OO for Database Programming (contd.)

Questions:

1. What SQL commands do the following:

   a) delete an existing row from a table

   b) delete all rows from a table without
      destroying the table itself

   c) erase a previously constructed table

   d) add an additional column to a
      previously constructed table

   e) display a table whose rows are ordered
      by the entries in a specific column

2. What facilities does SQL provide for calculating the various statistics for the entries in a specified column of a table.

3. What is JDBC?

4. An SQL statement in a JDBC program is transmitted to a database by supplying the statement as an argument to which method of what class?

5. How does one construct a Statement object in JDBC?

6. When a JDBC program requests information from a database (by invoking, say, SELECT), what is the type of the data object returned by the database?

7. How does one figure out the structure of the information (meaning its rows and columns) retrieved by JDBC from a database?

```
//DBFriends1.java

import java.sql.*;

class DBFriends1 {
    public static void main( String[] args )
    {
        try {
            Class.forName( "org.gjt.mm.mysql.Driver").newInstance();
            String url = "jdbc:mysql:///test";
            Connection con = DriverManager.getConnection( url );
            Statement stmt = con.createStatement();

            stmt.executeQuery( "SET AUTOCOMMIT=1" );
            stmt.executeQuery( "DROP TABLE IF EXISTS Friends" );
            stmt.executeQuery( "DROP TABLE IF EXISTS Rovers" );

            // new table (Friends):
            stmt.executeQuery(
                "CREATE TABLE Friends ( Name CHAR (30) PRIMARY KEY, " +
                                        "Phone INT, Email CHAR(30) )" );
            stmt.executeQuery(
                "INSERT INTO Friends VALUES ( 'Ziggy Zaphod',
                                    4569876, " +  "'ziggy@sirius' )" );
            stmt.executeQuery(
                "INSERT INTO Friends VALUES ( 'Yo Yo Ma', 3472828, " +
                                                " 'yoyo@yippy' )" );
            stmt.executeQuery(
                "INSERT INTO Friends VALUES ( 'Gogo Gaga',
                                    27278927, " + " 'gogo@garish' )" );
```

```
        //new table (Rovers):

        stmt.executeQuery(
            "CREATE TABLE Rovers ( Name CHAR (30) NOT NULL, " +
                                        "RovingTime CHAR(10) )" );
        stmt.executeQuery(
            "INSERT INTO Rovers VALUES ( 'Dusty Dodo', '2 pm' )" );
        stmt.executeQuery(
            "INSERT INTO Rovers VALUES ( 'Yo Yo Ma', '8 pm' )" );
        stmt.executeQuery(
            "INSERT INTO Rovers VALUES ( 'BeBe Beaut', '6 pm' )" );

        // Query: which Friends are Rovers ?
        ResultSet rs = stmt.executeQuery(
            "SELECT Friends.Name, Rovers.RovingTime FROM Friends, "
                    +  "Rovers WHERE Friends.Name = Rovers.Name" );

        ResultSetMetaData rsmd = rs.getMetaData();
        int numCols = rsmd.getColumnCount();
        while ( rs.next() ) {
            for ( int i = 1; i <= numCols; i++ ) {
                if ( i > 1 ) System.out.print( " | " );
                    System.out.print( rs.getString( i ) );
            }
            System.out.println( "" );
        }
        rs.close();
        con.close();
    } catch(Exception ex ) { System.out.println(ex); }
  }
}
```

To compile a JDBC program, you'd need to tell `javac` how to locate the database driver.

If the driver is in a JAR file named `mm.mysql-2.0.7-bin.jar`, an invocation like the following should work

```
javac -classpath .:~/mm.mysql-2.0.7-bin.jar DBFriends1.java
```

You'd also need to specify the classpath for the `java` application launcher:

```
java -classpath .:~/mm.mysql-2.0.7-bin.jar DBFriends1
```

Large databases can obviously not be created from within JDBC programs one row at a time.

The information that you want to enter into a database is more likely to be found in the form of what's known as a *flat file*.

For illustration, we may have the following information in a flat file called `Friends.txt`:

```
Doctor Cosmos   876-6547   zinger@zoros    68   0    73   galacticSoccer
Yo Yo Ma        838-9393   yoyo@yahoo      56   1    0    violaHockey
Zinger Zaphod   939-1717   dodo@dada       23   0    2    tennis
Bebe Beaut      84-83838   bebe@parlor     18   1    3    tennis
```

Each row of this text file has a name, a phone number, an e-mail address, age, whether or not married, number of kids, and the name of the favorite sport.

