearParameters\(\)](#) on a closed prepared statement causes NPE. (Bug #17587)
- Map "latin1" on MySQL server to CP1252 for MySQL > 4.1.0.
- Added additional accessor and mutator methods on ConnectionProperties so that DataSource users can use same naming as regular URL properties.
- Fixed data truncation and [getWarnings\(\)](#) only returns last warning in set. (Bug #18740)
- Improved performance of retrieving [BigDecimal](#), [Time](#), [Timestamp](#) and [Date](#) values from server-side prepared statements by creating fewer short-lived instances of [Strings](#) when the native type is not an exact match for the requested type. Fixes Bug #18496 for [BigDecimal](#)s.
- Fixed aliased column names where length of name > 251 are corrupted. (Bug #18554)
- Fixed [ResultSet.wasNull\(\)](#) not always reset correctly for booleans when done via conversion for server-side prepared statements. (Bug #17450)
- Fixed invalid classname returned for [ResultSetMetaData.getColumnClassName\(\)](#) for [BIGINT](#) type. (Bug #19282)
- Fixed case where driver wasn't reading server status correctly when fetching server-side prepared statement rows, which in some cases could cause warning counts to be off, or multiple result sets to not be read off the wire.
- Driver now aware of fix for [BIT](#) type metadata that went into MySQL-5.0.21 for server not reporting length consistently (Bug #13601).
- Fixed [PreparedStatement.setObject\(int, Object, int\)](#) doesn't respect scale of [BigDecimal](#)s. (Bug #19615)
- Fixed [ResultSet.wasNull\(\)](#) returns incorrect value when extracting native string from server-side prepared statement generated result set. (Bug #19282)

Changes in MySQL Connector/J 3.1.12 (30 November 2005)

- Fixed client-side prepared statement bug with embedded `?` characters inside quoted identifiers (it was recognized as a placeholder, when it was not).
- Don't allow [executeBatch\(\)](#) for [CallableStatements](#) with registered [OUT/INOUT](#) parameters (JDBC compliance).
- Fall back to platform-encoding for [URLDecoder.decode\(\)](#) when parsing driver URL properties if the platform doesn't have a two-argument version of this method.
- Java type conversion may be incorrect for [MEDIUMINT](#). (Bug #14562)
- Added configuration property [useGmtMillisForDatetimes](#) which when set to [true](#) causes [ResultSet.getDate\(\)](#), [.getTimestamp\(\)](#) to return correct millis-since GMT when [.getTime\(\)](#) is called on the return value (currently default is [false](#) for legacy behavior).
- Fixed [DatabaseMetaData.storesIdentifiers\(\)](#):
 - If [lower_case_table_names=0](#) (on server):
 - [storesLowerCaseIdentifiers\(\)](#) returns [false](#)
 - [storesLowerCaseQuotedIdentifiers\(\)](#) returns [false](#)

- `storesMixedCaseIdentifiers()` returns `true`
- `storesMixedCaseQuotedIdentifiers()` returns `true`
- `storesUpperCaseIdentifiers()` returns `false`
- `storesUpperCaseQuotedIdentifiers()` returns `true`
- If `lower_case_table_names=1` (on server):
 - `storesLowerCaseIdentifiers()` returns `true`
 - `storesLowerCaseQuotedIdentifiers()` returns `true`
 - `storesMixedCaseIdentifiers()` returns `false`
 - `storesMixedCaseQuotedIdentifiers()` returns `false`
 - `storesUpperCaseIdentifiers()` returns `false`
 - `storesUpperCaseQuotedIdentifiers()` returns `true`
- `DatabaseMetaData.getColumns()` doesn't return `TABLE_NAME` correctly. (Bug #14815)
- Escape processor replaces quote character in quoted string with string delimiter. (Bug #14909)
- OpenOffice expects `DBMD.supportsIntegrityEnhancementFacility()` to return `true` if foreign keys are supported by the datasource, even though this method also covers support for check constraints, which MySQL doesn't have. Setting the configuration property `overrideSupportsIntegrityEnhancementFacility` to `true` causes the driver to return `true` for this method. (Bug #12975)
- Added `com.mysql.jdbc.testsuite.url.default` system property to set default JDBC url for testsuite (to speed up bug resolution when I'm working in Eclipse).
- Unable to initialize character set mapping tables (due to J2EE classloader differences). (Bug #14938)
- Deadlock while closing server-side prepared statements from multiple threads sharing one connection. (Bug #14972)
- `logSlowQueries` should give better info. (Bug #12230)
- Extraneous sleep on `autoReconnect`. (Bug #13775)
- Driver incorrectly closes streams passed as arguments to `PreparedStatement`. Reverts to legacy behavior by setting the JDBC configuration property `autoClosePstmtStreams` to `true` (also included in the 3-0-Compat configuration 「bundle」). (Bug #15024)
- `maxQuerySizeToLog` is not respected. Added logging of bound values for `execute()` phase of server-side prepared statements when `profileSQL=true` as well. (Bug #13048)
- Usage advisor complains about unreferenced columns, even though they've been referenced. (Bug #15065)
- Don't increase timeout for failover/reconnect. (Bug #6577)
- Process escape tokens in `Connection.prepareStatement(...)`. (Bug #15141) You can disable this behavior by setting the JDBC URL configuration property `processEscapeCodesForPrepStmts` to `false`.
- Reconnect during middle of `executeBatch()` should not occur if `autoReconnect` is enabled. (Bug #13255)

Changes in MySQL Connector/J 3.1.11-stable (07 October 2005)

- Spurious `!` on console when character encoding is `utf8`. (Bug #11629)
- Fixed statements generated for testcases missing `;` for 「plain」 statements.
- Incorrect generation of testcase scripts for server-side prepared statements. (Bug #11663)

- Fixed regression caused by fix for Bug #11552 that caused driver to return incorrect values for unsigned integers when those integers were within the range of the positive signed type.
- Moved source code to Subversion repository.
- Escape tokenizer doesn't respect stacked single quotes for escapes. (Bug #11797)
- `GEOMETRY` type not recognized when using server-side prepared statements.
- `ReplicationConnection` won't switch to slave, throws 「Catalog can't be null」 exception. (Bug #11879)
- Properties shared between master and slave with replication connection. (Bug #12218)
- `Statement.getWarnings()` fails with NPE if statement has been closed. (Bug #10630)
- Only get `char[]` from SQL in `PreparedStatement.ParseInfo()` when needed.
- Geometry types not handled with server-side prepared statements. (Bug #12104)
- `StringUtils.getBytes()` doesn't work when using multi-byte character encodings and a length in characters is specified. (Bug #11614)
- `Pstmt.setObject(..., Types.BOOLEAN)` throws exception. (Bug #11798)
- `maxPerformance.properties` mis-spells 「elideSetAutoCommits」. (Bug #11976)
- `DBMD.supportsLower/Mixed/UpperIdentifiers()` reports incorrect values for servers deployed on Windows. (Bug #11575)
- `ResultSet.moveToCurrentRow()` fails to work when preceded by a call to `ResultSet.moveToInsertRow()`. (Bug #11190)
- `VARBINARY` data corrupted when using server-side prepared statements and `.setBytes()`. (Bug #11115)
- `explainSlowQueries` hangs with server-side prepared statements. (Bug #12229)
- Escape processor didn't honor strings demarcated with double quotes. (Bug #11498)
- Lifted restriction of changing streaming parameters with server-side prepared statements. As long as `all` streaming parameters were set before execution, `.clearParameters()` does not have to be called. (due to limitation of client/server protocol, prepared statements can not reset individual stream data on the server side).
- Reworked `Field` class, `*Buffer`, and `MysqlIO` to be aware of field lengths > `Integer.MAX_VALUE`.
- Updated `DBMD.supportsCorrelatedQueries()` to return `true` for versions > 4.1, `supportsGroupByUnrelated()` to return `true` and `getResultSetHoldability()` to return `HOLD_CURSORS_OVER_COMMIT`.
- Handling of catalog argument in `DatabaseMetaData.getIndexInfo()`, which also means changes to the following methods in `DatabaseMetaData`: (Bug #12541)
 - `getBestRowIdentifier()`
 - `getColumns()`
 - `getCrossReference()`
 - `getExportedKeys()`
 - `getImportedKeys()`
 - `getIndexInfo()`
 - `getPrimaryKeys()`
 - `getProcedures()` (and thus indirectly `getProcedureColumns()`)

- [getTables\(\)](#)

The [catalog](#) argument in all of these methods now behaves in the following way:

- Specifying [NULL](#) means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set [nullCatalogMeansCurrent=true](#) in your JDBC URL properties.
- Specifying [""](#) means 「current」 catalog, even though this isn't quite JDBC spec compliant, it's there for legacy users.
- Specifying a catalog works as stated in the API docs.
- Made [Connection.clientPrepare\(\)](#) available from 「wrapped」 connections in the [jdbc2.optional](#) package (connections built by [ConnectionPoolDataSource](#) instances).
- Added [Connection.isMasterConnection\(\)](#) for clients to be able to determine if a multi-host master/slave connection is connected to the first host in the list.
- Tokenizer for [=](#) in URL properties was causing [sessionVariables=...](#) to be parameterized incorrectly. (Bug #12753)
- Foreign key information that is quoted is parsed incorrectly when [DatabaseMetaData](#) methods use that information. (Bug #11781)
- The [sendBlobChunkSize](#) property is now clamped to [max_allowed_packet](#) with consideration of stream buffer size and packet headers to avoid [PacketTooBigExceptions](#) when [max_allowed_packet](#) is similar in size to the default [sendBlobChunkSize](#) which is 1M.
- [CallableStatement.clearParameters\(\)](#) now clears resources associated with [INOUT/OUTPUT](#) parameters as well as [INPUT](#) parameters.
- [Connection.prepareCall\(\)](#) is database name case-sensitive (on Windows systems). (Bug #12417)
- [cp1251](#) incorrectly mapped to [win1251](#) for servers newer than 4.0.x. (Bug #12752)
- [java.sql.Types.OTHER](#) returned for [BINARY](#) and [VARBINARY](#) columns when using [DatabaseMetaData.getColumns\(\)](#). (Bug #12970)
- [ServerPreparedStatement.getBinding\(\)](#) now checks if the statement is closed before attempting to reference the list of parameter bindings, to avoid throwing a [NullPointerException](#).
- [ResultSetMetaData](#) from [Statement.getGeneratedKeys\(\)](#) caused a [NullPointerException](#) to be thrown whenever a method that required a connection reference was called. (Bug #13277)
- Backport of [Field](#) class, [ResultSetMetaData.getColumnClassName\(\)](#), and [ResultSet.getObject\(int\)](#) changes from 5.0 branch to fix behavior surrounding [VARCHAR BINARY/VARBINARY](#) and related types.
- Fixed [NullPointerException](#) when converting [catalog](#) parameter in many [DatabaseMetaDataMethods](#) to [byte\[\]](#)s (for the result set) when the parameter is [null](#). ([null](#) isn't technically allowed by the JDBC specification, but we've historically allowed it).
- Backport of [VAR\[BINARY|CHAR\] \[BINARY\]](#) types detection from 5.0 branch.
- Read response in [MysqlIO.sendFileToServer\(\)](#), even if the local file can't be opened, otherwise next query issued will fail, because it's reading the response to the empty [LOAD DATA INFILE](#) packet sent to the server.
- Workaround for Bug #13374: [ResultSet.getStatement\(\)](#) on closed result set returns [NULL](#) (as per JDBC 4.0 spec, but not backward-compatible). Set the connection property [retainStatementAfterResultSetClose](#) to [true](#) to be able to retrieve a [ResultSet](#)'s statement after the [ResultSet](#) has been closed via [.getStatement\(\)](#) (the default is [false](#), to be JDBC-compliant and to reduce the chance that code using JDBC leaks [Statement](#) instances).
- URL configuration parameters don't allow ['&'](#) or ['='](#) in their values. The JDBC driver now parses configuration parameters as if they are encoded using the application/x-www-form-urlencoded format as specified by [java.net.URLDecoder](#) (<http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html>). (Bug #13453)

If the '%' character is present in a configuration property, it must now be represented as %25, which is the encoded form of '%' when using application/x-www-form-urlencoded encoding.

- The configuration property [sessionVariables](#) now allows you to specify variables that start with the '@' sign.
- When [gatherPerfMetrics](#) is enabled for servers older than 4.1.0, a [NullPointerException](#) is thrown from the constructor of [ResultSet](#) if the query doesn't use any tables. (Bug #13043)

Changes in MySQL Connector/J 3.1.10-stable (23 June 2005)

- Fixed connecting without a database specified raised an exception in [MysqlIO.changeDatabaseTo\(\)](#).
- Initial implementation of [ParameterMetadata](#) for [PreparedStatement.getParameterMetadata\(\)](#). Only works fully for [CallableStatements](#), as current server-side prepared statements return every parameter as a [VARCHAR](#) type.

Changes in MySQL Connector/J 3.1.9-stable (22 June 2005)

- Overhaul of character set configuration, everything now lives in a properties file.
- Driver now correctly uses CP932 if available on the server for Windows-31J, CP932 and MS932 java encoding names, otherwise it resorts to SJIS, which is only a close approximation. Currently only MySQL-5.0.3 and newer (and MySQL-4.1.12 or .13, depending on when the character set gets backported) can reliably support any variant of CP932.
- [com.mysql.jdbc.PreparedStatement.ParseInfo](#) does unnecessary call to [toCharArray\(\)](#). (Bug #9064)
- Memory leak in [ServerPreparedStatement](#) if [serverPrepare\(\)](#) fails. (Bug #10144)
- Actually write manifest file to correct place so it ends up in the binary jar file.
- Added [createDatabaseIfNotExist](#) property (default is [false](#)), which will cause the driver to ask the server to create the database specified in the URL if it doesn't exist. You must have the appropriate privileges for database creation for this to work.
- Unsigned [SMALLINT](#) treated as signed for [ResultSet.getInt\(\)](#), fixed all cases for [UNSIGNED](#) integer values and server-side prepared statements, as well as [ResultSet.getObject\(\)](#) for [UNSIGNED TINYINT](#). (Bug #10156)
- Double quotes not recognized when parsing client-side prepared statements. (Bug #10155)
- Made [enableStreamingResults\(\)](#) visible on [com.mysql.jdbc.jdbc2.optional.StatementWrapper](#).
- Made [ServerPreparedStatement.asSql\(\)](#) work correctly so auto-explain functionality would work with server-side prepared statements.
- Made JDBC2-compliant wrappers public in order to allow access to vendor extensions.
- Cleaned up logging of profiler events, moved code to dump a profiler event as a string to [com.mysql.jdbc.log.LogUtils](#) so that third parties can use it.
- [DatabaseMetaData.supportsMultipleOpenResults\(\)](#) now returns [true](#). The driver has supported this for some time, DBMD just missed that fact.
- Driver doesn't support [{?=CALL\(...\)}](#) for calling stored functions. This involved adding support for function retrieval to [DatabaseMetaData.getProcedures\(\)](#) and [getProcedureColumns\(\)](#) as well. (Bug #10310)
- [SQLException](#) thrown when retrieving [YEAR\(2\)](#) with [ResultSet.getString\(\)](#). The driver will now always treat [YEAR](#) types as [java.sql.Date](#)s and return the correct values for [getString\(\)](#). Alternatively, the [yearIsDateType](#) connection property can be set to [false](#) and the values will be treated as [SHORT](#)s. (Bug #10485)
- The datatype returned for [TINYINT\(1\)](#) columns when [tinyInt1isBit=true](#) (the default) can be switched between [Types.BOOLEAN](#) and [Types.BIT](#) using the new configuration property [transformedBitsBoolean](#), which defaults to [false](#). If set to [false](#) (the default), [DatabaseMetaData.getColumns\(\)](#) and [ResultSetMetaData.getColumnType\(\)](#) will return [Types.BOOLEAN](#) for [TINYINT\(1\)](#) columns. If [true](#), [Types.BOOLEAN](#) will be returned instead. Regardless

of this configuration property, if `tinyInt1isBit` is enabled, columns with the type `TINYINT(1)` will be returned as `java.lang.Boolean` instances from `ResultSet.getObject(...)`, and `ResultSetMetaData.getColumnClassName()` will return `java.lang.Boolean`.

- `SQLException` is thrown when using property `characterSetResults` with `cp932` or `eucjms`. (Bug #10496)
- Reorganized directory layout. Sources now are in `src` folder. Don't pollute parent directory when building, now output goes to `./build`, distribution goes to `./dist`.
- Added support/bug hunting feature that generates `.sql` test scripts to `STDERR` when `autoGenerateTestcaseScript` is set to `true`.
- 0-length streams not sent to server when using server-side prepared statements. (Bug #10850)
- Setting `cachePrepStmts=true` now causes the `Connection` to also cache the check the driver performs to determine if a prepared statement can be server-side or not, as well as caches server-side prepared statements for the lifetime of a connection. As before, the `prepStmtCacheSize` parameter controls the size of these caches.
- Try to handle `OutOfMemoryErrors` more gracefully. Although not much can be done, they will in most cases close the connection they happened on so that further operations don't run into a connection in some unknown state. When an OOM has happened, any further operations on the connection will fail with a 「Connection closed」 exception that will also list the OOM exception as the reason for the implicit connection close event.
- Don't send `COM_RESET_STMT` for each execution of a server-side prepared statement if it isn't required.
- Driver detects if you're running MySQL-5.0.7 or later, and does not scan for `LIMIT ?[,?]` in statements being prepared, as the server supports those types of queries now.
- `VARBINARY` data corrupted when using server-side prepared statements and `ResultSet.getBytes()`. (Bug #11115)
- `Connection.setCatalog()` is now aware of the `useLocalSessionState` configuration property, which when set to `true` will prevent the driver from sending `USE ...` to the server if the requested catalog is the same as the current catalog.
- Added the following configuration bundles, use one or many via the `useConfigs` configuration property:
 - `maxPerformance` — maximum performance without being reckless
 - `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can
 - `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality
- Added `maintainTimeStats` configuration property (defaults to `true`), which tells the driver whether or not to keep track of the last query time and the last successful packet sent to the server's time. If set to `false`, removes two syscalls per query.
- `autoReconnect` ping causes exception on connection startup. (Bug #11259)
- Connector/J dumping query into `SQLException` twice. (Bug #11360)
- Fixed `PreparedStatement.setClob()` not accepting `null` as a parameter.
- Production package doesn't include JBoss integration classes. (Bug #11411)
- Removed nonsensical 「costly type conversion」 warnings when using usage advisor.

Changes in MySQL Connector/J 3.1.8-stable (14 April 2005)

- Fixed `DatabaseMetaData.getTables()` returning views when they were not asked for as one of the requested table types.
- Added support for new precision-math `DECIMAL` type in MySQL 5.0.3 and up.
- Fixed `ResultSet.getTime()` on a `NULL` value for server-side prepared statements throws NPE.
- Made `Connection.ping()` a public method.

- `DATE_FORMAT()` queries returned as `BLOBs` from `getObject()`. (Bug #8868)
- `ServerPreparedStatement` now correctly 「stream」 `BLOB/CLOB` data to the server. You can configure the threshold chunk size using the JDBC URL property `blobSendChunkSize` (the default is 1MB).
- `BlobFromLocator` now uses correct identifier quoting when generating prepared statements.
- Server-side session variables can be preset at connection time by passing them as a comma-delimited list for the connection property `sessionVariables`.
- Fixed regression in `ping()` for users using `autoReconnect=true`.
- `PreparedStatement.addBatch()` doesn't work with server-side prepared statements and streaming `BINARY` data. (Bug #9040)
- `DBMD.supportsMixedCase*Identifiers()` returns wrong value on servers running on case-sensitive filesystems. (Bug #8800)
- Cannot use `UTF-8` for `characterSetResults` configuration property. (Bug #9206)
- A continuation of Bug #8868, where functions used in queries that should return non-string types when resolved by temporary tables suddenly become opaque binary strings (work-around for server limitation). Also fixed fields with type of `CHAR(n) CHARACTER SET BINARY` to return correct/matching classes for `RSMD.getColumnClassName()` and `ResultSet.getObject()`. (Bug #9236)
- `DBMD.supportsResultSetConcurrency()` not returning `true` for forward-only/read-only result sets (we obviously support this). (Bug #8792)
- `DATA_TYPE` column from `DBMD.getBestRowIdentifier()` causes `ArrayIndexOutOfBoundsException` when accessed (and in fact, didn't return any value). (Bug #8803)
- Check for empty strings (") when converting `CHAR/VARCHAR` column data to numbers, throw exception if `emptyStringsConvertToZero` configuration property is set to `false` (for backward-compatibility with 3.0, it is now set to `true` by default, but will most likely default to `false` in 3.2).
- `PreparedStatement.getMetaData()` inserts blank row in database under certain conditions when not using server-side prepared statements. (Bug #9320)
- `Connection.canHandleAsPreparedStatement()` now makes 「best effort」 to distinguish `LIMIT` clauses with placeholders in them from ones without in order to have fewer false positives when generating work-arounds for statements the server cannot currently handle as server-side prepared statements.
- Fixed `build.xml` to not compile `log4j` logging if `log4j` not available.
- Added support for the c3p0 connection pool's (<http://c3p0.sf.net/>) validation/connection checker interface which uses the lightweight `COM_PING` call to the server if available. To use it, configure your c3p0 connection pool's `connectionTesterClassName` property to use `com.mysql.jdbc.integration.c3p0.MysqlConnectionTester`.
- Better detection of `LIMIT` inside/outside of quoted strings so that the driver can more correctly determine whether a prepared statement can be prepared on the server or not.
- Stored procedures with same name in different databases confuse the driver when it tries to determine parameter counts/types. (Bug #9319)
- Added finalizers to `ResultSet` and `Statement` implementations to be JDBC spec-compliant, which requires that if not explicitly closed, these resources should be closed upon garbage collection.
- Stored procedures with `DECIMAL` parameters with storage specifications that contained `'` in them would fail. (Bug #9682)
- `PreparedStatement.setObject(int, Object, int type, int scale)` now uses scale value for `BigDecimal` instances.
- `Statement.getMoreResults()` could throw NPE when existing result set was `.close()`d. (Bug #9704)
- The performance metrics feature now gathers information about number of tables referenced in a `SELECT`.

- The logging system is now automatically configured. If the value has been set by the user, via the URL property [logger](#) or the system property [com.mysql.jdbc.logger](#), then use that, otherwise, autodetect it using the following steps:
 1. Log4j, if it's available,
 2. Then JDK1.4 logging,
 3. Then fallback to our [STDERR](#) logging.
- [DBMD.getTables\(\)](#) shouldn't return tables if views are asked for, even if the database version doesn't support views. (Bug #9778)
- Fixed driver not returning [true](#) for [-1](#) when [ResultSet.getBoolean\(\)](#) was called on result sets returned from server-side prepared statements.
- Added a [Manifest.MF](#) file with implementation information to the [.jar](#) file.
- More tests in [Field.isOpaqueBinary\(\)](#) to distinguish opaque binary (that is, fields with type [CHAR\(n\)](#) and [CHARACTER SET BINARY](#)) from output of various scalar and aggregate functions that return strings.
- Should accept [null](#) for catalog (meaning use current) in DBMD methods, even though it's not JDBC-compliant for legacy's sake. Disable by setting connection property [nullCatalogMeansCurrent](#) to [false](#) (which will be the default value in C/J 3.2.x). (Bug #9917)
- Should accept [null](#) for name patterns in DBMD (meaning '%'), even though it isn't JDBC compliant, for legacy's sake. Disable by setting connection property [nullNamePatternMatchesAll](#) to [false](#) (which will be the default value in C/J 3.2.x). (Bug #9769)

Changes in MySQL Connector/J 3.1.7-stable (18 February 2005)

- Timestamp key column data needed [_binary](#) stripped for [UpdatableResultSet.refreshRow\(\)](#). (Bug #7686)
- Timestamps converted incorrectly to strings with server-side prepared statements and updatable result sets. (Bug #7715)
- Detect new [sql_mode](#) variable in string form (it used to be integer) and adjust quoting method for strings appropriately.
- Added [holdResultsOpenOverStatementClose](#) property (default is [false](#)), that keeps result sets open over [statement.close\(\)](#) or new execution on same statement (suggested by Kevin Burton).
- Infinite recursion when 「falling back」 to master in failover configuration. (Bug #7952)
- Disable multi-statements (if enabled) for MySQL-4.1 versions prior to version 4.1.10 if the query cache is enabled, as the server returns wrong results in this configuration.
- Fixed duplicated code in [configureClientCharset\(\)](#) that prevented [useOldUTF8Behavior=true](#) from working properly.
- Removed [dontUnpackBinaryResults](#) functionality, the driver now always stores results from server-side prepared statements as is from the server and unpacks them on demand.
- Emulated locators corrupt binary data when using server-side prepared statements. (Bug #8096)
- Fixed synchronization issue with [ServerPreparedStatement.serverPrepare\(\)](#) that could cause deadlocks/crashes if connection was shared between threads.
- By default, the driver now scans SQL you are preparing via all variants of [Connection.prepareStatement\(\)](#) to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this by passing [emulateUnsupportedPstmts=false](#) in your JDBC URL. (Bug #4718)
- Remove [_binary](#) introducer from parameters used as in/out parameters in [CallableStatement](#).
- Always return [byte\[\]](#)s for output parameters registered as [*BINARY](#).

- Send correct value for 「boolean」 `true` to server for `PreparedStatement.setObject(n, "true", Types.BIT)`.
- Fixed bug with Connection not caching statements from `prepareStatement()` when the statement wasn't a server-side prepared statement.
- Choose correct 「direction」 to apply time adjustments when both client and server are in GMT time zone when using `ResultSet.get(..., cal)` and `PreparedStatement.set(..., cal)`.
- Added `dontTrackOpenResources` option (default is `false`, to be JDBC compliant), which helps with memory use for non-well-behaved apps (that is, applications that don't close `Statement` objects when they should).
- `ResultSet.getString()` doesn't maintain format stored on server, bug fix only enabled when `noDatetimeStringSync` property is set to `true` (the default is `false`). (Bug #8428)
- Fixed NPE in `ResultSet.realClose()` when using usage advisor and result set was already closed.
- `PreparedStatements` not creating streaming result sets. (Bug #8487)
- Don't pass `NULL` to `String.valueOf()` in `ResultSet.getNativeConvertToString()`, as it stringifies it (that is, returns `null`), which is not correct for the method in question.
- `ResultSet.getBigDecimal()` throws exception when rounding would need to occur to set scale. The driver now chooses a rounding mode of 「half up」 if non-rounding `BigDecimal.setScale()` fails. (Bug #8424)
- Added `useLocalSessionState` configuration property, when set to `true` the JDBC driver trusts that the application is well-behaved and only sets autocommit and transaction isolation levels using the methods provided on `java.sql.Connection`, and therefore can manipulate these values in many cases without incurring round-trips to the database server.
- Added `enableStreamingResults()` to `Statement` for connection pool implementations that check `Statement.setFetchSize()` for specification-compliant values. Call `Statement.setFetchSize(>=0)` to disable the streaming results for that statement.
- Added support for `BIT` type in MySQL-5.0.3. The driver will treat `BIT(1-8)` as the JDBC standard `BIT` type (which maps to `java.lang.Boolean`), as the server does not currently send enough information to determine the size of a bitfield when `< 9` bits are declared. `BIT(>9)` will be treated as `VARBINARY`, and will return `byte[]` when `getObject()` is called.

Changes in MySQL Connector/J 3.1.6-stable (23 December 2004)

- Fixed hang on `SocketInputStream.read()` with `Statement.setMaxRows()` and multiple result sets when driver has to truncate result set directly, rather than tacking a `LIMIT n` on the end of it.
- `DBMD.getProcedures()` doesn't respect catalog parameter. (Bug #7026)

Changes in MySQL Connector/J 3.1.5-gamma (02 December 2004)

- Fix comparisons made between string constants and dynamic strings that are converted with either `toUpperCase()` or `toLowerCase()` to use `Locale.ENGLISH`, as some locales 「override」 case rules for English. Also use `StringUtils.indexOfIgnoreCase()` instead of `.toUpperCase().indexOf()`, avoids creating a very short-lived transient `String` instance.
- Server-side prepared statements did not honor `zeroDateTimeBehavior` property, and would cause class-cast exceptions when using `ResultSet.getObject()`, as the all-zero string was always returned. (Bug #5235)
- Fixed batched updates with server prepared statements weren't looking if the types had changed for a given batched set of parameters compared to the previous set, causing the server to return the error 「Wrong arguments to mysql_stmt_execute()」.
- Handle case when string representation of timestamp contains trailing 「.」 with no numbers following it.
- Inefficient detection of pre-existing string instances in `ResultSet.getNativeString()`. (Bug #5706)
- Don't throw exceptions for `Connection.releaseSavepoint()`.

- Use a per-session [Calendar](#) instance by default when decoding dates from [ServerPreparedStatements](#) (set to old, less performant behavior by setting property [dynamicCalendars=true](#)).
- Added experimental configuration property [dontUnpackBinaryResults](#), which delays unpacking binary result set values until they're asked for, and only creates object instances for non-numerical values (it is set to [false](#) by default). For some usecase/jvm combinations, this is friendlier on the garbage collector.
- [UNSIGNED BIGINT](#) unpacked incorrectly from server-side prepared statement result sets. (Bug #5729)
- [ServerSidePreparedStatement](#) allocating short-lived objects unnecessarily. (Bug #6225)
- Removed unwanted new [Throwable\(\)](#) in [ResultSet](#) constructor due to bad merge (caused a new object instance that was never used for every result set created). Found while profiling for Bug #6359.
- Fixed too-early creation of [StringBuffer](#) in [EscapeProcessor.escapeSQL\(\)](#), also return [String](#) when escaping not needed (to avoid unnecessary object allocations). Found while profiling for Bug #6359.
- Use null-safe-equals for key comparisons in updatable result sets.
- [SUM\(\)](#) on [DECIMAL](#) with server-side prepared statement ignores scale if zero-padding is needed (this ends up being due to conversion to [DOUBLE](#) by server, which when converted to a string to parse into [BigDecimal](#), loses all 「padding」 zeros). (Bug #6537)
- Use [DatabaseMetaData.getIdentifierQuoteString\(\)](#) when building DBMD queries.
- Use 1MB packet for sending file for [LOAD DATA LOCAL INFILE](#) if that is < [max_allowed_packet](#) on server.
- [ResultSetMetaData.getColumnDisplaySize\(\)](#) returns incorrect values for multi-byte charsets. (Bug #6399)
- Make auto-deserialization of [java.lang.Objects](#) stored in [BLOB](#) columns configurable via [autoDeserialize](#) property (defaults to [false](#)).
- Re-work [Field.isOpaqueBinary\(\)](#) to detect [CHAR\(n\) CHARACTER SET BINARY](#) to support fixed-length binary fields for [ResultSet.getObject\(\)](#).
- Use our own implementation of buffered input streams to get around blocking behavior of [java.io.BufferedReader](#). Disable this with [useReadAheadInput=false](#).
- Failing to connect to the server when one of the addresses for the given host name is IPV6 (which the server does not yet bind on). The driver now loops through all IP addresses for a given host, and stops on the first one that [accepts\(\)](#) a [socket.connect\(\)](#). (Bug #6348)

Changes in MySQL Connector/J 3.1.4-beta (04 September 2004)

- Connector/J 3.1.3 beta does not handle integers correctly (caused by changes to support unsigned reads in [Buffer.readInt\(\)](#) -> [Buffer.readShort\(\)](#)). (Bug #4510)
- Added support in [DatabaseMetaData.getTables\(\)](#) and [getTableTypes\(\)](#) for views, which are now available in MySQL server 5.0.x.
- [ServerPreparedStatement.execute*\(\)](#) sometimes threw [ArrayIndexOutOfBoundsException](#) when unpacking field metadata. (Bug #4642)
- Optimized integer number parsing, enable 「old」 slower integer parsing using JDK classes via [useFastIntParsing=false](#) property.
- Added [useOnlyServerErrorMessages](#) property, which causes message text in exceptions generated by the server to only contain the text sent by the server (as opposed to the [SQLState](#)'s 「standard」 description, followed by the server's error message). This property is set to [true](#) by default.
- [ResultSet.wasNull\(\)](#) does not work for primitives if a previous [null](#) was returned. (Bug #4689)
- Track packet sequence numbers if [enablePacketDebug=true](#), and throw an exception if packets received out-of-order.

- [ResultSet.getObject\(\)](#) returns wrong type for strings when using prepared statements. (Bug #4482)
- Calling [MysqlPooledConnection.close\(\)](#) twice (even though an application error), caused NPE. Fixed.
- [ServerPreparedStatements](#) dealing with return of [DECIMAL](#) type don't work. (Bug #5012)
- [ResultSet.getObject\(\)](#) doesn't return type [Boolean](#) for pseudo-bit types from prepared statements on 4.1.x (shortcut for avoiding extra type conversion when using binary-encoded result sets obscured test in [getObject\(\)](#) for 「pseudo」 bit type). (Bug #5032)
- You can now use URLs in [LOAD DATA LOCAL INFILE](#) statements, and the driver will use Java's built-in handlers for retrieving the data and sending it to the server. This feature is not enabled by default, you must set the [allowUrlInLocalInfile](#) connection property to [true](#).
- The driver is more strict about truncation of numerics on [ResultSet.get*\(\)](#), and will throw an [SQLException](#) when truncation is detected. You can disable this by setting [jdbcCompliantTruncation](#) to [false](#) (it is enabled by default, as this functionality is required for JDBC compliance).
- Added three ways to deal with all-zero datetimes when reading them from a [ResultSet](#): [exception](#) (the default), which throws an [SQLException](#) with an SQLState of [S1009](#); [convertToNull](#), which returns [NULL](#) instead of the date; and [round](#), which rounds the date to the nearest closest value which is '0001-01-01'.
- Fixed [ServerPreparedStatement](#) to read prepared statement metadata off the wire, even though it's currently a placeholder instead of using [MysqlIO.clearInputStream\(\)](#) which didn't work at various times because data wasn't available to read from the server yet. This fixes sporadic errors users were having with [ServerPreparedStatements](#) throwing [ArrayIndexOutOfBoundsExceptions](#).
- Use [com.mysql.jdbc.Message](#)'s classloader when loading resource bundle, should fix sporadic issues when the caller's classloader can't locate the resource bundle.

