INSTITUTIONEN FÖR DATA-
OCH SYSTEMVETENSKAP
SU / KTH

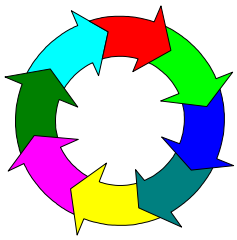
UPDATING A DB WITH SERVLETS

IS7

KOMPONENTBASERAD SYSTEMANALYS

VÅRTERMINEN 1999

http://L238.dsv.su.se/courses/is7/laboration.html



nikos dimitrakas

1 Introduction

This document is a complement to the Servlet Laboration Compendium (*nikos dimitrakas*, 1999). The techniques presented here are not supported by the wizards of Websphere Studio 1.0. In future versions of Websphere Studio, this functionality is going to be available.

2 Updating a database with servlets

This section concentrates on inserting and deleting records from the same sample database used in the database example in the Servlet Laboration Compendium. It is important that the database is already registered at the ODBC Administrator and an SQL is created in Websphere Studio (as described in the Servlet Laboration Compendium).

2.1 Inserting a record

The basic component for doing that is a Java Bean that takes care of establishing a connection to the database and also putting together an SQL statement and executing it. The following code is a very basic Java Bean that does exactly that:

```
package UD;

import java.sql.*;
import java.util.*;
import java.text.*;

public class DBBean {

  static protected Connection con;
  static protected Statement stmt;

// DB access variables

  private String URL = "";
  private String userID = "";
  private String driver = "";
  private String password = "";

// variables for the "shop" DB table

  private String shopname = "";
  private String reallocation = "";
  private String netadress = "";


  // DB get and set methods
```

This section is just declaration of variables and setters and getters

```
 public void setUserID(String s) { userID = s; }
 public String getUserID() { return userID; }

 public void setPassword(String s) { password = s; }
 public String getPassword() { return password; }

 public void setDriver(String s) { driver = s; }
 public String getDriver() { return driver; }

 public void setURL(String s) { URL = s; }
 public String getURL() { return URL; }

// get and set methods for "shop" DB table variables

 public void setShopname (String s) { shopname = s; }
 public String getShopname() { return shopname; }

 public void setReallocation(String s) { reallocation = s; }
 public String getReallocation() { return reallocation; }

 public void setNetadress(String s) { netadress = s; }
 public String getNetadress() { return netadress; }
```

```
// establish DB connection
 public void dbConnect()
 {
   try
   {
     // register the driver with DriverManager
     Class.forName(getDriver());
     con = DriverManager.getConnection(getURL(),
getUserID(), getPassword());
     con.setAutoCommit(true);
   }
   catch (Exception e)
   {
     e.printStackTrace();
   }
 }
```

This part of the code establishes a connection (or throws an exception).

First we ask the DriverManager for a connection (the URL, UserID and Password must already be defined)

Then we set the AutoCommit of the connection to true.

```
public void savenewshop() throws Exception
 {
    String query;
    query = "insert into shop (shopname,
reallocation, netadress) " + "values
('"+getShopname()+"', '"+getReallocation()+"',
'"+getNetadress()+"') ";
```

This is the method that actually does the insertion of the record to the database.

First we construct an SQL statement (as a string) that inserts the values that we have stored in the shopname, reallocation and netadress variables to the shop table.

stmt = con.createStatement();
 	
stmt.executeUpdate (query);
 	
stmt.close();
con.commit();
}
	Then we associate the Statement variable (stmt) to the connection.
	Now we can use the string that includes the SQL and execute it through the statement.
	
	Finally we close the statement and commit the changes to the connection.

2.2 Deleting a record

To delete a record we need to add one more method to our Java Bean:

public void removeshop() throws Exception
{
String query;
query = "delete from shop where shopname = '"+getShopname()+"'";
 	
stmt = con.createStatement();
//stmt.setString(1, getShopname());
//stmt.setString(2, getReallocation());
//stmt.setString(3, getNetadress());
 	
stmt.executeUpdate (query);
 	
stmt.close();
con.commit();
}
	This is the method that actually does the deletion of one or more records from the database.
	
	First we construct an SQL statement (as a string) that deletes the records that match the condition (in this case shopname equals the value of the variable shopname)
	
	Then we associate the Statement variable (stmt) to the connection.
	Now we can use the string that includes the SQL and execute it through the statement.
	
	Finally we close the statement and commit the changes to the connection.

2.3 Using the Update Java Bean from a Servlet

As we saw before the Java Bean that contains the logic for updating the database does not have any servlet functionality. We need therefore a servlet to invoke the Java Bean and instanciate the variables (shopname, reallocation and netadress, as well as the connection variables) (for example with values fetched from an input html). Here is an example of the code that can exist in a servlet:

public void performTask(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) { try { UD.DBBean AddShopServlet = null; AddShopServlet = (UD.DBBean) java.beans.Beans.instantiate(getClass().getClassLoader(),	First we have to create a new instance of the Java Bean.

"UD.DBBean");	

// Code for getting init-parameters for DB access
 	
// Initialize the bean id property from the parameters
AddShopServlet.setUserID(getParameter(request, "userID", true, true, true, null));
 	
// Initialize the bean password property from the parameters
AddShopServlet.setPassword(getParameter(request, "password", true, true, true, null));
 	
// Initialize the bean driver property from the parameters
AddShopServlet.setDriver(getParameter(request, "driver", true, true, true, null));
 	
// Initialize the bean URL property from the parameters
AddShopServlet.setURL(getParameter(request, "URL", true, true, true, null));
 	
// end code for DB access parameters
 	
// Initialize the bean city property from the parameters
AddShopServlet.setShopname(getParameter(request, "shopname", true, true, true, null));
 	
// Initialize the bean email property from the parameters
AddShopServlet.setReallocation(getParameter(request, "reallocation", true, true, true, null));
 	
// Initialize the bean firstname property from the parameters
AddShopServlet.setNetadress(getParameter(request, "netadress", true, true, true, null));
	Here we instanciate all the variables of the Java Bean with values either from the request (shopname, reallocation and netadress) or from the servlet configuration file (database connection variables).

