Java and Microsoft Access SQL Tutorial

Last Update : 30 April 2008
Revision : 1.03

Introduction:

This is a short tutorial on how to use Java and Microsoft Access to store and retrieve data in an SQL database. A mixture of source code and explanatory note's are used to introduce the student to the usage of SQL from Java to access and manipulate information. We hope to assist students in obtaining better marks for their projects and assignments as SQL is an often-neglected part of the Java curriculum. This tutorial is free and you are encouraged to share it with others as long as you retain it in it's original format. You can contact the author at **info@javak.co.za** if you have any corrections or comments on this tutorial.

All the source code for this tutorial is available at no charge. Simply send an email to **info@javak.co.za** to request a free zipped copy of all the Java source code examples in this Java SQL tutorial. We recommend students enter the code manually as it is often the most effective way to learn new code and have included line numbers to make this easier.

It is assumed that you have some knowledge of Microsoft Access, especially on how to create tables and enter information. Also, I am going to assume that you have some Java knowledge!

Content:

Creating the required Microsoft Access database:

The first thing we need to do is to create the Microsoft Access database that we will be using during this tutorial. We will begin by creating a blank database in Microsoft Access. Let's save the database to the **C:\Projects** folder and call it **tutorial**. So now you have a blank database in the **C:\Projects** folder.

We can now create a simple table in Design View. Insert the following fields into your table and remember to set the primary key as indicated!

Field Name	Data Type	Description	Key Type
RECIPEID	Autonumber	Our primary key	Primary
RECIPENAME	Text (100 characters)	Recipe name	
INGREDIENTS	Memo	Our recipe ingredients, separated by a ,	
INSTRUCTIONS	Memo	Recipe instructions, separated by a ,	

Save it as RECIPESTABLE and then enter the following information into the table :

1	Green Beans	Salt, Green Beans - Sliced, 1 Litre of water	Chop the beans, Boil the beans in the water until soft, Add salt, Serve
2	Hamburger	Pocket Money, Lift	Grab pocket money, Get lift from friend to nearest burger place, Buy burger, Enjoy

When all the information is entered as shown, you can close Microsoft Access and fire up your favourite Java editor. For those impatient individuals : The entire example program source code is at the end of the document.

Connecting to the Microsoft Access database:

Now we need to connect to the Microsoft Access database. Our first program will simply connect to the database and then disconnect.

Consider the following code segment :

ConnectDB.java

```
 1
 2 /**
 3  * Opens a Microsoft Access Database without having need to
 4  * have access rights to the Administrative Tools on Windows
 5  * to set up any ODBC connections.
 6  */
 7 import java.sql.*;
 8 import java.io.*;
 9
10 /**
11  * @author Ewald Horn
12  * @company JavaK
13  */
14 public class ConnectDB
15 {
16
17     public static Connection con;
18     public static final String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
19     public static final String url = "jdbc:odbc:" +
20             "DRIVER={Microsoft Access Driver (*.mdb)};DBQ=";
21     String path; // where the database can be found
22
23     ////////////////////////////////////////////////////////
24     /**
25      * Sets the path to the database.
26      */
27     public ConnectDB()
28     {
29         path = "c:" + File.separator + "projects" +
30                 File.separator + "tutorial.mdb";
31     }
32
33
34     ////////////////////////////////////////////////////////
35     /**
36      * Runner method for the ConnectDB class.
37      */
38     public void go()
39     {
40         makeConnection();
41         closeConnection();
42     }
43
44
45     ////////////////////////////////////////////////////////
46     /**
47      * Creates the database connection.
48      */
49     private void makeConnection()
50     {
51         System.out.println("Opening database...\n");
52         try
```

```
53          {
54              Class.forName(driver);
55              con = DriverManager.getConnection(url + path);
56          } catch (Exception ex)
57          {
58              System.out.println("Error opening the database!");
59              System.out.println(ex);
60              System.exit(0);
61          }
62          System.out.println("Success!");
63      }
64
65
66      ////////////////////////////////////////////////////////
67      /**
68       * Closes the connection cleanly.
69       */
70      private void closeConnection()
71      {
72          System.out.println("\nClosing database.");
73          try
74          {
75              con.close();
76          } catch (Exception ex)
77          {
78              System.out.println("Error closing the database!");
79              System.out.println(ex);
80          }
81      }
82
83
84      ////////////////////////////////////////////////////////
85      /**
86       * Main method for ConnectDB.java
87       */
88      public static void main(String args[])
89      {
90          ConnectDB testApp = new ConnectDB();
91          testApp.go();
92      }
93  }
```