Changes in MySQL Connector/J 3.1.3-beta (07 July 2004)

- Mangle output parameter names for [CallableStatements](#) so they will not clash with user variable names.
- Added support for [INOUT](#) parameters in [CallableStatements](#).
- Null bitmask sent for server-side prepared statements was incorrect. (Bug #4119)
- Use SQL Standard SQL states by default, unless [useSqlStateCodes](#) property is set to [false](#).
- Added packet debugging code (see the [enablePacketDebug](#) property documentation).
- Added constants for MySQL error numbers (publicly accessible, see [com.mysql.jdbc.MySQLErrorNumbers](#)), and the ability to generate the mappings of vendor error codes to SQLStates that the driver uses (for documentation purposes).
- Externalized more messages (on-going effort).
- Error in retrieval of [mediumint](#) column with prepared statements and binary protocol. (Bug #4311)
- Support new time zone variables in MySQL-4.1.3 when [useTimezone=true](#).
- Support for unsigned numerics as return types from prepared statements. This also causes a change in [ResultSet.getObject\(\)](#) for the [bigint unsigned](#) type, which used to return [BigDecimal](#) instances, it now returns instances of [java.lang.BigInteger](#).

Changes in MySQL Connector/J 3.1.2-alpha (09 June 2004)

- Fixed stored procedure parameter parsing info when size was specified for a parameter (for example, [char\(\)](#), [varchar\(\)](#)).
- Enabled callable statement caching via [cacheCallableStmts](#) property.
- Fixed case when no output parameters specified for a stored procedure caused a bogus query to be issued to retrieve out parameters, leading to a syntax error from the server.

- Fixed case when no parameters could cause a [NullPointerException](#) in [CallableStatement.setOutputParameters\(\)](#).
- Removed wrapping of exceptions in [MysqlIO.changeUser\(\)](#).
- Fixed sending of split packets for large queries, enabled nio ability to send large packets as well.
- Added [.toString\(\)](#) functionality to [ServerPreparedStatement](#), which should help if you're trying to debug a query that is a prepared statement (it shows SQL as the server would process).
- Added [gatherPerformanceMetrics](#) property, along with properties to control when/where this info gets logged (see docs for more info).
- [ServerPreparedStatements](#) weren't actually de-allocating server-side resources when [.close\(\)](#) was called.
- Added [logSlowQueries](#) property, along with [slowQueriesThresholdMillis](#) property to control when a query should be considered 「slow.」
- Correctly map output parameters to position given in [prepareCall\(\)](#) versus. order implied during [registerOutParameter\(\)](#). (Bug #3146)
- Correctly detect initial character set for servers $\geq 4.1.0$.
- Cleaned up detection of server properties.
- Support placeholder for parameter metadata for server $\geq 4.1.2$.
- [getProcedures\(\)](#) does not return any procedures in result set. (Bug #3539)
- [getProcedureColumns\(\)](#) doesn't work with wildcards for procedure name. (Bug #3540)
- [DBMD.getSQLStateType\(\)](#) returns incorrect value. (Bug #3520)
- Added [connectionCollation](#) property to cause driver to issue [set collation_connection=...](#) query on connection init if default collation for given charset is not appropriate.
- Fixed [DatabaseMetaData.getProcedures\(\)](#) when run on MySQL-5.0.0 (output of [SHOW PROCEDURE STATUS](#) changed between 5.0.0 and 5.0.1).
- [getWarnings\(\)](#) returns [SQLWarning](#) instead of [DataTruncation](#). (Bug #3804)
- Don't enable server-side prepared statements for server version 5.0.0 or 5.0.1, as they aren't compatible with the '4.1.2+' style that the driver uses (the driver expects information to come back that isn't there, so it hangs).

Changes in MySQL Connector/J 3.1.1-alpha (14 February 2004)

- Fixed bug with [UpdatableResultSets](#) not using client-side prepared statements.
- Fixed character encoding issues when converting bytes to ASCII when MySQL doesn't provide the character set, and the JVM is set to a multi-byte encoding (usually affecting retrieval of numeric values).
- Unpack 「unknown」 data types from server prepared statements as [Strings](#).
- Implemented long data (Blobs, Clobs, InputStreams, Readers) for server prepared statements.
- Implemented [Statement.getWarnings\(\)](#) for MySQL-4.1 and newer (using [SHOW WARNINGS](#)).
- Default result set type changed to [TYPE_FORWARD_ONLY](#) (JDBC compliance).
- Centralized setting of result set type and concurrency.
- Refactored how connection properties are set and exposed as [DriverPropertyInfo](#) as well as [Connection](#) and [DataSource](#) properties.
- Support for NIO. Use [useNIO=true](#) on platforms that support NIO.
- Support for transaction savepoints (MySQL $\geq 4.0.14$ or 4.1.1).

- Support for [mysql_change_user\(\)](#). See the [changeUser\(\)](#) method in [com.mysql.jdbc.Connection](#).
- Reduced number of methods called in average query to be more efficient.
- Prepared [Statements](#) will be re-prepared on auto-reconnect. Any errors encountered are postponed until first attempt to re-execute the re-prepared statement.
- Ensure that warnings are cleared before executing queries on prepared statements, as-per JDBC spec (now that we support warnings).
- Support 「old」 [profileSql](#) capitalization in [ConnectionProperties](#). This property is deprecated, you should use [profileSQL](#) if possible.
- Optimized [Buffer.readLenByteArray\(\)](#) to return shared empty byte array when length is 0.
- Allow contents of [PreparedStatement.setBlob\(\)](#) to be retained between calls to [.execute*\(\)](#).
- Deal with 0-length tokens in [EscapeProcessor](#) (caused by callable statement escape syntax).
- Check for closed connection on delete/update/insert row operations in [UpdatableResultSet](#).
- Fix support for table aliases when checking for all primary keys in [UpdatableResultSet](#).
- Removed [useFastDates](#) connection property.
- Correctly initialize datasource properties from JNDI Refs, including explicitly specified URLs.
- [DatabaseMetaData](#) now reports [supportsStoredProcedures\(\)](#) for MySQL versions $\geq 5.0.0$
- Fixed stack overflow in [Connection.prepareCall\(\)](#) (bad merge).
- Fixed [IllegalAccessError](#) to [Calendar.getTimeInMillis\(\)](#) in [DateTimeValue](#) (for JDK < 1.4).
- [DatabaseMetaData.getColumns\(\)](#) is not returning correct column ordinal info for non-'%' column name patterns. (Bug #1673)
- Merged fix of datatype mapping from MySQL type [FLOAT](#) to [java.sql.Types.REAL](#) from 3.0 branch.
- Detect collation of column for [RSMD.isCaseSensitive\(\)](#).
- Fixed sending of queries larger than 16M.
- Added named and indexed input/output parameter support to [CallableStatement](#). MySQL-5.0.x or newer.
- Fixed [NullPointerException](#) in [ServerPreparedStatement.setTimestamp\(\)](#), as well as year and month discrepancies in [ServerPreparedStatement.setTimestamp\(\)](#), [setDate\(\)](#).
- Added ability to have multiple database/JVM targets for compliance and regression/unit tests in [build.xml](#).
- Fixed NPE and year/month bad conversions when accessing some datetime functionality in [ServerPreparedStatements](#) and their resultant result sets.
- Display where/why a connection was implicitly closed (to aid debugging).
- [CommunicationsException](#) implemented, that tries to determine why communications was lost with a server, and displays possible reasons when [.getMessage\(\)](#) is called.
- [NULL](#) values for numeric types in binary encoded result sets causing [NullPointerExceptions](#). (Bug #2359)
- Implemented [Connection.prepareCall\(\)](#), and [DatabaseMetaData.getProcedures\(\)](#) and [getProcedureColumns\(\)](#).
- Reset [long binary](#) parameters in [ServerPreparedStatement](#) when [clearParameters\(\)](#) is called, by sending [COM_RESET_STMT](#) to the server.
- Merged prepared statement caching, and [.getMetaData\(\)](#) support from 3.0 branch.

- Fixed off-by-1900 error in some cases for years in `TimeUtil.fastDate/TimeCreate()` when unpacking results from server-side prepared statements.
- Fixed charset conversion issue in `getTables()`. (Bug #2502)
- Implemented multiple result sets returned from a statement or stored procedure.
- Server-side prepared statements were not returning datatype `YEAR` correctly. (Bug #2606)
- Enabled streaming of result sets from server-side prepared statements.
- Class-cast exception when using scrolling result sets and server-side prepared statements. (Bug #2623)
- Merged unbuffered input code from 3.0.
- Fixed `ConnectionProperties` that weren't properly exposed via accessors, cleaned up `ConnectionProperties` code.
- `NULL` fields were not being encoded correctly in all cases in server-side prepared statements. (Bug #2671)
- Fixed rare buffer underflow when writing numbers into buffers for sending prepared statement execution requests.
- Use DocBook version of docs for shipped versions of drivers.

Changes in MySQL Connector/J 3.1.0-alpha (18 February 2003)

- Added `requireSSL` property.
- Added `useServerPrepStmts` property (default `false`). The driver will use server-side prepared statements when the server version supports them (4.1 and newer) when this property is set to `true`. It is currently set to `false` by default until all bind/fetch functionality has been implemented. Currently only DML prepared statements are implemented for 4.1 server-side prepared statements.
- Track open `Statements`, close all when `Connection.close()` is called (JDBC compliance).

Changes in MySQL Connector/J 3.0.x

Changes in MySQL Connector/J 3.0.17-ga (23 June 2005)

- `Timestamp/Time` conversion goes in the wrong 「direction」 when `useTimeZone=true` and server time zone differs from client time zone. (Bug #5874)
- `DatabaseMetaData.getIndexInfo()` ignored `unique` parameter. (Bug #7081)
- Support new protocol type `MYSQL_TYPE_VARCHAR`.
- Added `useOldUTF8Behavior` configuration property, which causes JDBC driver to act like it did with MySQL-4.0.x and earlier when the character encoding is `utf-8` when connected to MySQL-4.1 or newer.
- Statements created from a pooled connection were returning physical connection instead of logical connection when `getConnection()` was called. (Bug #7316)
- `PreparedStatement` don't encode Big5 (and other multi-byte) character sets correctly in static SQL strings. (Bug #7033)
- Connections starting up failed-over (due to down master) never retry master. (Bug #6966)
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug #7061)
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug #7686)
- Backported `SQLState` codes mapping from Connector/J 3.1, enable with `useSqlStateCodes=true` as a connection property, it defaults to `false` in this release, so that we don't break legacy applications (it defaults to `true` starting with Connector/J 3.1).

- [PreparedStatement.fixDecimalExponent\(\)](#) adding extra `+`, making number unparseable by MySQL server. (Bug #7601)
- Escape sequence `{fn convert(..., type)}` now supports ODBC-style types that are prepended by `SQL_`.
- Fixed duplicated code in [configureClientCharset\(\)](#) that prevented `useOldUTF8Behavior=true` from working properly.
- Handle streaming result sets with more than 2 billion rows properly by fixing wraparound of row number counter.
- [MS932](#), [SHIFT_JIS](#), and [Windows_31J](#) not recognized as aliases for [sjis](#). (Bug #7607)
- Adding [CP943](#) to aliases for [sjis](#). (Bug #6549, fixed while fixing Bug #7607)
- Which requires hex escaping of binary data when using multi-byte charsets with prepared statements. (Bug #8064)
- [NON_UNIQUE](#) column from [DBMD.getIndexInfo\(\)](#) returned inverted value. (Bug #8812)
- Workaround for server Bug #9098: Default values of `CURRENT_*` for [DATE](#), [TIME](#), [DATETIME](#), and [TIMESTAMP](#) columns can't be distinguished from `string` values, so [UpdatableResultSet.moveToInsertRow\(\)](#) generates bad SQL for inserting default values.
- [EUCKR](#) charset is sent as `SET NAMES euc_kr` which MySQL-4.1 and newer doesn't understand. (Bug #8629)
- [DatabaseMetaData.supportsSelectForUpdate\(\)](#) returns correct value based on server version.
- Use hex escapes for [PreparedStatement.setBytes\(\)](#) for double-byte charsets including 「aliases」 [Windows-31J](#), [CP934](#), [MS932](#).
- Added support for the [EUC_JP_Solaris](#) character encoding, which maps to a MySQL encoding of [eucjpm](#)s (backported from 3.1 branch). This only works on servers that support [eucjpm](#)s, namely 5.0.3 or later.

Changes in MySQL Connector/J 3.0.16-ga (15 November 2004)

- Re-issue character set configuration commands when re-using pooled connections and/or [Connection.changeUser\(\)](#) when connected to MySQL-4.1 or newer.
- Fixed [ResultSetMetaData.isReadOnly\(\)](#) to detect non-writable columns when connected to MySQL-4.1 or newer, based on existence of 「original」 table and column names.
- [ResultSet.updateByte\(\)](#) when on insert row throws [ArrayOutOfBoundsException](#). (Bug #5664)
- Fixed [DatabaseMetaData.getTypes\(\)](#) returning incorrect (this is, non-negative) scale for the [NUMERIC](#) type.
- Off-by-one bug in [Buffer.readString\(string\)](#). (Bug #5664)
- Made [TINYINT\(1\)](#) -> [BIT/Boolean](#) conversion configurable via [tinyInt1isBit](#) property (default `true` to be JDBC compliant out of the box).
- Only set [character_set_results](#) during connection establishment if server version `>= 4.1.1`.
- Fixed regression where [useUnbufferedInput](#) was defaulting to `false`.
- [ResultSet.getTimestamp\(\)](#) on a column with [TIME](#) in it fails. (Bug #5664)

Changes in MySQL Connector/J 3.0.15-production (04 September 2004)

- [StringUtils.escapeEasternUnicodeByteStream](#) was still broken for GBK. (Bug #4010)
- Failover for [autoReconnect](#) not using port numbers for any hosts, and not retrying all hosts. (Warning: This required a change to the [SocketFactory.connect\(\)](#) method signature, which is now `public Socket connect(String host, int portNumber, Properties props)`; therefore, any third-party socket factories will have to be changed to support this signature. (Bug #4334)
- Logical connections created by [MysqlConnectionPoolDataSource](#) will now issue a [rollback\(\)](#) when they are closed and sent back to the pool. If your application server/connection pool already does this for you, you can set the [rollbackOnPooledClose](#) property to `false` to avoid the overhead of an extra [rollback\(\)](#).

- Removed redundant calls to [checkRowPos\(\)](#) in [ResultSet](#).
- [DOUBLE](#) mapped twice in [DBMD.getTypeInfo\(\)](#). (Bug #4742)
- Added FLOSS license exemption.
- Calling [.close\(\)](#) twice on a [PooledConnection](#) causes NPE. (Bug #4808)
- [DBMD.getColumns\(\)](#) returns incorrect JDBC type for unsigned columns. This affects type mappings for all numeric types in the [RSMD.getColumnType\(\)](#) and [RSMD.getColumnTypeNames\(\)](#) methods as well, to ensure that 「like」 types from [DBMD.getColumns\(\)](#) match up with what [RSMD.getColumnType\(\)](#) and [getColumnTypeNames\(\)](#) return. (Bug #4138, Bug #4860)
- 「Production」 is now 「GA」 (General Availability) in naming scheme of distributions.
- [RSMD.getPrecision\(\)](#) returning 0 for non-numeric types (should return max length in chars for non-binary types, max length in bytes for binary types). This fix also fixes mapping of [RSMD.getColumnType\(\)](#) and [RSMD.getColumnTypeName\(\)](#) for the [BLOB](#) types based on the length sent from the server (the server doesn't distinguish between [TINYBLOB](#), [BLOB](#), [MEDIUMBLOB](#) or [LONGBLOB](#) at the network protocol level). (Bug #4880)
- [ResultSet](#) should release [Field\[\]](#) instance in [.close\(\)](#). (Bug #5022)
- [ResultSet.getMetaData\(\)](#) should not return incorrectly initialized metadata if the result set has been closed, but should instead throw an [SQLException](#). Also fixed for [getRow\(\)](#) and [getWarnings\(\)](#) and traversal methods by calling [checkClosed\(\)](#) before operating on instance-level fields that are nullified during [.close\(\)](#). (Bug #5069)
- Parse new time zone variables from 4.1.x servers.
- Use [_binary](#) introducer for [PreparedStatement.setBytes\(\)](#) and [set*Stream\(\)](#) when connected to MySQL-4.1.x or newer to avoid misinterpretation during character conversion.

Changes in MySQL Connector/J 3.0.14-production (28 May 2004)

- Fixed URL parsing error.

Changes in MySQL Connector/J 3.0.13-production (27 May 2004)

- Using a [MySQLDataSource](#) without server name fails. (Bug #3848)
- [No Database Selected](#) when using [MysqlConnectionPoolDataSource](#). (Bug #3920)
- [PreparedStatement.getGeneratedKeys\(\)](#) method returns only 1 result for batched insertions. (Bug #3873)

Changes in MySQL Connector/J 3.0.12-production (18 May 2004)

- Add unsigned attribute to [DatabaseMetaData.getColumns\(\)](#) output in the [TYPE_NAME](#) column.
- Added [failOverReadOnly](#) property, to allow end-user to configure state of connection (read-only/writable) when failed over.
- Backported 「change user」 and 「reset server state」 functionality from 3.1 branch, to allow clients of [MysqlConnectionPoolDataSource](#) to reset server state on [getConnection\(\)](#) on a pooled connection.
- Don't escape SJIS/GBK/BIG5 when using MySQL-4.1 or newer.
- Allow [url](#) parameter for [MysqlDataSource](#) and [MysqlConnectionPool DataSource](#) so that passing of other properties is possible from inside appservers.
- Map duplicate key and foreign key errors to SQLState of [23000](#).
- Backport documentation tooling from 3.1 branch.
- Return creating statement for [ResultSets](#) created by [getGeneratedKeys\(\)](#). (Bug #2957)

- Allow `java.util.Date` to be sent in as parameter to `PreparedStatement.setObject()`, converting it to a `Timestamp` to maintain full precision. (Bug #103).
- Don't truncate `BLOB` or `CLOB` values when using `setBytes()` and/or `setBinary/CharacterStream()`. (Bug #2670).
- Dynamically configure character set mappings for field-level character sets on MySQL-4.1.0 and newer using `SHOW COLLATION` when connecting.
- Map `binary` character set to `US-ASCII` to support `DATETIME` charset recognition for servers $\geq 4.1.2$.
- Use `SET character_set_results` during initialization to allow any charset to be returned to the driver for result sets.
- Use `charsetnr` returned during connect to encode queries before issuing `SET NAMES` on MySQL $\geq 4.1.0$.
- Add helper methods to `ResultSetMetaData` (`getColumnCharacterEncoding()` and `getColumnCharacterSet()`) to allow end-users to see what charset the driver thinks it should be using for the column.
- Only set `character_set_results` for MySQL $\geq 4.1.0$.
- `StringUtils.escapeSJISByteStream()` not covering all eastern double-byte charsets correctly. (Bug #3511)
- Renamed `StringUtils.escapeSJISByteStream()` to more appropriate `escapeEasternUnicodeByteStream()`.
- Not specifying database in URL caused `MalformedURLException` exception. (Bug #3554)
- Auto-convert MySQL encoding names to Java encoding names if used for `characterEncoding` property.
- Added encoding names that are recognized on some JVMs to fix case where they were reverse-mapped to MySQL encoding names incorrectly.
- Use `junit.textui.TestRunner` for all unit tests (to allow them to be run from the command line outside of Ant or Eclipse).
- `UpdatableResultSet` not picking up default values for `moveToInsertRow()`. (Bug #3557)
- Inconsistent reporting of data type. The server still doesn't return all types for `*BLOBs` `*TEXT` correctly, so the driver won't return those correctly. (Bug #3570)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug #3520)
- Fixed regression in `PreparedStatement.setString()` and eastern character encodings.
- Made `StringRegressionTest` 4.1-unicode aware.

Changes in MySQL Connector/J 3.0.11-stable (19 February 2004)

- Trigger a `SET NAMES utf8` when encoding is forced to `utf8` or `utf-8` via the `characterEncoding` property. Previously, only the Java-style encoding name of `utf-8` would trigger this.
- `AutoReconnect` time was growing faster than exponentially. (Bug #2447)
- Fixed failover always going to last host in list. (Bug #2578)
- Added `useUnbufferedInput` parameter, and now use it by default (due to JVM issue <http://developer.java.sun.com/developer/bugParade/bugs/4401235.html>)
- Detect `on/off` or `1, 2, 3` form of `lower_case_table_names` value on server.
- Return `java.lang.Integer` for `TINYINT` and `SMALLINT` types from `ResultSetMetaData.getColumnClassName()`. (Bug #2852)
- Return `java.lang.Double` for `FLOAT` type from `ResultSetMetaData.getColumnClassName()`. (Bug #2855)
- Return `[B` instead of `java.lang.Object` for `BINARY`, `VARBINARY` and `LONGVARBINARY` types from `ResultSetMetaData.getColumnClassName()` (JDBC compliance).

- Issue connection events on all instances created from a [ConnectionPoolDataSource](#).

Changes in MySQL Connector/J 3.0.10-stable (13 January 2004)

- Don't count quoted IDs when inside a 'string' in [PreparedStatement](#) parsing. (Bug #1511)
- 「Friendlier」 exception message for [PacketTooLargeException](#). (Bug #1534)
- Backported fix for aliased tables and [UpdatableResultSets](#) in [checkUpdatability\(\)](#) method from 3.1 branch.
- Fix for [ArrayIndexOutOfBoundsException](#) exception when using [Statement.setMaxRows\(\)](#). (Bug #1695)
- Barge blobs and split packets not being read correctly. (Bug #1576)
- Fixed regression of [Statement.getGeneratedKeys\(\)](#) and [REPLACE](#) statements.
- Subsequent call to [ResultSet.updateFoo\(\)](#) causes NPE if result set is not updatable. (Bug #1630)
- Fix for 4.1.1-style authentication with no password.
- Foreign Keys column sequence is not consistent in [DatabaseMetaData.getImported/Exported/CrossReference\(\)](#). (Bug #1731)
- [DatabaseMetaData.getSystemFunction\(\)](#) returning bad function [VResultsSion](#). (Bug #1775)
- Cross-database updatable result sets are not checked for updatability correctly. (Bug #1592)
- [DatabaseMetaData.getColumns\(\)](#) should return [Types.LONGVARCHAR](#) for MySQL [LONGTEXT](#) type.
- [ResultSet.getObject\(\)](#) on [TINYINT](#) and [SMALLINT](#) columns should return Java type [Integer](#). (Bug #1913)
- Added [alwaysClearStream](#) connection property, which causes the driver to always empty any remaining data on the input stream before each query.
- Added more descriptive error message [Server Configuration Denies Access to DataSource](#), as well as retrieval of message from server.
- Autoreconnect code didn't set catalog upon reconnect if it had been changed.
- Implement [ResultSet.updateClob\(\)](#).
- [ResultSetMetaData.isCaseSensitive\(\)](#) returned wrong value for [CHAR/VARCHAR](#) columns.
- Connection property [maxRows](#) not honored. (Bug #1933)
- Statements being created too many times in [DBMD.extractForeignKeyFromCreateTable\(\)](#). (Bug #1925)
- Support escape sequence {fn convert ... }. (Bug #1914)
- [ArrayIndexOutOfBoundsException](#) when parameter number == number of parameters + 1. (Bug #1958)
- [ResultSet.findColumn\(\)](#) should use first matching column name when there are duplicate column names in [SELECT](#) query (JDBC-compliance). (Bug #2006)
- Removed static synchronization bottleneck from [PreparedStatement.setTimestamp\(\)](#).
- Removed static synchronization bottleneck from instance factory method of [SingleByteCharsetConverter](#).
- Enable caching of the parsing stage of prepared statements via the [cachePrepStmts](#), [prepStmtCacheSize](#), and [prepStmtCacheSqlLimit](#) properties (disabled by default).
- Speed up parsing of [PreparedStatements](#), try to use one-pass whenever possible.
- Fixed security exception when used in Applets (applets can't read the system property [file.encoding](#) which is needed for [LOAD DATA LOCAL INFILE](#)).

- Use constants for SQLStates.
- Map charset `ko18_ru` to `ko18r` when connected to MySQL-4.1.0 or newer.
- Ensure that `Buffer.writeString()` saves room for the `\0`.
- Fixed exception `Unknown character set 'danish'` on connect with JDK-1.4.0
- Fixed mappings in `SQLException` to report deadlocks with SQLStates of `41000`.
- `maxRows` property would affect internal statements, so check it for all statement creation internal to the driver, and set to 0 when it is not.

Changes in MySQL Connector/J 3.0.9-stable (07 October 2003)

- Faster date handling code in `ResultSet` and `PreparedStatement` (no longer uses `Date` methods that synchronize on static calendars).
- Fixed test for end of buffer in `Buffer.readString()`.
- Fixed `ResultSet.previous()` behavior to move current position to before result set when on first row of result set. (Bug #496)
- Fixed `Statement` and `PreparedStatement` issuing bogus queries when `setMaxRows()` had been used and a `LIMIT` clause was present in the query.
- `refreshRow` didn't work when primary key values contained values that needed to be escaped (they ended up being doubly escaped). (Bug #661)
- Support `InnoDB` constraint names when extracting foreign key information in `DatabaseMetaData` (implementing ideas from Parwinder Sekhon). (Bug #517, Bug #664)
- Backported 4.1 protocol changes from 3.1 branch (server-side SQL states, new field information, larger client capability flags, connect-with-database, and so forth).
- Fix `UpdatableResultSet` to return values for `getXXX()` when on insert row. (Bug #675)
- The `insertRow` in an `UpdatableResultSet` is now loaded with the default column values when `moveToInsertRow()` is called. (Bug #688)
- `DatabaseMetaData.getColumns()` wasn't returning `NULL` for default values that are specified as `NULL`.
- Change default statement type/concurrency to `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY` (spec compliance).
- Don't try and reset isolation level on reconnect if MySQL doesn't support them.
- Don't wrap `SQLExceptions` in `RowDataDynamic`.
- Don't change timestamp TZ twice if `useTimezone==true`. (Bug #774)
- Fixed regression in large split-packet handling. (Bug #848)
- Better diagnostic error messages in exceptions for 「streaming」 result sets.
- Issue exception on `ResultSet.getXXX()` on empty result set (wasn't caught in some cases).
- Don't hide messages from exceptions thrown in I/O layers.
- Don't fire connection closed events when closing pooled connections, or on `PooledConnection.getConnection()` with already open connections. (Bug #884)
- Clip +/- INF (to smallest and largest representative values for the type in MySQL) and NaN (to 0) for `setDouble/setFloat()`, and issue a warning on the statement when the server does not support +/- INF or NaN.
- Double-escaping of `'\'` when charset is SJIS or GBK and `'\'` appears in non-escaped input. (Bug #879)

- When emptying input stream of unused rows for 「streaming」 result sets, have the current thread [yield\(\)](#) every 100 rows in order to not monopolize CPU time.
- [DatabaseMetaData.getColumns\(\)](#) getting confused about the keyword 「set」 in character columns. (Bug #1099)
- Fixed deadlock issue with [Statement.setMaxRows\(\)](#).
- Fixed [CLOB.truncate\(\)](#). (Bug #1130)
- Optimized [CLOB.setChracterStream\(\)](#). (Bug #1131)
- Made [databaseName](#), [portNumber](#), and [serverName](#) optional parameters for [MysqlDataSourceFactory](#). (Bug #1246)
- [ResultSet.get/setString](#) mashing char 127. (Bug #1247)
- Backported authentication changes for 4.1.1 and newer from 3.1 branch.
- Added [com.mysql.jdbc.util.BaseBugReport](#) to help creation of testcases for bug reports.
- Added property to 「clobber」 streaming results, by setting the [clobberStreamingResults](#) property to [true](#) (the default is [false](#)). This will cause a 「streaming」 [ResultSet](#) to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.

Changes in MySQL Connector/J 3.0.8-stable (23 May 2003)

- Allow bogus URLs in [Driver.getPropertyInfo\(\)](#).
- Return list of generated keys when using multi-value [INSERTS](#) with [Statement.getGeneratedKeys\(\)](#).
- Use JVM charset with filenames and [LOAD DATA \[LOCAL\] INFILE](#).
- Fix infinite loop with [Connection.cleanup\(\)](#).
- Changed Ant target [compile-core](#) to [compile-driver](#), and made testsuite compilation a separate target.
- Fixed result set not getting set for [Statement.executeUpdate\(\)](#), which affected [getGeneratedKeys\(\)](#) and [getUpdateCount\(\)](#) in some cases.
- Unicode character 0xFFFF in a string would cause the driver to throw an [ArrayOutOfBoundsException](#). (Bug #378).
- Return correct number of generated keys when using [REPLACE](#) statements.
- Fix problem detecting server character set in some cases.
- Fix row data decoding error when using very large packets.
- Optimized row data decoding.
- Issue exception when operating on an already closed prepared statement.
- Fixed SJIS encoding bug, thanks to Naoto Sato.
- Optimized usage of [EscapeProcessor](#).
- Allow multiple calls to [Statement.close\(\)](#).

Changes in MySQL Connector/J 3.0.7-stable (08 April 2003)

- Fixed [MysqlPooledConnection.close\(\)](#) calling wrong event type.
- Fixed [StringIndexOutOfBoundsException](#) in [PreparedStatement.setClob\(\)](#).
- 4.1 Column Metadata fixes.
- Remove synchronization from [Driver.connect\(\)](#) and [Driver.acceptsUrl\(\)](#).

- [IOExceptions](#) during a transaction now cause the [Connection](#) to be closed.
- Fixed missing conversion for [YEAR](#) type in [ResultSetMetaData.getColumnTypeName\(\)](#).
- Don't pick up indexes that start with [pri](#) as primary keys for [DBMD.getPrimaryKeys\(\)](#).
- Throw [SQLExceptions](#) when trying to do operations on a forcefully closed [Connection](#) (that is, when a communication link failure occurs).
- You can now toggle profiling on/off using [Connection.setProfileSql\(boolean\)](#).
- Fixed charset issues with database metadata (charset was not getting set correctly).
- Updatable [ResultSets](#) can now be created for aliased tables/columns when connected to MySQL-4.1 or newer.
- Fixed [LOAD DATA LOCAL INFILE](#) bug when file > [max_allowed_packet](#).
- Fixed escaping of 0x5c ('\') character for GBK and Big5 charsets.
- Fixed [ResultSet.getTimestamp\(\)](#) when underlying field is of type [DATE](#).
- Ensure that packet size from [alignPacketSize\(\)](#) does not exceed [max_allowed_packet](#) (JVM bug)
- Don't reset [Connection.isReadOnly\(\)](#) when autoReconnecting.

Changes in MySQL Connector/J 3.0.6-stable (18 February 2003)

- Fixed [ResultSetMetaData](#) to return "" when catalog not known. Fixes [NullPointerExceptions](#) with Sun's [CachedRowSet](#).
- Fixed [DBMD.getTypeInfo\(\)](#) and [DBMD.getColumns\(\)](#) returning different value for precision in [TEXT](#) and [BLOB](#) types.
- Allow ignoring of warning for 「non transactional tables」 during rollback (compliance/usability) by setting [ignoreNonTxTables](#) property to [true](#).
- Fixed [SQLExceptions](#) getting swallowed on initial connect.
- Fixed [Statement.setMaxRows\(\)](#) to stop sending [LIMIT](#) type queries when not needed (performance).
- Clean up [Statement](#) query/method mismatch tests (that is, [INSERT](#) not allowed with [.executeQuery\(\)](#)).
- More checks added in [ResultSet](#) traversal method to catch when in closed state.
- Fixed [ResultSetMetaData.isWritable\(\)](#) to return correct value.
- Add 「window」 of different [NULL](#) sorting behavior to [DBMD.nullsAreSortedAtStart](#) (4.0.2 to 4.0.10, true; otherwise, no).
- Implemented [Blob.setBytes\(\)](#). You still need to pass the resultant [Blob](#) back into an updatable [ResultSet](#) or [PreparedStatement](#) to persist the changes, because MySQL does not support 「locators」.
- Backported 4.1 charset field info changes from Connector/J 3.1.

Changes in MySQL Connector/J 3.0.5-gamma (22 January 2003)

- Fixed [Buffer.fastSkipLenString\(\)](#) causing [ArrayIndexOutOfBoundsException](#) exceptions with some queries when unpacking fields.
- Implemented an empty [TypeMap](#) for [Connection.getTypeMap\(\)](#) so that some third-party apps work with MySQL (IBM WebSphere 5.0 Connection pool).
- Added missing [LONGTEXT](#) type to [DBMD.getColumns\(\)](#).
- Retrieve [TX_ISOLATION](#) from database for [Connection.getTransactionIsolation\(\)](#) when the MySQL version supports it, instead of an instance variable.

- Quote table names in [DatabaseMetaData.getColumns\(\)](#), [getPrimaryKeys\(\)](#), [getIndexInfo\(\)](#), [getBestRowIdentifier\(\)](#).
- Greatly reduce memory required for [setBinaryStream\(\)](#) in [PreparedStatement](#)s.
- Fixed [ResultSet.isBeforeFirst\(\)](#) for empty result sets.
- Added update options for foreign key metadata.

Changes in MySQL Connector/J 3.0.4-gamma (06 January 2003)

- Added quoted identifiers to database names for [Connection.setCatalog](#).
- Added support for quoted identifiers in [PreparedStatement](#) parser.
- Streamlined character conversion and [byte\[\]](#) handling in [PreparedStatement](#)s for [setByte\(\)](#).
- Reduce memory footprint of [PreparedStatement](#)s by sharing outbound packet with [MySQLIO](#).
- Added [strictUpdates](#) property to allow control of amount of checking for 「correctness」 of updatable result sets. Set this to [false](#) if you want faster updatable result sets and you know that you create them from [SELECT](#) statements on tables with primary keys and that you have selected all primary keys in your query.
- Added support for 4.0.8-style large packets.
- Fixed [PreparedStatement.executeBatch\(\)](#) parameter overwriting.

Changes in MySQL Connector/J 3.0.3-dev (17 December 2002)

- Changed [charsToByte](#) in [SingleByteCharConverter](#) to be non-static.
- Changed [SingleByteCharConverter](#) to use lazy initialization of each converter.
- Fixed charset handling in [Fields.java](#).
- Implemented [Connection.nativeSQL\(\)](#).
- More robust escape tokenizer: Recognize `--` comments, and allow nested escape sequences (see [testsuite.EscapeProcessingTest](#)).
- [DBMD.getImported/ExportedKeys\(\)](#) now handles multiple foreign keys per table.
- Fixed [ResultSetMetaData.getPrecision\(\)](#) returning incorrect values for some floating-point types.
- Fixed [ResultSetMetaData.getColumnTypeName\(\)](#) returning [BLOB](#) for [TEXT](#) and [TEXT](#) for [BLOB](#) types.
- Fixed [Buffer.isLastDataPacket\(\)](#) for 4.1 and newer servers.
- Added [CLIENT_LONG_FLAG](#) to be able to get more column flags ([isAutoIncrement\(\)](#) being the most important).
- Because of above, implemented [ResultSetMetaData.isAutoIncrement\(\)](#) to use [Field.isAutoIncrement\(\)](#).
- Honor [lower_case_table_names](#) when enabled in the server when doing table name comparisons in [DatabaseMetaData](#) methods.
- Some MySQL-4.1 protocol support (extended field info from selects).
- Use non-aliased table/column names and database names to fully qualify tables and columns in [UpdatableResultSet](#) (requires MySQL-4.1 or newer).
- Allow user to alter behavior of [Statement/ PreparedStatement.executeBatch\(\)](#) via [continueBatchOnError](#) property (defaults to [true](#)).
- Check for connection closed in more [Connection](#) methods ([createStatement](#), [prepareStatement](#), [setTransactionIsolation](#), [setAutoCommit](#)).
- More robust implementation of updatable result sets. Checks that all primary keys of the table have been selected.

