Lab # 9

Java to Database Connection

Eng. Haneen El-Masry

December, 2014
Objective

In this lab, we turn our attention to how SQL can be called from the Java object-oriented programming language.

**JDBC**

Java Database Connectivity (JDBC) is:

- An API for the Java programming language that defines how a client may access a database.
- The function libraries for SQL access from Java.
- It provides methods for querying and updating data in a database.

**JDBC Driver** is basically an implementation of the function calls specified in the JDBC API.

**Database Connection from Java**

1- Add JDBC Driver to the Java project.

- Right click on Libraries >> Add JAR/Folder.
➤ Browse to JDBC Driver >> Open.

➤ JDBC Driver is added to the project libraries.
2- Create a new class, especially for Connection. Let it to be DBConnection Class.

DBConnection Class

```java
package javadbconnection;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
public class DBConnection {
    static Connection connection;
    static Statement statement;
    public static void connect()
    {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            connection=DriverManager.getConnection(url,"hr","hr");
            System.out.println("Successful Connection");
            statement=connection.createStatement();
            System.out.println("Statement Created Successfully");
            System.out.println("Now, You can Access Database ^_^");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

DBConnection Class Description

- **import java.sql.*:**
  The JDBC library of classes must be imported into the Java program. These classes are called java.sql.*, and can be imported.

- **Class.forName("oracle.jdbc.driver.OracleDriver");**
  The method Class.forName(String s) is used to load the JDBC driver class.

Notes:
- When a Driver class is loaded, it creates an instance of itself and registers it with the **DriverManager**.
The Java exception “ClassNotFoundException” occurs if the driver is not loaded successfully.

Connection object: a Connection object is created using the getConnection function of the DriverManager.

Connection Parameters

1- URL

With JDBC, a database is represented by a URL (Uniform Resource Locator). With Oracle, URL takes the following form:

```
jdbc:oracle:thin:@host:port:database
```

The parameters have the following meanings:

- **Host**: Database server IP, or Name. Defaults to localhost.
- **Port**: The port number the server is listening on. Defaults to the Oracle standard port number (1521).
- **Database**: The database name. Defaults to xe.

2- **UserName**: The database user on whose behalf the connection is being made.

3- **Password**: The database user's password.

Statement object: In JDBC, there is a basic statement class, Statement. The Statement object is created using the createStatement function of the Connection class of JDBC.

**Note:**

Declare Statement and Connection objects as static, to be shared objects between the project classes.
3- Call connect method in your main method to connect to the database.

```java
package javadbconnection;
public class JavaDBConnection {
    public static void main(String[] args) {
        DatabaseConnection.connect();
    }
}
```

4- Run the file that contains the main method to ensure from your connection.

**Database Access**

Anywhere you want to access the database, you need to do the following steps:

1- **Check the connection is still open.**
2- Define a String holds SQL Statement that you want to execute.
3- Execute SQL Statement.

**Notes:**

If the statement is a Query (Select Statement), execute it by `executeQuery` method of the statement object that takes the Query as its parameter.

- `executeQuery` method returns an object of type ResultSet, that resembles a two-dimensional array or a table, where the tuples are the rows and the attributes returned are the columns.
- When the query is executed, ResultSet object refers to a tuple before the first tuple in the query result.
- The `rs.next()` function moves to the next tuple (row) in the ResultSet object and returns NULL if there are no more objects. This is used to control the looping.
- You can refer to the attributes in the current tuple using (getString, getInteger, getDouble, and so on) functions that depend on the type of each attribute.
- Also you can either use the attribute positions (1, 2) or the actual attribute names.

If the SQL statement is insert, update or delete statement, execute it by `executeUpdate` method of the statement object that takes the Query as its parameter.
- `executeUpdate` method returns an integer value indicating the number of tuples that were affected.
Let us to create HRAccess class to implement methods to access HR schema then call these methods in the main method.