The following table explains some code segments in ConnectDB.java

Lines	Explanation of code
7	Imports the java.sql.* libraries that are required for most database methods.
8	Import java.io.* which is used to get the File.separator value to set up the path to the database.
17 – 21	Declare the various variables required for this program. Of special interest is the **url** variable that is used to identify the driver as being intended for use on Microsoft Access MDB files. For the more adventurous readers : A similar driver exists for Microsoft Excel XLS documents and can be used to interrogate spreadsheet documents for information.
29 – 30	Set up the path to the database. You can modify this to point to your database, but remember to use File.separator instead of slashes (/ or \ as it is more reliable and platform independent .
38 – 42	A runner method used to call all the required methods to run the example program. It is used to keep the MAIN method as short as possible. The runner is included in the main class to shorten the code listing, but you can place the runner (main) method in a seperate class if you are more comfortable with that approach. Ask your teacher if you are unsure about which method to use.

49 – 63	A connection is established with the database using the variables previously declared. Notice that all operations are contained in a **try-catch** block to prevent the program from crashing if something goes wrong with connecting to the database. Lines 54 and 55 contain the actual code that connects to the database. Very easy!
70 – 81	Closes the connection made to the database. Again, a **try-catch** block is used to prevent exceptions from crashing the program. The statement on line 75 closes the database connection. It is a good practice to always close your database connections before closing your program. This will help prevent data corruption due to open database handles. Most databases close their connections automatically, but you should never rely on the computer to think for you.
88 – 92	The main method creates a new object of the type ConnectDB and calls the go() method. It is the entry point into the application.

When you run the program you should see the following output :

```
Opening database...

Success!

Closing database.
```

Certainly not *spectacular*, but it did create a connection to the database and closed it again. So what if something goes wrong ?

Problem	Possible cause
Error when opening the database.	If you get an error while trying to open the database, check the following things : Ensure Microsoft Access is closed, otherwise the database might appear to be locked and your program might not be able to access it. Ensure that you have set up the database path correctly. Maybe you left out a File.separator or did not include the entire path. Make sure that you typed the *driver* and *url* variable's contents correctly. Even a little mistake will cause your program to fail.

Now that we can connect to the database we should be able to display the information contained therein. This brings us to our next section...

Displaying table information to the screen:

Lets add a method to our program that will display some of the information contained in the database. This new method, showRecipes(), will display the names of all the recipes contained in the database. We are going to modify ConnectDB.java to include our new method to save ourselves some typing. I suggest that you insert the showRecipes() method between the makeConnection() and closeConnection() methods.

We are going to create a method called showRecipes() that will display all the records contained in the RECIPESTABLE on the screen. The method uses an SQL statement to SELECT all the records and then parses the resulting ResultSet to obtain the required information. The SQL string is declared in a separate variable and this makes it easier to reuse the code block. As always, the operation is enclosed in a **try-catch** block to prevent exceptions from crashing our program.

After all information has been obtained from the database and displayed on the screen, the statement and the result set variables are both closed. This is good programming practice as it frees system resources and indicates that we are done with the specified database table. In a larger program this will prevent database access deadlocks and other data access violations.

Insert the following method into your program:

```
1     ////////////////////////////////////////////////////////
2     /**
3      * Displays all the recipes in the database.
4      */
5     private void showRecipes ()
6     {
7         String sql = "SELECT * FROM RECIPESTABLE";
8
9         System.out.println ("\nRecipes in the database : \n");
10
11        try
12        {
13
14            Statement statement = con.createStatement ();
15            ResultSet rs = statement.executeQuery (sql);
16
17            if (rs != null)
18            {
19                while (rs.next ())
20                {
21                    String recipeName = rs.getString ("RECIPENAME");
22                    System.out.println (recipeName);
23                }
24            }
25
26            rs.close ();
27            statement.close ();
28
29        }
30        catch (Exception ex)
31        {
32            System.out.println ("Error reading database information!");
33            System.out.println (ex);
34        }
35    }
```

Now modify the go() method to call the showRecipes() method before it closes the database connection by adding the line in **bold** and yellow to the method.

```
public void go ()
{
   makeConnection ();
   showRecipes();
   closeConnection ();
}
```

The expected output from running ConnectDB.java with the modifications :

```
Opening database...

Success!

Recipes in the database :

Green Beans
Hamburger

Closing database.
```

Adding a new record to a table:

Now that we can display the records in the database it is time to add a new record. Once again we will modify the ConnectDB.java program and add another method called addRecipe(). This method will be called *before* the call to the showRecipes() method. This makes it easier to test that our program actually works.