- [LOAD DATA LOCAL INFILE ...](#) now works, if your server is configured to allow it. Can be turned off with the [allowLoadLocalInfile](#) property (see the [README](#)).
- Substitute '?' for unknown character conversions in single-byte character sets instead of '\0'.
- [NamedPipeSocketFactory](#) now works (only intended for Windows), see [README](#) for instructions.

Changes in MySQL Connector/J 3.0.2-dev (08 November 2002)

- Fixed issue with updatable result sets and [PreparedStatements](#) not working.
- Fixed [ResultSet.setFetchDirection\(FETCH_UNKNOWN\)](#).
- Fixed issue when calling [Statement.setFetchSize\(\)](#) when using arbitrary values.
- Fixed incorrect conversion in [ResultSet.getLong\(\)](#).
- Implemented [ResultSet.updateBlob\(\)](#).
- Removed duplicate code from [UpdatableResultSet](#) (it can be inherited from [ResultSet](#), the extra code for each method to handle updatability I thought might someday be necessary has not been needed).
- Fixed [UnsupportedEncodingException](#) thrown when 「forcing」 a character encoding via properties.
- Fixed various non-ASCII character encoding issues.
- Added driver property [useHostsInPrivileges](#). Defaults to true. Affects whether or not [@hostname](#) will be used in [DBMD.getColumn/TablePrivileges](#).
- All [DBMD](#) result set columns describing schemas now return [NULL](#) to be more compliant with the behavior of other JDBC drivers for other database systems (MySQL does not support schemas).
- Added SSL support. See [README](#) for information on how to use it.
- Properly restore connection properties when autoReconnecting or failing-over, including [autoCommit](#) state, and isolation level.
- Use [SHOW CREATE TABLE](#) when possible for determining foreign key information for [DatabaseMetaData](#). Also allows cascade options for [DELETE](#) information to be returned.
- Escape [0x5c](#) character in strings for the SJIS charset.
- Fixed start position off-by-1 error in [Clob.getSubString\(\)](#).
- Implemented [Clob.truncate\(\)](#).
- Implemented [Clob.setString\(\)](#).
- Implemented [Clob.setAsciiStream\(\)](#).
- Implemented [Clob.setCharacterStream\(\)](#).
- Added [com.mysql.jdbc.MinAdmin](#) class, which allows you to send [shutdown](#) command to MySQL server. This is intended to be used when 「embedding」 Java and MySQL server together in an end-user application.
- Added [connectTimeout](#) parameter that allows users of JDK-1.4 and newer to specify a maximum time to wait to establish a connection.
- Failover and [autoReconnect](#) work only when the connection is in an [autoCommit\(false\)](#) state, in order to stay transaction-safe.
- Added [queriesBeforeRetryMaster](#) property that specifies how many queries to issue when failed over before attempting to reconnect to the master (defaults to 50).
- Fixed [DBMD.supportsResultSetConcurrency\(\)](#) so that it returns true for [ResultSet.TYPE_SCROLL_INSENSITIVE](#) and [ResultSet.CONCUR_READ_ONLY](#) or [ResultSet.CONCUR_UPDATABLE](#).

- Fixed [ResultSet.isLast\(\)](#) for empty result sets (should return [false](#)).
- [PreparedStatement](#) now honors stream lengths in [setBinary/Ascii/Character Stream\(\)](#) unless you set the connection property [useStreamLengthsInPrepStmts](#) to [false](#).
- Removed some not-needed temporary object creation by smarter use of [Strings](#) in [EscapeProcessor](#), [Connection](#) and [DatabaseMetaData](#) classes.

Changes in MySQL Connector/J 3.0.1-dev (21 September 2002)

- Fixed [ResultSet.getRow\(\)](#) off-by-one bug.
- Fixed [RowDataStatic.getAt\(\)](#) off-by-one bug.
- Added limited [Clob](#) functionality ([ResultSet.getClob\(\)](#), [PreparedStatement.setClob\(\)](#), [PreparedStatement.setObject\(Clob\)](#)).
- Added [socketTimeout](#) parameter to URL.
- [Connection.isClosed\(\)](#) no longer 「pings」 the server.
- [Connection.close\(\)](#) issues [rollback\(\)](#) when [getAutoCommit\(\)](#) is [false](#).
- Added [paranoid](#) parameter, which sanitizes error messages by removing 「sensitive」 information from them (such as hostnames, ports, or usernames), as well as clearing 「sensitive」 data structures when possible.
- Fixed [ResultSetMetaData.isSigned\(\)](#) for [TINYINT](#) and [BIGINT](#).
- Charsets now automatically detected. Optimized code for single-byte character set conversion.
- Implemented [ResultSet.getCharacterStream\(\)](#).
- Added [LOCAL TEMPORARY](#) to table types in [DatabaseMetaData.getTableTypes\(\)](#).
- Massive code clean-up to follow Java coding conventions (the time had come).

Changes in MySQL Connector/J 3.0.0-dev (31 July 2002)

- !!! LICENSE CHANGE !!! The driver is now GPL. If you need non-GPL licenses, please contact me [<mark@mysql.com>](mailto:mark@mysql.com).
- JDBC-3.0 functionality including [Statement/PreparedStatement.getGeneratedKeys\(\)](#) and [ResultSet.getURL\(\)](#).
- Performance enhancements: Driver is now 50–100% faster in most situations, and creates fewer temporary objects.
- Repackaging: New driver name is [com.mysql.jdbc.Driver](#), old name still works, though (the driver is now provided by MySQL-AB).
- Better checking for closed connections in [Statement](#) and [PreparedStatement](#).
- Support for streaming (row-by-row) result sets (see [README](#)) Thanks to Doron.
- Support for large packets (new addition to MySQL-4.0 protocol), see [README](#) for more information.
- JDBC Compliance: Passes all tests besides stored procedure tests.
- Fix and sort primary key names in [DBMetaData](#) (SF bugs 582086 and 582086).
- Float types now reported as [java.sql.Types.FLOAT](#) (SF bug 579573).
- [ResultSet.getTimestamp\(\)](#) now works for [DATE](#) types (SF bug 559134).
- [ResultSet.getDate/Time/Timestamp](#) now recognizes all forms of invalid values that have been set to all zeros by MySQL (SF bug 586058).

- Testsuite now uses Junit (which you can get from <http://www.junit.org>).
- The driver now only works with JDK-1.2 or newer.
- Added multi-host failover support (see [README](#)).
- General source-code cleanup.
- Overall speed improvements via controlling transient object creation in [MysqlIO](#) class when reading packets.
- Performance improvements in string handling and field metadata creation (lazily instantiated) contributed by Alex Twisleton-Wykeham-Fiennes.

Changes in MySQL Connector/J 2.0.x

Changes in MySQL Connector/J 2.0.14 (16 May 2002)

- More code cleanup.
- [PreparedStatement](#) now releases resources on [.close\(\)](#). (SF bug 553268)
- Quoted identifiers not used if server version does not support them. Also, if server started with `--ansi` or `--sql-mode=ANSI_QUOTES`, `""` will be used as an identifier quote character, otherwise `''` will be used.
- [ResultSet.getDouble\(\)](#) now uses code built into JDK to be more precise (but slower).
- [LogicalHandle.isClosed\(\)](#) calls through to physical connection.
- Added SQL profiling (to `STDERR`). Set `profileSql=true` in your JDBC URL. See [README](#) for more information.
- Fixed typo for `relaxAutoCommit` parameter.

Changes in MySQL Connector/J 2.0.13 (24 April 2002)

- More code cleanup.
- Fixed unicode chars being read incorrectly. (SF bug 541088)
- Faster blob escaping for [PrepStmt](#).
- Added [set/getPortNumber\(\)](#) to [DataSource\(s\)](#). (SF bug 548167)
- Added [setURL\(\)](#) to [MySQLXADataSource](#). (SF bug 546019)
- [PreparedStatement.toString\(\)](#) fixed. (SF bug 534026)
- [ResultSetMetaData.getColumnClassName\(\)](#) now implemented.
- Rudimentary version of [Statement.getGeneratedKeys\(\)](#) from JDBC-3.0 now implemented (you need to be using JDK-1.4 for this to work, I believe).
- [DBMetaData.getIndexInfo\(\)](#) - bad PAGES fixed. (SF BUG 542201)

Changes in MySQL Connector/J 2.0.12 (07 April 2002)

- General code cleanup.
- Added [getIdleFor\(\)](#) method to [Connection](#) and [MysqlLogicalHandle](#).
- Relaxed synchronization in all classes, should fix 520615 and 520393.
- Added [getTable/ColumnPrivileges\(\)](#) to DBMD (fixes 484502).
- Added new types to [getTypeInfo\(\)](#), fixed existing types thanks to Al Davis and Kid Kalanon.

- Added support for [BIT](#) types (51870) to [PreparedStatement](#).
- Fixed [getRow\(\)](#) bug (527165) in [ResultSet](#).
- Fixes for [ResultSet](#) updatability in [PreparedStatement](#).
- Fixed time zone off-by-1-hour bug in [PreparedStatement](#) (538286, 528785).
- [ResultSet](#): Fixed updatability (values being set to `null` if not updated).
- [DataSources](#) - fixed [setUrl](#) bug (511614, 525565), wrong datasource class name (532816, 528767).
- Added identifier quoting to all [DatabaseMetaData](#) methods that need them (should fix 518108).
- Added support for [YEAR](#) type (533556).
- [ResultSet.insertRow\(\)](#) should now detect `auto_increment` fields in most cases and use that value in the new row. This detection will not work in multi-valued keys, however, due to the fact that the MySQL protocol does not return this information.
- [ResultSet.refreshRow\(\)](#) implemented.
- Fixed [testsuite.Traversal afterLast\(\)](#) bug, thanks to Igor Lastric.

Changes in MySQL Connector/J 2.0.11 (27 January 2002)

- Fixed missing [DELETE_RULE](#) value in [DBMD.getImported/ExportedKeys\(\)](#) and [getCrossReference\(\)](#).
- Full synchronization of [Statement.java](#).
- More changes to fix [Unexpected end of input stream](#) errors when reading [BLOB](#) values. This should be the last fix.

Changes in MySQL Connector/J 2.0.10 (24 January 2002)

- Fixed spurious [Unexpected end of input stream](#) errors in [MysqlIO](#) (bug 507456).
- Fixed null-pointer-exceptions when using [MysqlConnectionPoolDataSource](#) with Websphere 4 (bug 505839).

Changes in MySQL Connector/J 2.0.9 (13 January 2002)

- [Ant](#) build was corrupting included [jar](#) files, fixed (bug 487669).
- Fixed extra memory allocation in [MysqlIO.readPacket\(\)](#) (bug 488663).
- Implementation of [DatabaseMetaData.getExported/ImportedKeys\(\)](#) and [getCrossReference\(\)](#).
- Full synchronization on methods modifying instance and class-shared references, driver should be entirely thread-safe now (please let me know if you have problems).
- [DataSource](#) implementations moved to [org.gjt.mm.mysql.jdbc2.optional](#) package, and (initial) implementations of [PooledConnectionDataSource](#) and [XADataSource](#) are in place (thanks to Todd Wolff for the implementation and testing of [PooledConnectionDataSource](#) with IBM WebSphere 4).
- Added detection of network connection being closed when reading packets (thanks to Todd Lizambri).
- Fixed quoting error with escape processor (bug 486265).
- Report batch update support through [DatabaseMetaData](#) (bug 495101).
- Fixed off-by-one-hour error in [PreparedStatement.setTimestamp\(\)](#) (bug 491577).
- Removed concatenation support from driver (the `||` operator), as older versions of VisualAge seem to be the only thing that use it, and it conflicts with the logical `||` operator. You will need to start [mysqld](#) with the `--ansi` flag to use the `||` operator as concatenation (bug 491680).

- Fixed casting bug in [PreparedStatement](#) (bug 488663).

Changes in MySQL Connector/J 2.0.8 (25 November 2001)

- Batch updates now supported (thanks to some inspiration from Daniel Rall).
- [XADataSource/ConnectionPoolDataSource](#) code (experimental)
- [PreparedStatement.setAnyNumericType\(\)](#) now handles positive exponents correctly (adds + so MySQL can understand it).
- [DatabaseMetaData.getPrimaryKeys\(\)](#) and [getBestRowIdentifier\(\)](#) are now more robust in identifying primary keys (matches regardless of case or abbreviation/full spelling of [Primary Key](#) in [Key_type](#) column).

Changes in MySQL Connector/J 2.0.7 (24 October 2001)

- [PreparedStatement.setCharacterStream\(\)](#) now implemented
- Fixed dangling socket problem when in high availability ([autoReconnect=true](#)) mode, and finalizer for [Connection](#) will close any dangling sockets on GC.
- Fixed [ResultSetMetaData.getPrecision\(\)](#) returning one less than actual on newer versions of MySQL.
- [ResultSet.getBlob\(\)](#) now returns `null` if column value was `null`.
- Character sets read from database if [useUnicode=true](#) and [characterEncoding](#) is not set. (thanks to Dmitry Vereshchagin)
- Initial transaction isolation level read from database (if available). (thanks to Dmitry Vereshchagin)
- Fixed [DatabaseMetaData.supportsTransactions\(\)](#), and [supportsTransactionIsolationLevel\(\)](#) and [getTypeInfo\(\)](#) [SQL_DATETIME_SUB](#) and [SQL_DATA_TYPE](#) fields not being readable.
- Fixed [PreparedStatement](#) generating SQL that would end up with syntax errors for some queries.
- Fixed [ResultSet.isAfterLast\(\)](#) always returning `false`.
- Fixed time zone issue in [PreparedStatement.setTimestamp\(\)](#). (thanks to Erik Olofsson)
- Capitalize type names when [capitalizeTypeNames=true](#) is passed in URL or properties (for WebObjects. (thanks to Anjo Krank)
- Updatable result sets now correctly handle `NULL` values in fields.
- [PreparedStatement.setDouble\(\)](#) now uses full-precision doubles (reverting a fix made earlier to truncate them).
- [PreparedStatement.setBoolean\(\)](#) will use 1/0 for values if your MySQL version is 3.21.23 or higher.

Changes in MySQL Connector/J 2.0.6 (16 June 2001)

- Fixed [PreparedStatement](#) parameter checking.
- Fixed case-sensitive column names in [ResultSet.java](#).

Changes in MySQL Connector/J 2.0.5 (13 June 2001)

- Fixed [ResultSet.getBlob\(\)](#) [ArrayIndex](#) out-of-bounds.
- Fixed [ResultSetMetaData.getColumnTypeName](#) for `TEXT/BLOB`.
- Fixed [ArrayIndexOutOfBounds](#) when sending large `BLOB` queries. (Max size packet was not being set)
- Added `ISOLATION` level support to [Connection.setIsolationLevel\(\)](#)
- Fixed NPE on [PreparedStatement.executeUpdate\(\)](#) when all columns have not been set.

- Fixed data parsing of `TIMESTAMP` values with 2-digit years.
- Added `Byte` to `PreparedStatement.setObject()`.
- `ResultSet.getBoolean()` now recognizes `-1` as `true`.
- `ResultSet` has `+/-Inf/inf` support.
- `ResultSet.insertRow()` works now, even if not all columns are set (they will be set to `NULL`).
- `DataBaseMetaData.getCrossReference()` no longer `ArrayIndexOOB`.
- `getObject()` on `ResultSet` correctly does `TINYINT->Byte` and `SMALLINT->Short`.

Changes in MySQL Connector/J 2.0.3 (03 December 2000)

- Implemented `getBigDecimal()` without scale component for JDBC2.
- Fixed composite key problem with updatable result sets.
- Added detection of `-/+INF` for doubles.
- Faster ASCII string operations.
- Fixed incorrect detection of `MAX_ALLOWED_PACKET`, so sending large blobs should work now.
- Fixed off-by-one error in `java.sql.Blob` implementation code.
- Added `ultraDevHack` URL parameter, set to `true` to allow (broken) Macromedia UltraDev to use the driver.

Changes in MySQL Connector/J 2.0.1 (06 April 2000)

- Fixed `RSMD.isWritable()` returning wrong value. Thanks to Moritz Maass.
- Cleaned up exception handling when driver connects.
- Columns that are of type `TEXT` now return as `Strings` when you use `getObject()`.
- `DatabaseMetaData.getPrimaryKeys()` now works correctly with respect to `key_seq`. Thanks to Brian Slesinsky.
- No escape processing is done on `PreparedStatement` anymore per JDBC spec.
- Fixed many JDBC-2.0 traversal, positioning bugs, especially with respect to empty result sets. Thanks to Ron Smits, Nick Brook, Cessar Garcia and Carlos Martinez.
- Fixed some issues with updatability support in `ResultSet` when using multiple primary keys.

Changes in MySQL Connector/J 2.0.0pre5 (21 February 2000)

- Fixed Bad Handshake problem.

Changes in MySQL Connector/J 2.0.0pre4 (10 January 2000)

- Fixes to `ResultSet` for `insertRow()` - Thanks to Cesar Garcia
- Fix to Driver to recognize JDBC-2.0 by loading a JDBC-2.0 class, instead of relying on JDK version numbers. Thanks to John Baker.
- Fixed `ResultSet` to return correct row numbers
- `Statement.getUpdateCount()` now returns rows matched, instead of rows actually updated, which is more SQL-92 like.

10-29-99

- Statement/PreparedStatement.getMoreResults() bug fixed. Thanks to Noel J. Bergman.
- Added Short as a type to PreparedStatement.setObject(). Thanks to Jeff Crowder
- Driver now automagically configures maximum/preferred packet sizes by querying server.
- Autoreconnect code uses fast ping command if server supports it.
- Fixed various bugs with respect to packet sizing when reading from the server and when alloc'ing to write to the server.

Changes in MySQL Connector/J 2.0.0pre (17 August 1999)

- Now compiles under JDK-1.2. The driver supports both JDK-1.1 and JDK-1.2 at the same time through a core set of classes. The driver will load the appropriate interface classes at runtime by figuring out which JVM version you are using.
- Fixes for result sets with all nulls in the first row. (Pointed out by Tim Endres)
- Fixes to column numbers in SQLExceptions in ResultSet (Thanks to Blas Rodriguez Somoza)
- The database no longer needs to be specified to connect. (Thanks to Christian Motschke)

Changes in MySQL Connector/J 1.2b (04 July 1999)

- Better Documentation (in progress), in doc/mm.doc/book1.html
- DBMD now allows null for a column name pattern (not in spec), which it changes to '%'
- DBMD now has correct types/lengths for getXXX().
- ResultSet.getDate(), getTime(), and getTimestamp() fixes. (contributed by Alan Wilken)
- EscapeProcessor now handles \{ \} and { or } inside quotes correctly. (thanks to Alik for some ideas on how to fix it)
- Fixes to properties handling in Connection. (contributed by Juho Tikkala)
- ResultSet.getObject() now returns null for NULL columns in the table, rather than bombing out. (thanks to Ben Grosman)
- ResultSet.getObject() now returns Strings for types from MySQL that it doesn't know about. (Suggested by Chris Perdue)
- Removed DataInput/Output streams, not needed, 1/2 number of method calls per IO operation.
- Use default character encoding if one is not specified. This is a work-around for broken JVMs, because according to spec, EVERY JVM must support "ISO8859_1", but they don't.
- Fixed Connection to use the platform character encoding instead of "ISO8859_1" if one isn't explicitly set. This fixes problems people were having loading the character- converter classes that didn't always exist (JVM bug). (thanks to Fritz Elfert for pointing out this problem)
- Changed MySQLIO to re-use packets where possible to reduce memory usage.
- Fixed escape-processor bugs pertaining to {} inside quotes.

Changes in MySQL Connector/J 1.2.x and lower

Changes in MySQL Connector/J 1.2a (14 April 1999)

- Fixed character-set support for non-Javasoft JVMs (thanks to many people for pointing it out)
- Fixed ResultSet.getBoolean() to recognize 'y' & 'n' as well as '1' & '0' as boolean flags. (thanks to Tim Pizey)

- Fixed `ResultSet.getTimestamp()` to give better performance. (thanks to Richard Swift)
- Fixed `getBytes()` for numeric types. (thanks to Ray Bellis)
- Fixed `DatabaseMetaData.getTypeInfo()` for DATE type. (thanks to Paul Johnston)
- Fixed `EscapeProcessor` for "fn" calls. (thanks to Piyush Shah at locomotive.org)
- Fixed `EscapeProcessor` to not do extraneous work if there are no escape codes. (thanks to Ryan Gustafson)
- Fixed Driver to parse URLs of the form "jdbc:mysql://host:port" (thanks to Richard Lobb)

Changes in MySQL Connector/J 1.1i (24 March 1999)

- Fixed Timestamps for PreparedStatements
- Fixed null pointer exceptions in RSMD and RS
- Re-compiled with jikes for valid class files (thanks ms!)

Changes in MySQL Connector/J 1.1h (08 March 1999)

- Fixed escape processor to deal with unmatched { and } (thanks to Craig Coles)
- Fixed escape processor to create more portable (between DATETIME and TIMESTAMP types) representations so that it will work with BETWEEN clauses. (thanks to Craig Longman)
- `MySQLIO.quit()` now closes the socket connection. Before, after many failed connections some OS's would run out of file descriptors. (thanks to Michael Brinkman)
- Fixed `NullPointerException` in `Driver.getPropertyInfo`. (thanks to Dave Potts)
- Fixes to `MySQLDefs` to allow all *text fields to be retrieved as Strings. (thanks to Chris at Leverage)
- Fixed `setDouble` in `PreparedStatement` for large numbers to avoid sending scientific notation to the database. (thanks to J.S. Ferguson)
- Fixed `getScale()` and `getPrecision()` in RSMD. (contrib'd by James Klicman)
- Fixed `getObject()` when field was DECIMAL or NUMERIC (thanks to Bert Hobbs)
- `DBMD.getTables()` bombed when passed a null table-name pattern. Fixed. (thanks to Richard Lobb)
- Added check for "client not authorized" errors during connect. (thanks to Hannes Wallnoefer)

Changes in MySQL Connector/J 1.1g (19 February 1999)

- Result set rows are now byte arrays. Blobs and Unicode work bidirectionally now. The `useUnicode` and encoding options are implemented now.
- Fixes to `PreparedStatement` to send binary set by `setXXXStream` to be sent untouched to the MySQL server.
- Fixes to `getDriverPropertyInfo()`.

Changes in MySQL Connector/J 1.1f (31 December 1998)

- Changed all `ResultSet` fields to Strings, this should allow Unicode to work, but your JVM must be able to convert between the character sets. This should also make reading data from the server be a bit quicker, because there is now no conversion from `StringBuffer` to `String`.
- Changed `PreparedStatement.toString()` to be more efficient (code from Uwe Schaefer).
- URL parsing is more robust (throws SQL exceptions on errors rather than `NullPointerExceptions`)
- `PreparedStatement` now can convert Strings to Time/Date values via `setObject()` (code from Robert Currey).

- IO no longer hangs in `Buffer.readInt()`, that bug was introduced in 1.1d when changing to all byte-arrays for result sets. (Pointed out by Samo Login)

Changes in MySQL Connector/J 1.1b (03 November 1998)

- Fixes to `DatabaseMetaData` to allow both IBM VA and J-Builder to work. Let me know how it goes. (thanks to Jac Kersing)
- Fix to `ResultSet.getBoolean()` for NULL strings (thanks to Barry Lagerweij)
- Beginning of code cleanup, and formatting. Getting ready to branch this off to a parallel JDBC-2.0 source tree.
- Added "final" modifier to critical sections in `MysqlIO` and `Buffer` to allow compiler to inline methods for speed.

9-29-98

- If object references passed to `setXXX()` in `PreparedStatement` are null, `setNull()` is automatically called for you. (Thanks for the suggestion goes to Erik Ostrom)
- `setObject()` in `PreparedStatement` will now attempt to write a serialized representation of the object to the database for objects of `Types.OTHER` and objects of unknown type.
- Util now has a static method `readObject()` which given a `ResultSet` and a column index will re-instantiate an object serialized in the above manner.

Changes in MySQL Connector/J 1.1 (02 September 1998)

- Got rid of "ugly hack" in `MysqlIO.nextRow()`. Rather than catch an exception, `Buffer.isLastDataPacket()` was fixed.
- `Connection.getCatalog()` and `Connection.setCatalog()` should work now.
- `Statement.setMaxRows()` works, as well as setting by property `maxRows`. `Statement.setMaxRows()` overrides `maxRows` set via properties or url parameters.
- Automatic re-connection is available. Because it has to "ping" the database before each query, it is turned off by default. To use it, pass in "autoReconnect=true" in the connection URL. You may also change the number of reconnect tries, and the initial timeout value via "maxReconnects=n" (default 3) and "initialTimeout=n" (seconds, default 2) parameters. The timeout is an exponential backoff type of timeout; for example, if you have initial timeout of 2 seconds, and maxReconnects of 3, then the driver will timeout 2 seconds, 4 seconds, then 16 seconds between each re-connection attempt.

Changes in MySQL Connector/J 1.0 (24 August 1998)

- Fixed handling of blob data in `Buffer.java`
- Fixed bug with authentication packet being sized too small.
- The JDBC Driver is now under the LPGL

8-14-98

- Fixed `Buffer.readLenString()` to correctly read data for BLOBS.
- Fixed `PreparedStatement.toStringStream` to correctly read data for BLOBS.
- Fixed `PreparedStatement.setDate()` to not add a day. (above fixes thanks to Vincent Partington)
- Added URL parameter parsing (`?user=...` and so forth).

Changes in MySQL Connector/J 0.9d (04 August 1998)

- Big news! New package name. Tim Endres from ICE Engineering is starting a new source tree for GNU GPL'd Java software. He's graciously given me the `org.gjt.mm` package directory to use, so now the driver is in the `org.gjt.mm.mysql` package scheme. I'm "legal" now. Look for more information on Tim's project soon.

- Now using dynamically sized packets to reduce memory usage when sending commands to the DB.
- Small fixes to `getTypeInfo()` for parameters, and so forth.
- `DatabaseMetaData` is now fully implemented. Let me know if these drivers work with the various IDEs out there. I've heard that they're working with JBuilder right now.
- Added JavaDoc documentation to the package.
- Package now available in .zip or .tar.gz.

Changes in MySQL Connector/J 0.9 (28 July 1998)

- Implemented `getTypeInfo()`. `Connection.rollback()` now throws an `SQLException` per the JDBC spec.
- Added `PreparedStatement` that supports all JDBC API methods for `PreparedStatement` including `InputStreams`. Please check this out and let me know if anything is broken.
- Fixed a bug in `ResultSet` that would break some queries that only returned 1 row.
- Fixed bugs in `DatabaseMetaData.getTables()`, `DatabaseMetaData.getColumns()` and `DatabaseMetaData.getCatalogs()`.
- Added functionality to `Statement` that allows `executeUpdate()` to store values for IDs that are automatically generated for `AUTO_INCREMENT` fields. Basically, after an `executeUpdate()`, look at the `SQLWarnings` for warnings like "LAST_INSERTED_ID = 'some number', COMMAND = 'your SQL query'". If you are using `AUTO_INCREMENT` fields in your tables and are executing a lot of `executeUpdate()`s on one `Statement`, be sure to `clearWarnings()` every so often to save memory.

Changes in MySQL Connector/J 0.8 (06 July 1998)

- Split `MysqlIO` and `Buffer` to separate classes. Some `ClassLoaders` gave an `IllegalAccessException` error for some fields in those two classes. Now `mm.mysql` works in applets and all classloaders. Thanks to Joe Ennis <jce@mail.boone.com> for pointing out the problem and working on a fix with me.

Changes in MySQL Connector/J 0.7 (01 July 1998)

- Fixed `DatabaseMetadadata` problems in `getColumns()` and bug in switch statement in the `Field` constructor. Thanks to Costin Manolache <costin@tdiinc.com> for pointing these out.

Changes in MySQL Connector/J 0.6 (21 May 1998)

- Incorporated efficiency changes from Richard Swift <Richard.Swift@kanatek.ca> in [MysqlIO.java](#) and [ResultSet.java](#):
- We're now 15% faster than gwe's driver.
- Started working on [DatabaseMetaData](#).
- The following methods are implemented:
 - [getTables\(\)](#)
 - [getTableTypes\(\)](#)
 - [getColumns](#)
 - [getCatalogs\(\)](#)

付録D 制限と規制

目次

ストアード ルーチンとトリガの規制	1703
サーバサイドカーソルの規制	1705
サブクエリの規制	1705
ビューの規制	1707
XA トランザクションの規制	1709
MySQL の制限	1709
結合の制限	1709

以下は、サブクエリやビューを含む MySQL 機能の使用に関する制限と規制についての考察です。

ストアード ルーチンとトリガの規制

ここに記載されている規制のうちには、すべてのストアード ルーチン、つまりストアード プロシージャとストアード ファンクションの両方に適用するものがあります。またあるものは、ストアード ファンクションのみにあてはまり、ストアード プロシージャには関係しません。

ストアード ファンクションに適用されるすべての規制は、トリガにもあてはまります。

ストアード ルーチンは、任意の SQL 文を取り込むことはできません。次のステートメントは利用不可です:

- 固定文 [LOCK TABLES](#)、[UNLOCK TABLES](#)。
- [LOAD DATA](#) および [LOAD TABLE](#)。
- SQL プリペアド ステートメント ([PREPARE](#)、[EXECUTE](#)、[DEALLOCATE PREPARE](#))。解説: ストアード ルーチン内 (ステートメントをストリングとして動的に構築し、実行する場) で動的 SQL を使うことはできない。この規制は MySQL 5.0.13 からストアード プロシージャに適用され、引き続きストアード ファンクションおよびトリガにも用いられる。

ストアード ファンクションでは、次の予備のステートメントまたは操作が不許可になっています (ストアード プロシージャには適合しない):

- 明示的もしくは暗黙の遂行、またはロールバックを行うステートメント。
- 結果セットを返すステートメント。これには [INTO var_list](#) 句を持たない [SELECT](#) 文と、[SHOW](#) 文が含まれる。関数は [SELECT ... INTO var_list](#) または、カーソルと [FETCH](#) 文を使って結果セットを処理することができる。[SELECT ... INTO ステートメント](#) 参照。
- [FLUSH](#) ステートメント
- 再帰的ステートメント。ストアード ファンクションを再帰的に使用することはできない。
- ストアード ファンクションまたはトリガ内では、その関数やトリガを実行したステートメントが (読み取り、または書き込み) にすでに使用しているテーブルを改変することはできない。

規制のあるものは通常、ストアード ファンクションとトリガに適用されるがストアード プロシージャには当てはまらないということになっていますが、ストアード ファンクション内またはトリガ内から実行されたストアード プロシージャには、それらの規制が適用されますので注意してください。例えば、[FLUSH](#) をストアード プロシージャで使用することはできませんが、そうしたストアード プロシージャをストアード ファンクションやトリガから呼び出すことができません。

ルーチン パラメータ、ローカル変数、テーブル カラムに同じ識別子を使用することは可能です。また、同じローカル変数名を入れ子になったブロックで使うこともできます。例:

```
CREATE PROCEDURE p (i INT)
BEGIN
```

```
DECLARE i INT DEFAULT 0;
SELECT i FROM t;
BEGIN
  DECLARE i INT DEFAULT 1;
  SELECT i FROM t;
END;
END;
```

その場合、識別子が曖昧になるため、次の優先順位ルールが適用されます：

- ローカル変数は、ルーチン パラメータもしくはテーブル カラムより優先される。
- ルーチン パラメーターをテーブル カラムより優先。
- 内側ブロックのローカル変数を、外側ブロックのローカル変数より優先。

テーブル カラムが変数より優先されないという場合、その挙動は標準のものではありません。

ストアド ルーチンの使用は、反復の問題の原因になる場合があります。この問題については [ストアドルーチンとトリガのバイナリログ](#) で詳しく述べられています。

INFORMATION_SCHEMA スキーマは **PARAMETERS** テーブルをまだ持っていないので、ランタイムでルーチン パラメータ情報を得る必要のあるアプリケーションは、**SHOW CREATE** 文のアウトプットを構文解析するなどの回避策を取らなければなりません。

ストアド ルーチンのデバッグ機能は存在しません。

CALL 文を準備することはできません。これはサーバ側のプリペアド ステートメントでも、SQL プリペアド ステートメントでも同じです。

UNDO ハンドラはサポートされていません。

FOR ループはサポートされていません。

サーバ スレッド間の作業の問題を防ぐため、クライアントがステートメントを発行する際、サーバはステートメントの実行に利用できるルーチンとトリガのスナップショットを使用します。これは、ステートメントの実行の間に使用される可能性のあるプロシージャ、関数、トリガのリストを算出してロードし、そしてステートメントの実行に進むということです。これにより、ステートメントが実行される間、他のスレッドが実行するルーチンに変化がないということになります。

RENAME DATABASE 文が、ストアド ルーチンを新たなスキーマ名に変えることはありません。 **RENAME DATABASE** [構文](#) 参照。

トリガには、次の予備のステートメントまたは操作が利用不可になっています：

- トリガは現在、外部キーのアクションでは起動されない。
- **RETURN** 文は、値を返すことのできないトリガでは利用不可。すぐさまトリガから出るには、**LEAVE** 文を使用する。
- **mysql** データベースのテーブルではトリガを使用できない。

MySQL Cluster のストアド ルーチンおよびトリガ。 スストアド ファンクション、ストアド プロシージャ、およびトリガはすべて、**NDB** 保存エンジンを使用したテーブルにサポートされていますが、それらは MySQL サーバ間に Cluster SQL ノードとして自動的に伝播しないということを忘れないでください。その原因は次になります：

- スストアド ルーチンの定義は、**MyISAM** 保存エンジンを使用して、**mysql** システム データベース内のテーブルに保管されます。
- トリガの定義を含む **.TRN** および **.TRG** ファイルを、**NDB** 保存エンジンが読み取ることはなく、また Cluster ノード間でのコピーもされません。

