POSTGRESQL - C/C++ INTERFACE

This tutorial is going to use **libpqxx** library, which is the official C++ client API for PostgreSQL. The source code for libpqxx is available under the BSD license, so you're free to download it, pass it on to others, change it, sell it, include it in your own code, and share your changes with anyone you choose.

Installation

The the latest version of libpqxx is available to be downloaded from the link Download Libpqxx. So download the latest version and follow the following steps:

```
wget http://pqxx.org/download/software/libpqxx/libpqxx-4.0.tar.gz
tar xvfz libpqxx-4.0.tar.gz
cd libpqxx-4.0
./configure
make
make install
```

Before you start using C/C++ PostgreSQL interface, find **pg_hba.conf** file in your PostgreSQL installation directory and add the following line:

```
# IPv4 local connections:
host    all         all         127.0.0.1/32          md5
```

You can start/restart postgres server in case it is not running using the following command:

```
[root@host]# service postgresql restart
Stopping postgresql service:                               [  OK  ]
Starting postgresql service:                               [  OK  ]
```

C/C++ Interface APIs

Following are important interface routines which can sufice your requirement to work with PostgreSQL database from your C/C++ program. If you are looking for a more sophisticated application then you can look into libpqxx official documentation, or you can use commercially available APIs.

S.N.	API & Description
1	**pqxx::connection C**`const std::string & dbstring` This is a typedef which will be used to connect to the database. Here, dbstring provides required parameters to connect to the datbase, for example **dbname=testdb user=postgres password=pass123 hostaddr=127.0.0.1 port=5432**. If connection is setup successfully then it creates C with connection object which provides various useful function public function.
2	**C.is_open** The method is_open is a public method of connection object and returns boolean value. If connection is active, then this method returns true otherwise it returns false.
3	**C.disconnect** This method is used to disconnect an opened database connection.

4 **pqxx::work W*C***

This is a typedef which will be used to create a transactional object using connection C, which ultimately will be used to execute SQL statements in transactional mode.

If transaction object gets created successfully, then it is assigned to variable W which will be used to access public methods related to transactional object.

5 **W.exec** `const std::string & sql`

This public method from transactional object will be used to execute SQL statement.

6 **W.commit**

This public method from transactional object will be used to commit the transaction.

7 **W.abort**

This public method from transactional object will be used to rollback the transaction.

8 **pqxx::nontransaction N*C***

This is a typedef which will be used to create a non-transactional object using connection C, which ultimately will be used to execute SQL statements in non-transactional mode.

If transaction object gets created successfully, then it is assigned to variable N which will be used to access public methods related to non-transactional object.

9 **N.exec** `const std::string & sql`

This public method from non-transactional object will be used to execute SQL statement and returns a result object which is actually an interator holding all the returned records.

Connecting To Database

Following C code segment shows how to connect to an existing database running on local machine at port 5432. Here, I used backslash \ for line continuation.

```
#include <iostream>
#include <pqxx/pqxx>

using namespace std;
using namespace pqxx;

int main(int argc, char* argv[])
{
   try{
      connection C("dbname=testdb user=postgres password=cohondob \
      hostaddr=127.0.0.1 port=5432");
      if (C.is_open()) {
         cout << "Opened database successfully: " << C.dbname() << endl;
      } else {
         cout << "Can't open database" << endl;
         return 1;
      }
      C.disconnect ();
   }catch (const std::exception &e){
      cerr << e.what() << std::endl;
      return 1;
   }
}
```

Now, let's compile and run above program to connect to our database **testdb**, which is already available in your schema and can be accessed using user *postgres* and password *pass123*. You can use user ID and password based on your database setting. Remember to keep the -lpqxx and -lpq in the given order! Otherwise, the linker will complain bitterly about missing functions with names starting with "PQ."

```
$g++ test.cpp -lpqxx -lpq
$./a.out
Opened database successfully: testdb
```