The entries in each row are tab separated, but can also be supplied in what's known as the *comma separated values* (csv) form.

You can also have flat files in which each field is given a fixed number of positions.

So let's say that we want a database system to read this flat file and create a `Friends` database.

In some systems this can be done with the help of another text file, known usually as a database table's *schema*, which tells the system how to interpret the position of each data item in a row of the flat file.

In MySQL, the same is most easily accomplished by first creating a table directly with the `CREATE TABLE` command and then invoking `LOAD DATA INFILE` to read in the data from the flat file into the database table.

The next JDBC program executes the MySQL statement `LOAD DATA INFILE` for the creation of two database tables named `Friends` and `SportsClub`.

The table `Friends` is created from the flat file shown earlier.

The table `SportsClub` is created from the flat file:

```
Hobo Hooter       45    hockey           4
Doctor Cosmos     68    galacticSoccer   9
Zinger Zaphod     23    tennis           2
Bebe Beaut        84    tennis           10
```

where the entries are in the following order: name, age, sport, and the level at which sport is played.

For the query, the program executes the SQL statement

```
SELECT ... WHERE
```

to seek out friends who play tennis at the sports club.

```
//DBFriends2.java

import java.sql.*;

class DBFriends2 {
    public static void main( String[] args )
    {
        try {
            Class.forName( "org.gjt.mm.mysql.Driver").newInstance();
            String url = "jdbc:mysql:///test";
            Connection con = DriverManager.getConnection( url );
            Statement stmt = con.createStatement();

            stmt.executeQuery( "SET AUTOCOMMIT=1" );
            stmt.executeQuery( "DROP TABLE IF EXISTS Friends" );
            stmt.executeQuery( "DROP TABLE IF EXISTS SportsClub" );

            stmt.executeQuery(
                "CREATE TABLE Friends ( Name CHAR (30) PRIMARY KEY, " +
                                    "Phone CHAR (15), Email CHAR(30), " +
                                      "Age TINYINT (3), Married BOOL, " +
                                "NumKids TINYINT (3), Sport CHAR(20) )"
            );
            stmt.executeQuery(
             "CREATE TABLE SportsClub ( Name CHAR (30) PRIMARY KEY, " +
                                    "Age TINYINT (3), Sport CHAR(20), " +
                                     "Level Char(20) )"
            );
            stmt.executeQuery(
               "LOAD DATA LOCAL INFILE 'Friends.txt' INTO TABLE " +
                                          " Friends" );
```

```
        stmt.executeQuery(
            "LOAD DATA LOCAL INFILE 'SportsClub.txt' INTO " +
                          " TABLE SportsClub" );

        // which of the Friends also play tennis at the club:
        ResultSet rs = stmt.executeQuery(
           "SELECT Friends.Name, SportsClub.Level FROM Friends, "
            + "SportsClub WHERE "
            + "Friends.Name = SportsClub.Name AND "
            + "Friends.Sport = SportsClub.Sport AND "
            + "Friends.Sport = 'tennis' " );

        ResultSetMetaData rsmd = rs.getMetaData();
        int numCols = rsmd.getColumnCount();

        while ( rs.next() ) {
            for ( int i = 1; i <= numCols; i++ ) {
                if ( i > 1 )
                    System.out.print( " plays tennis at level " );
            System.out.print( rs.getString( i ) );
            }
            System.out.println( "" );
        }
        rs.close();
        con.close();
    } catch(Exception ex ) { System.out.println(ex); }
  }
}
```

The output of this program is

```
Zinger Zaphod plays tennis at level 2
Bebe Beaut plays tennis at level 10
```

Mysql++ Programming: Invoking SQL through C++

The open source Mysql++ is a C++-based programming interface for communicating with a MySQL database.

The future releases of Mysql++ releases are expected to work with SQL databases in general.

Mysql++ is available from `http://www.mysql.com/download_mysql++.html`.

Just like JDBC, Mysql++ can communicate your command to a database and can retrieve, analyze, and display the results.

Basic to Mysql++ programming are the Connection, Query, and Result classes.

They play the same roles in Mysql++ that Connection, Statement, and ResultSet classes play in JDBC.