MySQL Cluster テーブルに作用するすべてのストアド ルーチンまたはトリガは、そのストアド ルーチンまたはトリガを使用したいクラスタに参与する各 MySQL サーバの、適切な **CREATE PROCEDURE**、**CREATE FUNCTION**、ま

たは `CREATE TRIGGER` 文を実行することによって再作成されなければなりません。同様に、既存のストアド ルーチンまたはトリガのいかなる変更も、クラスタにアクセスする各 MySQL Server の適切な `ALTER` または `DROP` 文を使用して、すべての Cluster SQL ノードで明示的に実行される必要があります。

警告

NDB 保存エンジンを使用するために、`mysql` データベース テーブルを変換して、上の最初の項で説明のあった問題を回避しようと試みるのはやめてください。`mysql` データベースのシステム テーブルを変更すると希望しない結果を引き起こす可能性が高く、MySQL AB では支持していません。

サーバサイドカーソルの規制

サーバサイドカーソルは、`mysql_stmt_attr_set()` 関数を介して C API に実装されます。ストアド ルーチンのカーソルにも同じ実装が使用されます。サーバサイドカーソルによって、結果セットをサーバ側で生成することが可能になりますが、クライアントが請求した行以外をクライアントに転送することはできません。例えば、もしクライアントがクエリを実行し、しかし最初の行しか必要としない場合は、残りの行は転送されません。

MySQL では、サーバサイドカーソルは一時テーブルへと出力されます。最初、これは `MEMORY` テーブルになりますが、そのサイズが `max_heap_table_size` システム変数の値に達すると、`MyISAM` テーブルに変換されます。この実装の制限のひとつとして、大きな結果セットでは、カーソルでの行の呼び出しに時間がかかる場合があります。

カーソルは読み取り専用で、カーソルを行のアップデートに使用することはできません。

`UPDATE WHERE CURRENT OF` および `DELETE WHERE CURRENT OF` は、アップデート可能なカーソルはサポートされていないため実装されていません。

カーソルは保持不可能 (コミットの後で開いたままにしておくことができない)。

カーソルはセンシティブです。

カーソルはスクロール不可能。

カーソルに名称は付いていません。ステートメント ハンドラがカーソル ID として作用します。

プリペアド ステートメントごとに、カーソルをひとつだけ開いておくことができます。複数のカーソルが必要な場合は、複数のステートメントを準備しなければなりません。

結果セットを生成するステートメントで、準備モードでサポートされていないものにはカーソルを使うことはできません。そのようなステートメントには、`CHECK TABLES`、`HANDLER READ`、そして `SHOW BINLOG EVENTS` があります。

サブクエリの規制

- 後に修正される既知のバグ :`ALL`、`ANY`、または `SOME` を使用して `NULL` 値をサブクエリと比較し、サブクエリが空の結果を返す場合、その比較は `TRUE` もしくは `FALSE` よりも、`NULL` の非標準結果を評価した可能性があります。
- サブクエリの外側のステートメントが次のどれかである可能性があります :`SELECT`、`INSERT`、`UPDATE`、`DELETE`、`SET` または `DO`
- `IN` のサブクエリの最適化は、`=` オペレータ、または `IN(value_list)` 構文に対する最適化ほど効果的ではありません。

貧弱な `IN` サブクエリの動作の一般的なケースで、サブクエリが少数の行を返すのに対し、外側のクエリは多数の行をサブクエリの結果と比較するために戻します。

その問題は、`IN` サブクエリを使用するステートメントでは、最適化機能がそれを相関サブクエリとして書き換えるということにあります。非相関サブクエリを使用する次のステートメントを検討してください：

```
SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
```

最適化機能がステートメントを相関サブクエリに書き換えます：

```
SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
```

内側と外側のクエリが M および N 行を戻す場合、それぞれ、実行時間は非相関サブクエリに対してと同様に、 $O(M+N)$ ではなく、 $O(M \times N)$ の順になります。

それはつまり、**IN** サブクエリが、サブクエリが戻す同じ値をリストアップする **IN(value_list)** 構文を使って書かれたクエリよりも、格段に実行速度が遅い場合があることを示しています。

- 基本的に、テーブルを改変したり、サブクエリの同じテーブルから選択することはできません。例えば、この制限は次のフォームのステートメントに適用されます：

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

例外：**FROM** 句にある改変されたテーブルにサブクエリを使用する場合、前述の禁止事項は適用されません。例：

```
UPDATE t ... WHERE col = (SELECT (SELECT ... FROM t...) AS t...);
```

FROM 句のサブクエリからの結果が一時テーブルとして保存されるため、禁止事項は適用されず、**t** の該当する行が、**t** へのアップデートまでには選択されているということになります。

- 行の比較のオペレーションは一部のみサポートされています：
 - expr IN (subquery)** では、**expr** が n タプル (行コンストラクタ シンタックスを介して指定) であることがあり、サブクエリが n タプルの行を戻すことができます。
 - expr op {ALL|ANY|SOME} (subquery)** では、**expr** がスカラー値である必要があり、またサブクエリはカラム サブクエリでなくてはならず、複数段の行を戻すことができません。

つまり、 n タプルの行を戻すサブクエリには、次がサポートされています：

```
(val_1, ..., val_n) IN (subquery)
```

しかし、次はサポートされていません：

```
(val_1, ..., val_n) op {ALL|ANY|SOME} (subquery)
```

IN の行の比較がサポートされているのに、他はされていない理由は、**IN** が **=** 比較および **AND** オペレーションへ書き換えることによって実装されているためです。この方法は、**ALL**、**ANY**、または **SOME** には使うことができません。

- 行コンストラクタの最適化は的確に行われていません。次のふたつの式は等価ですが、ふたつ目のみが最適化されます：

```
(col1, col2, ...) = (val1, val2, ...)
col1 = val1 AND col2 = val2 AND ...
```

- FROM** 句のサブクエリは相関サブクエリにはなりえません。それらは外側のクエリを評価する前に出力 (結果セットを生成するために実行される) されているので、外側のクエリの行ごとに評価を受けることはできません。
- 最適化機能は、サブクエリに対してより結合に対するほうが完成度が高く、そのため多くの場合は、サブクエリを使用するステートメントを結合として書き換えるほうが、より効率的に実行されます。

IN サブクエリが **SELECT DISTINCT** 結合として書き換えられる場合には例外が生じます。例：

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
```

そのステートメントは次のように書き換えられます：

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
```

しかしこの場合は、追加の **DISTINCT** オペレーションが結合に必要になり、サブクエリより効率的とは言えません。

- 可能な今後の最適化 :MySQL はサブクエリ評価のために結合順を書き換えることはしません。場合によっては、MySQL がサブクエリを結合として書き換えたほうが効率的に実行されるようです。これをふまれば、最適化機能により豊富な実行計画の選択肢を与えることができます。例えば、特定のテーブルを先に読み込むか、他を先にするか決定することが可能です。

例 :

```
SELECT a FROM outer_table AS ot
WHERE a IN (SELECT a FROM inner_table AS it WHERE ot.b = it.b);
```

そのクエリでは、MySQL は常に `outer_table` を先にスキャンし、それから `inner_table` の各行でサブクエリを実行します。`outer_table` の行数が多く、`inner_table` の行数が少ない場合は、クエリの速度が比較的落ちるでしょう。

上記のクエリは次のように書くこともできます :

```
SELECT a FROM outer_table AS ot, inner_table AS it
WHERE ot.a = it.a AND ot.b = it.b;
```

この場合、小さなテーブル (`inner_table`) をスキャンし、`outer_table` で行を検査することで、`(ot.a,ot.b)` にインデックスがあれば速くなります。

- 可能な今後の最適化 : 関連サブクエリは、外側のクエリのそれぞれの行に対して評価されます。よりよい方法としては、外側の行の値がその前の行と同じである場合、サブクエリを再度評価せず、かわりに前の結果を使用します。
- 可能な今後の最適化 : `FROM` 句のサブクエリは、結果を一時テーブルに出力することによって評価されます。また、そのテーブルはインデックスを使用しません。これにより、クエリの他のテーブルとの比較にインデックスを使えると便利ではありますが、それは許可されていません。
- 可能な今後の最適化 : `FROM` 句のサブクエリが、組合せアルゴリズムの適用が可能なビューに類似している場合、クエリを書き換えて組合せアルゴリズムを適用すると、インデックスが使用できるようになります。次のステートメントにはその種のサブクエリが含まれています :

```
SELECT * FROM (SELECT * FROM t1 WHERE t1.t1_col)
AS _t1, t2 WHERE t2.t2_col;
```

このステートメントは次のように、結合として書き換えることができます :

```
SELECT * FROM t1, t2 WHERE t1.t1_col AND t2.t2_col;
```

このタイプの書き換えには、ふたつの利点があります :

- インデックスが使えない一時テーブルの使用を避ける。書き換えられたクエリでは、最適化機能は `t1` でインデックスを使うことができる。
- 最適化機能に、複数の実行計画から選択する自由を与える。例えば、クエリを結合として書き換えることで、最適化機能が `t1` または `t2` を最初に使用できるようになる。
- 可能な今後の最適化 : 非関連サブクエリを持つ `IN`、`= ANY`、`<> ANY`、`= ALL`、および `<> ALL` には、メモリ内ハッシュを結果に使用するか、大きな結果にはインデックスのある一時テーブルを使用してください。例 :

```
SELECT a FROM big_table AS bt
WHERE non_key_field IN (SELECT non_key_field FROM table WHERE condition)
```

この場合、一時テーブルを作成することもできます :

```
CREATE TABLE t (key (non_key_field))
(SELECT non_key_field FROM table WHERE condition)
```

そして、`big_table` の各行では、`bt.non_key_field` に基づいて、`t` でキー検査を行ってください。

ビューの規制

ビューの処理は最適化されていません :

- ビューにインデックスを作成することはできません。

- 組合せアルゴリズムを使用して処理されたビューに、インデックスを使うことは可能です。しかし、誘導可能なアルゴリズムで処理されたビューは、その背後にあるテーブルのインデックスを活用することができません（一時テーブルの作成中にインデックスを使用することはできません）。

ビューの **FROM** 句でサブクエリを使用することはできません。この制限はいずれ取り除かれる予定です。

一般原則として、テーブルを改変したり、サブクエリの同じテーブルから選択することはできません。[サブクエリの規制](#) 参照。

また、テーブルから選択するビューを選ぶ場合、ビューがサブクエリのテーブルから選択する場合、そして、ビューが組合せアルゴリズムを使って評価される場合、同じ原則が適用されます。例：

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

一時テーブルを使ってビューが評価される場合、ビュー サブクエリのテーブルから選択し、さらに外側のクエリのテーブルで改変することが可能です。その場合、ビューは一時テーブルに格納されることになり、したがってサブクエリのテーブルから選択し、「同時に」改変するというにはなりません。（これもまた、ビュー定義で **ALGORITHM = TEMPTABLE** を指定して、誘導可能なアルゴリズムをMySQL が使用するよう強制できると便利だと考える理由です）。

DROP TABLE または **ALTER TABLE** を使用して、ビュー定義（ビューを無効化するもの）や削除または変更オペレーションからの無警告結果に使われるテーブルを、削除または変更することができます。エラーは、後でビューが使用された時に発生します。

ビュー定義は特定のステートメントによって「凍結」されています：

- **PREPARE** によって準備されたステートメントがビューを参照する場合、後でステートメントが実行される度に参照されるビューの内容が、ステートメントが準備できた時のビューの内容になります。これは、ステートメントが準備された後、実行される前にビュー定義が変更されても同じことです。例：

```
CREATE VIEW v AS SELECT 1;
PREPARE s FROM 'SELECT * FROM v';
ALTER VIEW v AS SELECT 2;
EXECUTE s;
```

EXECUTE ステートメントによって返される結果は 2 ではなく、1 です。

- ストアド ルーチンのステートメントがビューを参照する場合、ステートメントによって参照されたビューの内容は、最初にステートメントが実行された時のその内容です。これは例えば、ステートメントがループで実行された場合、さらなるステートメントの繰り返しですが、後でビュー定義がループで変更されたとしても、同じビューの内容を参照するという意味になります。例：

```
CREATE VIEW v AS SELECT 1;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  WHILE i < 5 DO
    SELECT * FROM v;
    SET i = i + 1;
    ALTER VIEW v AS SELECT 2;
  END WHILE;
END;
//
delimiter ;
CALL p();
```

プロシージャ **p()** が呼び出される際、**SELECT** は、たとえビュー定義がループ内で変更されても、ループを通るたびに 1 を返します。

ビューの更新可能性に関しては、すべてのビューが理論的には更新可能ならば、実際に更新可能であるべきだというのがビューに対する全体的な目標です。これには、定義に **UNION** を含むビューも含まれています。現時点では、理論的には更新可能なすべてのビューが、実際に更新可能というわけではありません。最初のビュー実装は、できるだ

け速く MySQL に使用可能で更新可能なビューを提供するため、故意にこのように書かれていました。理論的には更新可能なビューの多くは、今でも更新することができますが、制限はまだ存在します：

- **WHERE** 句以外にある、サブクエリを持った更新可能なビュー。**SELECT** リストにある、サブクエリを持つビューのいくつかは、更新可能な場合がある。
- **UPDATE** を使用して、結合として定義されたビューの背後のテーブルをひとつ以上更新することはできない。
- **DELETE** を使用して、結合として定義されたビューを更新することはできない。

ビューの現在の実装には欠点があります。もしユーザがビューの作成に必要な基本権限 (**CREATE VIEW** と **SELECT** 権限) を取得した場合、**SHOW VIEW** 権限も取得しない限り、そのユーザはオブジェクトの **SHOW CREATE VIEW** を呼び出すことはできないでしょう。

この欠点は、**mysqldump** を持つデータベースのバックアップの問題につながり、権限の不足のため失敗する原因になる恐れがあります。この問題は バグ #22062 で説明されています。

問題に対する迂回策としては、ビューが作成された時に MySQL が暗黙的に **SHOW VIEW** 権限を与えないので、管理権限者が手動でその権限を、**CREATE VIEW** を与えられたユーザに提供することです。

XA トランザクションの規制

XA トランザクションのサポートは、**InnoDB** 保存エンジンに限られています。

MySQL XA の実装は、MySQL サーバが Resource manager の役割をしたり、クライアント プログラムが Transaction Manager の役割をする「外部 XA_i」のものです。「内部 XA」は実装されていません。これによって、MySQL サーバ内の個々の保存エンジンが RM の役割をしたり、サーバそのものが TM の役割をしたりすることができます。内部 XA は、ひとつ以上の保存エンジンが関与する XA トランザクションを処理するために必要になります。テーブル ハンドラのレベルで 2 相コミットをサポートする保存エンジンを必要とするため、内部 XA の実装は不完全なもので、現時点では **InnoDB** だけです。

XA START では、**JOIN** および **RESUME** 句はサポートされていません。

XA END では、**SUSPEND [FOR MIGRATE]** 句はサポートされていません。

xid 値の **bqual** 部分が、大規模トランザクション内の各 XA トランザクションによって異なる必要があるという条件は、現在の MySQL XA の実装の制限です。これは XA の仕様によるものではありません。

XA トランザクションが **PREPARED** 状態に至ると、MySQL サーバが切断される (例えば、Unix の **kill -9** で)、または不自然に停止する場合、サーバが再起動した後にトランザクションを続けることができるはずですが、しかし、クライアントが再接続をし、トランザクションをコミットした場合、そのトランザクションは、たとえコミットされていても、バイナリ ログには記録されないでしょう。これは、データとバイナリ ログの同調が途切れたということですから、XA を複製と一緒に安全に使用することはできないと思われます。

サーバが保留になっている XA トランザクションを、たとえ **PREPARED** 状態に至っていても、ロールバックするということはありえます。これは、クライアントの接続が切断されて、サーバは起動を続ける、またはクライアントが接続されているのに、サーバが自然に停止する場合に起こります。(後者のケースでは、サーバが切断される各接続をマークし、それに関連する **PREPARED** XA トランザクションをロールバックする)。**PREPARED** XA トランザクションをコミットする、またはロールバックすることは可能であるはずですが、これはバイナリのロギング メカニズムを変更なしには実行できません。

MySQL の制限

このセクションでは、現在の MySQL の制限 5.1 をリストアップします。

結合の制限

ひとつの結合で参照できるテーブルの最大数は 61 です。これはビューの定義で参照できるテーブルの数と同じです。

付録E Credits

目次

Developers at MySQL AB	1711
Contributors to MySQL	1716
Documenters and translators	1719
Libraries used by and included with MySQL	1721
Packages that support MySQL	1721
Tools that were used to create MySQL	1722
Supporters of MySQL	1722

This appendix lists the developers, contributors, and supporters that have helped to make MySQL what it is today.

Developers at MySQL AB

These are the developers that are or have been employed by MySQL AB to work on the [MySQL](#) database software, roughly in the order they started to work with us. Following each developer is a small list of the tasks that the developer is responsible for, or the accomplishments they have made. All developers are involved in support.

- Michael (Monty) Widenius
 - Lead developer and main author of the MySQL server ([mysqld](#)).
 - New functions for the string library.
 - Most of the [mysys](#) library.
 - The [ISAM](#) and [MyISAM](#) libraries (B-tree index file handlers with index compression and different record formats).
 - The [HEAP](#) library. A memory table system with our superior full dynamic hashing. In use since 1981 and published around 1984.
 - The [replace](#) program (take a look at it, it's COOL!).
 - Connector/ODBC (MyODBC), the ODBC driver for Windows.
 - Fixing bugs in MIT-pthreads to get it to work for MySQL Server. And also Unireg, a curses-based application tool with many utilities.
 - Porting of [mSQL](#) tools like [msqlperl](#), [DBD/DBI](#), and [DB2mysql](#).
 - Most of [crash-me](#) and the foundation for the MySQL benchmarks.
- David Axmark
 - Initial main writer of the Reference Manual, including enhancements to [texi2html](#).
 - Automatic Web site updating from the manual.
 - Initial Autoconf, Automake, and Libtool support.
 - Licensing.
 - Parts of all the text files. (Nowadays only the [README](#) is left. The rest ended up in the manual.)
 - Lots of testing of new features.
 - Our in-house Free Software legal expert.

- Mailing list maintainer (who never has the time to do it right...).
- Our original portability code (now more than 10 years old). Nowadays only some parts of [mysys](#) are left.
- Someone for Monty to call in the middle of the night when he just got that new feature to work.
- Chief "Open Sourcerer" (MySQL community relations).
- Jani Tolonen
 - [mysqlimport](#)
 - A lot of extensions to the command-line clients.
 - [PROCEDURE ANALYSE\(\)](#)
- Sinisa Milivojevic (now in support)
 - Compression (with [zlib](#)) in the client/server protocol.
 - Perfect hashing for the lexical analyzer phase.
 - Multi-row [INSERT](#)
 - [mysqldump -e](#) option
 - [LOAD DATA LOCAL INFILE](#)
 - [SQL_CALC_FOUND_ROWS SELECT](#) option
 - [--max-user-connections=...](#) option
 - [net_read](#) and [net_write_timeout](#)
 - [GRANT/REVOKE](#) and [SHOW GRANTS FOR](#)
 - New client/server protocol for 4.0
 - [UNION](#) in 4.0
 - Multiple-table [DELETE/UPDATE](#)
 - Subqueries in the [FROM](#) clause (4.1).
 - User resources management
 - Initial developer of the [MySQL++](#) C++ API and the [MySQLGUI](#) client.
- Tonu Samuel (past developer)
 - VIO interface (the foundation for the encrypted client/server protocol).
 - MySQL Filesystem (a way to use MySQL databases as files and directories).
 - The [CASE](#) expression.
 - The [MD5\(\)](#) and [COALESCE\(\)](#) functions.
 - [RAID](#) support for [MyISAM](#) tables.
- Sasha Pachev (past developer)
 - Initial implementation of replication (up to version 4.0).
 - [SHOW CREATE TABLE](#).

- [mysql-bench](#)
- Matt Wagner
 - MySQL test suite.
 - Webmaster (until 2002).
- Miguel Solorzano (now in support)
 - Win32 development and release builds.
 - Windows NT server code.
 - WinMySQLAdmin
- Timothy Smith (now in development)
 - Dynamic character sets support.
 - configure, RPMs and other parts of the build system.
 - Initial developer of [libmysqld](#), the embedded server.
- Sergei Golubchik
 - Full-text search.
 - Added keys to the [MERGE](#) library.
 - Precision math.
- Jeremy Cole (past developer)
 - Proofreading and editing this fine manual.
 - [ALTER TABLE ... ORDER BY](#)
 - [UPDATE ... ORDER BY](#)
 - [DELETE ... ORDER BY](#)
- Indrek Siitan
 - Designing/programming of our Web interface.
 - Author of our newsletter management system.
- Jorge del Conde (past developer)
 - [MySQLCC \(MySQL Control Center\)](#)
 - Win32 development
 - Initial implementation of the Web site portals.
- Venu Anuganti (past developer)
 - MyODBC 3.51
 - New client/server protocol for 4.1 (for prepared statements).
- Arjen Lentz (also handled community, 2004-2006; now works in Support)
 - Maintainer of the MySQL Reference Manual (2001-2004).

- Preparing the O'Reilly printed edition of the manual (2002).
- Alexander (Bar) Barkov, Alexey (Holyfoot) Botchkov, and Ramil Kalimullin
 - Spatial data (GIS) and R-Trees implementation for 4.1
 - Unicode and character sets for 4.1; documentation for same
- Oleksandr (Sanja) Byelkin
 - Query cache in 4.0
 - Implementation of subqueries (4.1).
 - Implementation of views (5.0).
- Aleksey (Walrus) Kishkin and Alexey (Ranger) Stroganov
 - Benchmarks design and analysis.
 - Maintenance of the MySQL test suite.
- Zak Greant (past employee)
 - Open Source advocate, MySQL community relations.
- Carsten Pedersen
 - The MySQL Certification program.
- Lenz Grimmer
 - Production (build and release) engineering.
- Peter Zaitsev
 - [SHA1\(\)](#), [AES_ENCRYPT\(\)](#) and [AES_DECRYPT\(\)](#) functions.
 - Debugging, cleaning up various features.
- Alexander (Salle) Keremidarski
 - Support.
 - Debugging.
- Per-Erik Martin
 - Lead developer for stored procedures (5.0).
- Jim Winstead
 - Former lead Web developer.
 - Improving server, fixing bugs.
- Mark Matthews
 - Connector/J driver (Java).
- Peter Gultzan
 - SQL standards compliance.
 - Documentation of existing MySQL code/algorithms.

- Character set documentation.
- Guilhem Bichot
 - Replication, from [MySQL](#) version 4.0.
 - Fixed handling of exponents for [DECIMAL](#).
 - Author of [mysql_tableinfo](#).
 - Backup (in 5.1).
- Antony T. Curtis
 - Porting of the MySQL Database software to OS/2.
- Mikael Ronstrom
 - Much of the initial work on NDB Cluster until 2000. Roughly half the code base at that time. Transaction protocol, node recovery, system restart and restart code and parts of the API functionality.
 - Lead Architect, developer, debugger of NDB Cluster 1994-2004
 - Lots of optimizations
- Jonas Orelund
 - On-line Backup
 - The automatic test environment of MySQL Cluster
 - Portability Library for NDB Cluster
 - Lots of other things
- Pekka Nouisiainen
 - Ordered index implementation of MySQL Cluster
 - BLOB support in MySQL Cluster
 - Charset support in MySQL Cluster
- Martin Skold
 - Unique index implementation of MySQL Cluster
 - Integration of NDB Cluster into MySQL
- Magnus Svensson
 - The test framework for MySQL Cluster
 - Integration of NDB Cluster into MySQL
- Tomas Ulin
 - Lots of work on configuration changes for simple installation and use of MySQL Cluster
- Konstantin Osipov
 - Prepared statements.
 - Cursors.
- Dmitri Lenev

- Time zone support.
- Triggers (in 5.0).

Contributors to MySQL

Although MySQL AB owns all copyrights in the [MySQL server](#) and the [MySQL manual](#), we wish to recognize those who have made contributions of one kind or another to the [MySQL distribution](#). Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola <qwert@mbx.vol.it> or <qwert@tin.it>

The initial port to Win32/NT.

- Per Eric Olsson

For more or less constructive criticism and real testing of the dynamic record format.

- Irena Pancirov <irena@mail.yacc.it>

Win32 port with Borland compiler. [mysqlshutdown.exe](#) and [mysqlwatch.exe](#)

- David J. Hughes

For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with [mSQL](#), but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. [mysqladmin](#) and [mysql](#) client are programs that were largely influenced by their [mSQL](#) counterparts. We have put a lot of effort into making the MySQL syntax a superset of [mSQL](#). Many of the API's ideas are borrowed from [mSQL](#) to make it easy to port free [mSQL](#) programs to the MySQL API. The MySQL software doesn't contain any code from [mSQL](#). Two files in the distribution ([client/insert_test.c](#) and [client/select_test.c](#)) are based on the corresponding (non-copyrighted) files in the [mSQL](#) distribution, but are modified as examples showing the changes necessary to convert code from [mSQL](#) to MySQL Server. ([mSQL](#) is copyrighted David J. Hughes.)

- Patrick Lynch

For helping us acquire <http://www.mysql.com/>.

- Fred Lindberg

For setting up gmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko <igor@frog.kiev.ua>

[mysqldump](#) (previously [msqldump](#), but ported and enhanced by Monty).

- Yuri Dario

For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

Author of [mysqlhotcopy](#).

- Zarko Mocnik <zarko.mocnik@dem.si>

Sorting for Slovenian language.

- "TAMITO" <tommy@valley.ne.jp>

The [_MB](#) character set macros and the [ujis](#) and [sjis](#) character sets.

- Joshua Chamas <joshua@chamas.com>

Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier <Yves.Carlier@rug.ac.be>
[mysqlaccess](#), a program to show the access rights for a user.
- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)
For one of the early JDBC drivers.
- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>
Further development of one of the early JDBC drivers and other MySQL-related Java tools.
- James Cooper <pixel@organic.com>
For setting up a searchable mailing list archive at his site.
- Rick Mehalick <Rick_Mehalick@i-o.com>
For [xmysql](#), a graphical X client for MySQL Server.
- Doug Sisk <sisk@wix.com>
For providing RPM packages of MySQL for Red Hat Linux.
- Diemand Alexander V. <axeld@vial.ethz.ch>
For providing RPM packages of MySQL for Red Hat Linux-Alpha.
- Antoni Pamies Olive <toni@readysoft.es>
For providing RPM versions of a lot of MySQL clients for Intel and SPARC.
- Jay Bloodworth <jay@pathways.sde.state.sc.us>
For providing RPM versions for MySQL 3.21.
- David Sacerdote <davids@secnet.com>
Ideas for secure checking of DNS hostnames.
- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>
Some support for Chinese(BIG5) characters.
- Wei He <hewei@mail.ied.ac.cn>
A lot of functionality for the Chinese(GBK) character set.
- Jan Pazdziora <adelton@fi.muni.cz>
Czech sorting order.
- Zeev Suraski <bourbon@netvision.net.il>
[FROM_UNIXTIME\(\)](#) time formatting, [ENCRYPT\(\)](#) functions, and [bison](#) advisor. Active mailing list member.
- Luuk de Boer <luuk@wxs.nl>
Ported (and extended) the benchmark suite to [DBI/DBD](#). Have been of great help with [crash-me](#) and running benchmarks. Some new date functions. The [mysql_setpermission](#) script.
- Alexis Mikhailov <root@medinf.chuvashia.su>
User-defined functions (UDFs); [CREATE FUNCTION](#) and [DROP FUNCTION](#).
- Andreas F. Bobak <bobak@relog.ch>

The [AGGREGATE](#) extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

The [libmysql.dll](#) library.

- James Pereria <jpereira@iafrica.com>

Mysqmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

Examples in the MySQL Tutorial.

- Steve Harvey

For making [mysqlaccess](#) more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

<http://www.pspl.co.in/konark/>. Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

Emulation of [pthread_mutex\(\)](#) for OS/2.

- Benjamin Pflugmann

Extended [MERGE](#) tables to handle [INSERTS](#). Active member on the MySQL mailing lists.

- Jocelyn Fournier

Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X PKGs.

- Robert Rutherford

Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataulah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, <jehamby@lightside>, <psmith@BayNetworks.com>, <duane@connect.com.au>, Ted Deppner <ted@psyber.com>, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch <dkoch@amcity.com>

Irix setup.

- Luuk de Boer <luuk@wxs.nl>

Benchmark questions.

- Tim Sailer <tps@users.buoy.com>

DBD::mysql questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

xmysql-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

General questions.

- Jonathan J Smith <jsmith@wtp.net>

Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

Is flexible and can handle Linux and perhaps HP-UX. Tries to get users to use [mysqlbug](#).

- John Lyon <jllyon@imag.net>

Questions about installing MySQL on Linux systems, using either [.rpm](#) files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

DBD, Linux, some SQL syntax questions.

Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (<http://mysql.hitstar.com/>) versions were based. [Personal home page at linuxdb.yeah.net](#).

- Jay Flaherty <fty@mediapulse.com>

Big parts of the Perl [DBI/DBD](#) section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

French error messages.

- Petr Snajdr, <snajdr@pvt.net>

Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

Polish error messages.

- Miguel Angel Fernandez Roiz

Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

German error messages.

- Billik Stefan <billik@sun.uniag.sk>

Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

Romanian error messages.

- Peter Feher
Hungarian error messages.
- Roberto M. Serqueira
Portuguese error messages.
- Carsten H. Pedersen
Danish error messages.
- Arjen Lentz
Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

Libraries used by and included with MySQL

The following is a list of the creators of the libraries we have included with the MySQL server source to make it easy to compile and install MySQL. We are very thankful to all individuals that have created these and it has made our life much easier.

- Fred Fish
For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).
- Richard A. O'Keefe
For his public domain string library.
- Henry Spencer
For his regex library, used in [WHERE column REGEXP regexp](#).
- Chris Provenzano
Portable user level pthreads. From the copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. We are currently using version 1_60_beta6 patched by Monty (see [mit-pthreads/Changes-mysql](#)).
- Jean-loup Gailly and Mark Adler
For the zlib library (used on MySQL on Windows).
- Bjorn Benson
For his safe_malloc (memory checker) package which is used in when you configure MySQL with `--debug`.
- Free Software Foundation
The [readline](#) library (used by the [mysql](#) command-line client).
- The NetBSD foundation
The [libedit](#) package (optionally used by the [mysql](#) command-line client).

Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We can't list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at <http://solutions.mysql.com/software/>.

- Tim Bunce, Alligator Descartes
For the [DBD](#) (Perl) interface.
- Andreas Koenig <a.koenig@mind.de>
For the Perl interface for MySQL Server.
- Jochen Wiedmann <wiedmann@neckar-alb.de>
For maintaining the Perl [DBD::mysql](#) module.
- Eugene Chan <eugene@acenet.com.sg>
For porting PHP for MySQL Server.
- Georg Richter
MySQL 4.1 testing and bug hunting. New PHP 5.0 [mysqli](#) extension (API) for use with MySQL 4.1 and up.
- Giovanni Maruzzelli <maruzz@matrice.it>
For porting iODBC (Unix ODBC).
- Xavier Leroy <Xavier.Leroy@inria.fr>
The author of LinuxThreads (used by the MySQL Server on Linux).

Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation
From whom we got an excellent compiler ([gcc](#)), an excellent debugger ([gdb](#)) and the [libc](#) library (from which we have borrowed [strto.c](#) to get some code working in Linux).
- Free Software Foundation & The XEmacs development team
For a really great editor/environment used by almost everybody at MySQL AB.
- Julian Seward
Author of [valgrind](#), an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.
- Dorothea Lütkehaus and Andreas Zeller
For [DDD](#) (The Data Display Debugger) which is an excellent graphical front end to [gdb](#).

Supporters of MySQL

Although MySQL AB owns all copyrights in the [MySQL server](#) and the [MySQL manual](#), we wish to recognize the following companies, which helped us finance the development of the [MySQL server](#), such as by paying us for developing a new feature or giving us hardware for development of the [MySQL server](#).

- VA Linux / Andover.net
Funded replication.
- NuSphere
Editing of the MySQL manual.

- Stork Design studio

The MySQL Web site in use between 1998-2000.

- Intel

Contributed to development on Windows and Linux platforms.

- Compaq

Contributed to Development on Linux/Alpha.

- SWSOft

Development on the embedded `mysqld` version.