Here is the code for our simple addRecipe() method:

```
1     ////////////////////////////////////////////////////////
2     /**
3      * Adds a recipe to the database.
4      */
5     private void addRecipe ()
6     {
7         String sql = "INSERT INTO RECIPESTABLE(RECIPENAME,INGREDIENTS)" +
8                      " VALUES('Any Recipe','Ingredients')";
9
10        System.out.print ("\nAdding a recipe : ");
11
12        try
13        {
14            Statement statement = con.createStatement ();
15            int result = statement.executeUpdate (sql);
16            System.out.println (" Added " + result + " recipe(s).");
17        }
18        catch (Exception ex)
19        {
20            System.out.println ("Error adding a recipe!");
21            System.out.println (ex);
22        }
23    }
```

Now modify the go() method to call the addRecipe() method before it closes the database connection by adding the line in **bold** and yellow to the method.

```
public void go ()
{
   makeConnection ();
   addRecipe();
   showRecipes();
   closeConnection ();
}
```

After running the program you should see the following output:

```
Opening database...

Success!

Adding a recipe :  Added 1 recipe(s).

Recipes in the database :

Green Beans
Hamburger
Any Recipe

Closing database.
```

Lines 7 and 8 deserve special attention :

An INSERT statement is used to add a new record to the database. When compiling the SQL string we specify that we want to INSERT a new record into RECIPESTABLE and that we wish to provide content for both the RECIPENAME and the INGREDIENTS values. The database will automatically update the RECIPEID value since it was declared as an autonumber. The other fields will simply receive the default values as specified in the database.

Adding a record is pretty easy! Question : What happens when you run the program a few times in a row ? Now you have a lot of “Any Recipe” entries in the database. Not very smart is it ? You could open Access and delete some of the extra rows, but that is not really the point of this exercise, right ?

If we are going to delete records from the database we will need some way to uniquely identify each record to prevent us from deleting the wrong record by accident. Remember we gave each recipe a unique ID that the database automatically assigns ? We can make use of that unique number to identify each recipe as the number is guaranteed to be unique for each record. Let's look at how we can modify our program to also display the unique ID (key) of each record contained in the database.

We can modify the showRecipes() methods to also display the ID number of each recipe :

```
 1     ////////////////////////////////////////////////////////
 2     /**
 3      * Displays all the recipes in the database.
 4      */
 5     private void showRecipes ()
 6     {
 7         String sql = "SELECT * FROM RECIPESTABLE";
 8
 9         System.out.println ("\nRecipes in the database : \n");
10
11         try
12         {
13
14             Statement statement = con.createStatement ();
15             ResultSet rs = statement.executeQuery (sql);
16
17             if (rs != null)
18             {
19                 while (rs.next ())
20                 {
21                     int recipeID = rs.getInt("RECIPEID");
22                     String recipeName = rs.getString ("RECIPENAME");
23                     System.out.println (recipeID + " " + recipeName);
24                 }
25             }
26
27             rs.close ();
28             statement.close ();
29
30         }
31         catch (Exception ex)
32         {
33             System.out.println ("Error reading database information!");
34             System.out.println (ex);
35         }
36     }
```

With these modifications we are now able to see each recipe's name AND unique ID number. When we run the program again, we see all the records and their unique ID numbers.

After the modifications our new screen output will be something like this :

```
Opening database...

Success!

Adding a recipe :  Added 1 recipe(s).

Recipes in the database :

1 Green Beans
2 Hamburger
8 Any Recipe
9 Any Recipe
10 Any Recipe
11 Any Recipe
12 Any Recipe
13 Any Recipe
14 Any Recipe
15 Any Recipe
16 Any Recipe
17 Any Recipe

Closing database.
```

Notice that I have removed some records previously to demonstrate that the database keeps track of the RECIPEID values automatically and that it is a unique identifier for each recipe. Now we are ready to remove records from the database.

Removing a record from a table:

Removing a record from a database is a very straightforward exercise and is virtually the same procedure as for adding a record. For the purpose of the exercise we will be removing recipe number 11 from the database. The first time your program runs it will delete the recipe (if it exists) and report that one record was deleted. Subsequent runs will not delete any other records and the program will report that no records were deleted.

