CSC9Q5 / ITNP33 Practical

Introduction to Java and MySQL

Computing Science, University of Stirling

So far we have been using PHPMyAdmin as an interface to MySQL. For some things, this is fine, but for many other applications, it is of no use at all. Now we will start to access a MySQL database from java code.

The JDBC API
Connecting to a database from Java is done using the JDBC API. This is not MySQL specific – it will connect to any DB, but we will use it with MySQL. To use JDBC with MySQL you need to use the MySQL Connector, which is available to download for free from MySQL. The current version is called mysql-connector-java-5.1.26-bin.jar. This has been placed in C:\Program Files\Java\jre7\lib\ext on the lab machines, but you might need to tell the development environment (BlueJ in our case) about it.

You won't need to download it for the practical, but if you need it for your own computer, you can find it here:

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

Don't write any code until you have read this whole sheet. The first part is mostly for information. The second part (**What to Do**) gives more specific instructions.

Import Libraries
You will need to include the following libraries:

```
java.sql.Connection;
java.sql.DriverManager;
java.sql.ResultSet;
java.sql.SQLException;
java.sql.Statement;
```

Loading the Driver
It used to be the case that before you can connect to a database, you need to load the MySQL JDBC driver. This is done in your Java code using:

```
Class.forName("com.mysql.jdbc.Driver");
```

but JDBC 4.0 drivers must include the file META-INF/services/java.sql.Driver, which contains the name of the driver class. This is loaded automatically and so the forName call is no longer needed. You will still see it in old code and online examples though.

Have a look in the META-INF file for the MySQL driver .jar and see what it contains.

Connecting to a Database
To connect to a database, use

Connection connect =
DriverManager.getConnection("jdbc:mysql://**url**?user=**user**&password=**pass**");

Replace the **bold** terms with your own details. The database we are connecting to is at the URL shark.cs.stir.ac.uk/**username**.

Connecting in this way can throw an SQLException, so deal with that with a try{}catch structure or have your method throw the same exception.

There are other ways of calling getConnection. You can read about them here:

http://docs.oracle.com/javase/6/docs/api/java/sql/DriverManager.html

From JDBC 2.0, connection via a DataSource is preferred. You can read about this here:

http://docs.oracle.com/javase/tutorial/jdbc/basics/sqldatasources.html

Running a Query
Once you have connected, you are ready to construct and execute a query. One format for this is as follows:

```
String sql = "your SQL query";
Statement statement = connect.createStatement();
ResultSet rs = statement.executeQuery(sql);
Type res;                                   // Type can be int, String, … etc
while (rs.next())
{
        res = rs.getType("Field name");     // Type is type of res
// Do something with res
 }
```

To make this work, you will need to decide on the proper data type for the variable rs and make the right call to get…..(). For example, you might use:

String res = rs.getString("TelephoneNum");

To get the field called TelephoneNum from the result. Note the while loop. If you know there will be only one row in your result, you needn't use it, but you still need to call *next*() once.

What to Do

Open BlueJ and add the MySQL driver to the build path. Create a project with a single class called testDB.

We will start with a simple example that just runs in the main() method, so create one for your class. Make sure you have imported the libraries you need.

Now write the code to create a connection to your database on shark. Use a local variable for the Connection object.

Choose a table in your database and write a simple query to select everything from it. Inside the loop that gets each row, use System.out.println() to output the contents of the row. For this first example, you can hard-code it all, knowing the exact structure of the table.

Once you have that working, you can try something more general. First, move the Connection object out of the *main*() method and make it a member of your testDB class.

In your existing class, create a new method called *connect*() and move the connection code into it.

Now add a new method called *queryArray*(*String qry*), which takes a String as its only parameter. The string will be the SQL query to be run. Write the code to run the query given in the parameter *qry* and put the results into an array of Strings. Make the return type of the method *String[]* and make sure your method returns an array that is the first result of running the query. Note, your method does not need to loop through the results – it should only return one row.

For example:

String[] row = queryArray("SELECT * FROM Employees WHERE `Employee Number`=234621");

would return an array containing the details of the entry in the *Employees* table from the row where Employee Number =234621. Note that you still need to call resultSet.next() once.

How might you improve this method? If you have time once you have completed the rest of the practical, take a look at Java HashMaps and make use of them to change the return type to *Map*.

Also, if you have time, write another similar method that picks a single field and returns an array containing the value of the chosen field for every row in the query result.

Building Queries and Prepared Statements

Rather than hard coding the contents of a query, you will probably want to make a query that looks up the value held in a variable in your Java code. Let's try two ways of doing that.

First, you can build a query string using Java String concatenation, which is done as

String = String + String;

For example "SELECT * FROM Employees WHERE `Employee Number`="+en;

Where *en* is a variable that holds the employee number. Note that you can concatenate Strings with other types, for example, the variable *en* might be an int.

Write a new method which takes an employee number as an integer parameter and returns the Employee name as a String:

```
public String getEmployeeName(int num)
{
        // Construct a string and execute the query
}
```

An alternative way is to use prepared statements. You will need to import the class:

java.sql.PreparedStatement

and use the following syntax:

PreparedStatement p = connect.PreparedStatement(String sql)

The sql in your String parameter should have a ? in place of the value you wish to look up, so in this case, it should replace the employee number.

To set a specific value, you use:

p.setString(int index, String value);

Indices start at 1, so in the example here, that is the index of the ? for employee number. The String is the value you want to search for. In this case, it is the employee number, which needs to be changed from an int to a String. You can use parseInt or cheat and use ""+num;