- FutureQuest

`--skip-show-database`

索引

シンボル

!(NOT 演算), 576
!= (等しくない), 572
", 501
#mysql50識別子の接頭辞, 501, 504
%, 593
% (モジユロ), 596
% (ワイルドカード文字), 498
& (ビット単位の論理積), 630
&& (AND 演算), 576
() (丸括弧), 570
(Control-Z) \Z, 498
* (乗算), 593
+ (加算), 592
- (単項マイナス), 593
- (減算), 592
--password option, 266
-p オプション, 266
.my.cnf file, 143, 144, 246, 253, 266, 311
.mysql_history file, 456
.pid (process ID) file, 289
/ (除算), 593
/etc/passwd, 236, 701
16 進値, 499
2000年コンプライアンス, 556
2000年問題, 556
< (より少ない), 573
<< (左シフト), 630
<= (より多いか等しい), 573
<= (より少ないか等しい), 572
<=> (等しい), 572
<> (等しくない), 572
= (等しい), 572
> (より多い), 573
>> (右シフト), 630
[API] (MySQL Cluster), 922
[MGM] (MySQL Cluster), 921
[NDBD] (MySQL Cluster), 918
[NDBD_DEFAULT] (MySQL Cluster), 918
[NDB_MGMD] (MySQL Cluster), 921
[SQL] (MySQL Cluster), 922
\' (二重引用符), 498
\' (単一引用符), 498
\0 (ASCII 0), 498
\b (バックスペース), 498
\n (改行), 498, 498
\r (復帰改行), 498
\t (タブ), 498
\Z (Control-Z) ASCII 26, 498
\\ (エスケープ), 498
^ (ビット単位の排他的論理和), 630
_ (ワイルドカード文字), 498
_rowid, 666
`, 501
|| (OR 演算), 576
|| (ビット単位の論理積), 630

~, 630
かっこ
角, 541
その他の関数, 640
でのバックアップ
MySQL Cluster, 945, 945, 946, 948
でのバックアップからの保存
MySQL Cluster レプリケーション, 970
によるインストール
ソース ディストリビューション, 73
にインストールする際の問題
IBM-AIX, 121
の tar の問題
Solaris, 69, 114
のアップグレード
MySQL, 924
MySQL Cluster, 924, 926
異なるアーキテクチャ, 105
のインストール
Solaris PKG パッケージ, 69
バイナリ ディストリビューション, 71
のインストール時の問題
Solaris, 114
のエラー
ディレクトリ チェックサム, 114
のコマンド
バイナリ ディストリビューション, 72
のダウングレード
MySQL Cluster, 924, 926
のダウンロード
MySQL Cluster, 924
のテスト
インストール, 90
のバックアップの復旧
MySQL Cluster, 946
のバックアップの設定
MySQL Cluster, 948
のフェールオーバー
MySQL Cluster レプリケーション, 970
の起動
サーバ, 90
の追加
新しいユーザーの, 73, 76
より多い (>), 573
より多いか等しい (>=), 573
より少ない (<), 573
より少ないか等しい (<=), 572
アカウント
ルート, 99
匿名ユーザー, 99
アカウント権限
追加, 261
アクセス制御, 246, 247
アクセス権限, 238
アップグレード, 101
権限テーブル, 229
アップデート
テーブル, 22
イベント スケジューラ
ステータス情報を得る, 769

- イベント タイプ (MySQL Cluster), 938, 940
- イベント ログ (MySQL Cluster), 937, 939, 939
- イベント ログ フォーマット (MySQL Cluster), 940
- イベントの深刻度タイプ (MySQL Cluster), 939
- インストール
 - の概要, 32
- インストールする
 - Windows に Perl を, 135
- インストールのレイアウト, 45, 45
- インストールの概要, 73
- インストール後
 - 複数サーバ, 306
- インストール後の
 - 設定とテスト, 89
- インターネット リレー チャット, 13
- インデックス, 660
 - およびBLOBカラム, 413
 - およびTEXTカラム, 413
 - と BLOB カラム, 666
 - と NULL 値, 666
 - と TEXT カラム, 666
 - と IS NULL, 415
 - と LIKE, 415
 - の使用, 414
 - カラム, 413
 - キー キャッシュに割り当てる, 776
 - ブロック サイズ, 172
 - 削除する, 653, 678
 - 名, 500
 - 最も左側のプリフィックス, 415
 - 複合, 413
 - 複数部分, 660
- インデックス ヒント, 699, 705
- インデックスの最も左側プリフィックス, 415
- イントロデューサ
 - 文字列リテラル, 497, 518
- エイリアス
 - での GROUP BY 句, 648
 - テーブルの, 699
 - 名, 500
 - 式, 648
 - 式の上で, 699
- エイリアス名
 - 大文字/小文字の区別, 502
- エスケープ(\\), 498
- エスケープ文字, 497
- エッフェルラッパー (Eiffel Wrapper), 1231
- エラー
 - テーブルのチェック, 281
 - 報告, 2, 14
- エラー メッセージ
 - 表示, 494
 - 言語, 290, 290
- エラーログ (MySQL Cluster),
- オプション
 - libmysqld, 76, 1137
 - myisamchk, 440
 - コマンド ライン
 - mysql, 452
 - コマンドライン
 - mysqladmin, 467
 - レプリケーション, 352
 - 埋め込まれたサーバ, 1137
 - オプションファイル, 143, 253
 - オプティマイザ
 - 管理, 428
 - オペレーティング システム
 - サポート, 33
 - オンライン マニュアルのロケーション, 2
 - オープニング
 - テーブル, 422
 - オープン ソース
 - 定義, 5
 - オープンテーブル, 422, 466
 - カスタマ
 - MySQLの, 373
 - カラム
 - インデックス, 413
 - タイプ, 541
 - 他のタイプ, 566
 - 名, 500
 - 変更, 652
 - 必要とする記憶容量, 564
 - カラム コメント, 665
 - カラム名
 - 大文字/小文字の区別, 502
 - カレンダー, 613
 - カンマで区切られた値データ、読み込み, 693, 702
 - カーソル, 1063
 - キャスト, 570, 572, 624
 - キャスト演算子, 624
 - キャスト関数, 624
 - キャッシュ
 - クリアする, 777
 - キャラクタ セット, 289
 - 追加, 291
 - キャラクタセット, 513
 - キー, 413
 - 外部, 24
 - 複合, 413
 - キー キャッシュ
 - インデックスを割り当てる, 776
 - キーによるパーティショニング, 1006
 - キースキャッシュ
 - MyISAM, 416
 - キースペース
 - MyISAM, 802
 - キーパーティショニング
 - 分離と結合, 1019
 - 管理, 1019
 - キーワード, 507
 - クエリ
 - スピード, 375
 - 推定パフォーマンス, 384
 - クエリ オプション
 - ndb_config, 951, 951
 - クエリ キャッシュ, 311
 - クライアント
 - スレッド付き, 1228
 - クライアント プログラム, 435

- クライアントツール, 1135
- クライアントプログラム
 - 構築, 1228
- クラス ログ, 937, 939
- クラスタリング, 872
- クラッカー
 - セキュリティ対策, 234
- クラッシュ
 - リカバリ, 280
- クリアする
 - キャッシュ, 777
- クロージング
 - テーブル, 422
- グラント テーブル
 - の再作成, 94
 - 再作成, 94
- グループ分け
 - 式, 570
- グローバル権限, 734, 741
- コマンド オプション
 - mysql, 452
 - MySQL Cluster, 932
 - MySQL Cluster の, 933, 933, 934, 935
 - mysqld, 152
- コマンド構文, 3
- コメント
 - 追加, 511
 - 開始, 26
- コメント構文, 511
- コンパイラ
 - C++ gcc, 77
- コンパイル
 - Windows での, 88
 - スピード, 429
 - 最適化, 423
 - 静的, 77
- コンパイルの問題, 82, 82
- コンパイル中の仮想メモリ
 - の問題, 82
- コンプライアンス
 - Y2K, 556
- サイズ
 - ディスプレイ, 541
- サイレント カラムの変更, 675
- サブクエリ, 711, 711
- サブクエリ規制, 1705
- サブセレクト, 710
- サブパーティショニング, 1007, 1007
- サポートしている
 - オペレーティングシステム, 33
- サーバ
 - のシャットダウン, 92
 - の起動, 90, 92
 - の起動と停止, 95
 - 接続, 246
 - 複数, 306
 - 起動時の問題, 97
- サーバ サイド プログラム, 150
- サーバなしのオプション, 76
- サーバを
 - 停止する, 95
 - 自動的に起動する, 95
- サーバオプションなし
 - configure, 77
- サーバサイドカーソルの規制, 1705
- サーバ変数, 162, 773
- シェル構文, 3
- システム
 - セキュリティ, 232
 - 権限, 239
- システムテーブル, 377
- システム変数, 162, 187, 773
- システム最適化, 423
- シャットダウン
 - サーバ, 92
- シンボリックリンク, 432, 434
- シークエンス エミュレーション, 638
- スキーマ
 - 作成する, 659
 - 削除する, 677
 - 変更する, 650
- スクリプト, 210, 213
 - mysqlbug, 17
 - mysql_install_db, 93
 - SQL, 452
- スケール
 - 算数, 1127
- スタンダード SQL
 - との違い, 21, 740
- スタートアップ パラメータ
 - mysql, 452
- スタートアップのオプション
 - デフォルト, 143
- スタートアップパラメータ, 424
 - mysqladmin, 467
 - チューニング, 423
- ステータス
 - テーブル, 771
- ステータス変数, 195, 770
- ステートメント
 - GRANT, 261
 - INSERT, 262
 - 複製スレーブ, 783
 - 複製マスタ, 780
- ストアド ルーチンの規制, 1703
- ストアドプロシージャ, 1053
- ストアドプロシージャとトリガ
 - 定義された, 24
- ストアドルーチン
 - LAST_INSERT_ID(), 1066
- ストップワード リスト
 - ユーザ定義, 623
- ストライピング
 - 定義された, 432
- ストレージ ノード - データ ノード参照、ndbd, 930
- ストレージエンジン
 - ARCHIVE, 866
 - 選択, 793
- スピード

- クエリの, 375, 384
- コンパイル, 429
- リンク, 429
- レプリケーションで高める, 317
- スレッド, 764
 - 表示, 764
- スレッド サポート
 - 非ネイティブ, 84
- スレッドのサポート, 33
- スレッド付きクライアント, 1228
- スロー クエリ ログ, 304
- スーパーユーザー, 99
- セキュリティ システム, 238
- セキュリティ対策
 - クラッカー, 234
- ソケット ロケーションの
 - 変更, 77, 77
- ソケット ロケーションの変更, 77, 96
- ソース ディストリビューション
 - Linux, 108
- ソース ディストリビューションによるインストール, 73
- ソート
 - キャラクタ セット, 289
 - 権限テーブル, 249, 250
- タイプ
 - カラム, 541, 566
 - テーブルの, 793
 - データ, 541, 564
 - ポータビリティ, 566
 - 数値, 564
 - 文字列, 557
 - 日付と時刻, 550
 - 時刻, 564
- タイプ オプション
 - ndb_config, 951
- タイプの選択, 566
- タイプ変換, 570, 572
- タイム ゾーン テーブル, 232
- タイムアウト, 689
- タブ(\t), 498
- ダウングレード, 106
- ダウンロード, 43
- チェック
 - テーブルのエラー, 281
- チェックサムのエラー, 114
- ツール
 - mysqld_multi, 213
 - mysqld_safe, 210
- テスト
 - インストール後, 89
 - サーバ接続, 247
- テーブル
 - BLACKHOLE, 869
 - CSV, 867
 - EXAMPLE, 862
 - FEDERATED, 862
 - HEAP, 860
 - InnoDB, 806
 - MEMORY, 860
 - MERGE, 856
 - MyISAM, 799
 - アップデート, 22
 - エラー チェック, 281
 - オープニング, 422
 - オープン, 422
 - クロー징ング, 422
 - コピーする, 673
 - システム, 377
 - シンボリックリンク, 433
 - チェック, 442
 - デフラグ, 289, 745, 803
 - フラグメンテーション, 745
 - ホスト, 251
 - マーキング, 856
 - 付与, 252
 - 保守計画, 289
 - 修復, 281
 - 削除する, 678
 - 動的, 803
 - 名, 500
 - 圧縮された, 446
 - 圧縮フォーマット, 804
 - 変更, 650, 653
 - 定数, 377, 385
 - 性能向上, 412
 - 情報, 284
 - 数が多すぎる, 423
 - 最後の列に対するユニークID, 1227
 - 最適化, 283
 - 表示ステータス, 771
 - 領域確保, 856
 - テーブル エイリアス, 699
 - テーブルは満杯です。 , 749
 - テーブルをコピーする, 673
 - テーブルキャッシュ, 422
 - テーブルスキャン
 - 避ける, 403
 - テーブルタイプ
 - 選択, 793
 - テーブルレベルロック, 408
 - テーブル名
 - 大文字/小文字の区別, 502
 - 大文字と小文字を区別する, 19
 - ディスク
 - データを横に分布する, 434
 - ディスク データ テーブル, 975
 - ディスク関連の問題, 431
 - ディスプレイ トリガ, 772
 - ディスプレイサイズ, 541
 - ディスプレイ幅, 541
 - ディレクトリの構成
 - デフォルト, 45
 - デバッグ オプション付き
 - configure, 79
 - デバッグのサポート, 76
 - デフォルト
 - 埋め込まれた, 1137
 - デフォルトの
 - 権限, 99

デフォルトのインストールの場所, 45
デフォルトのホスト名, 246
デフォルトオプション, 143
デフォルト値, 372, 547, 665, 685
 BLOB と TEXT カラム, 560
 抑制, 27
 明示的な, 547
 暗黙の, 547
データ
 REAL, 544
 キャラクタ セット, 289
 サイズ, 412
データ タイプ
 DECIMAL, 1127
データ ノード (MySQL Cluster), 930
データの保存, 411
データをスタートするために複合ディスクを使用する, 434
データタイプ, 541, 564
 BIGINT, 543
 BINARY, 546, 558
 BIT, 542
 BLOB, 547, 559
 BOOL, 542, 566
 BOOLEAN, 542, 566
 C API, 1140
 CHAR, 546, 557
 CHAR VARYING, 546
 CHARACTER, 546
 CHARACTER VARYING, 546
 DATE, 544, 551
 DATETIME, 544, 551
 DEC, 544
 DECIMAL, 544
 DOUBLE, 543
 DOUBLE PRECISION, 544
 ENUM, 547, 560
 FIXED, 544
 FLOAT, 543, 543, 544
 INT, 543
 INTEGER, 543
 LONG, 559
 LONGBLOB, 547
 LONGTEXT, 547
 MEDIUMBLOB, 547
 MEDIUMINT, 543
 MEDIUMTEXT, 547
 NATIONAL CHAR, 546
 NCHAR, 546
 NUMERIC, 544
 SET, 547, 562
 SMALLINT, 543
 TEXT, 547, 559
 TIME, 545, 555
 TIMESTAMP, 545, 551
 TINYBLOB, 547
 TINYINT, 542
 TINYTEXT, 547
 VARBINARY, 547, 558
 VARCHAR, 546, 557
 VARCHARACTER, 546
 Y2K問題, 556
 YEAR, 545, 556
 概要, 541
データノード (MySQL Cluster)
 の定義, 875
データベース
 の複製, 317
 シンボリックリンク, 432
 バックアップ, 274
 作成する, 659
 削除する, 677
 名, 500
 変更する, 650
 定義, 4
データベース メタデータ, 1099
データベースの
 コピー, 105
データベースのコピー, 105
データベース名
 大文字/小文字の区別, 502
 大文字と小文字を区別する, 19
データベース情報
 得る, 752
データベース設計, 411
データ関数
 Y2Kコンプライアンス, 556
トラブルシューティング
 FreeBSD, 83
 Solaris, 83
トランザクション
 サポート, 22, 806
トランザクション オプション
 ndb_delete_all, 952
トランザクション セーフ テーブル, 22, 806
トリガ, 24, 772, 1071
 LAST_INSERT_ID(), 1066
トリガ、ドロップする, 1074
トリガ、作成, 1071
トリガの規制, 1703
トレース ファイル (MySQL Cluster),
ドロップ
 ユーザ, 264
ドロップする
 ユーザ, 734
ネイティブ スレッドのサポート, 33
ネスト化したクエリ, 711
ネチケット, 13
ネットマスク表記
 mysql.user テーブル, 247
ノード グループ (MySQL Cluster), 876
ノード ログ (MySQL Cluster), 937
ノード識別子 (MySQL Cluster), 915, 916
ハッシュによるパーティショニング, 1003
ハッシュパーティショニング
 分離と結合, 1019
 管理, 1019
ハンドラ, 1061
バイナリ ディストリビューション
 のインストール, 71
バイナリ ログ, 301

バイナリの配布, 38
バイナリデータの引用, 499
バグ
 MySQL Cluster
 レポート, 955
 報告, 14, 14
バグ データベース, 14
バグ報告
 のためのクライテリア, 14
バックアップ, 274
 MySQL Cluster レプリケーション, 970
バックアップする
 データベース, 743
バックアップのトラブルシューティング
 MySQL Cluster, 949
バックスペース(b), 498
バックスラッシュ
 エスケープ文字, 497
バッチSQLファイル, 452
バッファサイズ
 mysqldサーバ, 424
バッファースize
 クライアント, 1135
バージョン オプション
 ndb_config, 951
バージョン間のアップグレードおよびダウングレードの
 互換性 (MySQL Cluster), 926
パスワード
 セキュリティ, 239
 ユーザ用, 260
 ルート ユーザー, 99
 設定, 265
 設定する, 738, 742
パスワードの暗号化
 可逆性, 634
パスワードを設定する, 742
パターン照会, 589
パフォーマンス
 ディスク関連の問題, 431
 向上, 358, 360
パラメータ
 サーバ, 424
パーティショニング, 995
 と FULLTEXT インデックス, 1027
 と外部キー, 1027
 と日付, 998
 キャラクタセット/照合, 1027
 キーによる, 1006
 コンセプト, 996
 サポート, 996
 サーバSQL モード, 1027
 ストレージ エンジン(限界), 1027
 タイプ, 998
 チェック, 1019
 テンポラリテーブル, 1027
 ハッシュによる, 1003
 パーティションの最大数, 1027
 リストによる, 1001
 リニアキーによる, 1007
 リニアハッシュによる, 1005
 レンジによる, 999
 修復, 1019
 分析, 1019
 分離と結合, 1013
 利点, 997
 制限, 1025
 情報, 995
 改良, 1013
 最適化, 1019, 1021, 1022
 有効化, 996
 演算子。パーティショニング表現では許容されない。 , 1026
 演算子。パーティショニング表現でサポートされる。 , 1026
 管理, 1013
 追加と削除, 1013
 関数。パーティショニング表現でサポートされている。 , 1026
 関数は、パーティショニング表現では許容されていません。 , 1025
パーティショニングのサポート
 MySQL Cluster,
パーティショニングキーとプライマリキー, 1028
パーティショニングキーとユニークキー,
パーティショニング管理, 1013
パーティションの刈り込み, 1022
パーティションの情報を取得する, 1020
パーティション情報のステートメント, 1020
パーティション (MySQL Cluster), 876
ヒント
 インデックス, 699, 705
 最適化, 406
ビュー, 25, 1091, 1091
 問題, 1709
 更新可能, 25, 1091
 権限, 1709
 規制, 1709
ビュー規制, 1707
ファイル
 config.cache, 82
 my.cnf, 352
 tmp, 94
 エラー メッセージ, 290
 スロー クエリ ログ, 304
 バイナリ ログ, 301
 ログ, 76, 305
 一般クエリ ログ, 300
 修復, 443
 更新ログ (旧式), 301
ファイルソート最適化, 400
フィールド
 変更, 652
フィールド オプション
 ndb_config, 951
フォーラム, 13
プログラム
 crash-me, 372
 クライアント, 435, 1228
 サーバ サイド, 150
 ユーティリティ, 435
プログラム変数
 セット, 148

プログラム変数を設定する, 148
プロシージャ
 保存された, 24, 1053
プロセス
 表示, 764
プロセスのサポート, 33
プロセス管理 (MySQL Cluster), 929
ベンチマーク, 374
ベンチマークスイート, 374
ホスト オプション
 ndb_config, 951
ホストネームキャッチング, 431
ホスト名
 デフォルト, 246
ポータビリティ
 タイプ, 566
マイナス
 単項 (-), 593
マニュアル
 オンライン ロケーション, 2
 利用可能なフォーマット, 2
 表記規則, 2
マネジメント クライアント (MySQL Cluster), 932
マネジメント ノード (MySQL Cluster), 931
 の定義, 875
マルチ バイト文字, 293
ミディアム・チェックオプション
 myisamchk, 476
ミラー サイト, 43
メソッド
 ロック, 408
メタデータ
 データベース, 1099
メモリの使用, 430, 467
メモリ使用量
 myisamchk, 445
メーリング リスト, 11
 保管場所, 11
メーリング リスト アドレス, 2
メーリングリスト
 ガイドライン, 13
モジュロ (%), 596
モジュロ (MOD), 596
ユニークID, 1227
ユニークキー
 パーティショニングキー, ,
ユーザ
 削除, 264
 削除する, 734
ユーザ アカウント
 リネームする, 741
 作成する, 734
ユーザ アカウントをリネームする, 741
ユーザ アカウントを作成する, 734
ユーザーの
 追加, 73, 76, 99
ユーザ名
 パスワード, 260
ユーザ変数, 510
ユーザ権限
 ドロップ, 264
 ドロップする, 734
 削除, 264
 削除する, 734
 追加, 261
ユーティリティ プログラム, 435
ライブラリー
 mysqlclient, 1135
 mysqld, 1135
ラッパー (wrappers)
 (エッフェル)Eiffel, 1231
リカバリ
 クラッシュから, 280
 任意時点, 279
リストによるパーティショニング, 1001
リストパーティショニング
 管理, 1014
 追加と削除, 1014
リテラル, 497
リニアキーによるパーティショニング, 1007
リニアハッシュによるパーティショニング, 1005
リリース
 の命名概要, 35
 更新, 37
リリースの
 テスト, 36
リリース番号, 35
リレーショナル データベース
 定義, 5
リンキング, 1228
 問題, 1227
リンク
 シンボリック, 432
 速度, 429
ルート パスワード, 99
レプリカ (MySQL Cluster), 876
レプリケーション, 317
レプリケーション オプション, 352
レプリケーション、非同期, 962
レプリケーションでの制約, 352
レプリケーションで高める
 スピード, 317
レプリケーションの実装, 363
レポート
 Connector/NET の問題, 1375
レンジによるパーティショニング, 999
レンジパーティショニング
 管理, 1014
 追加と削除, 1014
ログ コマンド (MySQL Cluster), 939
ログ ファイル, 76, 297
 保守, 305
 名前, 275
ログ ファイル (MySQL Cluster), 930
ロック, 423
 テーブルレベル, 408
 外部, 155, 159, 183, 280
 行レベル, 24, 408
ロックする
 外部, 768

- ロックメソッド, 408
- ローカル チェックポイント (MySQL Cluster), 923
- ローリング アップグレードおよびダウングレード (MySQL Cluster), 924
- ローリング再起動 (MySQL Cluster), 924
- ワイルドカード
 - とLIKE, 415
- ワイルドカード文字
 - mysql.columns_priv テーブル, 250
 - mysql.db テーブル, 250
 - mysql.host テーブル, 250
 - mysql.procs_priv テーブル, 250
 - mysql.tables_priv テーブル, 250
 - mysql.user テーブル, 247
- ワイルドカード文字(%), 498
- ワイルドカード文字(_), 498
- 一時ファイル
 - 書き込みアクセス, 94
- 一般クエリ ログ, 300
- 一般公有使用許諾, 5
- 一般情報, 1
- 丸め, 1127
- 丸め誤差, 543
- 丸括弧 (および), 570
- 主キー
 - 削除, 653
- 乗算 (*), 593
- 予備の文字セット付き
 - configure, 78
- 予約語, 507
- 二重引用符 (\"), 498
- 互換性
 - MySQL バージョン間の, 102
 - ODBC 利用時の, 665, 705
 - ODBCとの, 502, 544
 - Oracle, 20, 645
 - Oracle 利用時の, 721
 - PostgreSQLとの, 21
 - Sybase 利用時の, 724
 - 標準SQLとの, 17
 - mSQL, 588
- 任意時点のリカバリ, 279
- 作成
 - デフォルトのスタートアップのオプション, 143
 - バグ報告, 14
- 作成する
 - スキーマ, 659
 - データベース, 659
- 使用
 - MySQLの, 373
- 使用オプション
 - ndb_config, 951
- 例
 - mysamchk 出力, 284
- 供与する
 - 権限, 734
- 保存スペース
 - 縮小, 412
- 保守
 - テーブル, 289
- ログファイル, 305
- 修復
 - テーブル, 282
- 全文
 - ストップワード リスト, 623
- 全文検索, 613
- 共有メモリのトランスポート, 915
- 内部ロック, 408
- 再設定, 82, 82
- 再起動
 - サーバの, 92
- 制御フロー関数, 577
- 制約, 26
 - レプリケーション, 352
 - 設計上, 372
- 制限
 - MySQL 制限, 1709
 - MySQL 制限、MySQL の制限, 1709
- 削除
 - ユーザ, 264, 264
 - 主キー, 653
 - 外部キー, 653, 824
- 削除する
 - インデックス, 653, 678
 - スキーマ, 677
 - テーブル, 678
 - データベース, 677
 - ユーザ, 734, 734
 - 権限, 741
- 前に実行後に configure
 - を実行する, 82
- 前の実行の後に configure を実行する, 82
- 加算 (+), 592
- 動的テーブルの特徴, 803
- 区切り文字が無い文字列, 552
- 匿名のユーザー, 99, 99
- 匿名ユーザ, 248, 250
- 十進演算, 1127
- 単一ユーザーモード (MySQL Cluster), , 944
 - および ndb_restore, 946
- 単一引用符 (\'), 498
- 単項マイナス (-), 593
- 参照, 653
- 同時挿入, 408, 411
- 名, 500
- 名付けられていないビュー, 715
- 名前
 - 変数, 510
 - 大文字/小文字の区別, 502
- 名前付きパイプ, 57, 61
- 問題
 - ODBC, 1315
 - Perl をインストールする際の, 136
 - サーバの起動時, 97
 - テーブルロック, 410
 - 報告, 14
 - 日付値, 553
- 固定小数点数演算, 1127
- 圧縮オプション
 - mysqladmin, 467

圧縮テーブル, 446, 804
埋め込まれたMySQLサーバライブラリ, 1135
埋め込みサーバオプション付き
 configure, 77
報告
 Connector/ODBC の問題点, 1315, 1315
 エラー, 2
 バグ, 14, 14
変数
 mysqld, 424
 サーバ, 162, 773
 システム, 162, 187, 773
 ステータス, 195, 770
 ユーザ, 510
変更
 カラム, 652
 テーブル, 650, 653
 フィールド, 652
変更する
 スキーマ, 650
 データベース, 650
変更のみチェック・オプション
 myisamchk, 475
外部キー, 24, 653
 制約, 27
 削除, 653, 824
外部ロック, 155, 159, 183, 280, 768
外部ロック オプション
 mysqld, 155
大/小文字の区別
 アクセス チェック, 241
大きなテーブルのオプション, 76
大きなテーブルのオプション付き
 configure, 79
大文字/小文字の区別
 名前における, 502
 識別子における, 502
大文字と小文字を区別する
 テーブル名の, 19
 データベース名の, 19
大文字小文字の区別
 文字列比較, 587
定数テーブル, 377, 385
実行
 ANSIモード, 18
 複数サーバ, 306
小数点, 541
幅
 ディスプレイ, 541
式のエイリアス, 699
式エイリアス, 648
引用, 499
引用符
 文字列内, 498
復帰改行(\r), 498, 498
必要とする記憶容量
 データタイプ, 563
性能
 ベンチマーク, 374
 向上, 412

 推定, 384
情報関数, 634
手がかり, 19
抑制
 デフォルト値, 27
拡張子
 標準SQLに対する, 17
挿入
 の速度, 403
 同時, 408, 411
排他的論理和
 ビット単位, 630
接続
 SSH/遠隔, 274
 サーバへ, 246
 確認, 247
接続文字列, 894
推定
 クエリパフォーマンス, 384
改行 (\n), 498
改行(\n), 498
数値, 499
数値タイプ, 564
数学関数, 593
整数, 499
整数演算, 1127
文字
 マルチ バイト, 293
文字セット, 78
文字列
 エスケープ文字, 497
 区切り文字が無い, 552
 確定, 497
文字列タイプ, 557
文字列リテラルイントロデューサ, 497, 518
文字列比較
 大文字小文字の区別, 587
文字列比較関数, 587
文字列照合, 292
文字列置き換え
 replace ユーティリティ, 495
文字列関数, 578
新しいユーザーの
 追加, 73, 76
日付
 パーティショニングと使用, 998
 パーティショニングと使用される(例), 1004, 1007, 1023
 パーティショニングにおいて使用される(例), 1001
日付と時刻タイプ, 550
日付値
 問題, 553
日付時刻関数, 599
明示的なデフォルト値, 547
時刻タイプ, 564
暗号化, 267
暗号化関数, 630
暗黙のデフォルト値, 547
更新
 MySQL のリリース, 37
書き込みアクセス

- tmp, 94
- 最大数
 - 各結合の最大テーブル数, 1709
- 最後の列
 - ユニークID, 1227
- 最新の
 - バージョン, 43
- 最適化, 384, 389
 - DISTINCT, 391
 - GROUP BY, 401
 - LEFT JOIN, 392
 - LIMIT, 402
 - テーブル, 283
 - ヒント, 405
 - ファイルソート, 400
- 有効な数値
 - の例, 499
- 条件, 1061
- 桁, 541
- 検索
 - MySQL ウェブページ, 14
 - 全文, 613
- 概要, 1
- 構文
 - 正規表現, 589
- 構築
 - クライアントプログラム, 1228
- 標準SQL
 - に対する拡張子, 17, 19
- 標準互換性, 17
- 権限
 - アクセス, 238
 - デフォルト, 99
 - ドロップ, 264
 - ドロップする, 734
 - 供与する, 734
 - 削除, 264
 - 削除する, 734, 741
 - 変更, 252
 - 表示, 761
 - 追加, 261
- 権限の変更, 252
- 権限システム, 239
 - 説明, 239
- 権限テーブル, 252
 - アップグレード, 229
 - ソート, 249, 250
- 権限情報
 - 位置, 243
- 機能
 - C API, 1144
 - C準備されたステートメントAPI, 1198
- 正規表現の構文, 589
- 比較演算子, 572
- 法的名, 500
- 派生テーブル, 715
- 浮動小数点, 499
- 浮動小数点数, 544
- 減算 (-), 592
- 演算
 - 算術, 592
- 演算子, 569
 - キャスト, 592, 624
 - 優先順位, 570, 570
 - 論理, 575
- 演算関数, 629
- 無効値
 - 制約, 27
- 照合
 - 文字列, 292
- 環境変数, 147, 253, 437
 - CC, 77, 77, 83, 133
 - CFLAGS, 77, 83, 133
 - CXX, 77, 77, 83, 83, 83, 133
 - CXXFLAGS, 77, 83, 133
 - DBI_TRACE, 133
 - DBI_USER, 133
 - HOME, 133, 456
 - LD_LIBRARY_PATH, 136
 - LD_RUN_PATH, 109, 116, 133, 136
 - MYSQL_DEBUG, 133, 437
 - MYSQL_GROUP_SUFFIX, 133
 - MYSQL_HISTFILE, 133, 456
 - MYSQL_HOME, 133
 - MYSQL_HOST, 133, 246
 - MYSQL_PS1, 133
 - MYSQL_PWD, 133, 246, 437
 - MYSQL_TCP_PORT, 133, 310, 311, 437
 - MYSQL_UNIX_PORT, 94, 133, 310, 311, 437
 - PATH, 73, 133, 141
 - TMPDIR, 94, 133
 - TZ, 133
 - UMASK, 133
 - UMASK_DIR, 133
 - USER, 133, 246
 - リスト, 133
- 発音
 - MySQL, 5
- 移植性, 372
- 等しい (=), 572
- 等しくない (!=), 572
- 等しくない (<>), 572
- 算術演算, 592
- 精密値リテラル, 1127
- 精密計算, 1127
- 精度
 - 算数, 1127
- 縮小
 - データサイズ, 412
- 行
 - ロック, 24
- 行オプション
 - ndb_config, 952
- 行レベルロック, 408
- 表示する
 - テーブル ステータス, 771
- 情報
 - SHOW, 752, 754, 761, 762, 772
 - 濃度, 762
 - 照合, 762

- 表記規則, 2
- 複合インデックス, 413
- 複合パーティショニング, 1007
- 複数サーバ, 306
- 複数サーバの起動, 306
- 複数部分のインデックス, 660
- 複製スレーブ
 - ステートメント, 783
- 複製マスタ
 - ステートメント, 780
- 規制
 - サブクエリ, 1705
 - サーバサイドカーソル, 1705
 - ストアド ルーチン, 1703
 - トリガ, 1703
 - ビュー, 1707
- 規則
 - 表記, 2
- 角カッコ, 541
- 言語サポート
 - エラー メッセージ, 290
- 計算, 1127
- 設定
 - MySQL Cluster, 917
 - インストール後, 89
 - パスワード, 265
- 設定による
 - MySQL Cluster の制限, 985
- 設定オプション, 76
 - with-low-memory, 82
- 設定ファイル, 253
- 設計
 - 選択, 411
- 設計上
 - 制約, 372
- 許可確認の
 - スピードへの影響, 375
- 論理演算子, 575
- 論理積
 - ビット単位, 630, 630
- 識別子, 500
 - 大文字/小文字の区別, 502
 - 引用, 501
- 識別子の引用, 501
- 負の値, 499
- 質問
 - 回答, 13
- 質問に答える
 - 工チケット, 13
- 起動
 - mysqld, 238
- 近似値リテラル, 1127
- 追加
 - キャラクタ セット, 291
 - 新規ユーザの権限, 261, 261
- 速度
 - 挿入, 403
- 適合性
 - ODBC, 571, 573
- 開始

- コメント, 26
- 開発ソース ツリー, 80
- 関数, 569
 - GROUP BY, 642
 - その他, 640
 - キャスト, 624
 - グループ分け, 570
 - ビット, 629
 - 制御フロー, 577
 - 情報, 634
 - 数学, 593
 - 文字列, 578
 - 文字列比較, 587
 - 日付時刻, 599
 - 暗号化, 630
 - 演算, 629
- 関数名
 - あいまいさの解消, 505
 - 構文解析, 505
- 除算 (/), 593
- 電子メール リスト, 11
- 静的
 - コンパイル, 77
- 非同期のレプリケーション, 962
- 高速化
 - パフォーマンス, 358, 360
- 5.1へのアップグレード, 102
- MySQL Cluster でのバックアップ, 944

A

- aborted clients, 1499
- aborted connection, 1499
- ABS(), 593
- access denied errors, 1492
- ACID, 22, 806
- ACL, 238
- ACOS(), 594
- Active Server Pages (ASP), 1310
- ActiveState Perl, 135
- add-drop-database option
 - mysqldump, 478
- add-drop-table option
 - mysqldump, 478
- add-locks option
 - mysqldump, 478
- add-user オプション
 - mysqlmanager, 217
- ADDDATE(), 600
- adding
 - native functions, 1455
 - new functions, 1446
 - procedures, 1456
 - user-defined functions, 1448
- ADDTIME(), 600
- addtodest option
 - mysqlhotcopy, 486
- administration
 - server, 464
- AES_DECRYPT(), 631

AES_ENCRYPT(), 631
alias, 1512
ALL, 702
all-databases option
 mysqlcheck, 475
 mysqldump, 478
all-in-1 option
 mysqlcheck, 475
all-tablespaces option
 mysqldump, 478
allow-keywords option
 mysqldump, 478
allow-suspicious-udfs オプション
 mysqld, 153, 236
allowold option
 mysqlhotcopy, 486
ALLOW_INVALID_DATES SQL モード, 205
ALTER COLUMN, 652
ALTER DATABASE, 650
ALTER EVENT, 1083
ALTER FUNCTION, 1058
ALTER LOGFILE GROUP, 657
ALTER PROCEDURE, 1058
ALTER SCHEMA, 650
ALTER SERVER, 659
ALTER TABLE, 650, 653, 1515
ALTER TABLESPACE, 658
ALTER VIEW, 1091
analyze option
 mysqlcheck, 475
ANALYZE TABLE, 742
analyze オプション
 myisamchk, 444
AND
 論理, 576
angel-pid-file オプション
 mysqlmanager, 217
ANSI SQL モード, 205, 208
ansi オプション
 mysqld, 153
ANSI_QUOTES SQL モード, 205
ANSIモード
 実行, 18
API, 1135
API ノード, 929
API ノード (MySQL Cluster)
 の定義, 875
API's
 list of, 1721
APIs
 Perl, 1230
apply_status テーブル (廃番), 966
ArbitrationDelay, 897, 913
ArbitrationRank, 896, 913
ArbitrationTimeout (DEPRECATED), 910
arbitrator, 1471, 1477
ARCHIVE ストレージエンジン, 793, 866
Area(), 1045, 1046
argument processing, 1451
AS, 699, 703

