JDBC: A simple tutorial

The following tutorial will be useful if you are developing the application in Java. Note, you are not restricted to using the java environment only. Java provides the programmers an API popularly called the JDBC (Java Database connectivity) to connect and. We will in this document provide the details of utilizing the power of this API.

Installing the JDBC driver:

The first step is to install the JBDC driver. The MySQL JDBC driver is available at:

http://dev.mysql.com/downloads/connector/j/3.0.html

The JBDC driver is a jar file. All you need to do is to place this jar file in the classpath. Once this is done, you will be able to access the functionality provided by the API.

Simple Test Program:

Once you are done installing the JDBC driver. You are in a position to execute the following simple java program. This java program illustrates the following aspects of JDBC: a) creation of a table b) deletion of a table c) inserting into a table and d) querying a table.

```
import java.sql.*;

public class DBConnector {

  private Connection con = null;
  private String username = "rjammala";
  private String password = "********";
  private String url =
              "jdbc:mysql://MYComputer_IP/Project2";


  public void ConnecToDB() {

      try {
              Statement stmt;
              ResultSet rs;
              //Register the JDBC driver for MySQL.
              Class.forName("com.mysql.jdbc.Driver");

              //Get a connection to a database...
              con =      DriverManager.getConnection(
                           url, username, password);

       }catch(Exception e){
              System.out.println(e);
```

```
     }

     return ;
}


public Connection ReturnConnectionObject() {
       return con;
    }


 public static void main ( String args[]) throws Exception  {


      DBConnector DBCon  = new DBConnector();
      DBCon.ConnecToDB();

      Connection con = DBCon.ReturnConnectionObject();


      Statement stmt = con.createStatement();

      String SQL = "DROP TABLE simple";
      stmt.execute(SQL);

      SQL = "CREATE TABLE simple( temp varchar(300))";

      stmt.execute(SQL);

      System.out.println("table created");


    // Inserting something into the

      SQL = "Insert into simple values('10')";

      stmt.execute(SQL);

      System.out.println("Tuple inserted in the table");


      SQL = "Select * from simple";

      ResultSet rs;
      rs = stmt.executeQuery(SQL);

          while ( rs.next()) {

            System.out.println( "temp:" + rs.getString(1));

        }

      System.out.println("Congratulations you are done");


  }
```

We will now explain some parts of the above java program.

“ Class.forName("com.mysql.jdbc.Driver");}

The above statement loads all the classes related to the JDBC driver in the program. Now you will able to access all the required classes in JDBC.

```
private String username = "rjammala";
private String password = "********";
private String url =
            "jdbc:mysql://MYComputer_IP/Project2";
```

The above information provides the java program with the necessary information to connect to your database. Assume that you end up creating a Project2 database in your MYSQL server. Then string url specified above will direct the java program to look at the right place for storing and retrieving data. MyComputer_IP represents the IP address of the machine on which the SQL server is running. Variables username and password have the obvious meaning. Such information is required for the user to authenticate with the SQL database.

“ con = DriverManager.getConnection(
url, username, password); “

The above statement creates a connection object that can now communicate with your database.

Now let us see what can be done with the *con* object.

Consider the following statements:

“

```
Statement stmt = con.createStatement();
String SQL = "Select * from simple";
rs = stmt.execute(SQL);
```

The connection class will allow you to create statement objects that can now directly execute the SQL statements, as illustrated in the above example.

After executing the queries, JDBC allows you to capture the output in the form of a resultset object. Use can then use the resultset object to print out the final results.

“

```
while ( rs.next()) {
    System.out.println( "temp:" + rs.getString(1));
}”
```

The above code loops through the resultset object to printout the relevant results. The *getString(i)* function returns the string representation of the value of i th column in table. Note, that you need to make sure when you use getString function, that the corresponding column in the table is a string. There are other functions available that other data types.

Please contact me rjammala@ics.uci.edu if you have further questions about the material presented above.