Create a Table

Following C code segment will be used to create a table in previously created database:

```
#include <iostream>
#include <pqxx/pqxx>

using namespace std;
using namespace pqxx;

int main(int argc, char* argv[])
{
   char * sql;

   try{
      connection C("dbname=testdb user=postgres password=cohondob \
      hostaddr=127.0.0.1 port=5432");
      if (C.is_open()) {
         cout << "Opened database successfully: " << C.dbname() << endl;
      } else {
         cout << "Can't open database" << endl;
         return 1;
      }
      /* Create SQL statement */
      sql = "CREATE TABLE COMPANY("  \
      "ID INT PRIMARY KEY     NOT NULL," \
      "NAME           TEXT    NOT NULL," \
      "AGE            INT     NOT NULL," \
      "ADDRESS        CHAR(50)," \
      "SALARY         REAL );";

      /* Create a transactional object. */
      work W(C);

      /* Execute SQL query */
      W.exec( sql );
      W.commit();
      cout << "Table created successfully" << endl;
      C.disconnect ();
   }catch (const std::exception &e){
      cerr << e.what() << std::endl;
      return 1;
   }

   return 0;
}
```

When above program is compiled and executed, it will create COMPANY table in your testdb database and will display the following statements:

```
Opened database successfully: testdb
Table created successfully
```

INSERT Operation

Following C code segment shows how we can create records in our COMPANY table created in above example:

```
#include <iostream>
#include <pqxx/pqxx>

using namespace std;
using namespace pqxx;

int main(int argc, char* argv[])
{
   char * sql;

   try{
      connection C("dbname=testdb user=postgres password=cohondob \
      hostaddr=127.0.0.1 port=5432");
      if (C.is_open()) {
         cout << "Opened database successfully: " << C.dbname() << endl;
      } else {
         cout << "Can't open database" << endl;
         return 1;
      }
      /* Create SQL statement */
      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) "  \
      "VALUES (1, 'Paul', 32, 'California', 20000.00 ); " \
      "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) "  \
      "VALUES (2, 'Allen', 25, 'Texas', 15000.00 ); "     \
      "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)" \
      "VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );" \
      "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)" \
      "VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );";

      /* Create a transactional object. */
      work W(C);

      /* Execute SQL query */
      W.exec( sql );
      W.commit();
      cout << "Records created successfully" << endl;
      C.disconnect ();
   }catch (const std::exception &e){
      cerr << e.what() << std::endl;
      return 1;
   }

   return 0;
}
```

When above program is compiled and executed, it will create given records in COMPANY table and will display the following two lines:

```
Opened database successfully: testdb
Records created successfully
```

SELECT Operation

Following C code segment shows how we can fetch and display records from our COMPANY table created in above example:

```
#include <iostream>
#include <pqxx/pqxx>

using namespace std;
using namespace pqxx;

int main(int argc, char* argv[])
{
```

```
   char * sql;

   try{
      connection C("dbname=testdb user=postgres password=cohondob \
      hostaddr=127.0.0.1 port=5432");
      if (C.is_open()) {
         cout << "Opened database successfully: " << C.dbname() << endl;
      } else {
         cout << "Can't open database" << endl;
         return 1;
      }
      /* Create SQL statement */
      sql = "SELECT * from COMPANY";

      /* Create a non-transactional object. */
      nontransaction N(C);

      /* Execute SQL query */
      result R( N.exec( sql ));

      /* List down all the records */
      for (result::const_iterator c = R.begin(); c != R.end(); ++c) {
         cout << "ID = " << c[0].as<int>() << endl;
         cout << "Name = " << c[1].as<string>() << endl;
         cout << "Age = " << c[2].as<int>() << endl;
         cout << "Address = " << c[3].as<string>() << endl;
         cout << "Salary = " << c[4].as<float>() << endl;
      }
      cout << "Operation done successfully" << endl;
      C.disconnect ();
   }catch (const std::exception &e){
      cerr << e.what() << std::endl;
      return 1;
   }

   return 0;
}
```

When above program is compiled and executed, it will produce the following result:

```
Opened database successfully: testdb
ID = 1
Name = Paul
Age = 32
Address = California
Salary = 20000
ID = 2
Name = Allen
Age = 25
Address = Texas
Salary = 15000
ID = 3
Name = Teddy
Age = 23
Address = Norway
Salary = 20000
ID = 4
Name = Mark
Age = 25
Address = Rich-Mond
Salary = 65000
Operation done successfully
```