AsBinary(), 1041
ASCII(), 578
ASIN(), 594
AsText(), 1042
ATAN(), 594
ATAN2(), 594
auto-generate-sql オプション
 mysqlslap, 492, 493, 493
AUTO-INCREMENT
 ODBC, 1306
auto-rehash オプション
 mysql, 452
auto-repair option
 mysqlcheck, 475
autoclose オプション
 mysqld_safe, 211
AUTO_INCREMENT
 and NULL values, 1512
AVG(), 643
AVG(DISTINCT), 643

B

BACKUP TABLE, 743
backup オプション
 myisamchk, 443
 myisampack, 447
BackupDataBufferSize, 948
BackupDataBufferSize (MySQL Cluster 設定パラメータ), 912
BackupDataDir, 898
BackupLogBufferSize, 912, 948
BackupMaxWriteSize, 912, 949
BackupMemory, 912, 949
backups
 databases and tables, 477, 485
BackupWriteSize, 912, 949
base64-output option
 mysqlbinlog, 469
basedir オプション
 mysqld, 153
 mysqld_safe, 211
 mysql_upgrade, 231
batch オプション
 mysql, 452
BatchByteSize, 913
BatchSize, 913
BatchSizePerLocalScan, 903
BdMPolyFromText(), 1038
BdMPolyFromWKB(), 1039
BdPolyFromText(), 1038
BdPolyFromWKB(), 1039
BEGIN, 724, 1059
 XA トランザクション, 731
BENCHMARK(), 634
BETWEEN ...AND, 573
big5, 1478, 1481
Big5 Chinese character encoding, 1510
BIGINTデータタイプ, 543
BIN(), 578
BINARY, 624

- binary distributions
 - on Linux, 107
- BINARYデータタイプ, 546, 558
- bind-address オプション
 - mysqld, 153
 - mysqlmanager, 218
- binlog-do-db オプション
 - mysqld, 302
- binlog-format オプション
 - mysqld, 153
- binlog-ignore-db オプション
 - mysqld, 302
- binlog-row-event-max-size オプション
 - mysqld, 153
- binlog_index テーブル (廃番), 965
- BitKeeper ツリー, 80
- BIT_AND(), 643
- BIT_COUNT(), 630
- BIT_LENGTH(), 579
- BIT_OR(), 643
- BIT_XOR(), 643
- BITデータタイプ, 542
- BLACKHOLE ストレージエンジン, 793, 869
- BLOB
 - サイズ, 565
 - バイナリデータの挿入, 499
- BLOB カラム
 - インデックス, 666
 - デフォルト値, 560
- BLOBカラム
 - インデックス, 413
- BLOBデータタイプ, 547, 559
- block-search オプション
 - myisamchk, 444
- BOOLEANデータタイプ, 542
- BOOLデータタイプ, 542
- bootstrap オプション
 - mysqld, 153
- Borland Builder 4, 1310
- Borland C++ コンパイラ, 88
- Boundary(), 1043
- brief option
 - mysqlaccess, 463
- Buffer(), 1047
- bugs
 - known, 1517
- bugs.mysql.com, 14

C

- C API
 - データタイプ, 1140
 - リンク問題, 1227
 - 機能, 1144
- C++ APIs, 1231
- C++ Builder, 1311
- C++ コンパイラ
 - gcc, 77
- C++ コンパイラが実行ファイルを作成できません。 , 83
- C:\my.cnf file, 311

- CACHE INDEX, 776
- CALL, 1058
- calling sequences for aggregate functions
 - UDF, 1450
- calling sequences for simple functions
 - UDF, 1449
- can't create/write to file, 1501
- CASE, 577, 1064
- case sensitivity
 - in searches, 1510
- CAST, 624
- CC 環境変数, 77, 77, 83, 133
- cc1plus 問題, 82
- CEILING(), 594
- Centroid(), 1046
- CFLAGS 環境変数, 77, 83, 133
- CHANGE MASTER TO, 783
 - MySQL Cluster, 967
- ChangeLog, 1564
- changes
 - log, 1564
 - MySQL 5.1, 1564
- changing
 - column order, 1516
 - table, 1515
- changing socket location, 1509
- CHAR VARYINGデータタイプ, 546
- CHAR(), 579
- CHARACTER VARYINGデータタイプ, 546
- character-set-client-handshake オプション
 - mysqld, 153
- character-set-filesystem オプション
 - mysqld, 153
- character-set-server オプション
 - mysqld, 154
- character-sets-dir option
 - mysqladmin, 467
 - mysqlbinlog, 469
 - mysqlcheck, 475
 - mysqldump, 478
 - mysqlexport, 488
 - mysqlshow, 490
- character-sets-dir オプション
 - myisamchk, 443
 - myisampack, 447
 - mysql, 452
 - mysqld, 153
- CHARACTER_LENGTH(), 579
- CHARACTER_SETS
 - INFORMATION_SCHEMA テーブル, 1106
- CHARACTERデータタイプ, 546
- CHARSET(), 635
- CHAR_LENGTH(), 579
- CHARデータタイプ, 546, 557
- check option
 - mysqlcheck, 475
- CHECK TABLE, 743
- check オプション
 - myisamchk, 442, 442
- check-only-changed オプション

myisamchk, 442
check-password-file オプション
 mysqlmanager, 218
check-upgrade option
 mysqlcheck, 475
checkpoint option
 mysqlhotcopy, 486
CHECKPOINT イベント (MySQL Cluster), 940
Checksum, 914
Checksum (MySQL Cluster), 916, 917
CHECKSUM TABLE, 745
Chinese, 1510
Chinese, Japanese, Korean character sets
 frequently asked questions, 1478
chroot オプション
 mysqld, 154
CJK
 FAQ, 1478
CJK (Chinese, Japanese, Korean)
 Access, PHP, etc., 1479, 1483
 availability of specific characters, 1479, 1486
 available character sets, 1479, 1485
 big5, 1478, 1481
 character sets available, 1479, 1485
 characters displayed as question marks, 1478
 CJKV, 1479, 1488
 collations, 1479, 1479, 1487, 1488
 conversion problems with Japanese character sets, 1478, 1481
 data truncation, 1478, 1482
 Database and table names, 1479, 1489
 documentation in Chinese, 1479, 1489
 documentation in Japanese, 1479, 1489
 documentation in Korean, 1479, 1489
 gb2312, gbk, 1478, 1480
 Japanese character sets, 1478, 1481
 Korean character set, 1478, 1482
 LIKE and FULLTEXT, 1479, 1485
 MySQL 4.0 behavior, 1479, 1484
 ORDER BY treatment, 1479, 1479, 1487, 1488
 problems with Access, PHP, etc., 1479, 1483
 problems with Big5 character sets (Chinese), 1478, 1481
 problems with data truncation, 1478, 1482
 problems with euckr character set (Korean), 1478, 1482
 problems with GB character sets (Chinese), 1478, 1480
 problems with LIKE and FULLTEXT, 1479, 1485
 problems with Yen sign (Japanese), 1478, 1482
 rejected characters, 1479, 1488
 sort order problems, 1479, 1479, 1487, 1488
 sorting problems, 1479, 1479, 1487, 1488
 testing availability of characters, 1479, 1486
 Unicode collations, 1479, 1488
 Vietnamese, 1479, 1488
 Yen sign, 1478, 1482
clean-password-file オプション
 mysqlmanager, 218
CLOSE, 1063
cluster データベース (廃番), 965
cluster.binlog_index テーブル (廃番), 965
CLUSTERLOG STATISTICS コマンド (MySQL Cluster), 943
CLUSTERLOG コマンド (MySQL Cluster), 939
cluster_replication データベース (廃番), 965
CMake, 86
COALESCE(), 574
COERCIBILITY(), 635
ColdFusion, 1311
COLLATION(), 635
collation-server オプション
 mysqld, 154
COLLATIONS
 INFORMATION_SCHEMA テーブル, 1106
COLLATION_CHARACTER_SET_APPLICABILITY
 INFORMATION_SCHEMA テーブル, 1106
column-names オプション
 mysql, 452
columns
 changing, 1516
 displaying, 490
COLUMNS
 INFORMATION_SCHEMA テーブル, 1102
columns option
 mysqlimport, 488
COLUMN_PRIVILEGES
 INFORMATION_SCHEMA テーブル, 1105
command options
 mysqladmin, 467
command-line history
 mysql, 456
command-line tool, 452
commands out of sync, 1502
comments option
 mysqldump, 478
COMMIT, 22, 724
 XA トランザクション, 731
commit option
 mysqlaccess, 463
compact option
 mysqldump, 479
compatible option
 mysqldump, 479
compiling
 user-defined functions, 1453
complete-insert option
 mysqldump, 479
compress option
 mysqlcheck, 475
 mysqldump, 479
 mysqlimport, 488
 mysqlshow, 490
compress オプション
 mysql, 452
 mysqslap, 492
COMPRESS(), 631
CONCAT(), 579
CONCAT_WS(), 580
concurrency オプション
 mysqslap, 492
config-file option
 ndb_config, 951
config-file オプション

- mysqld_multi, 214
- my_print_defaults, 437, 437
- config.cache, 82
- config.cache ファイル, 82
- config.ini (MySQL Cluster), 883, 892, 892,
- configure
 - disable-grant-options option, 79
 - enable-thread-safe-client option, 79
 - previx オプション, 77, 77
 - with-unix-socket-path option, 77, 77
 - サーバオプションなし, 77
 - デバッグ オプション付き, 79
 - 予備の文字セット付き, 78, 78
 - 埋め込みサーバオプション付き, 77
 - 大きなテーブルのオプション付き, 79
 - 文字セット付き, 78
 - 照合オプション付き, 78
- configure スクリプト, 76
- connection
 - aborted, 1499
- CONNECTION イベント (MySQL Cluster), 940
- CONNECTION_ID(), 635
- Connector/JDBC, 1235
- Connector/MXJ, 1235
- Connector/NET, 1235, 1316
 - 問題のレポート, 1375
- Connector/ODBC, 1235, 1236
 - Borland, 1310
 - Borland Database Engine, 1310
 - 問題を報告, 1315, 1315
- connect_timeout variable, 456, 468
- console オプション
 - mysqld, 154
- CONSTRAINTS
 - INFORMATION_SCHEMA テーブル, 1106
- Contains(), 1049
- contributing companies
 - list of, 1722
- contributors
 - list of, 1716
- CONV(), 580
- CONVERT, 624
- CONVERT TO, 654
- CONVERT_TZ(), 600
- ConvexHull(), 1047
- copy option
 - mysqlaccess, 463
- core-file オプション
 - mysqld, 154
- core-file-size オプション
 - mysqld_safe, 211
- correct-checksum オプション
 - myisamchk, 443
- COS(), 594
- COT(), 595
- count option
 - mysqladmin, 467
 - mysqlshow, 490
- count オプション
 - myisam_ftdump, 439

- COUNT(), 643
- COUNT(DISTINCT), 643
- crash
 - repeated, 1506
- crash-me, 374
- crash-me プログラム, 372, 374
- CRC32(), 595
- CREATE DATABASE, 659
- CREATE EVENT, 1080
- CREATE FUNCTION, 1054, 1447
- CREATE INDEX, 660
- CREATE LOGFILE GROUP, 675
- CREATE PROCEDURE, 1054
- CREATE SCHEMA, 659
- CREATE SERVER, 677
- CREATE TABLE, 661
- CREATE TABLESPACE, 676
- CREATE TRIGGER, 1071
- CREATE USER, 734
- CREATE VIEW, 1091
- create オプション
 - mysqlslap, 492
- create-options option
 - mysqldump, 479
- create-schema オプション
 - mysqlslap, 492
- creating
 - function, 1447
- CROSS JOIN, 703
- Crosses(), 1049
- CR_SERVER_GONE_ERROR, 1497
- CR_SERVER_LOST_ERROR, 1497
- csv オプション
 - mysqlslap, 492
- CSV ストレージエンジン, 793, 867
- CSV データ、読み込み, 693, 702
- CURDATE(), 600
- CURRENT_DATE, 600
- CURRENT_TIME, 601
- CURRENT_TIMESTAMP, 601
- CURRENT_USER(), 635
- CURTIME(), 600
- CXX 環境変数, 77, 77, 83, 83, 83, 133
- CXXFLAGS 環境変数, 77, 83, 133
- C準備されたステートメントAPI
 - 機能, 1198

D

- data
 - importing, 487
- Data truncation with CJK characters, 1478, 1482
- data type
 - GEOMETRY, 1037
 - GEOMETRYCOLLECTION, 1037
 - LINestring, 1037
 - MULTILINESTRING, 1037
 - MULTIPOINT, 1037
 - MULTIPOLYGON, 1037
 - POINT, 1037

POLYGON, 1037
data-file-length オプション
 myisamchk, 443
database option
 mysqlbinlog, 469
database オプション
 mysql, 453
DATABASE(), 636
databases
 displaying, 490
 dumping, 477, 485
databases option
 mysqlcheck, 475
 mysqldump, 479
DataDir, 897, 898
datadir オプション
 mysqld, 154
 mysqld_safe, 211
 mysql_upgrade, 231
DataJunction, 1311
DataMemory, 899, 923
DATE, 1510
DATE columns
 problems, 1510
DATE(), 601
DATEDIFF(), 601
DATETIMEデータタイプ, 544, 551
DATE_ADD(), 601
DATE_FORMAT(), 603
DATE_SUB(), 601, 604
DATEデータタイプ, 544, 551
DAY(), 604
DAYNAME(), 604
DAYOFMONTH(), 604
DAYOFWEEK(), 604
DAYOFYEAR(), 604
db option
 mysqlaccess, 463
db テーブル
 ソート, 250
DB2 SQL モード, 208
DBI インターフェース, 1231
DBI->quote, 499
DBI/DBD インターフェース, 1231
DBI_TRACE 環境変数, 133
DBI_USER 環境変数, 133
DEALLOCATE PREPARE, 790, 791
debug option
 mysqlaccess, 463
 mysqladmin, 467
 mysqlbinlog, 469
 mysqlcheck, 475
 mysqldump, 479
 mysqlhotcopy, 486
 mysqlimport, 488
 mysqlshow, 490
debug オプション
 make_win_bin_dist, 229
 myisamchk, 440
 myisampack, 447
 mysql, 453
 mysqld, 154
 mysqlmanager, 218
 mysqlslap, 492
 my_print_defaults, 437
debug-info オプション
 mysql, 453
DECIMAL データタイプ, 1127
DECIMALデータタイプ, 544
DECLARE, 1060
DECODE(), 632
decode_bits myisamchk 変数, 440
DECデータタイプ, 544
DEFAULT
 制約, 27
DEFAULT 値条項, 665
DEFAULT(), 640
default-character-set option
 mysqladmin, 467
 mysqlcheck, 475
 mysqldump, 479
 mysqlimport, 488
 mysqlshow, 490
default-character-set オプション
 mysql, 453
 mysqld, 154
default-collation オプション
 mysqld, 154
default-mysqld-path オプション
 mysqlmanager, 218
default-storage-engine オプション
 mysqld, 154
default-table-type オプション
 mysqld, 154
default-time-zone オプション
 mysqld, 155
defaults-extra-file オプション, 147
 mysqld_safe, 211
 my_print_defaults, 438
defaults-file オプション, 147
 mysqld_safe, 211
 mysqlmanager, 218
defaults-group-suffix オプション
 my_print_defaults, 438
defaults-group-suffix=str オプション, 147
DEFAULT値条項, 547
DEGREES(), 595
delay-key-write オプション
 mysqld, 155, 801
DELAYED, 688
delayed-insert option
 mysqldump, 479
delayed_insert_limit, 689
DELETE, 681
 および MySQL Cluster,
delete option
 mysqlimport, 488
delete-master-logs option
 mysqldump, 479
deleting

- function, 1447
- rows, 1513
- deletion
 - mysql.sock, 1509
- delimiter オプション
 - mysql, 453
 - mysqlslap, 492
- Delphi, 1310
- des-key-file オプション
 - mysqld, 155
- DESC, 721
- DESCRIBE, 721
- description オプション
 - myisamchk, 444
- design
 - issues, 1517
- DES_DECRYPT(), 632
- DES_ENCRYPT(), 632
- developers
 - list of, 1711
- Difference(), 1047
- Dimension(), 1042
- disable-grant-options option
 - configure, 79
- disable-keys option
 - mysqldump, 479
- disable-log-bin option
 - mysqlbinlog, 469
- DISCARD TABLESPACE, 654, 811
- Disjoint(), 1049
- disk full, 1508
- DiskCheckpointSpeed, 909
- DiskCheckpointSpeedInRestart, 909
- Diskless, 906
- DiskPageBufferMemory, 978
- DiskSyncSize, 909
- displaying
 - database information, 490
- Distance(), 1049
- DISTINCT, 391, 702
 - AVG(), 643
 - COUNT(), 643
 - MAX(), 644
 - MIN(), 644
 - SUM(), 645
- DISTINCTROW, 702
- DIV, 593
- DNS, 431
- DO, 683
- DocBook XML
 - 文書ソース フォーマット, 2
- Documentation
 - in Chinese, 1479, 1489
 - in Japanese, 1479, 1489
 - in Korean, 1479, 1489
- Documenters
 - list of, 1719
- DOUBLE PRECISIONデータタイプ, 544
- DOUBLEデータタイプ, 543
- DROP DATABASE, 677

- DROP EVENT, 1085
- DROP FOREIGN KEY, 653, 824
- DROP FUNCTION, 1058, 1447
- DROP INDEX, 653, 678
- DROP LOGFILE GROUP, 679
- DROP PREPARE, 791
- DROP PRIMARY KEY, 653
- DROP PROCEDURE, 1058
- DROP SCHEMA, 677
- DROP SERVER, 679
- DROP TABLE, 678
 - および MySQL Cluster,
- DROP TABLESPACE, 679
- DROP TRIGGER, 1074
- DROP USER, 734
- DROP VIEW, 1097
- drop-user オプション
 - mysqlmanager, 218
- dryrun option
 - mysqlhotcopy, 486
- DUAL, 699
- dump オプション
 - myisam_ftdump, 439
- DUMPFIL, 702
- dumping
 - databases and tables, 477, 485

E

- edit-user オプション
 - mysqlmanager, 218
- ELT(), 580
- embedded オプション
 - make_win_bin_dist, 229
- enable-named-pipe オプション
 - mysqld, 155
- enable-thread-safe-client option
 - configure, 79
- ENCODE(), 632
- ENCRYPT(), 633
- END, 1059
- EndPoint(), 1044
- engine オプション
 - mysqlslap, 492
- ENGINES
 - INFORMATION_SCHEMA テーブル, 1111
- ENTER SINGLE USER MODE コマンド (MySQL Cluster),

,

- ENUM
 - サイズ, 566
- ENUMデータタイプ, 547, 560
- Envelope(), 1042
- Environment variable
 - TZ, 1510
 - UMASK, 1504
 - UMASK_DIR, 1504
- Equals(), 1049
- Errcode, 494
- errno:, 494
- error messages

- can't find file, 1504
- ERROR イベント (MySQL Cluster), 942
- errors
 - access denied, 1492
 - common, 1491
 - handling for UDFs, 1452
 - known, 1517
 - linking, 1503
 - list of, 1492
- ERROR_FOR_DIVISION_BY_ZERO SQL モード, 205
- Event Scheduler, 1077
 - altering events, 1083
 - and MySQL privileges, 1086
 - and mysqladmin debug, 1085
 - and SHOW PROCESSLIST, 1078
 - concepts, 1077
 - creating events, 1080
 - dropping events, 1085
 - enabling and disabling, 1078
 - event metadata, 1085
 - obtaining status information, 1085
 - SQL statements, 1080
 - starting and stopping, 1078
- event-scheduler オプション
 - mysqld, 155
- events, 1077
 - altering, 1083
 - creating, 1080
 - dropping, 1085
 - limitations, 1088
 - metadata, 1085
 - status variables, 1088
- EVENTS
 - INFORMATION_SCHEMA table, 1086
 - INFORMATION_SCHEMA テーブル, 1114
- events option
 - mysqldump, 480
- example オプション
 - mysqld_multi, 214
- EXAMPLE ストレージエンジン, 793, 862
- examples
 - 圧縮テーブル, 448
- exe-suffix オプション
 - make_win_bin_dist, 229
- EXECUTE, 790, 791
- execute オプション
 - mysql, 453
- ExecuteOnComputer, 896, 897, 913
- EXIT コマンド (MySQL Cluster),
- exit-info オプション
 - mysqld, 155
- EXP(), 595
- EXPLAIN, 375
- EXPLAIN PARTITIONS, 1020, 1021
- EXPLAIN をパーティショニングされたテーブルと使用する, 1020
- EXPORT_SET(), 580
- extend-check オプション
 - myisamchk, 442, 443
- extended option

- mysqlcheck, 475
- extended-insert option
 - mysqldump, 480
- ExteriorRing(), 1045
- extra-file オプション
 - my_print_defaults, 438
- extra-partition-info option
 - ndb_desc, 953
- EXTRACT(), 604
- ExtractValue(), 626

F

- FALSE, 499, 500
 - テスト, 573
- fast option
 - mysqlcheck, 476
- fast オプション
 - myisamchk, 442
- fatal signal 11, 82
- FEDERATED ストレージエンジン, 793, 862
- FETCH, 1063
- FIELD(), 580
- fields-enclosed-by option
 - mysqldump, 480, 488
- fields-escaped-by option
 - mysqldump, 480, 488
- fields-optionally-enclosed-by option
 - mysqldump, 480, 488
- fields-terminated-by option
 - mysqldump, 480, 488
- FILE, 582
- files
 - not found message, 1504
 - permissions, 1504
 - size limits, 1500
 - text, 487
- FILES
 - INFORMATION_SCHEMA テーブル, 1117
- FilePath, 898
- FIND_IN_SET(), 581
- first-slave option
 - mysqldump, 480
- fix-db-names option
 - mysqlcheck, 476
- fix-table-names option
 - mysqlcheck, 476
- FIXEDデータタイプ, 544
- FLOATデータタイプ, 543, 543, 544
- FLOOR(), 595
- FLUSH, 777
- flush tables, 466
- flush オプション
 - mysqld, 155
- flush-logs option
 - mysqldump, 480
- flush-privileges option
 - mysqldump, 480
- flushlog option
 - mysqlhotcopy, 486

FOR UPDATE, 702
FORCE INDEX, 699, 705, 1515
FORCE KEY, 699, 705
force option
 mysql, 453
 mysqladmin, 467
 mysqlcheck, 476
 mysqldump, 480
 mysqlimport, 488
force オプション
 myisamchk, 442, 443
 myisampack, 447
 mysql_upgrade, 231
force-read option
 mysqlbinlog, 470
FORMAT(), 581
FOUND_ROWS(), 636
FreeBSD のトラブルシューティング, 83
frequently-asked questions about MySQL Cluster, 1470
FROM, 699
FROM_DAYS(), 605
FROM_UNIXTIME(), 605
ft_max_word_len myisamchk 変数, 440
ft_min_word_len myisamchk 変数, 440
ft_stopword_file myisamchk 変数, 440
full disk, 1508
FULLTEXT, 613
function
 creating, 1447
 deleting, 1447
functions
 native
 adding, 1455
 new, 1446
 user-defined, 1446
 adding, 1448
Functions
 user-defined, 1447, 1447

G

gb2312, gbk, 1478, 1480
gcc, 77
general-log オプション
 mysqld, 155
geographic feature, 1030
GeomCollFromText(), 1037
GeomCollFromWKB(), 1038
geometry, 1030
GEOMETRY data type, 1037
GEOMETRYCOLLECTION data type, 1037
GeometryCollection(), 1039
GeometryCollectionFromText(), 1037
GeometryCollectionFromWKB(), 1038
GeometryFromText(), 1037
GeometryFromWKB(), 1038
GeometryN(), 1046
GeometryType(), 1042
GeomFromText(), 1037, 1042
GeomFromWKB(), 1038, 1042

geospatial feature, 1030
GET_FORMAT(), 605
GET_LOCK(), 640
GIS, 1029, 1030
GLength(), 1044, 1045
GLOBAL_STATUS
 INFORMATION_SCHEMA テーブル, 1123
GLOBAL_VARIABLES
 INFORMATION_SCHEMA テーブル, 1123
GRANT, 734
GRANT ステートメント, 261
GRANTS, 761
GREATEST(), 574
GROUP BY, 401
 エイリアス, 648
 スタンダード SQL の拡張子, 700
 標準 SQL の拡張機能, 647
GROUP BY 関数, 642
GROUP_CONCAT(), 644

H

HANDLER, 683
handling
 errors, 1452
HAVING, 700
HEAP ストレージエンジン, 793, 860
HeartbeatIntervalDbApi (MySQL Cluster 設定パラメータ), 907
HeartbeatIntervalDbDb (MySQL Cluster 設定パラメータ), 907
help option
 mysqlaccess, 463
 mysqladmin, 467
 mysqlbinlog, 469
 mysqlcheck, 475
 mysqldump, 478
 mysqlhotcopy, 486
 mysqlimport, 487
 mysqlshow, 490
 mysqslap, 492
 perror, 495
help オプション
 myisamchk, 440
 myisampack, 447
 myisam_ftdump, 438
 mysql, 452
 mysqld, 153
 mysqld_multi, 214
 mysqld_safe, 211
 mysqlmanager, 217
 mysql_upgrade, 231
 my_print_defaults, 437
HELP コマンド (MySQL Cluster),
HELP ステートメント, 721
HEX(), 581
hex-blob option
 mysqldump, 480
hexdump option
 mysqlbinlog, 470
HIGH_NOT_PRECEDENCE SQL モード, 205
HIGH_PRIORITY, 703

HOME 環境変数, 133
HOME環境変数, 456
host option
 mysql, 453
 mysqlaccess, 463
 mysqladmin, 467
 mysqlbinlog, 470
 mysqlcheck, 476
 mysqldump, 480
 mysqlhotcopy, 486
 mysqlimport, 488
 mysqlshow, 491
host オプション
 mysqlslap, 492
host テーブル, 251
 ソート, 250
Host*Scild* parameters, 916
host.frm
 problems finding, 91
HostName, 896, 897, 913
HOUR(), 606
howto option
 mysqlaccess, 463
html option
 mysql, 453

I

i-am-a-dummy option
 mysql, 455
Id, 895, 897, 913
ID
 ユニーク, 1227
id オプション
 ndb_config, 951
IF, 1064
IF(), 577
IFNULL(), 578
IGNORE INDEX, 699, 705
IGNORE KEY, 699, 705
ignore option
 mysqlimport, 488
ignore-lines option
 mysqlimport, 488
ignore-spaces option
 mysql, 453
ignore-table option
 mysqldump, 480
IGNORE_SPACE SQL モード, 206
IMPORT TABLESPACE, 654, 811
importing
 data, 487
IN, 574
IndexMemory, 899, 923
INET_ATON(), 640
INET_NTOA(), 641
INFO イベント (MySQL Cluster), 943
information オプション
 myisamchk, 442
INFORMATION_SCHEMA, 1099

init-file オプション
 mysqld, 156
INNER JOIN, 703
InnoDB, 806
 Solaris 10 x86_64 の問題, 114
innodb オプション
 mysqld, 812
InnoDB ストレージ エンジン, 806
InnoDB ストレージエンジン, 793
InnoDB テーブル, 22
innodb_status_file オプション
 mysqld, 812
INSERT, 403, 684
INSERT ... SELECT, 687
INSERT DELAYED, 688, 688
INSERT ステートメント
 権限付与, 262
INSERT(), 581
insert-ignore option
 mysqldump, 480
INSTALL PLUGIN, 1433
install オプション
 mysqlmanager, 218
installing
 user-defined functions, 1453
installing plugins, 1433
INSTR(), 582
INTEGERデータタイプ, 543
InteriorRingN(), 1045
internal compiler errors, 82
internals, 1429
Intersection(), 1047
Intersects(), 1049
INTERVAL(), 575
INTデータタイプ, 543
IRC, 13
IS boolean_value, 573
IS NOT boolean_value, 573
IS NOT NULL, 573
IS NULL, 391, 573
 とインデックス, 415
IsClosed(), 1045
IsEmpty(), 1043
ISNULL(), 575
ISOLATION LEVEL, 729
IsRing(), 1045
IsSimple(), 1043
IS_FREE_LOCK(), 641
IS_USED_LOCK(), 641
ITERATE, 1065
iterations オプション
 mysqlslap, 492

J

Japanese character sets
 conversion, 1478, 1481
Japanese, Korean, Chinese character sets
 frequently asked questions, 1478
JOIN, 703

join オプション
 mysampack, 447

K

keepold option
 mysqlhotcopy, 486
keys option
 mysqlshow, 491
keys-used オプション
 mysamchk, 443
key_buffer_size mysamchk 変数, 440
KEY_COLUMN_USAGE
 INFORMATION_SCHEMA テーブル, 1107
KILL, 779
known errors, 1517
Korean, 1478, 1482
Korean, Chinese, Japanese character sets
 frequently asked questions, 1478

L

language オプション
 mysqld, 156
large-pages オプション
 mysqld, 156
LAST_DAY(), 606
LAST_INSERT_ID(), 24, 687
LAST_INSERT_ID() とストアルーチン, 1066
LAST_INSERT_ID() とトリガ, 1066
LAST_INSERT_ID([expr]), 637
LCASE(), 582
LD_LIBRARY_PATH 環境変数, 136
LD_RUN_PATH 環境変数, 109, 116, 133, 136
LEAST(), 575
LEAVE, 1064
ledir オプション
 mysqld_safe, 211
LEFT JOIN, 392, 703
LEFT OUTER JOIN, 703
LEFT(), 582
length オプション
 mysam_ftdump, 439
LENGTH(), 582
libmysqld, 1135
 オプション, 1137
libraries
 list of, 1721
LIKE, 587
 とインデックス, 415
 とワイルドカード, 415
LIMIT, 402, 636, 701
limits
 file-size, 1500
line-numbers option
 mysql, 453
LineFromText(), 1037
LineFromWKB(), 1038
lines-terminated-by option
 mysqldump, 480, 488
LINESTRING data type, 1037

LineString(), 1039
LineStringFromText(), 1037
LineStringFromWKB(), 1038
linking
 errors, 1503
Linux
 ソース ディストリビューション, 108
 バイナリ ディストリビューション, 107
Linux RPM パッケージ
 のインストール, 65
list-users オプション
 mysqlmanager, 218
LN(), 595
LOAD DATA FROM MASTER, 784
LOAD DATA INFILE, 690, 1512
load emulation, 491
LOAD TABLE FROM MASTER, 785
LOAD_FILE(), 582
local option
 mysqlimport, 488
local-infile option
 mysql, 453
local-infile オプション
 mysqld, 236
local-load option
 mysqlbinlog, 470
localstatedir オプション
 configure, 77
LOCALTIME, 606
LOCALTIMESTAMP, 606
LOCATE(), 582
LOCK IN SHARE MODE, 702
LOCK TABLES, 727
lock-all-tables option
 mysqldump, 481
lock-directory オプション
 mysqslap, 492, 493
lock-tables option
 mysqldump, 481
 mysqlimport, 488
LockPagesInMainMemory, 906
log
 changes, 1564
log オプション
 mysqld, 156
 mysqld_multi, 214
 mysqlmanager, 218
LOG(), 595
log-bin オプション
 mysqld, 156
log-bin-index オプション
 mysqld, 156
log-bin-trust-function-creators オプション
 mysqld, 156
log-error オプション
 mysqld, 157
 mysqld_safe, 211
log-isam オプション
 mysqld, 157
log-long-format オプション

mysqld, 157
log-output オプション
 mysqld, 157
log-queries-not-using-indexes オプション
 mysqld, 157
log-short-format オプション
 mysqld, 157
log-slave-updates オプション
 mysqld, 331
log-slow-admin-statements オプション
 mysqld, 157
log-slow-queries オプション
 mysqld, 157
log-tc オプション
 mysqld, 157
log-tc-size オプション
 mysqld, 157
log-warnings オプション
 mysqld, 158, 332
LOG10(), 596
LOG2(), 596
LogDestination, 896
LogLevelCheckpoint (MySQL Cluster 設定パラメータ), 911
LogLevelConnection (MySQL Cluster 設定パラメータ), 911
LogLevelError, 911
LogLevelInfo, 911
LogLevelNodeRestart (MySQL Cluster 設定パラメータ), 911
LogLevelShutdown (MySQL Cluster 設定パラメータ), 911
LogLevelStartup (MySQL Cluster 設定パラメータ), 911
LogLevelStatistic (MySQL Cluster 設定パラメータ), 911
LONGBLOBデータタイプ, 547
LongMessageBuffer, 903
LONGTEXTデータタイプ, 547
LONGデータタイプ, 559
LOOP, 1064
low-priority option
 mysqlimport, 489
low-priority-updates オプション
 mysqld, 158
LOWER(), 582
LPAD(), 583
LTRIM(), 583

M

Mac OS X, 1236
 へのインストール, 67
Mac OS X PKG パッケージ
 のインストール, 67
maintenance
 tables, 474
MAKEDATE(), 606
MAKETIME(), 606
make_binary_distribution, 151
MAKE_SET(), 583
make_win_bin_dist, 151, 228
 debug オプション, 229
 embedded オプション, 229
 exe-suffix オプション, 229
 no-debug オプション, 229
 no-embedded オプション, 229
 only-debug オプション, 229
master-connect-retry オプション
 mysqld, 332
master-data option
 mysqldump, 481
master-host オプション
 mysqld, 332
master-info-file オプション
 mysqld, 332
master-password オプション
 mysqld, 332
master-port オプション
 mysqld, 332
master-retry-count オプション
 mysqld, 332
master-ssl オプション
 mysqld, 332
master-ssl-ca オプション
 mysqld, 332
master-ssl-capath オプション
 mysqld, 332
master-ssl-cert オプション
 mysqld, 332
master-ssl-cipher オプション
 mysqld, 332
master-ssl-key オプション
 mysqld, 332
master-user オプション
 mysqld, 332
MASTER_POS_WAIT(), 641, 786
MATCH ... AGAINST(), 613
MAX(), 644
MAX(DISTINCT), 644
max-record-length オプション
 myisamchk, 443
max-relay-log-size オプション
 mysqld, 332
MAXDB SQL モード, 209
maximum memory used, 467
MaxNoOfAttributes, 904
MaxNoOfConcurrentIndexOperations, 902
MaxNoOfConcurrentOperations, 901
MaxNoOfConcurrentScans, 903
MaxNoOfConcurrentTransactions, 901
MaxNoOfFiredTriggers, 902
MaxNoOfIndexes, 905
MaxNoOfLocalOperations, 902
MaxNoOfLocalScans, 903
MaxNoOfOpenFiles, 904
MaxNoOfOrderedIndexes, 905
MaxNoOfSavedMessages, 904
MaxNoOfTables, 905
MaxNoOfTriggers, 905
MaxNoOfUniqueHashIndexes, 905
MaxScanBatchSize, 913
max_allowed_packet variable, 456
MAX_CONNECTIONS_PER_HOUR, 264
max_join_size variable, 456
MAX_QUERIES_PER_HOUR, 264