The Connection class gives you a connection with a database. Its constructor can be invoked directly with the database name as the sole argument, as in

```
Connection con( "myDatabase" );
```

where "myDatabase" is the name of a MySQL database on the local machine and if the database user name is the same as the login name.

For accessing a database on a different machine or when the database user name is not the same as the login name and if a password is required for accessing the database, you'd need to invoke the Connection constructor with additional arguments.

The prototype of the constructor is

```
Connection( cchar* db, cchar* host="",
                       cchar* user="", cchar* passwd="" );
```

Another way to establish a connection with a database is to first carry out a partial construction of a Connection object with the use_exceptions option turned on and to then separately establish a connection with the designated database:

```
Connection con( use_exceptions );
con.connect( "myDatabase" );
```

There is also available a 9-argument constructor for Connection that allows specification of a port, a socket, connection timeout, and so on.

When a connection is established with the two-call invocation shown above, the connect method can also be invoked with up to four arguments, the additional arguments allowing specificatio of a hostname, a user name, and password.

Invoking the function query on a Connection object returns
a Query object that can then be used to communicate SQL statements
to the database:

```
Connection con( .... );
Query query = con.query();
```

A Query object behaves much like an output stream object in
C++, such as cout.

You can use the insertion operator ‘<<’ to insert strings
in a Query object. These strings will usually be SQL statements.

How you get those SQL statements to execute depends upon whether or not their execution is supposed to return something.

When an SQL statement fed into a Query object is not supposed to return anything --- such as when inserting a new row into a table --- the inserted statement can be executed by invoking the function execute on the Query object, as in

```
Query query = ....
query <<  ...  SQL statement ...
query.execute();
```

If it is desired to reset the state of the Query object, the last call can also be of the form that includes the flag RESET_QUE

```
query.execute( RESET_QUERY );
```

On the other hand, when the SQL statement is supposed to return a result set, you need to invoke store, which causes execution of the SQL statement and retrieval of a result-set object of type Result:

```
Query query = ....
query <<  ...  SQL SELECT statement ...
Result res = query.store();
```

A Result object is an STL-like container, supporting a random-acce
read-only iterator and array-like indexing.

Each element stored in a Result object corresponds to a row
of the result set and is again an STL-like container that
also supports array-like indexing.

Both the Result object and the rows stored therein as elements
can be accessed via iterators.

Since both the row elements of a result set and the elements inside each row can be accessed by array-like indexing, the result set can be thought of as a two-dimensional table directly addressable by a pair of indices, as in

```
Result res = query.store();
cout << res[2][5];
```

which would display the datum in the column indexed 5 of the result set row indexed 2.

```
//DBFriends1.cc

#include <iostream>
#include <sqlplus.hh>
#include <iomanip>

int main()
{
    try {
        Connection con( use_exceptions );
        con.connect( "test" );
        Query query = con.query();
        query << "SET AUTOCOMMIT=1";
        query.execute();
        query << "DROP TABLE IF EXISTS Friends";
        query.execute();
        query << "DROP TABLE IF EXISTS Rovers";
        query.execute();

        // Friends table:
        query << "CREATE TABLE Friends ( Name CHAR (30) "
              << "PRIMARY KEY, Phone INT, Email CHAR(30) )";
        query.execute();
        query << "INSERT INTO Friends VALUES ( 'Ziggy Zaphod', "
              << "4569876, 'ziggy@sirius' )";
        query.execute();
        query << "INSERT INTO Friends VALUES ( 'Yo Yo Ma', "
              << "3472828, 'yoyo@yippy' )";
        query.execute();
        query << "INSERT INTO Friends VALUES ( 'Gogo Gaga', "
              << "27278927, 'gogo@garish' )" ;
```

```
query.execute();

// Rovers table:
query << "CREATE TABLE Rovers ( Name CHAR (30) NOT NULL, "
      << "RovingTime CHAR(10) )";
query.execute();
query << "INSERT INTO Rovers VALUES ( 'Dusty Dodo', '2 pm' )";
query.execute();
query << "INSERT INTO Rovers VALUES ( 'Yo Yo Ma', '8 pm' )";
query.execute();
query << "INSERT INTO Rovers VALUES ( 'BeBe Beaut', '6 pm' )";
query.execute();

query << "SELECT Friends.Name, Rovers.RovingTime "
      << "FROM Friends, Rovers WHERE Friends.Name=Rovers.Name";

// The result set:
Result res = query.store();
cout << "Query: " << query.preview() << endl;
cout << "Records Found: " << res.size() << endl << endl;

Row row_rs;
cout.setf(ios::left);
Result::iterator i;
for ( i = res.begin(); i != res.end(); i++ ) {
    row_rs = *i;
    int numFields = row_rs.size();
    if ( i == res.begin() ) {
        for ( int j = 0; j < numFields; j++ )
            cout << setw( 17 ) << res.names( j ) << "\t\t";
        cout << endl << endl;
```

```
            }
            for ( int j = 0; j < numFields; j++ )
                cout << setw( 17 ) << row_rs[ j ] << "\t";
            cout << endl;
        }
    } catch ( BadQuery& er ) {
        cerr << "Query Error: " << er.error << endl;
        return -1;
    } catch( BadConversion& er ) {
        cerr << "Conversion Error: Tried to convert \""
             << er.data << "\" to a \""
             << er.type_name << "\"." << endl;
        return -1;
    }
}
```