Let's add a method called removeRecipe() to our program:

```
 1     ////////////////////////////////////////////////////////
 2     /**
 3      * Removes a recipe from the database.
 4      */
 5     private void removeRecipe ()
 6     {
 7         String sql = "DELETE FROM RECIPESTABLE WHERE RECIPEID=11";
 8         System.out.print ("\nRemoving a recipe : ");
 9
10         try
11         {
12             Statement statement = con.createStatement ();
13             int result = statement.executeUpdate (sql);
14             System.out.println (" Removed " + result + " recipe(s).");
15         }
16         catch (Exception ex)
17         {
18             System.out.println ("Error removing a recipe!");
19             System.out.println (ex);
20         }
21     }
```

Modify the go() method to call the removeRecipe() method:

```
1     ////////////////////////////////////////////////////////
2     /**
3      * Runner method for the TestAccess class.
4      */
5     public void go ()
6     {
7         makeConnection ();
8         addRecipe ();
9         removeRecipe ();
10        showRecipes ();
11        closeConnection ();
12    }
```

The output will look something like this on the first run:

```
Opening database...

Success!

Adding a recipe :  Added 1 recipe(s).

Removing a recipe :  Removed 1 recipe(s).

Recipes in the database :

1 Green Beans
2 Hamburger
8 Any Recipe
9 Any Recipe
10 Any Recipe
12 Any Recipe
27 Any Recipe

Closing database.
```

And on subsequent runs :

```
Opening database...

Success!

Adding a recipe :  Added 1 recipe(s).

Removing a recipe :  Removed 0 recipe(s).

Recipes in the database :

1 Green Beans
2 Hamburger
8 Any Recipe
9 Any Recipe
10 Any Recipe
12 Any Recipe
27 Any Recipe
28 Any Recipe

Closing database.
```

Updating (altering) a record in a table:

Let's alter a record in the database. Now that we can easily identify a recipe we should be able to modify any recipe quite easily. Let's write a small method called updateRecipe() that will the name of recipe number 8. We will be using the SQL command UPDATE to modify the existing record in the database.

Insert the following method into your program:

```
1     ////////////////////////////////////////////////////////
2     /**
3      * Modifies an existing record.
4      */
5     private void updateRecipe ()
6     {
7         String sql = "UPDATE RECIPESTABLE SET " +
8                      "RECIPENAME='Pizza' WHERE RECIPEID=8";
9         System.out.print ("Updating a record : ");
10        try
11        {
12            Statement statement = con.createStatement ();
13            int result = statement.executeUpdate (sql);
14            System.out.println (" Updated " + result + " recipe(s).");
15        }
16        catch (Exception ex)
17        {
18            System.out.println ("Error removing a recipe!");
19            System.out.println (ex);
20        }
21    }
```

We can now make our final modifications to the go() method to call the updateRecipe() method.

Modify the go() method to call the updateRecipe():

```
1     ////////////////////////////////////////////////////////
2     /**
3      * Runner method for the TestAccess class.
4      */
5     public void go ()
6     {
7         makeConnection ();
8         addRecipe ();
9         removeRecipe ();
10        updateRecipe ();
11        showRecipes ();
12        closeConnection ();
13    }
```

The output from the program will look something like this:

```
Opening database...

Success!

Adding a recipe :  Added 1 recipe(s).

Removing a recipe :  Removed 0 recipe(s).
Updating a record :  Updated 1 recipe(s).

Recipes in the database :

1 Green Beans
2 Hamburger
8 Pizza
9 Any Recipe
10 Any Recipe
12 Any Recipe
27 Any Recipe
28 Any Recipe
29 Any Recipe
30 Any Recipe

Closing database.
```