```java
package jdbcconnection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.*;
public class HRAccess {
    public void getRegions (){
        try {
            if(DBConnection.connection==null || DBConnection.connection.isClosed())
                DBConnection.connect();
            String sql="SELECT * FROM REGIONS";
            ResultSet rs=DBConnection.createStatement().executeQuery(sql);
            int id;
            String name;
            System.out.printf("%10s%30s","REGION_ID","REGION_NAME\n");
            while(rs.next()){
                id=rs.getInt("region_id");
                name=rs.getString(2);
                System.out.printf("%10d%30s",id,name);
            }
        } catch (SQLException ex) {
            Logger.getLogger(HRAccess.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```java
package jdbcconnection;
public class JavaDBConnection {
    public static void main(String[] args) {
        DBConnection.connect();
        HRAccess HR=new HRAccess ();
        HR.getRegions();
    }
}
```

Output - JavaDBConnection [run] X

```
Successful Connection
Statement Created Successfully
Now You can Access Database ^_^
REGION_ID  REGION_NAME
1  Europe
2  Americas
3  Asia
4  Middle East and Africa
BUILD SUCCESSFUL (total time: 2 seconds)
```
```java
public static void main(String[] args) {
    DBConnection.connect();
    HRAccess HR=new HRAccess();
    if(HR.insertRegion(5,"Test Region")>0)
        System.out.println("New Region inserted Successfully :)");
    HR.getRegions();
}
```
```java
38 int updateRegion(int id, String name) {
39     int rows = 0;
40     try {
41         if (DBConnection.connection == null || DBConnection.connection.isClosed())
42             DBConnection.connect();
43         String sql = "UPDATE REGIONS SET REGION_NAME='" + name + "' WHERE REGION_ID='" + id + ";"
44         rows = DBConnection.statement.executeUpdate(sql);
45     } catch (SQLException ex) {
46         Logger.getLogger(HRAccess.class.getName()).log(Level.SEVERE, null, ex);
47     }
48     return rows;
49 }
```

```java
4 public static void main(String[] args) {
 5     DBConnection.connect();
 6     HRAccess HR = new HRAccess();
 7     /*if(HR.insertRegion(5,"Test Region")>0)
 8         System.out.println("New Region inserted Successfully :");*/
 9     if (HR.updateRegion(5, "Updated Region")>0)
10         System.out.println("Region updated Successfully :"));
11     HR.getRegions();
```

Output - JavaDBConnection (run) X

- Successful Connection
- Statement Created Successfully
- Now, You can Access Database ^_^
- Region updated Successfully :)

<table>
<thead>
<tr>
<th>REGION_ID</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Europe</td>
</tr>
<tr>
<td>2</td>
<td>Americas</td>
</tr>
<tr>
<td>3</td>
<td>Asia</td>
</tr>
<tr>
<td>4</td>
<td>Middle East and Africa</td>
</tr>
<tr>
<td>5</td>
<td>Updated Region</td>
</tr>
</tbody>
</table>

BUILD SUCCESSFUL (total time: 1 second)
```java
public static void main(String[] args) {
    DBConnection.connect();
    HRAccess HR = new HRAccess();
    /*if(HR.insertRegion(5,"Test Region")>0)
     */ System.out.println("New Region inserted Successfully : ");
    /*if(HR.updateRegion(5, "Updated Region")>0)
     */ System.out.println("Region updated Successfully : ");
    if(HR.deleteRegion(5)>0)
     System.out.println("Region deleted Successfully : ");
    HR.getRegions();
}
```
Exercise

Add an Authentication table to any of yours schems, then Design a simple Java GUI application that authenticate the user who want to access the schema. If he is authenticated, let him to access the Database using various DML statements.

Authentication Table Example

<table>
<thead>
<tr>
<th>FName</th>
<th>LName</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>snow</td>
<td>white</td>
<td>123456</td>
</tr>
</tbody>
</table>

Your work can be like the following:
Best Wishes 😊