Using SQL with Java

Summary

This guide provides a brief overview of how to manipulate an MS-Access database with SQL statements in a Java program. It was compiled using MS-Access 2016, Netbeans 8.1 (using JDK 1.8) and incorporated the UCanAccess 3.0.3.1 JDBC driver.

Step 1: Gather your resources

Ensure that you have MS-Access installed as well as Netbeans and JDK. Download the UCanAccess driver package from http://ucanaccess.sourceforge.net/site.html and extract all the files to a convenient location. Suggestion: copying all six of the *.jar* files into a separate, single folder makes it easier to import them later, but this isn't entirely necessary.

Step 2: Create a database

Use MS-Access to create a database saved as *books.accdb*, add a table called *tblBooks* with the following structure:

Field	*Type*
id	Autonumber
title	Text[200]
author	Text[50]
price	Currency
hardcover	Yes/No

Populate the table with at least 10 records.

Step 3: Create a Netbeans project

Create a new Netbeans project (Java Application without a main class). Create a simple GUI screen with a text area for output and two buttons labelled *Show books* and *Add book*.

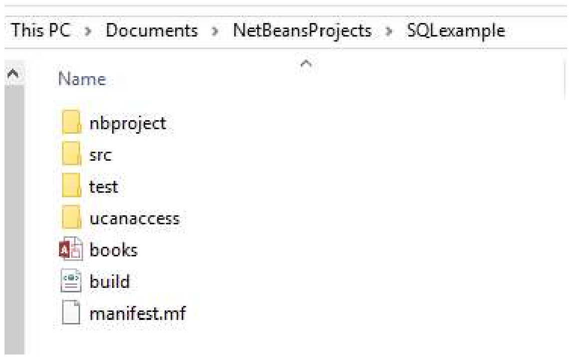
Step 4: Set up the project for it to access to the database

Create a new class called *DatabaseConnection* that looks as follows:

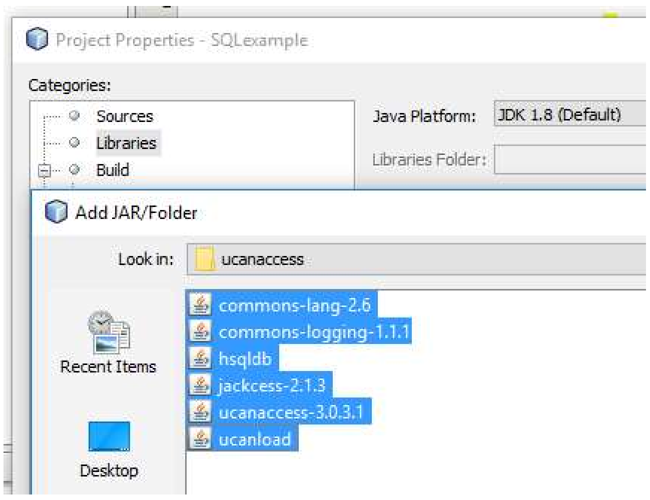
```
import java.sql.*;

public class DatabaseConnection {

    private static Connection conn = null;

    //constructor method: sets up connection
    public DatabaseConnection(){
        try{
            conn=DriverManager.getConnection("jdbc:ucanaccess://books.accdb");
            System.out.println("Connection successful");
        }
        catch(Exception e){
            System.out.println("Connection NOT successful\n");
            e.printStackTrace();
        }
    }
}
```

You may want to save a copy of this class for future use before you edit it any further. Later, once you are sure your connection is working, you can comment out the "Connection successful" line. To connect to a different database, simply replace the name of the database file "books" with the name of the database you want to use.