COMPLETE EXAMPLE PROGRAM SOURCE CODE:

```
 1 /**
 2  * Opens a Microsoft Access Database without having need to
 3  * have access rights to the Administrative Tools on Windows
 4  * to set up any ODBC connections.
 5  */
 6
 7 import java.sql.*;
 8 import java.io.*;
 9
10 /**
11  * @author Ewald Horn
12  * @company JavaK
13  */
14 public class ConnectDB
15 {
16
17     public static Connection con;
18     public static final String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
19     public static final String url = "jdbc:odbc:" +
20         "DRIVER={Microsoft Access Driver (*.mdb)};DBQ=";
21
22     String path; // where the database can be found
23
24     ////////////////////////////////////////////////////////
25     /**
26      * Sets the path to the database.
27      */
28     public ConnectDB ()
29     {
30         path = "c:" + File.separator + "projects" +
31             File.separator + "tutorial.mdb";
32     }
33
```

```
34
35      ////////////////////////////////////////////////////////////
36      /**
37       * Runner method for the TestAccess class.
38       */
39      public void go ()
40      {
41          makeConnection ();
42          addRecipe ();
43          removeRecipe ();
44          updateRecipe ();
45          showRecipes ();
46          closeConnection ();
47      }
48
49
50      ////////////////////////////////////////////////////////////
51      /**
52       * Creates the database connection.
53       */
54      private void makeConnection ()
55      {
56          System.out.println ("Opening database...\n");
57          try
58          {
59              Class.forName (driver);
60              con = DriverManager.getConnection (url + path);
61          }
62          catch (Exception ex)
63          {
64              System.out.println ("Error opening the database!");
65              System.out.println (ex);
66              System.exit (0);
67          }
68          System.out.println ("Success!");
69      }
70
71
72      ////////////////////////////////////////////////////////////
73      /**
74       * Removes a recipe from the database.
75       */
76      private void removeRecipe ()
77      {
78          String sql = "DELETE FROM RECIPESTABLE WHERE RECIPEID=11";
79          System.out.print ("\nRemoving a recipe : ");
80
81          try
82          {
83              Statement statement = con.createStatement ();
84              int result = statement.executeUpdate (sql);
85              System.out.println (" Removed " + result + " recipe(s).");
86          }
87          catch (Exception ex)
88          {
89              System.out.println ("Error removing a recipe!");
90              System.out.println (ex);
91          }
92      }
93
94
```

```
 95     ////////////////////////////////////////////////////////
 96     /**
 97      * Modifies an existing record.
 98      */
 99     private void updateRecipe ()
100     {
101         String sql = "UPDATE RECIPESTABLE SET " +
102             "RECIPENAME='Pizza' WHERE RECIPEID=8";
103         System.out.print ("Updating a record : ");
104         try
105         {
106             Statement statement = con.createStatement ();
107             int result = statement.executeUpdate (sql);
108             System.out.println (" Updated " + result + " recipe(s).");
109         }
110         catch (Exception ex)
111         {
112             System.out.println ("Error removing a recipe!");
113             System.out.println (ex);
114         }
115     }
116
117
118     ////////////////////////////////////////////////////////
119     /**
120      * Adds a recipe to the database.
121      */
122     private void addRecipe ()
123     {
124         String sql = "INSERT INTO RECIPESTABLE(RECIPENAME,INGREDIENTS)" +
125             " VALUES('Any Recipe','Ingredients')";
126
127         System.out.print ("\nAdding a recipe : ");
128
129         try
130         {
131             Statement statement = con.createStatement ();
132             int result = statement.executeUpdate (sql);
133             System.out.println (" Added " + result + " recipe(s).");
134         }
135         catch (Exception ex)
136         {
137             System.out.println ("Error adding a recipe!");
138             System.out.println (ex);
139         }
140     }
141
142
143     ////////////////////////////////////////////////////////
144     /**
145      * Displays all the recipes in the database.
146      */
147     private void showRecipes ()
148     {
149         String sql = "SELECT * FROM RECIPESTABLE";
150
151         System.out.println ("\nRecipes in the database : \n");
152
153         try
154         {
155
```

```
156             Statement statement = con.createStatement ();
157             ResultSet rs = statement.executeQuery (sql);
158 
159             if (rs != null)
160             {
161                 while (rs.next ())
162                 {
163                     int recipeID = rs.getInt ("RECIPEID");
164                     String recipeName = rs.getString ("RECIPENAME");
165                     System.out.println (recipeID + " " + recipeName);
166                 }
167             }
168 
169             rs.close ();
170             statement.close ();
171 
172         }
173         catch (Exception ex)
174         {
175             System.out.println ("Error reading database information!");
176             System.out.println (ex);
177         }
178     }
179 
180 
181     ////////////////////////////////////////////////////////
182     /**
183      * Closes the connection cleanly.
184      */
185     private void closeConnection ()
186     {
187         System.out.println ("\nClosing database.");
188         try
189         {
190             con.close ();
191         }
192         catch (Exception ex)
193         {
194             System.out.println ("Error closing the database!");
195             System.out.println (ex);
196         }
197     }
198 
199 
200     ////////////////////////////////////////////////////////
201     /**
202      * Main method for ConnectDB.java
203      */
204     public static void main (String args[])
205     {
206         ConnectDB testApp = new ConnectDB ();
207         testApp.go ();
208     }
209 }
```

About this tutorial:

This tutorial was prepared by Ewald Horn of NoFuss Solutions. Ewald is the web master of JavaK and has been active in the programming and training field for more than 10 years. You can visit the **http://www.javak.co.za** website for more free resources on Java programming. JavaK is aimed at Afrikaans-speaking Java students and provides a host of free resources to assist them with programming assignments and homework.