MAX_UPDATES_PER_HOUR, 264
MAX_USER_CONNECTIONS, 264
MBR, 1048
MBRContains(), 1048
MBRDisjoint(), 1048
MBREqual(), 1048
MBRIntersects(), 1048
MBROverlaps(), 1048
MBRTouches(), 1048
MBRWithin(), 1048
MD5(), 633
medium-check オプション
 myisamchk, 442
MEDIUMBLOBデータタイプ, 547
MEDIUMINTデータタイプ, 543
MEDIUMTEXTデータタイプ, 547
memlock オプション
 mysqld, 158
MEMORY ストレージエンジン, 793, 860
MERGE ストレージエンジン, 793, 856
MERGEテーブル
 定義された, 856
method option
 mysqlhotcopy, 486
mgm (MySQL Cluster プロセス), 932
mgmd (MySQL Cluster プロセス), 931
MICROSECOND(), 606
Microsoft Access, 1308
Microsoft ADO, 1309
Microsoft Excel, 1309
Microsoft Visual Basic, 1309
Microsoft Visual InterDev, 1309
MID(), 583
MIN(), 644
MIN(DISTINCT), 644
Minimum Bounding Rectangle, 1048
MINUTE(), 607
MIT-pthreads, 84
MLineFromText(), 1038
MLineFromWKB(), 1038
MOD (モジ ュ ロ), 596
MOD(), 596
monitoring-interval オプション
 mysqlmanager, 219
Mono, 1316
MONTH(), 607
MONTHNAME(), 607
MPointFromText(), 1038
MPointFromWKB(), 1038
MPolyFromText(), 1038
MPolyFromWKB(), 1038
mSQL の互換性, 588
msql2mysql, 1232
MSSQL SQL モード, 209
multi mysqld, 213
multi-byte character sets, 1502
MULTILINESTRING data type, 1037
MultiLineString(), 1039
MultiLineStringFromText(), 1038
MultiLineStringFromWKB(), 1038

MULTIPOINT data type, 1037
MultiPoint(), 1039
MultiPointFromText(), 1038
MultiPointFromWKB(), 1038
MULTIPOLYGON data type, 1037
MultiPolygon(), 1039
MultiPolygonFromText(), 1038
MultiPolygonFromWKB(), 1038
My
 由来, 5
mys.cnf
 および MySQL Cluster, 883, 892, 892
mys.cnf ファイル, 352
MyISAM
 サイズ, 564
 圧縮テーブル, 446, 804
MyISAM キーキャッシュ, 416
MyISAM ストレージエンジン, 793, 799
myisam-recover オプション
 mysqld, 158, 801
myisamchk, 78, 435, 439
 analyze オプション, 444
 backup オプション, 443
 block-search オプション, 444
 character-sets-dir オプション, 443
 check オプション, 442
 check-only-changed オプション, 442
 correct-checksum オプション, 443
 data-file-length オプション, 443
 debug オプション, 440
 description オプション, 444
 extend-check オプション, 442, 443
 fast オプション, 442
 force オプション, 442, 443
 help オプション, 440
 information オプション, 442
 keys-used オプション, 443
 max-record-length オプション, 443
 medium-check オプション, 442
 parallel-recover オプション, 443
 quick オプション, 443
 read-only オプション, 442
 recover オプション, 443
 safe-recover オプション, 443
 set-auto-increment[オプション, 444
 set-collation オプション, 444
 silent オプション, 440
 sort-index オプション, 444
 sort-records オプション, 444
 sort-recover オプション, 444
 tmpdir オプション, 444, 444
 unpack オプション, 444
 update-state オプション, 442
 verbose オプション, 440
 version オプション, 440
 wait オプション, 440
 出力例, 284
myisamlog, 436, 445
myisampack, 436, 446, 675, 804

- backup オプション, 447
- character-sets-dir オプション, 447
- debug オプション, 447
- force オプション, 447
- help オプション, 447
- join オプション, 447
- packlength オプション, 447
- silent オプション, 447
- test オプション, 447
- tmpdir オプション, 447
- verbose オプション, 447
- version オプション, 448
- wait オプション, 448
- mysam_block_size mysamchk 変数, 440
- mysam_ftdump, 435, 438
 - count オプション, 439
 - dump オプション, 439
 - help オプション, 438
 - length オプション, 439
 - stats オプション, 439
 - verbose オプション, 439
- MyODBC, 1236
- MySQL
 - の選択, 35
 - のテスト, 36
 - アップグレード, 231
 - リリースの命名規則, 35
 - 定義, 4
 - 発音, 5
 - 紹介, 4
- mysql, 436, 452
 - auto-rehash オプション, 452
 - batch オプション, 452
 - character-sets-dir オプション, 452
 - column-names オプション, 452
 - compress オプション, 452
 - database オプション, 453
 - debug オプション, 453
 - debug-info オプション, 453
 - default-character-set オプション, 453
 - delimiter オプション, 453
 - execute オプション, 453
 - force option, 453
 - help オプション, 452
 - host option, 453
 - html option, 453
 - i-am-a-dummy option, 455
 - ignore-spaces option, 453
 - line-numbers option, 453
 - local-infile option, 453
 - named-commands option, 453
 - no-auto-rehash option, 453
 - no-beep option, 454
 - no-named-commands option, 454
 - no-pager option, 454
 - no-tee option, 454
 - one-database option, 454
 - pager option, 454
 - password option, 454
 - port option, 454
 - prompt option, 454
 - protocol option, 454
 - quick option, 454
 - raw option, 454
 - reconnect option, 454
 - safe-updates option, 455
 - secure-auth option, 455
 - show-warnings option, 455
 - sigint-ignore option, 455
 - silent option, 455
 - skip-column-names option, 455
 - skip-line-numbers option, 455
 - socket option, 455
 - SSL options, 455
 - table option, 455
 - tee option, 455
 - unbuffered option, 455
 - user option, 455
 - verbose option, 455
 - version option, 456
 - vertical option, 456
 - wait option, 456
 - xml option, 456
- MySQL 5.0 および 5.1 の MySQL Cluster, 988
- MySQL AB
 - 定義, 3
- MySQL C タイプ, 1141
- MySQL Cluster, 872
 - administration, 935
 - and transactions, 1471, 1475
 - API ノード, 875, 913
 - arbitrator, 1471, 1477
 - CHECKPOINT イベント, 940
 - CLUSTERLOG STATISTICS コマンド, 943
 - CLUSTERLOG コマンド, 939
 - CONNECTION イベント, 940
 - data types supported, 1471, 1477
 - ENTER SINGLE USER MODE command,
 - error messages, 1471, 1475
 - ERROR イベント, 942
 - EXIT SINGLE USER MODE コマンド, 937
 - EXIT コマンド, 937
 - FAQ, 1470
 - hardware requirements, 1471, 1473
 - HELP コマンド, 937
 - how to obtain, 1471, 1475
 - importing existing tables, 1471, 1476
 - INFO イベント, 943
 - memory requirements, 1471, 1473
 - mgm, 932
 - mgm クライアント, 937
 - mgm プロセス, 932, 935
 - mgm マネジメント クライアント, 943
 - mgmd, 932
 - mgmd プロセス, 931, 934
 - mysqld プロセス, 929, 933
 - ndbd, 932
 - ndbd プロセス, 930, 933, 943
 - ndb_mgm, 885
 - ndb_size.pl (utility), 1473

networking requirements, 1471, 1471, 1472, 1474
node types, 1472
NODERESTART イベント, 941
number of computers required, 1471, 1472
platforms supported, 1471, 1472
QUIT コマンド,
roles of computers, 1471, 1472
SCI (Scalable Coherent Interface), 916
SCI (スケーラブル コヒーレント インターフェース), 978
SCI ソフトウェアのインストール, 979
SCI ソフトウェア要件, 979
SCI ドライバ, 980
SCI ネットワーク設定, 979
SCI、パフォーマンス vs TCP/IP, 982
security, 1474
SHOW コマンド, 937
SHUTDOWN コマンド, 937
SQL statements, 1471, 1474
SQL ノード, 875, 913, 929
START BACKUPコマンド, 945, 971
START コマンド, 937, 937
starting and stopping, 1471, 1477
STARTUP イベント, 940
STATISTICS イベント, 942
STATUS コマンド, 937
STOP コマンド, 937
Table is full error, 1471, 1474
vs replication, 1471, 1472
 のメモリ使用, 984
および DNS, 879
および IP アドレス, 879
およびネットワーク構築, 880
のアップグレードおよびダウングレード, 926
のアップグレードおよびダウンロード, 924, 924
のイベント タイプ, 938, 940
のイベント ログ, 937, 939
のイベント ログ フォーマット, 940
のイベント ログ閾値, 939
のイベントの深刻度レベル, 939
のインストール, 878, 881, 890
のエラーログ, 931
のクラスタ ログ, 937, 939
のコマンド, 932, 933, 933, 934, 935, 937
のコンセプト, 875
のトランザクション, 899
のノード ログ, 937
のノードと種類, 875
のノード不良 (単一ユーザーモード), 944
のノード識別子,
のノード間に直接接続, 915
のバックアップ, 944, 944, 945, 946, 948, 949
のパフォーマンス, 981
のプロセス管理, 929
のベンチマーク, 981
のマネジメント コマンド, 943
のメモリの使用およびリカバリ, 925,
のレプリケーション, 962, 962
のレプリケーションの準備, 967
のログ コマンド, 939
の共有メモリのトランスポート, 915

の再設定, 925
の単一ユーザーモード, 937, 944
の取得, 881
の情報源, 873
の接続文字列, 894
の用語, 989
の管理, 932, 932, 933, 933, 934, 935, 937, 943
の簡単な 設定, 890
の要件, 880
の設定, 878, 889, 890, 895, 895, 897, 913, 923, 930, 931, 948
の設定 (参考例), 892
の設定ファイル, 892
の設定変更, 925
インターコネクト, 978
クエリの実行, 886
シャットダウン, 889
ソースからのコンパイル, 890
テーブルおよびデータの使用, 886
ディスク データ テーブル, 975
データ ノード, 930
データノード, 875, 897
トランスポーター
 Scalable Coherent Interface (SCI), 916
 TCP/IP, 915
 共有メモリ (SHM), 915
トレース ファイル, 931
ネットワーク, 915, 915
ネットワーク トランスポーター, 978, 978
ネットワークの設定 (SCI), 979
ネットワーク化, 916
ノード ホストの定義, 895
ノードおよびノード グループ, 876
ノードの起動, 885
バックアップのトラブルシューティング, 949
バックアップの復旧, 946
パーティショニングのサポート, 983
パーテション, 876
マネジメント クライアント (mgm), 932
マネジメント ノード, 875, 895, 931
ランタイム統計, 943
レプリカ, 876
ログ ファイル, 930
一般概要, 873
再起動, 889
利用可能なプラットフォーム, 872
設定パラメータ, 917, 918, 921, 922
設定ファイル, 883
起動および再起動, 935
MySQL Cluster の
 --initialによる起動, 890
 m y .cnf, 930
MySQL Cluster のインストール, 878, 881, 890
MySQL Cluster のバックアップからのレプリケーション
 による保存, 970
MySQL Cluster のプロセスの管理, 929
MySQL Cluster のユーティリティ, 949
MySQL Cluster のレプリケーション, 962, 969
MySQL Cluster の以前のバージョンから
 今回のバージョンで解決され制限, 987

- MySQL Cluster の制限, 983, 983
- MySQL Cluster の取得, 881
- MySQL Cluster の手引き, 878
- MySQL Cluster の新しい機能, 988
- MySQL Cluster の用語, 989
- MySQL Cluster の管理, 932, 935
- MySQL Cluster の設定, 878, 889
- MySQL Cluster の設定 (コンセプト), 875
- MySQL Cluster の開発予定, 988
- MySQL Cluster を設定する, 930,
- MySQL Cluster ディスク データ, 975
 - のテーブルの作成, 976
 - のテーブルスペースの作成, 976
 - のログ ファイルの作成, 975
 - ストレージの要件, 977
 - テーブルの作成, 975
 - ディスク データ オブジェクトの削除, 977
- MySQL Cluster プロセス (タイプ), 929
- MySQL Cluster マネジメント クライアントの SHOW, 892
- MySQL Cluster レプリケーション
 - reset-slave.pl バックアップの自動化, 972
 - の既知の問題, 964
 - の概念, 963, 964
 - の準備, 967
 - を始める, 968
 - システム テーブル, 965
 - バックアップ, 970, 972
 - フェールオーバー, 970
- MySQL Cluster 制限
 - によるエラー, 984
 - の導入, 986
 - サポートされていない機能, 985, 986
 - ジオメトリデータ タイプ, 983
 - データベース オブジェクト, 985
 - トランザクション, 984
 - パフォーマンス, 985
 - パーティショニング,
 - レプリケーション, 987
 - 構文, 983
 - 自動検索, 988
 - 複数お MySQL サーバー, 986
 - 複数のマネジメントサーバー, 987
- MySQL Clustern の新機能
 - MySQL 5.1 の, 988
- mysql commands
 - list of, 457
- mysql history file, 456
- MySQL Instance Manager, 217
- mysql prompt command, 458
- mysql status command, 457
- MySQL のアップグレード, 231
- MySQL の主な機能, 6
- MySQL の取得, 43
- MySQL の機能, 6
- MySQL の歴史, 5, 5
- MySQL の目的, 5
- MySQL コネクタ
 - MySQL, 1235
- mysql コマンド オプション, 452
- MySQL サーバ
 - mysqld, 151
 - MySQL ソースの配布, 35
 - MySQL ダウンロード用 URL, 43
 - MySQL ドルフィン名, 5
 - MySQL バイナリの配布, 35
 - MySQL バージョン, 43
 - MySQL メーリング リスト, 11
 - MySQL 名, 5
 - MySQL++, 1231
 - mysql.event table, 1087
 - mysql.ndb_binlog_index テーブル, 965
 - mysql.server, 151, 213
 - mysql.sock
 - protection, 1509
 - MYSQL323 SQL モード, 209
 - MYSQL40 SQL モード, 209
 - mysqlaccess, 436, 462
 - brief option, 463
 - commit option, 463
 - copy option, 463
 - db option, 463
 - debug option, 463
 - help option, 463
 - host option, 463
 - howto option, 463
 - old_server option, 463
 - password option, 463
 - plan option, 463
 - preview option, 463
 - relnotes option, 463
 - rhost option, 464
 - rollback option, 464
 - spassword option, 464
 - superuser option, 464
 - table option, 464
 - user option, 464
 - version option, 464
 - mysqladmin, 436, 464, 659, 678, 770, 773, 777, 779
 - character-sets-dir option, 467
 - count option, 467
 - debug option, 467
 - default-character-set option, 467
 - force option, 467
 - help option, 467
 - host option, 467
 - password option, 467
 - port option, 467
 - protocol option, 467
 - relative option, 468
 - silent option, 468
 - sleep option, 468
 - socket option, 468
 - SSL options, 468
 - user option, 468
 - verbose option, 468
 - version option, 468
 - vertical option, 468
 - wait option, 468
 - 圧縮オプション, 467
 - mysqladmin command options, 467

mysqladmin オプション
 mysqld_multi, 214

mysqlbinlog, 436, 468
 base64-output option, 469
 character-sets-dir option, 469
 database option, 469
 debug option, 469
 disable-log-bin option, 469
 force-read option, 470
 help option, 469
 hexdump option, 470
 host option, 470
 local-load option, 470
 offset option, 470
 password option, 470
 port option, 470
 position option, 470
 protocol option, 470
 read-from-remote-server option, 470
 result-file option, 470
 server-id option, 470
 set-charset option, 470
 short-form option, 470
 socket option, 471
 start-datetime option, 471
 start-position option, 471
 stop-datetime option, 471
 stop-position option, 471
 to-last-log option, 471
 user option, 471
 version option, 471

mysqlbugスクリプト, 17

mysqlcheck, 436, 474
 all-databases option, 475
 all-in-1 option, 475
 analyze option, 475
 auto-repair option, 475
 character-sets-dir option, 475
 check option, 475
 check-only-changed option, 475
 check-upgrade option, 475
 compress option, 475
 databases option, 475
 debug option, 475
 default-character-set option, 475
 extended option, 475
 fast option, 476
 fix-db-names option, 476
 fix-table-names option, 476
 force option, 476
 help option, 475
 host option, 476
 medium-check option, 476
 optimize option, 476
 password option, 476
 port option, 476
 protocol option, 476
 quick option, 476
 repair option, 476
 silent option, 476
 socket option, 477
 SSL options, 477
 tables option, 477
 use_frm option, 477
 user option, 477
 verbose option, 477
 version option, 477

mysqlclient ライブラリー, 1135

mysqld, 150
 allow-suspicious-udfs オプション, 153, 236
 ansi オプション, 153
 basedir オプション, 153
 bind-address オプション, 153
 binlog-do-db オプション, 302
 binlog-format オプション, 153
 binlog-ignore-db オプション, 302
 binlog-row-event-max-size オプション, 153
 bootstrap オプション, 153
 character-set-client-handshake オプション, 153
 character-set-filesystem オプション, 153
 character-set-server オプション, 154
 character-sets-dir オプション, 153
 chroot オプション, 154
 collation-server オプション, 154
 console オプション, 154
 core-file オプション, 154
 datadir オプション, 154
 debug オプション, 154
 default-character-set オプション, 154
 default-collation オプション, 154
 default-storage-engine オプション, 154
 default-table-type オプション, 154
 default-time-zone オプション, 155
 delay-key-write オプション, 155, 801
 des-key-file オプション, 155
 enable-named-pipe オプション, 155
 event-scheduler オプション, 155
 exit-info オプション, 155
 flush オプション, 155
 general-log オプション, 155
 help オプション, 153
 init-file オプション, 156
 innodb オプション, 812
 innodb_status_file オプション, 812
 language オプション, 156
 large-pages オプション, 156
 local-infile オプション, 236
 log オプション, 156
 log-bin オプション, 156
 log-bin-index オプション, 156
 log-bin-trust-function-creators オプション, 156
 log-error オプション, 157
 log-isam オプション, 157
 log-long-format オプション, 157
 log-output オプション, 157
 log-queries-not-using-indexes オプション, 157
 log-short-format オプション, 157
 log-slave-updates オプション, 331
 log-slow-admin-statements オプション, 157
 log-slow-queries オプション, 157

log-tc オプション, 157
log-tc-size オプション, 157
log-warnings オプション, 158, 332
low-priority-updates オプション, 158
master-connect-retry オプション, 332
master-host オプション, 332
master-info-file オプション, 332
master-password オプション, 332
master-port オプション, 332
master-retry-count オプション, 332
master-ssl オプション, 332
master-ssl-ca オプション, 332
master-ssl-capath オプション, 332
master-ssl-cert オプション, 332
master-ssl-cipher オプション, 332
master-ssl-key オプション, 332
master-user オプション, 332
max-relay-log-size オプション, 332
memlock オプション, 158
myisam-recover オプション, 158, 801
MySQL Cluster プロセスとしての, 929, 933
MySQL サーバ, 151
ndb-connectstring オプション, 158
ndbcluster オプション, 158
old-passwords オプション, 159, 236
one-thread オプション, 159
open-files-limit オプション, 159
pid-file オプション, 159
port オプション, 159
port-open-timeout オプション, 159
read-only オプション, 333
relay-log オプション, 333
relay-log-index オプション, 333
relay-log-info-file オプション, 333
relay-log-purge オプション, 333
relay-log-space-limit オプション, 333
replicate-do-db オプション, 333
replicate-do-table オプション, 334
replicate-ignore-db オプション, 334
replicate-ignore-table オプション, 334
replicate-rewrite-db オプション, 334
replicate-same-server-id オプション, 334
replicate-wild-do-table オプション, 335
replicate-wild-ignore-table オプション, 335
report-host オプション, 335
report-port オプション, 335
safe-mode オプション, 159
safe-show-database オプション, 159, 236
safe-user-create オプション, 159, 236
secure-auth オプション, 159, 236
shared-memory オプション, 159
shared-memory-base-name オプション, 159
skip-concurrent-insert オプション, 159
skip-external-locking オプション, 159
skip-grant-tables オプション, 160, 236
skip-host-cache オプション, 160
skip-innodb オプション, 160
skip-merge オプション, 237
skip-name-resolve オプション, 160, 237
skip-ndbcluster オプション, 160
skip-networking オプション, 160, 237
skip-safemalloc オプション, 161
skip-show-database オプション, 161, 237
skip-slave-start オプション, 335
skip-stack-trace オプション, 161
skip-symbolic-links オプション, 160
skip-thread-priority オプション, 161
slave-load-tmpdir オプション, 336
slave-net-timeout オプション, 336
slave-skip-errors オプション, 336
slave_compressed_protocol オプション, 335
slow-query-log オプション, 161
socket オプション, 161
sql-mode オプション, 161
SSL オプション, 160, 237
standalone オプション, 160
symbolic-links オプション, 160
sysdate-is-now オプション, 161
tc-heuristic-recover オプション, 161
temp-pool オプション, 161
tmpdir オプション, 162
transaction-isolation オプション, 162
user オプション, 162
version オプション, 162
コマンド オプション, 152
ロール (MySQL Cluster), 875
外部ロック オプション, 155
起動, 238
mysqld オプション
 mysqld_multi, 215
 mysqld_safe, 211
mysqld ライブラリー, 1135
mysqld-safe-compatible オプション
 mysqlmanager, 219
mysqld-version オプション
 mysqld_safe, 212
mysqldump, 106, 436, 477
 add-drop-database option, 478
 add-drop-table option, 478
 add-locks option, 478
 all-databases option, 478
 all-tablespaces option, 478
 allow-keywords option, 478
 character-sets-dir option, 478
 comments option, 478
 compact option, 479
 compatible option, 479
 complete-insert option, 479
 compress option, 479
 create-options option, 479
 databases option, 479
 debug option, 479
 default-character-set option, 479
 delayed-insert option, 479
 delete-master-logs option, 479
 disable-keys option, 479
 events option, 480
 extended-insert option, 480
 fields-enclosed-by option, 480, 488
 fields-escaped-by option, 480, 488

fields-optionally-enclosed-by option, 480, 488
fields-terminated-by option, 480, 488
first-slave option, 480
flush-logs option, 480
flush-privileges option, 480
force option, 480
hex-blob option, 480
host option, 480
ignore-table option, 480
insert-ignore option, 480
lines-terminated-by option, 480, 488
lock-all-tables option, 481
lock-tables option, 481
master-data option, 481
no-autocommit option, 481
no-create-db option, 481
no-create-info option, 481
no-data option, 481
opt option, 481
order-by-primary option, 481
password option, 482
port option, 482
protocol option, 482
quick option, 482
quote-names option, 482
replace option, 482
result-file option, 482
routines option, 482
set-charset option, 482
single-transaction option, 482
skip-comments option, 483
skip-opt option, 483
socket option, 483
SSL options, 483
tab option, 483
tables option, 483
triggers option, 483
tz-utc option, 483
user option, 483
verbose option, 484
version option, 484
views, 485
where option, 484
xml option, 484
ビュー, 1709
ヘルプオプション, 478
問題, 485, 1709
問題の回避, 485, 1709
mysql_multi, 151, 213
 config-file オプション, 214
 example オプション, 214
 help オプション, 214
 log オプション, 214
 mysqladmin オプション, 214
 mysqld オプション, 215
 no-log オプション, 215
 password オプション, 215
 silent オプション, 215
 tcp-ip オプション, 215
 user オプション, 215
 verbose オプション, 215
 version オプション, 215
mysql_safe, 151, 210
 autoclose オプション, 211
 basedir オプション, 211
 core-file-size オプション, 211
 datadir オプション, 211
 defaults-extra-file オプション, 211
 defaults-file オプション, 211
 help オプション, 211
 ledir オプション, 211
 log-error オプション, 211
 mysqld オプション, 211
 mysqld-version オプション, 212
 nice オプション, 212
 no-defaults オプション, 212
 open-files-limit オプション, 212
 pid-file オプション, 212
 port オプション, 212
 socket オプション, 212
 timezone オプション, 212
 user オプション, 212
mysqld オプション, 424
mysqld サーバ
 バッファサイズ, 424
mysqlhotcopy, 436, 485
 addtodest option, 486
 allowold option, 486
 checkpoint option, 486
 chroot option, 486, 486
 debug option, 486
 dryrun option, 486
 flushlog option, 486
 help option, 486
 host option, 486
 keepold option, 486
 method option, 486
 noindices option, 486
 password option, 486
 port option, 486
 quiet option, 486
 record_log_pos option, 487
 regexp option, 487
 resetmaster option, 487
 resetslave option, 487
 socket option, 487
 suffix option, 487
 tmpdir option, 487
 user option, 487
mysqlimport, 106, 436, 487, 690
 character-sets-dir option, 488
 columns option, 488
 compress option, 488
 debug option, 488
 default-character-set option, 488
 delete option, 488
 force option, 488
 help option, 487
 host option, 488
 ignore option, 488

- ignore-lines option, 488
- local option, 488
- lock-tables option, 488
- low-priority option, 489
- password option, 489
- port option, 489
- protocol option, 489
- replace option, 489
- silent option, 489
- socket option, 489
- SSL options, 489
- user option, 489
- verbose option, 489
- version option, 489
- mysqlmanager, 151, 217
 - add-user オプション, 217
 - angel-pid-file オプション, 217
 - bind-address オプション, 218
 - check-password-file オプション, 218
 - clean-password-file オプション, 218
 - debug オプション, 218
 - default-mysqld-path オプション, 218
 - defaults-file オプション, 218
 - drop-user オプション, 218
 - edit-user オプション, 218
 - help オプション, 217
 - install オプション, 218
 - list-users オプション, 218
 - log オプション, 218
 - monitoring-interval オプション, 219
 - mysqld-safe-compatible オプション, 219
 - password オプション, 219
 - password-file オプション, 219
 - pid-file オプション, 219
 - port オプション, 219
 - print-defaults オプション, 219
 - print-password-line オプション, 219
 - remove オプション, 219
 - run-as-service オプション, 220
 - socket オプション, 220
 - standalone オプション, 220
 - user オプション, 220
 - username オプション, 220
 - version オプション, 220
 - wait-timeout オプション, 220
- mysqlshow, 436, 490
 - character-sets-dir option, 490
 - compress option, 490
 - count option, 490
 - debug option, 490
 - default-character-set option, 490
 - help option, 490
 - host option, 491
 - keys option, 491
 - password option, 491
 - port option, 491
 - protocol option, 491
 - show-table-type option, 491
 - socket option, 491
 - SSL options, 491
 - status option, 491
 - user option, 491
 - verbose option, 491
 - version option, 491
- mysqlslap, 436, 491
 - auto-generate-sql オプション, 492, 493, 493
 - compress オプション, 492
 - concurrency オプション, 492
 - create オプション, 492
 - create-schema オプション, 492
 - csv オプション, 492
 - debug オプション, 492
 - delimiter オプション, 492
 - engine オプション, 492
 - help option, 492
 - host オプション, 492
 - iterations オプション, 492
 - lock-directory オプション, 492, 493
 - number-char-cols オプション, 493
 - number-int-cols オプション, 493
 - number-of-queries オプション, 493
 - only-print オプション, 493
 - password オプション, 493, 493
 - port オプション, 493
 - preserve-schema オプション, 493
 - protocol オプション, 493
 - query オプション, 493
 - silent オプション, 493
 - skip-query option, 493
 - slave オプション, 493
 - socket オプション, 493
 - SSL オプション, 493
 - use-threads オプション, 493
 - user オプション, 494
 - verbose オプション, 494
 - version オプション, 494
- mysqltest
 - MySQL Test Suite, 1430
- mysql_affected_rows(), 1148
- mysql_autocommit(), 1149
- MYSQL_BIND C タイプ, 1194
- mysql_change_user(), 1149
- mysql_character_set_name(), 1150
- mysql_close(), 1151
- mysql_commit(), 1151
- mysql_config, 1232
- mysql_connect(), 1151
- mysql_create_db(), 1152
- mysql_data_seek(), 1152
- MYSQL_DEBUG 環境変数, 133, 437
- mysql_debug(), 1153
- mysql_drop_db(), 1153
- mysql_dump_debug_info(), 1154
- mysql_eof(), 1154
- mysql_errno(), 1155
- mysql_error(), 1156
- mysql_escape_string(), 1156
- mysql_fetch_field(), 1156
- mysql_fetch_fields(), 1157
- mysql_fetch_field_direct(), 1157

mysql_fetch_lengths(), 1158
mysql_fetch_row(), 1159
MYSQL_FIELD C タイプ, 1141
mysql_field_count(), 1159, 1172
MYSQL_FIELD_OFFSET C タイプ, 1141
mysql_field_seek(), 1160
mysql_field_tell(), 1161
mysql_fix_privilege_tables, 151, 229
mysql_free_result(), 1161
mysql_get_character_set_info(), 1161
mysql_get_client_info(), 1162
mysql_get_client_version(), 1162
mysql_get_host_info(), 1162
mysql_get_proto_info(), 1162
mysql_get_server_info(), 1163
mysql_get_server_version(), 1163
mysql_get_ssl_cipher(), 1163
MYSQL_GROUP_SUFFIX 環境変数, 133
mysql_hex_string(), 1163
MYSQL_HISTFILE 環境変数, 133, 456
MYSQL_HOME 環境変数, 133
MYSQL_HOST 環境変数, 133, 246
mysql_info(), 656, 686, 697, 720, 1164
mysql_init(), 1165
mysql_insert_id(), 24, 687, 1165
mysql_install_db, 151, 230
mysql_install_db スクリプト, 93
mysql_kill(), 1167
mysql_library_end(), 1167
mysql_library_init(), 1167
mysql_list_dbs(), 1168
mysql_list_fields(), 1169
mysql_list_processes(), 1169
mysql_list_tables(), 1170
mysql_more_results(), 1171
mysql_next_result(), 1171
mysql_num_fields(), 1172
mysql_num_rows(), 1173
mysql_options(), 1173
mysql_ping(), 1176
MYSQL_PS1 環境変数, 133
MYSQL_PWD 環境変数, 133, 246, 437
mysql_query(), 1177, 1226
mysql_real_connect(), 1178
mysql_real_escape_string(), 499, 1181
mysql_real_query(), 1181
mysql_refresh(), 1182
mysql_reload(), 1183
MYSQL_RES C タイプ, 1141
mysql_rollback(), 1184
MYSQL_ROW C タイプ, 1141
mysql_row_seek(), 1184
mysql_row_tell(), 1184
mysql_select_db(), 1185
mysql_server_end(), 1225
mysql_server_init(), 1225
mysql_set_character_set(), 1185
mysql_set_local_infile_default(), 1186, 1186
mysql_set_server_option(), 1187
mysql_shutdown(), 1188
mysql_sqlstate(), 1188
mysql_ssl_set(), 1189
mysql_stat(), 1189
MYSQL_STMT C タイプ, 1194
mysql_stmt_affected_rows(), 1201
mysql_stmt_attr_get(), 1201
mysql_stmt_attr_set(), 1202
mysql_stmt_bind_param(), 1203
mysql_stmt_bind_result(), 1203
mysql_stmt_close(), 1204
mysql_stmt_data_seek(), 1205, 1218
mysql_stmt_errno(), 1205
mysql_stmt_error(), 1205
mysql_stmt_execute(), 1206
mysql_stmt_fetch(), 1209
mysql_stmt_fetch_column(), 1213
mysql_stmt_field_count(), 1213
mysql_stmt_free_result(), 1213
mysql_stmt_init(), 1214
mysql_stmt_insert_id(), 1214
mysql_stmt_num_rows(), 1214
mysql_stmt_param_count(), 1215
mysql_stmt_param_metadata(), 1215
mysql_stmt_prepare(), 1215
mysql_stmt_reset(), 1216
mysql_stmt_result_metadata, 1216
mysql_stmt_row_seek(), 1217
mysql_stmt_row_tell(), 1218
mysql_stmt_sqlstate(), 1219
mysql_stmt_store_result(), 1220
mysql_store_result(), 1190, 1226
MYSQL_TCP_PORT 環境変数, 133, 310, 311, 437
mysql_thread_end(), 1224
mysql_thread_id(), 1191
mysql_thread_init(), 1224
mysql_thread_safe(), 1225
MYSQL_TIME C タイプ, 1196
mysql_tzinfo_to_sql, 151, 232
MYSQL_UNIX_PORT 環境変数, 94, 133, 310, 311, 437
mysql_upgrade, 151, 231, 253
 basedir オプション, 231
 datadir オプション, 231
 force オプション, 231
 help オプション, 231
 user オプション, 231
 verbose オプション, 232
mysql_use_result(), 1191
mysql_warning_count(), 1192
mysql_zap, 437, 494
MySQL クラスタ
 必要とする記憶容量, 564
MySQL ストレージエンジン, 793
my_bool C タイプ, 1141
my_bool 値
 印刷, 1141
my_init(), 1224
my_print_defaults, 435, 437
 config-file オプション, 437, 437
 debug オプション, 437
 defaults-extra-file オプション, 438

- defaults-group-suffix オプション, 438
- extra-file オプション, 438
- help オプション, 437
- no-defaults オプション, 438
- verbose オプション, 438
- version オプション, 438
- my_ulonglong C タイプ, 1141
- my_ulonglong 値
印刷, 1141