This program can be compiled by the following command line that can be conveniently placed in a shell script:

```
g++ -o DBFriends1 DBFriends1.cc -I/usr/include/mysql    \
            -lsqlplus -Wl,--rpath -Wl,/usr/local/lib
```

This assumes a standard installation of Mysql++ on a Linux machine. The output of the program is the same as for the Java example DBFriends1.java.

```
//DBFriends2.cc

#include <iostream>
#include <sqlplus.hh>
#include <iomanip>

int main()
{
    try {
        Connection con( use_exceptions );
        con.connect( "test" );
        Query query = con.query();
        query << "SET AUTOCOMMIT=1";
        query.execute();
        query << "DROP TABLE IF EXISTS Friends";
        query.execute();
        query << "DROP TABLE IF EXISTS SportsClub";
        query.execute();

        query << "CREATE TABLE Friends ( Name CHAR (30) PRIMARY KEY, "
              << "Phone CHAR (15), Email CHAR(30), "
              << "Age TINYINT (3), Married BOOL, "
              << "NumKids TINYINT (3), Sport CHAR(20) )";
        query.execute();

        query << "CREATE TABLE SportsClub (Name CHAR (30) PRIMARY KEY,"
              << "Age TINYINT (3), Sport CHAR(20), "
              << "Level Char(20) )";
        query.execute();

        query << "LOAD DATA LOCAL INFILE 'Friends.txt' "
```

```
      << "INTO TABLE Friends";
query.execute();
query << "LOAD DATA LOCAL INFILE 'SportsClub.txt' INTO "
      << " TABLE SportsClub";
query.execute();

// which of the Friends also play tennis at the club:
query << "SELECT Friends.Name, SportsClub.Level FROM Friends, "
      << "SportsClub WHERE "
      << "Friends.Name = SportsClub.Name AND "
      << "Friends.Sport = SportsClub.Sport AND "
      << "Friends.Sport = 'tennis' ";

Result res = query.store();
cout << "Query: " << query.preview() << endl;
cout << "Records Found: " << res.size() << endl << endl;

Row row_rs;
cout.setf(ios::left);

Result::iterator i;
for ( i = res.begin(); i != res.end(); i++ ) {
    row_rs = *i;
    int numFields = row_rs.size();
    if ( i == res.begin() ) {
        for ( int j = 0; j < numFields; j++ )
            cout << setw( 17 ) << res.names( j ) << "\t\t";
        cout << endl << endl;
    }
    for ( int j = 0; j < numFields; j++ )
        cout << setw( 17 ) << row_rs[ j ] << "\t";
```

```
            cout << endl;
        }
    } catch ( BadQuery& er ) {
        cerr << "Query Error: " << er.error << endl;
        return -1;
    } catch( BadConversion& er ) {
        cerr << "Conversion Error: Tried to convert \""
             << er.data << "\" to a \""
             << er.type_name << "\"." << endl;
        return -1;
    }
}
```

This program can be compiled by the following command line that can be conveniently placed in a shell script:

```
g++ -o DBFriends2 DBFriends2.cc -I/usr/include/mysql    \
            -lsqlplus -Wl,--rpath -Wl,/usr/local/lib
```

As before, this assumes a standard installation of Mysql++ on a Linux machine. The output of the program is the same as for the Java example DBFriends2.java.