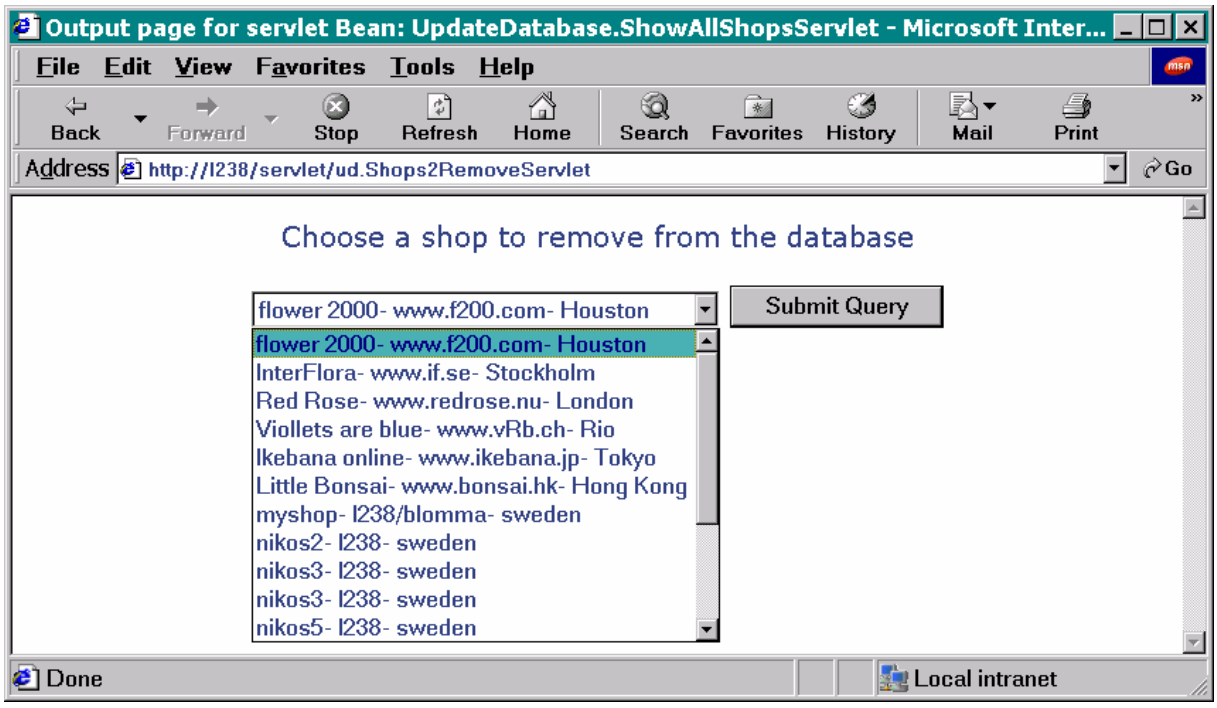
// call the dbConnect action on the bean
AddShopServlet.dbConnect();
 	
// call the savenewshop action on the bean
AddShopServlet.savenewshop();
 	
// Call the output page. If the output page is not passed
// as part of the URL, the default page is called.
callPage(getPageNameFromRequest(request), request, response);
}
catch (Exception theException)
{
handleError(request, response, theException);
}
}
	The only thing left to do now is to call the methods of the Java Bean that establish the database connection and construct and execute the SQL statement.
	
	Finally we call the output page!

Of course there must be an html input page that lets the user input values for the variables that are to be stored in the new record.

Removing a record can be a little trickier. It can be desirable to show to the user all the available records and let the user choose one of these records. Here is a simple solution how to let the user choose:
We can create an output page that shows the contents of the database table (exactly as in the example in the Compendium) and then modify the jsp file to be the input page for the record deletion servlet. Here is a simple modification of the jsp file:

```
<FORM ACTION="/servlet/UD.RemoveShopServlet" METHOD="get" ENCODE="application/x-www-form-urlencoded">
  <SELECT NAME="shopname">
    <REPEAT INDEX="i">
    <%shopsServletBean.getShop_shopname(i);%>
        <OPTION VALUE="<INSERT BEAN="shopsServletBean" PROPERTY="shop_shopname"></INSERT>"></INSERT>
        <INSERT BEAN="shopsServletBean" PROPERTY="shop_shopname"></INSERT>-
        <INSERT BEAN="shopsServletBean" PROPERTY="shop_netadress"></INSERT>-
        <INSERT BEAN="shopsServletBean" PROPERTY="shop_reallocation"></INSERT>
    </REPEAT>
  </SELECT>
  <INPUT TYPE="submit">
</FORM>
```

The result of this code will be something like this:



When the user chooses one of the alternatives and presses the Submit button then the shopname is passed to the servlet as a parameter in the request. Then the servlet initiates the shopename variable of the Java Bean and then it calls the removeshop method:

```
// Initialize the bean firstname property from the parameters
                RemoveShopServlet.setShopname(getParameter(request, "shopname", true, true, true, null));

                // call the dbConnect action on the bean
                RemoveShopServlet.dbConnect();
```

```
                // call the savenewshop action on the bean
                RemoveShopServlet.removeshop();
```

3 Epilogue

The techniques presented here are basically adjusted for databases using an ODBC driver. When using a native driver it is possible to use more features provided in java.sql libraries. A good source of information about these techniques can be the Websphere examples and their documentation.