N

- named-commands option
 - mysql, 453
- NAME_CONST(), 641
- NATIONAL CHARデータタイプ, 546
- native functions
 - adding, 1455
- NATURAL LEFT JOIN, 703
- NATURAL LEFT OUTER JOIN, 703
- NATURAL RIGHT JOIN, 703
- NATURAL RIGHT OUTER JOIN, 703
- NCHARデータタイプ, 546
- NDB, 1470, 1472
- NDB storage engine
 - FAQ, 1470
- ndb オプション
 - perror, 495
- NDB ストレージ エンジン, 872
- ndb-connectstring option
 - ndb_config, 951
- ndb-connectstring オプション
 - mysqld, 158
- ndbcluster オプション
 - mysqld, 158
- ndbd (MySQL Cluster プロセス), 930
- ndbd (MySQL Cluster)
 - の定義, 875, 875
- ndb_apply_status テーブル (MySQL Cluster のレプリケーション), 966
- ndb_apply_status テーブル (MySQL Cluster レプリケーション), 970
- ndb_binlog_index テーブル (MySQL Cluster レプリケーション), 965
- ndb_binlog_index テーブル (MySQL Cluster レプリケーション), 970
- ndb_config, 949, 950
 - config-file option, 951
 - id オプション, 951
 - ndb-connectstring option, 951
 - nodeid オプション, 951
 - nodes オプション, 951
 - クエリ オプション, 951, 951
 - タイプ オプション, 951
 - バージョン オプション, 951
 - フィールド オプション, 951
 - ホスト オプション, 951
 - 使用オプション, 951
 - 行オプション, 952
- ndb_delete_all, 949, 952

- トランザクション オプション, 952
- ndb_desc, 949, 953
 - extra-partition-info option, 953
- ndb_drop_index, 949, 954
- ndb_drop_table, 949, 954
- ndb_error_reporter, 949, 955
- ndb_mgm, 932
- ndb_mgm (MySQL Cluster マネジメント ノード クライアント), 885
- ndb_mgmd, 931
- ndb_mgmd (MySQL Cluster)
 - の定義, 875
- ndb_print_backup_file, 949, 955
- ndb_print_schema_file, 949, 955
- ndb_print_sys_file, 949, 956
- ndb_restore, 946
 - エラー, 948
- ndb_select_all, 949, 956
- ndb_select_count, 949, 958
- ndb_show_tables, 949, 958
- ndb_size.pl, 949, 959
- ndb_size.pl (utility), 1473
- ndb_waiter, 949, 961
- NetWare, 70
- net_buffer_length variable, 456
- new procedures
 - adding, 1456
- nice オプション
 - mysqld_safe, 212
- no matching rows, 1513
- no-auto-rehash option
 - mysql, 453
- no-autocommit option
 - mysqldump, 481
- no-beep option
 - mysql, 454
- no-create-db option
 - mysqldump, 481
- no-create-info option
 - mysqldump, 481
- no-data option
 - mysqldump, 481
- no-debug オプション
 - make_win_bin_dist, 229
- no-defaults オプション, 146
 - mysqld_safe, 212
 - my_print_defaults, 438
- no-embedded オプション
 - make_win_bin_dist, 229
- no-log オプション
 - mysqld_multi, 215
- no-named-commands option
 - mysql, 454
- no-pager option
 - mysql, 454
- no-tee option
 - mysql, 454
- nodeid オプション
 - ndb_config, 951
- NODERESTART イベント (MySQL Cluster), 941

- nodes オプション
 - ndb_config, 951
- noindices option
 - mysqlhotcopy, 486
- Non-transactional tables, 1513
- NoOfDiskPagesToDiskAfterRestartACCの計算, 923
- NoOfDiskPagesToDiskAfterRestartACC (DEPRECATED), 909
- NoOfDiskPagesToDiskAfterRestartTUPの計算, 923
- NoOfDiskPagesToDiskAfterRestartTUP (DEPRECATED), 909
- NoOfDiskPagesToDiskDuringRestartACC (DEPRECATED), 910
- NoOfDiskPagesToDiskDuringRestartTUP (DEPRECATED), 910
- NoOfFragmentLogFiles, 904
 - の計算, 923
- NoOfReplicas, 898
- NOT
 - 論理, 576
- NOT BETWEEN, 574
- NOT IN, 574
- NOT LIKE, 588
- NOT NULL
 - 制約, 27
- NOT REGEXP, 588
- Novell NetWare, 70
- NOW(), 607
- NO_AUTO_CREATE_USER SQL モード, 206
- NO_AUTO_VALUE_ON_ZERO SQL モード, 206
- NO_BACKSLASH_ESCAPES SQL モード, 206
- NO_DIR_IN_CREATE SQL モード, 206
- NO_FIELD_OPTIONS SQL モード, 206
- NO_KEY_OPTIONS SQL モード, 206
- NO_TABLE_OPTIONS SQL モード, 207
- NO_UNSIGNED_SUBTRACTION SQL モード, 207
- NO_ZERO_DATE SQL モード, 207
- NO_ZERO_IN_DATE SQL モード, 207
- NUL, 498
- NULL, 1512
 - null のテスト, 572, 573, 574, 578
 - ORDER BY, 401, 700
- NULL values
 - and AUTO_INCREMENT columns, 1512
 - and TIMESTAMP columns, 1512
 - vs. empty values, 1511
- NULL 値, 500
 - そしてインデックス, 666
- NULLIF(), 578
- number-char-cols オプション
 - mysqslap, 493
- number-int-cols オプション
 - mysqslap, 493
- number-of-queries オプション
 - mysqslap, 493
- NUMERICデータタイプ, 544
- NumGeometries(), 1046
- NumInteriorRings(), 1046
- NumPoints(), 1044

O

- OCT(), 583
- OCTET_LENGTH(), 583
- ODBC, 1236
- ODBC との適合性, 571, 573
- ODBC 互換性, 502, 544, 665, 705
- offset option
 - mysqlbinlog, 470
- OLAP, 645
- old-passwords オプション
 - mysqld, 159, 236
- OLD_PASSWORD(), 633
- old_server option
 - mysqlaccess, 463
- ON DUPLICATE KEY, 684
- one-database option
 - mysql, 454
- one-thread オプション
 - mysqld, 159
- only-debug オプション
 - make_win_bin_dist, 229
- only-print オプション
 - mysqslap, 493
- ONLY_FULL_GROUP_BY SQL モード, 647
- ONLY_FULL_GROUP_BY SQL モード, 207
- OPEN, 1063
- open-files-limit オプション
 - mysqld, 159
 - mysqld_safe, 212
- OpenGIS, 1030
- opens, 466
- OpenSSL, 267, 268
- open_files_limit variable, 471
- operating systems
 - file-size limits, 1500
- opt option
 - mysqldump, 481
- optimize option
 - mysqlcheck, 476
- OPTIMIZE TABLE, 745
- OR, 389
 - 演算, 576
- OR インデックス結合最適化, 389
- ORACLE SQL モード, 209
- Oracle との互換性, 20, 645
- Oracle 互換性, 721
- ORD(), 583
- ORDER BY, 653, 700
 - NULL, 401, 700
- order-by-primary option
 - mysqldump, 481
- OUTFILE, 701
- Overlaps(), 1049

P

- packages
 - list of, 1721
- packlength オプション

- myisampack, 447
- pack_isam, 446
- pager option
 - mysql, 454
- parallel-recover オプション
 - mysamchk, 443
- PARTITION, 995
- PARTITIONS
 - INFORMATION_SCHEMA テーブル, 1111
- password option
 - mysql, 454
 - mysqlaccess, 463
 - mysqladmin, 467
 - mysqlbinlog, 470
 - mysqlcheck, 476
 - mysqldump, 482
 - mysqlhotcopy, 486
 - mysqlimport, 489
 - mysqlshow, 491
- password オプション
 - mysqld_multi, 215
 - mysqlmanager, 219
 - mysqlslap, 493, 493
- PASSWORD(), 248, 265, 633, 1502
- password-file オプション
 - mysqlmanager, 219
- passwords
 - forgotten, 1505
 - lost, 1505
 - resetting, 1505
- PATH 環境変数, 73, 133, 141
- PERIOD_ADD(), 607
- PERIOD_DIFF(), 607
- Perl
 - のインストール, 134, 134
- Perl API, 1231
- Perl DBI/DBD
 - インストールの問題, 136
- perror, 437, 494
 - ndb option, 1475
 - help option, 495
 - ndb オプション, 495
 - silent オプション, 495
 - verbose オプション, 495
 - version option, 495
- PHP API, 1229
- PI(), 596
- pid-file オプション
 - mysqld, 159
 - mysqld_safe, 212
 - mysqlmanager, 219
- PIPES_AS_CONCAT SQL モード, 208
- plan option
 - mysqlaccess, 463
- plugin API, 1430
- PLUGINS
 - INFORMATION_SCHEMA テーブル, 1111
- plugins
 - adding, 1430
 - installing, 1433
 - uninstalling, 1433
- POINT data type, 1037
- Point(), 1039
- PointFromText(), 1038
- PointFromWKB(), 1039
- PointN(), 1044
- PointOnSurface(), 1046
- PolyFromText(), 1038
- PolyFromWKB(), 1039
- POLYGON data type, 1037
- Polygon(), 1039
- PolygonFromText(), 1038
- PolygonFromWKB(), 1039
- port option
 - mysql, 454
 - mysqladmin, 467
 - mysqlbinlog, 470
 - mysqlcheck, 476
 - mysqldump, 482
 - mysqlhotcopy, 486
 - mysqlimport, 489
 - mysqlshow, 491
- port オプション
 - mysqld, 159
 - mysqld_safe, 212
 - mysqlmanager, 219
 - mysqlslap, 493
- port-open-timeout オプション
 - mysqld, 159
- PortNumber, 896, 915
- position option
 - mysqlbinlog, 470
- POSITION(), 584
- POSTGRES SQL モード, 209
- PostgreSQL 互換性, 21
- POW(), 597
- POWER(), 597
- PREPARE, 790, 791
 - XA トランザクション, 731
- preserve-schema オプション
 - mysqlslap, 493
- preview option
 - mysqlaccess, 463
- previx オプション
 - configure, 77
- PRIMARY KEY, 653, 665
 - 制約, 27
- print-defaults オプション, 146
 - mysqlmanager, 219
- print-password-line オプション
 - mysqlmanager, 219
- problems
 - access denied errors, 1492
 - common errors, 1491
 - DATE columns, 1510
 - linking, 1503
 - time zone, 1510
- PROCEDURE, 702
- procedures
 - adding, 1456

- processing
 - arguments, 1451
- PROCESSLIST, 764
 - INFORMATION_SCHEMA テーブル, 1121
- prompt option
 - mysql, 454
- protocol option
 - mysql, 454
 - mysqladmin, 467
 - mysqlbinlog, 470
 - mysqlcheck, 476
 - mysqldump, 482
 - mysqlimport, 489
 - mysqlshow, 491
- protocol オプション
 - mysqlslap, 493
- PURGE MASTER LOGS, 781
- PURGE STALE SESSIONS, 936
- Python API, 1231

Q

- QUARTER(), 608
- query オプション
 - mysqlslap, 493
- questions, 466
- quick option
 - mysql, 454
 - mysqlcheck, 476
 - mysqldump, 482
- quick オプション
 - myisamchk, 443
- quiet option
 - mysqlhotcopy, 486
- QUIT コマンド (MySQL Cluster),
- QUOTE(), 584
- quote-names option
 - mysqldump, 482

R

- RADIANS(), 597
- RAND(), 597
- raw option
 - mysql, 454
- read-from-remote-server option
 - mysqlbinlog, 470
- read-only オプション
 - myisamchk, 442
 - mysqld, 333
- read_buffer_size myisamchk 変数, 440
- REAL_AS_FLOAT SQL モード, 208
- REALデータタイプ, 544
- ReceiveBufferMemory, 915
- reconnect option
 - mysql, 454
- record_log_pos option
 - mysqlhotcopy, 487
- RECOVER
 - XA トランザクション, 731
- recover オプション

- myisamchk, 443
- RedoBuffer, 910
- REFERENTIAL_CONSTRAINTS
 - INFORMATION_SCHEMA テーブル, 1122
- ref_or_null, 391
- REGEXP, 588
- regexp option
 - mysqlhotcopy, 487
- REGEXP 演算子, 589
- Related(), 1049
- relative option
 - mysqladmin, 468
- relay-log オプション
 - mysqld, 333
- relay-log-index オプション
 - mysqld, 333
- relay-log-info-file オプション
 - mysqld, 333
- relay-log-purge オプション
 - mysqld, 333
- relay-log-space-limit オプション
 - mysqld, 333
- RELEASE SAVEPOINT, 726
- RELEASE_LOCK(), 641
- relnotes option
 - mysqlaccess, 463
- remove オプション
 - mysqlmanager, 219
- RENAME DATABASE, 680
- RENAME TABLE, 680
- RENAME USER, 741
- reordering
 - columns, 1516
- repair
 - tables, 474
- repair option
 - mysqlcheck, 476
- REPAIR TABLE, 746
- repair オプション
 - myisamchk, 442
- REPEAT, 1065
- REPEAT(), 584
- replace, 437
- REPLACE, 697
- replace option
 - mysqldump, 482
 - mysqlimport, 489
- replace ユーティリティ, 495
- REPLACE(), 584
- replicate-do-db オプション
 - mysqld, 333
- replicate-do-table オプション
 - mysqld, 334
- replicate-ignore-db オプション
 - mysqld, 334
- replicate-ignore-table オプション
 - mysqld, 334
- replicate-rewrite-db オプション
 - mysqld, 334
- replicate-same-server-id オプション

- mysqld, 334
- replicate-wild-do-table オプション
 - mysqld, 335
- replicate-wild-ignore-table オプション
 - mysqld, 335
- report-host オプション
 - mysqld, 335
- report-port オプション
 - mysqld, 335
- REQUIRE GRANT オプション, 739
- RESET MASTER, 781
- RESET SLAVE, 786
- reset-slave.pl, 972
- resetmaster option
 - mysqldhotcopy, 487
- resetslave option
 - mysqldhotcopy, 487
- RESTART コマンド (MySQL Cluster),
- RestartOnErrorInsert, 906
- RESTORE TABLE, 747
- result-file option
 - mysqlbinlog, 470
 - mysqldump, 482
- return values
 - UDFs, 1452
- REVERSE(), 584
- REVOKE, 741
- rhost option
 - mysqlaccess, 464
- RIGHT JOIN, 703
- RIGHT OUTER JOIN, 703
- RIGHT(), 584
- RLIKE, 588
- ROLLBACK, 22, 724
 - XA トランザクション, 731
- rollback option
 - mysqlaccess, 464
- ROLLBACK TO SAVEPOINT, 726
- ROLLUP, 645
- root user
 - password resetting, 1505
- ROUND(), 597
- ROUTINES
 - INFORMATION_SCHEMA テーブル, 1108
- routines option
 - mysqldump, 482
- rows
 - deleting, 1513
 - matching problems, 1513
- ROW_COUNT(), 639
- RPAD(), 584
- RPM パッケージ マネージャ, 65
- RPM ファイル, 65
- RTRIM(), 584
- run-as-service オプション
 - mysqlmanager, 220

S

- safe-mode オプション

- mysqld, 159
- safe-recover オプション
 - myisamchk, 443
- safe-show-database オプション
 - mysqld, 159, 236
- safe-updates option, 461
 - mysql, 455
- safe-user-create オプション
 - mysqld, 159, 236
- Sakila, 5
- SAVEPOINT, 726
- SCHEMA(), 639
- SCHEMATA
 - INFORMATION_SCHEMA テーブル, 1101
- SCHEMA_PRIVILEGES
 - INFORMATION_SCHEMA テーブル, 1104
- SCI (Scalable Coherent Interface), 916
- SCI (スケーラブル コヒーレント インターフェース), 978
- searching
 - and case sensitivity, 1510
- SECOND(), 608
- secure-auth option
 - mysql, 455
- secure-auth オプション
 - mysqld, 159, 236
- SEC_TO_TIME(), 608
- SELECT
 - LIMIT, 698
 - クエリ キャッシュ, 312
 - 最適化, 375
- SELECT INTO, 1061
- SELECT INTO TABLE, 22
- SELECT および WHERE 句の関数, 569
- SELECT 速度, 384
- select_limit variable, 456
- SendBufferMemory, 914
- SendLimit, 917
- SendSignalId, 914, 916, 917
- server administration, 464
- server-id option
 - mysqlbinlog, 470
- ServerPort, 898
- SESSION_STATUS
 - INFORMATION_SCHEMA テーブル, 1123
- SESSION_USER(), 639
- SESSION_VARIABLES
 - INFORMATION_SCHEMA テーブル, 1123
- SET, 747, 1060
 - AUTOCOMMIT, 747
 - BIG_TABLES, 747
 - CHARACTER SET, 520, 747
 - FOREIGN_KEY_CHECKS, 747
 - IDENTITY, 747
 - INSERT_ID, 747
 - LAST_INSERT_ID, 747
 - NAMES, 520, 747
 - ONE_SHOT, 747
 - SQL_AUTO_IS_NULL, 747
 - SQL_BIG_SELECTS, 747
 - SQL_BUFFER_SELECT, 747

SQL_LOG_BIN, 747
SQL_LOG_OFF, 747
SQL_LOG_UPDATE, 747
SQL_NOTES, 747
SQL_QUOTE_SHOW_CREATE, 747
SQL_SAFE_UPDATES, 747
SQL_SELECT_LIMIT, 747
SQL_WARNINGS, 747
TIMESTAMP, 747
UNIQUE_CHECKS, 747
サイズ, 566
SET GLOBAL SQL_SLAVE_SKIP_COUNTER, 786
SET OPTION, 747
SET PASSWORD, 742
SET PASSWORD ステートメント, 265
SET SQL_LOG_BIN, 781
SET TRANSACTION, 729
set-auto-increment[オプション
 myisamchk, 444
set-charset option
 mysqlbinlog, 470
 mysqldump, 482
set-collation オプション
 myisamchk, 444
SETデータタイプ, 547, 562
SHA(), 634
SHA1(), 634
shared-memory オプション
 mysqld, 159
shared-memory-base-name オプション
 mysqld, 159
SharedBufferSize, 916
SharedGlobalMemory, 978
ShmKey, 916
ShmSize, 916
short-form option
 mysqlbinlog, 470
SHOW AUTHORS, 752, 753
SHOW BINARY LOGS, 782
SHOW BINLOG EVENTS, 753, 781
SHOW CHARACTER SET, 752, 753
SHOW COLLATION, 752, 754
SHOW COLUMNS, 752, 754
SHOW CONTRIBUTORS, 752, 754
SHOW CREATE DATABASE, 752, 754
SHOW CREATE EVENT, 752
SHOW CREATE FUNCTION, 752, 755
SHOW CREATE PROCEDURE, 752, 755
SHOW CREATE SCHEMA, 752, 755
SHOW CREATE TABLE, 752, 755
SHOW CREATE VIEW, 752, 756
SHOW DATABASES, 752, 756
SHOW ENGINE, 752, 756
SHOW ENGINE INNODB STATUS, 756
SHOW ENGINE NDB STATUS, 756
SHOW ENGINES, 752, 758
SHOW ERRORS, 752, 759
 and MySQL Cluster, 1475
SHOW EVENTS, 752
SHOW FIELDS, 752
SHOW FUNCTION CODE, 752, 764
SHOW FUNCTION STATUS, 752, 764
SHOW GRANTS, 752, 761
SHOW INDEX, 752, 761
SHOW INNODB STATUS, 752
SHOW KEYS, 752, 761
SHOW MASTER LOGS, 753, 782
SHOW MASTER STATUS, 753, 782
SHOW OPEN TABLES, 752, 762
SHOW PLUGINS, 752, 763
SHOW PRIVILEGES, 752, 763
SHOW PROCEDURE CODE, 752, 764
SHOW PROCEDURE STATUS, 752, 764
SHOW PROCESSLIST, 752, 764
SHOW SCHEDULER STATUS, 752, 1085
SHOW SCHEMAS, 752, 756
SHOW SLAVE HOSTS, 753, 782
SHOW SLAVE STATUS, 753, 786
SHOW STATUS, 752
SHOW STORAGE ENGINES, 758
SHOW TABLE STATUS, 752
SHOW TABLE TYPES, 752, 758
SHOW TABLES, 752, 772
SHOW TRIGGERS, 752, 772
SHOW VARIABLES, 752
SHOW WARNINGS, 752, 774
 and MySQL Cluster,
SHOW with WHERE, 1099, 1123
SHOW コマンド (MySQL Cluster),
SHOW 拡張, 1123
show-table-type option
 mysqlshow, 491
show-warnings option
 mysql, 455
showing
 database information, 490
SHUTDOWN コマンド (MySQL Cluster),
shutdown_timeout variable, 468
sigint-ignore option
 mysql, 455
SIGN(), 598
silent option
 mysql, 455
 mysqladmin, 468
 mysqlcheck, 476
 mysqlimport, 489
silent オプション
 myisamchk, 440
 myisampack, 447
 mysqld_multi, 215
 mysqlslap, 493
 perror, 495
SIN(), 598
single-transaction option
 mysqldump, 482
size of tables, 1500
skip-column-names option
 mysql, 455
skip-comments option
 mysqldump, 483

skip-concurrent-insert オプション
 mysqld, 159

skip-external-locking オプション
 mysqld, 159

skip-grant-tables オプション
 mysqld, 160, 236

skip-host-cache オプション
 mysqld, 160

skip-innodb オプション
 mysqld, 160

skip-line-numbers option
 mysql, 455

skip-merge オプション
 mysqld, 237

skip-name-resolve オプション
 mysqld, 160, 237

skip-ndbcluster オプション
 mysqld, 160

skip-networking オプション
 mysqld, 160, 237

skip-opt option
 mysqldump, 483

skip-query option
 mysqslap, 493

skip-safemalloc オプション
 mysqld, 161

skip-show-database オプション
 mysqld, 161, 237

skip-slave-start オプション
 mysqld, 335

skip-stack-trace オプション
 mysqld, 161

skip-symbolic-links オプション
 mysqld, 160

skip-thread-priority オプション
 mysqld, 161

slave オプション
 mysqslap, 493

slave-load-tmpdir オプション
 mysqld, 336

slave-net-timeout オプション
 mysqld, 336

slave-skip-errors オプション
 mysqld, 336

slave_compressed_protocol オプション
 mysqld, 335

sleep option
 mysqladmin, 468

SLEEP(), 642

slow queries, 466

slow-query-log オプション
 mysqld, 161

SMALLINTデータタイプ, 543

socket option
 mysql, 455
 mysqladmin, 468
 mysqlbinlog, 471
 mysqlcheck, 477
 mysqldump, 483
 mysqlhotcopy, 487
 mysqlimport, 489
 mysqlshow, 491

socket オプション
 mysqld, 161
 mysqld_safe, 212
 mysqlmanager, 220
 mysqslap, 493

Solaris
 へのインストール, 69

Solaris x86_64 issues, 838

Solaris のインストール時の問題, 114

Solaris のトラブルシューティング, 83

sort-index オプション
 myisamchk, 444

sort-records オプション
 myisamchk, 444

sort-recover オプション
 myisamchk, 444

sort_buffer_size myisamchk 変数, 440

sort_key_blocks myisamchk 変数, 440

SOUNDEX(), 585

SOUNDS LIKE, 585

SPACE(), 585

spassword option
 mysqlaccess, 464

Spatial Extensions in MySQL, 1030

SQL
 定義, 5

SQL スクリプト, 452

SQL ステートメント
 複製スレーブ, 783
 複製マスタ, 780

SQL ノード (MySQL Cluster), 929
 の定義, 875

SQL モード
 ONLY_FULL_GROUP_BY, 647

SQL-92
 に対する拡張子, 17

sql-mode オプション
 mysqld, 161

SQL_BIG_RESULT, 703

SQL_BUFFER_RESULT, 703

SQL_CACHE, 314, 703

SQL_CALC_FOUND_ROWS, 703

SQL_NO_CACHE, 314, 703

SQL_SMALL_RESULT, 703

sql_yacc.cc 問題, 82

SQRT(), 598

SRID(), 1043

SSH, 274

SSL, 268

SSL options
 mysql, 455
 mysqladmin, 468
 mysqlcheck, 477
 mysqldump, 483
 mysqlimport, 489
 mysqlshow, 491

SSL と X509 の基本概念, 267

SSL オプション

- mysqld, 160, 237
- mysqlslap, 493
- ssl オプション, 270
- SSL コマンド オプション, 269
- SSL 関連オプション, 739
- ssl-ca オプション, 270
- ssl-capath オプション, 270
- ssl-cert オプション, 270
- ssl-cipher オプション, 270
- ssl-key オプション, 270
- ssl-verify-server-cert オプション, 270
- standalone オプション
 - mysqld, 160
 - mysqlmanager, 220
- START
 - XA トランザクション, 731
- START BACKUP コマンド (MySQL Cluster),
- START SLAVE, 789
- START TRANSACTION, 724
- START コマンド (MySQL Cluster),
- start-datetime option
 - mysqlbinlog, 471
- start-position option
 - mysqlbinlog, 471
- StartFailureTimeout (MySQL Cluster 設定パラメータ), 907
- StartPartialTimeout, 907
- StartPartitionedTimeout (MySQL Cluster 設定パラメータ), 907
- StartPoint(), 1044
- STARTUP イベント (MySQL Cluster), 940
- STATISTICS
 - INFORMATION_SCHEMA テーブル, 1103
- STATISTICS イベント (MySQL Cluster), 942
- stats オプション
 - mysam_ftdump, 439
- stats_method mysamchk 変数, 440
- status command
 - results, 466
- status option
 - mysqlshow, 491
- STATUS コマンド (MySQL Cluster),
- STD(), 645
- STDDEV(), 645
- STDDEV_POP(), 645
- STDDEV_SAMP(), 645
- STOP SLAVE, 790
- STOP コマンド (MySQL Cluster),
- stop-datetime option
 - mysqlbinlog, 471
- stop-position option
 - mysqlbinlog, 471
- StopOnError, 906
- STRAIGHT_JOIN, 703, 703
- STRCMP(), 589
- STRICT SQL モード, 205
- STRICT_ALL_TABLES SQL モード, 208
- STRICT_TRANS_TABLES SQL モード, 205, 208
- StringMemory, 900
- STR_TO_DATE(), 608
- SUBDATE(), 608
- SUBSTR(), 585

- SUBSTRING(), 585
- SUBSTRING_INDEX(), 586
- SUBTIME(), 609
- suffix option
 - mysqlhotcopy, 487
- SUM(), 645
- SUM(DISTINCT), 645
- superuser option
 - mysqlaccess, 464
- Sybase 互換性, 724
- symbolic-links オプション
 - mysqld, 160
- SymDifference(), 1047
- SYSDATE(), 609
- sysdate-is-now オプション
 - mysqld, 161
- SYSTEM_USER(), 639

T

- tab option
 - mysqldump, 483
- table
 - changing, 1515
- table is full, 1500
- Table is full error
 - MySQL Cluster, 1471, 1474
- table option
 - mysql, 455
 - mysqlaccess, 464
- tables
 - changing column order, 1516
 - deleting rows, 1513
 - displaying, 490
 - dumping, 477, 485
 - flush, 466
 - maintenance, 474
 - maximum size, 1500
 - repair, 474
- TABLES
 - INFORMATION_SCHEMA テーブル, 1101
- tables option
 - mysqlcheck, 477
 - mysqldump, 483
- table_open_cache, 422
- TABLE_PRIVILEGES
 - INFORMATION_SCHEMA テーブル, 1105
- TAN(), 598
- tc-heuristic-recover オプション
 - mysqld, 161
- Tcl API, 1231
- tcp-ip オプション
 - mysqld_multi, 215
- TCP/IP, 57, 61
- tee option
 - mysql, 455
- temp-pool オプション
 - mysqld, 161
- temporary tables
 - problems, 1516

test オプション
 mysampack, 447

testing mysql
 mysqltest, 1430

TEXT
 サイズ, 565

text files
 importing, 487

TEXT カラム
 インデックスする, 666
 デフォルト値, 560

TEXTカラム
 インデックス, 413

TEXTデータタイプ, 547, 559

threads, 466, 1429

time zone problems, 1510

TIME(), 609

TimeBetweenGlobalCheckpoints (MySQL Cluster 設定パラメータ), 908

TimeBetweenInactiveTransactionAbortCheck (MySQL Cluster 設定パラメータ, 908

TimeBetweenLocalCheckpoints (MySQL Cluster 設定パラメータ), 907

TimeBetweenWatchDogCheck, 906

TIMEDIFF(), 609

timeout, 168, 640
 connect_timeout variable, 456, 468
 shutdown_timeout variable, 468

TIMESTAMP
 and NULL values, 1512

TIMESTAMP(), 610

TIMESTAMPADD(), 610

TIMESTAMPDIFF(), 610

TIMESTAMPデータタイプ, 545, 551

timezone オプション
 mysqld_safe, 212

TIME_FORMAT(), 610

TIME_TO_SEC(), 610

TIMEデータタイプ, 545, 555

TINYBLOBデータタイプ, 547

TINYINTデータタイプ, 542

TINYTEXTデータタイプ, 547

tmpdir option
 mysqlhotcopy, 487

tmpdir オプション
 mysampack, 447
 mysqld, 162

TMPDIR 環境変数, 94, 133

to-last-log option
 mysqlbinlog, 471

TODO
 シンボリックリンク, 434

tools
 command-line, 452
 list of, 1722

Touches(), 1049

TO_DAYS(), 610

TRADITIONAL SQL モード, 205, 209

transaction-isolation オプション
 mysqld, 162

TransactionBufferMemory, 902

TransactionDeadlockDetectionTimeout (MySQL Cluster 設定パラメータ), 908

TransactionInactiveTimeout (MySQL Cluster 設定パラメータ), 908

Translators
 list of, 1719

TRIGGERS
 INFORMATION_SCHEMA テーブル, 1109

triggers option
 mysqldump, 483

TRIM(), 586

TRUE, 499, 500
 テスト, 573

TRUNCATE, 719
 および MySQL Cluster,

TRUNCATE(), 599

TZ environment variable, 1510

TZ 環境変数, 133

tz-utc option
 mysqldump, 483

U

UCASE(), 586

UCS-2, 513

UDFs, 1447, 1447
 compiling, 1453
 defined, 1446
 return values, 1452

ulimit, 1503

UMASK environment variable, 1504

UMASK 環境変数, 133

UMASK_DIR environment variable, 1504

UMASK_DIR 環境変数, 133

unbuffered option
 mysql, 455

UNCOMPRESS(), 634

UNCOMPRESSED_LENGTH(), 634

UndoDataBuffer, 910

UndoIndexBuffer, 910

UNHEX(), 586

Unicode, 513

Unicode 照合順序アルゴリズム, 532

UNINSTALL PLUGIN, 1433

uninstalling plugins, 1433

UNION, 709

Union(), 1047

UNIQUE, 653
 制約, 27

Unix, 1236, 1316

UNIX_TIMESTAMP(), 611

UNKNOWN
 テスト, 573

UNLOCK TABLES, 727

unpack オプション
 myisamchk, 444

UNTIL, 1065

UPDATE, 719

update-state オプション

- myisamchk, 442
- UpdateXML(), 627
- UPPER(), 587
- uptime, 466
- USE, 723
- USE INDEX, 699, 705
- USE KEY, 699, 705
- use-frm option
 - mysqlcheck, 477
- use-threads オプション
 - mysqslap, 493
- user option
 - mysql, 455
 - mysqlaccess, 464
 - mysqladmin, 468
 - mysqlbinlog, 471
 - mysqlcheck, 477
 - mysqldump, 483
 - mysqlhotcopy, 487
 - mysqlimport, 489
 - mysqlshow, 491
- user オプション
 - mysqld, 162
 - mysqld_multi, 215
 - mysqld_safe, 212
 - mysqlmanager, 220
 - mysqslap, 494
 - mysql_upgrade, 231
- user テーブル
 - ソート, 249
- USER 環境変数, 133, 246
- USER(), 639
- user-defined functions
 - adding, 1446, 1448
- User-defined functions, 1447, 1447
- username オプション
 - mysqlmanager, 220
- USER_PRIVILEGES
 - INFORMATION_SCHEMA テーブル, 1104
- UTC_DATE(), 611
- UTC_TIME(), 611
- UTC_TIMESTAMP(), 612
- UTF-8, 513
- UUID(), 642

V

- VALUES(), 642
- VARBINARYデータタイプ, 547, 558
- VARCHAR
 - サイズ, 565
- VARCHARACTERデータタイプ, 546
- VARCHARデータタイプ, 546, 557
- VARIANCE(), 645
- VAR_POP(), 645
- VAR_SAMP(), 645
- verbose option
 - mysql, 455
 - mysqladmin, 468
 - mysqlcheck, 477

- mysqldump, 484
- mysqlimport, 489
- mysqlshow, 491
- verbose オプション
 - myisamchk, 440
 - myisampack, 447
 - myisam_ftdump, 439
 - mysqld_multi, 215
 - mysqslap, 494
 - mysql_upgrade, 232
 - my_print_defaults, 438
 - perror, 495
- version
 - の選択, 35
- version option
 - mysql, 456
 - mysqlaccess, 464
 - mysqladmin, 468
 - mysqlbinlog, 471
 - mysqlcheck, 477
 - mysqldump, 484
 - mysqlimport, 489
 - mysqlshow, 491
 - perror, 495
- version オプション
 - myisamchk, 440
 - myisampack, 448
 - mysqld, 162
 - mysqld_multi, 215
 - mysqlmanager, 220
 - mysqslap, 494
 - my_print_defaults, 438
- VERSION(), 640
- vertical option
 - mysql, 456
 - mysqladmin, 468
- Vietnamese, 1479, 1488
- VIEWS
 - INFORMATION_SCHEMA テーブル, 1109
- Vision, 1311
- Visual Objects, 1309
- Visual Studio, 86
- Visual Studio プラグイン, 1235

W

- wait option
 - mysql, 456
 - mysqladmin, 468
- wait オプション
 - myisamchk, 440
 - myisampack, 448
- wait-timeout オプション
 - mysqlmanager, 220
- WEEK(), 612
- WEEKDAY(), 612
- WEEKOFYEAR(), 613
- Well-Known Binary format, 1036
- Well-Known Text format, 1035
- WHERE, 384

- with SHOW, 1099, 1123
- where option
 - mysqldump, 484
- WHILE, 1065
- Windows, 1236, 1316
 - のアップグレード, 62
 - の周知の問題, 65
 - 対 Unix, 63
 - 対 Unix のオペレーティング システム, 63
- Windows での
 - コンパイル, 88
- Windows に Perl を
 - インストールする, 135
- Within(), 1049
- WKB format, 1036
- WKT format, 1035
- write_buffer_size myisamchk 変数, 440

X

- X(), 1043
- X509 証明書, 267
- XA BEGIN, 731
- XA COMMIT, 731
- XA PREPARE, 731
- XA RECOVER, 731
- XA ROLLBACK, 731
- XA START, 731
- XA トランザクション, 730
 - トランザクション識別子, 731
- xid
 - XA トランザクション 識別子, 731
- xml option
 - mysql, 456
 - mysqldump, 484
- XOR
 - 論理, 576

Y

- Y(), 1043
- yaSSL, 267, 268
- YEAR(), 613
- YEARWEEK(), 613
- YEARデータタイプ, 545, 556
- Yen sign (Japanese), 1478, 1482