Place a copy of your *books.accdb* database file in the root of the Netbeans project folder. Then copy the UCanAccess folder that you extracted earlier into the root of the project folder. You can either use the complete extracted folder as is, or you can use a folder that you created that contains only the six *.jar* files. Here is a screenshot of my folder at this point (I copied the *.jar* files into a single folder called *ucanaccess*):

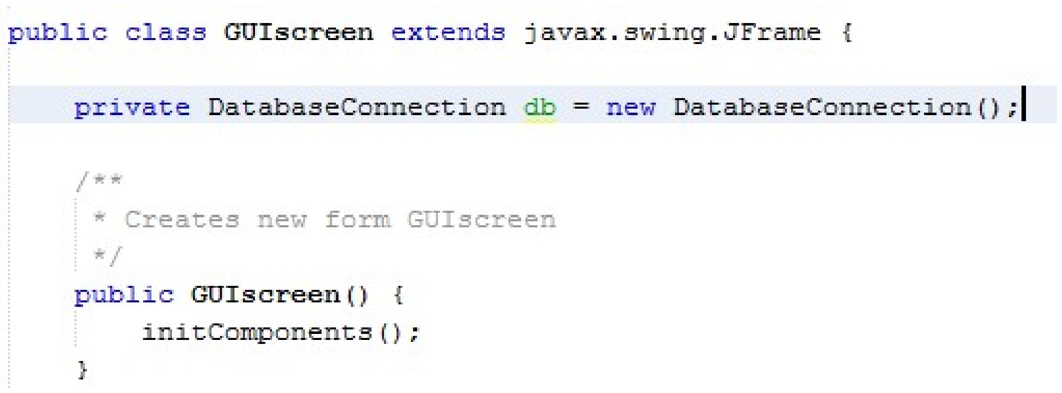


Now add the *UCanAccess .jar* files to the project by clicking: *File | Project Properties*. Select *Libraries* from the list on the left and then click the button on the right labelled *Add JAR/Folder.* Navigate to your Netbeans project folder and then into your folder containing the *UCanAccess .jar* files. You need to add all six of the *.jar* files. If they are all in one folder it is much easier to highlight and add them all at once (click the first one, press shift and click the last one). Here is a screenshot of this:



Make sure that “Relative Path” is selected on the right side of the screen before clicking the *Open* button on the bottom right of the window and then *OK*.

Create a private object in your GUI class of type *DatabaseConnection* called *db* that has class-wide scope. Here is a screenshot to help you:



```
public class GUIscreen extends javax.swing.JFrame {

    private DatabaseConnection db = new DatabaseConnection();

    /**
     * Creates new form GUIscreen
     */
    public GUIscreen() {
        initComponents();
    }
```

Step 5: Use SQL to manipulate the database

To allow for the separation of our GUI from the working class we place all the database manipulation methods in the *DatabaseConnection* class and call them using the *db* object in our GUI class.

Create the following method in your DatabaseConnection class:

```
public String displayRecords(){
    String str = "";
    try{
        Statement s = conn.createStatement();
        String qry = "SELECT * FROM tblSample";
        ResultSet rs = s.executeQuery(qry);
        while (rs.next()) {
            str += rs.getInt("id") + "  "
              + rs.getString("title") + " "
              + rs.getString("author")+ " "
              + rs.getDouble("price") + " "
              + rs.getBoolean("hardcover") + "\n";
        }
        rs.close();
        s.close();
        return str;
    }catch(Exception e){
        e.printStackTrace();
        return "An error occurred";
    }
}
```

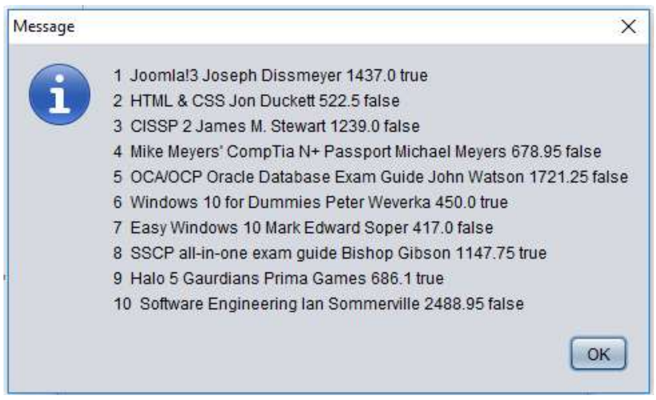
Points to note:

- A *Statement* object is created. If you wish to make an analogy, it is like a teacher giving a student permission to ask a question
- When the query is executed (much like the student then asking the question and getting an answer) the result produced is put into a *ResultSet* object. This is a temporary table that contains only the data that we want. If you run a MS-Access query, you will see the results are also displayed in a table, a *ResultSet* can be viewed as much the same, but useable to our Java program.
- We need to work through the ResultSet line by line (using `rs.next()` which moves the file pointer down one record each time), capturing the parts of each line/record we need (using `get`*`Type`*`(`*`fieldName`*`)` methods denoting the datatype we wish to obtain and the field from which we wish to capture the data) and placing these in some form that is directly accessible by our Java program. In this example we are building a String from the pieces, but they could also be assigned to variables, saved in arrays, used to create objects or manipulate object properties.
- Close the ResultSet and then the Statement.

Create a click event method for your “Show books” button in your GUI class (i.e. by double clicking the button). Add the following line of code to the method:

```
JOptionPane.showMessageDialog(null, db.displayRecords());
```

Run your project. If you click your “Show books” button, you should see the contents of your database in a dialogue box:



```
1  Joomla!3 Joseph Dissmeyer 1437.0 true
2  HTML & CSS Jon Duckett 522.5 false
3  CISSP 2 James M. Stewart 1239.0 false
4  Mike Meyers' CompTia N+ Passport Michael Meyers 678.95 false
5  OCA/OCP Oracle Database Exam Guide John Watson 1721.25 false
6  Windows 10 for Dummies Peter Weverka 450.0 true
7  Easy Windows 10 Mark Edward Soper 417.0 false
8  SSCP all-in-one exam guide Bishop Gibson 1147.75 true
9  Halo 5 Gaurdians Prima Games 686.1 true
10  Software Engineering Ian Sommerville 2488.95 false
```

All SELECT queries can be executed using this basic pattern.

To edit records in our database using UPDATE, INSERT or DELETE the pattern is slightly different. Since the database doesn't return values there is no need for a ResultSet object and the executeQuery() method is replaced with the executeUpdate() method. Take note that the executeUpdate() method returns an int value and that this value can sometimes be used to indicate how many records were changed in the table. We will, in this example, ignore this possibly useful function of the method to keep things simple.

To work through an example of editing a database table, we will obtain values from the user and use these to add a record to our database. The editing and deleting of records will work in much the same way, using the correct SQL statement.

Create a click event method for your "Add books" button in your GUI class. Obtain suitable values from the user for the title, author, price and hardcover status of a book (but not the id since this uses an autonumber datatype in our database) saving these in suitable variables.

Add the following method to your *DatabaseConnection* class:

```
public void addRecord(String t, String a, double p, boolean h){
    int num;
    try{
        Statement s = conn.createStatement();
        String qry = "INSERT INTO tblBooks (title, author, price, hardcover)"
          + " VALUES ('" + t + "', '" + a + "', " + p + ", " + h + ");";
        num = s.executeUpdate(qry);
        s.close();
    }
    catch(Exception e){
        e.printStackTrace();
        System.out.println("An error occurred");
    }
}
```

Points to note:

- Be careful to construct your SQL statement correctly with the parameters received. It is often useful to add a println() of the *qry* value before we attempt to run the executeUpdate() method so that we can check our SQL statement.
- The other SQL editing commands will work in the same way.
- The value in *num* is not being used here, but it could be returned to the calling statement with a message to say how many records were edited in our table (arguably most useful for UPDATE statements).

Call the above method from your “Add book” method. My complete method looks as follows:

```java
private void btnAddBookActionPerformed(java.awt.event.ActionEvent evt) {
    String t = JOptionPane.showInputDialog("Title?");
    String a = JOptionPane.showInputDialog("Author?");
    double p = Double.parseDouble(JOptionPane.showInputDialog("Price?"));
    char hc = JOptionPane.showInputDialog("Hardcover Y/N").charAt(0);
    boolean h;
    if(hc == 'Y' || hc == 'y')
        h = true;
    else
        h = false;
    db.addRecord(t, a, p, h);
}
```

Final step: create your own program
Using the above as a guide, you should now be able to build a much more extensive program that accesses and manipulates a database using SQL from a Java program.

Happy coding!

Compiled by:
G. Kenning
February 2016
Comments & feedback welcome, send to:
teacher.kenning@gmail.com