UPDATE Operation

Following C code segment shows how we can use UPDATE statement to update any record and then fetch and display updated records from our COMPANY table:

```
#include <iostream>
#include <pqxx/pqxx>

using namespace std;
using namespace pqxx;

int main(int argc, char* argv[])
{
   char * sql;

   try{
      connection C("dbname=testdb user=postgres password=cohondob \
      hostaddr=127.0.0.1 port=5432");
      if (C.is_open()) {
         cout << "Opened database successfully: " << C.dbname() << endl;
      } else {
         cout << "Can't open database" << endl;
         return 1;
      }

      /* Create a transactional object. */
      work W(C);
      /* Create  SQL UPDATE statement */
      sql = "UPDATE COMPANY set SALARY = 25000.00 where ID=1";
      /* Execute SQL query */
      W.exec( sql );
      W.commit();
      cout << "Records updated successfully" << endl;

      /* Create SQL SELECT statement */
      sql = "SELECT * from COMPANY";

      /* Create a non-transactional object. */
      nontransaction N(C);

      /* Execute SQL query */
      result R( N.exec( sql ));

      /* List down all the records */
      for (result::const_iterator c = R.begin(); c != R.end(); ++c) {
         cout << "ID = " << c[0].as<int>() << endl;
         cout << "Name = " << c[1].as<string>() << endl;
         cout << "Age = " << c[2].as<int>() << endl;
         cout << "Address = " << c[3].as<string>() << endl;
         cout << "Salary = " << c[4].as<float>() << endl;
      }
      cout << "Operation done successfully" << endl;
      C.disconnect ();
   }catch (const std::exception &e){
      cerr << e.what() << std::endl;
      return 1;
   }

   return 0;
}
```

When above program is compiled and executed, it will produce the following result:

```
Opened database successfully: testdb
Records updated successfully
ID = 2
Name = Allen
Age = 25
Address = Texas
Salary = 15000
ID = 3
Name = Teddy
Age = 23
Address = Norway
```

```
Salary = 20000
ID = 4
Name = Mark
Age = 25
Address = Rich-Mond
Salary = 65000
ID = 1
Name = Paul
Age = 32
Address = California
Salary = 25000
Operation done successfully
```

DELETE Operation

Following C code segment shows how we can use DELETE statement to delete any record and then fetch and display remaining records from our COMPANY table:

```
#include <iostream>
#include <pqxx/pqxx>

using namespace std;
using namespace pqxx;

int main(int argc, char* argv[])
{
   char * sql;

   try{
      connection C("dbname=testdb user=postgres password=cohondob \
      hostaddr=127.0.0.1 port=5432");
      if (C.is_open()) {
         cout << "Opened database successfully: " << C.dbname() << endl;
      } else {
         cout << "Can't open database" << endl;
         return 1;
      }

      /* Create a transactional object. */
      work W(C);
      /* Create  SQL DELETE statement */
      sql = "DELETE from COMPANY where ID = 2";
      /* Execute SQL query */
      W.exec( sql );
      W.commit();
      cout << "Records deleted successfully" << endl;

      /* Create SQL SELECT statement */
      sql = "SELECT * from COMPANY";

      /* Create a non-transactional object. */
      nontransaction N(C);

      /* Execute SQL query */
      result R( N.exec( sql ));

      /* List down all the records */
      for (result::const_iterator c = R.begin(); c != R.end(); ++c) {
         cout << "ID = " << c[0].as<int>() << endl;
         cout << "Name = " << c[1].as<string>() << endl;
         cout << "Age = " << c[2].as<int>() << endl;
         cout << "Address = " << c[3].as<string>() << endl;
         cout << "Salary = " << c[4].as<float>() << endl;
      }
      cout << "Operation done successfully" << endl;
      C.disconnect ();
   }catch (const std::exception &e){
      cerr << e.what() << std::endl;
```

```
      return 1;
   }

   return 0;
}
```

When above program is compiled and executed, it will produce the following result:

```
Opened database successfully: testdb
Records deleted successfully
ID = 3
Name = Teddy
Age = 23
Address = Norway
Salary = 20000
ID = 4
Name = Mark
Age = 25
Address = Rich-Mond
Salary = 65000
ID = 1
Name = Paul
Age = 32
Address = California
Salary = 25000
Operation done successfully
```