Web Development in a Server-Centric Environment Using Java Server Pages (JSP)

John D. Haney
Craig A. VanLengen
College of Business Administration, Northern Arizona University
Flagstaff, AZ 86011

Abstract

The development of interactive Web page applications, where data is extracted from a database or a database is updated, can be a tedious process. Several options are available. The preference for this study is Java Server Pages (JSP). The fundamental processes for interactive Web page development are - querying from a database, adding a row to a database table, changing the fields within a row, or deleting a row from a database table. These processes form the foundation for any type of database interaction. The scripting language used for Java Server Pages is Java and the interaction with the database is done with Structured Query Language (SQL) against an Oracle database. In each of these processes a connection must be made to the database. A Java Database Connectivity (JDBC) connection to Open Database Connectivity (ODBC) connection is used since the Jakarta Java Web Server is used which runs on Internet Information Server (IIS) and NT. In the case of the query, where information is extracted from the database and placed into a table on the web page, only one Web page is required. For adding a record, changing a record, or deleting a record two Web pages are required. The first page contains a form with objects, which are posted to the second JSP page. The second JSP page then, by using SQL, inserts a record, modifies a record or deletes a record.

Keywords: Java Server Pages, Java, database connectivity, structured query language, database interaction

1. INTRODUCTION

JavaServer Pages™ (JSP) and Microsoft Active Server Pages (ASP) were both designed to allow the creation of server based Web applications that interacts with a database. JSP is based on Sun Microsystems, Inc. Java 2 Platform, Enterprise Edition (J2EE) specification. J2EE includes the definitions of JSP, JavaBeans, Enterprise JavaBeans components and Java servlets. ASP is platform and server dependent where JSP was designed to be platform and server independent. This paper presents examples of extracting information from a database table and displaying the data in a Web page table using Java Server Pages (JSP), and maintaining a database with add, change, and delete options developed in JSP. An Oracle database is used for these examples.

JSP pages can □e run on any Web server that has a servlet container with a JSP environment. Microsoft IIS server and a servlet container with a JSP environment were used for the examples. The JSP specification allows us to combine static HTML and Java servlet programming to generate dynamic content. JSP pages get compiled into Java servlets that generate the user page. The resulting Web page combines the HTML from the JSP page with the results of the dynamic content specified within the JSP tags. This way the JSP page provides dynamic content for each specific HTTP request. The JSP container inserts the results of JavaBeans for business logic processing and database interaction into the static HTML and returns the request to the client.

JSP follows Sun's "Write Once, Run Anywhere(tm)" policy of Java. Also another advantage of JSP is that the first time a page is requested it is compiled into a servlet and executed. The next request for the same page will execute the compiled version as long as no changes were made to the original JSP page.

JSP also uses JavaBean components. Developing most of the business logic and database access as JavaBean components more cleanly separates the presentation from the business and data layers. Developing with JavaBeans promotes code reusability. Several JSP applications can access the needed functionality by calling the same JavaBean classes. Since the JSP specification is part of Sun's Enterprise Edition (J2EE) specification it also works with Enterprise JavaBeans (EJB) for development of enterprise solutions.

Dynamic Content
End users want content tailored to their specific situation. On the other hand providers of Web content want a solution that is easy to create and modify and also powerful and flexible. Dynamic content is defined as content provided on Web pages that are created when requested. The content is created for each request based on criteria specified in the request received from the client.

JSP is a presentation layer technology that allows static Web content to be mixed with Java code. JSP allows the use of standard HTML, but adds the power and flexibility of the Java programming language. JSP does not modify static data, so page layout and "look-and-feel" can continue to be designed with current methods. This allows for a clear separation between the page design and the application. JSP also enables Web applications to be broken down into separate components.

Tiered Development Approach
Organizations and developers want to move from a client-centric to a server-centric model. In a client-centric model changes to an application require the application to be redeployed to all client workstations. With a server-centric model, with applications running from a Web server, redeployment to the client is not required. Processing of transactions and preparing of user responses also takes place on the server.

The appropriate way for tiered development is to separate the static presentation (look-and-feel) from the dynamic content generation. By separating the static from the dynamic we make it easier to make changes in either the presentation or the generation of the dynamic content. Developers can concentrate on writing Java code within tags and building JavaBean components for the business and database logic while the designer/artists create the "look-and-feel" of the Web pages. By separating the presentation and the programming logic we are creating a system that allows for future growth and is easier to maintain.

JSP Compared to CGI Implementation
The Common Gateway Interface (CGI) is used to allow users of the World Wide Web access to data and processes that do not specifically exist within the realm of the HTTP server and HTML documents. The CGI request basically calls an external program. This external program could be written in C, C++, Perl, or other programming language. The external program is loaded, executed to obtain the result and then unloaded. The CGI programs run as a separate process from the Web server and may be platform specific thereby reducing the portability of the application. Another problem with CGI is that each request creates a new server process.

Sun Microsystems proposed servlets as an alternative to CGI. Servlets would provide a more flexible, portable, and faster solution to providing the additional functionality. Servlets are written in Java and can call other Java classes or JavaBeans to provide additional functionality. Since the servlets are written in Java they provide application portability. The server handles multiple requests for the same servlet by having each request run as a separate thread.

JSP is an extension of Java Servlets. The JSP page is compiled into a servlet and executed by the Java Virtual Machine. Therefore a JSP page implementation is more scaleable than a CGI implementation. JSP pages require less server processes than a CGI implementation. Requests for data from multiple JSP pages can be handled by a single connection to the database with a JDBC JavaBean.

Mechanics of JSP
The Web server recognizes that a page with a .jsp extension requires additional processing. The JSP file is compiled into a Java servlet. The servlet can execute any code that is within the <% … %> tags or request information from JavaBeans that are referenced in the JSP page. The response to a user request for a JSP page is an HTML document that is displayed in a Web browser.

Component Development with JavaBeans and EJBs
Component development makes it easier to separate the presentation, business logic, and the data layers. In JSP we have the following components: JavaBeans, Enterprise JavaBeans (EJB), XML definitions, and tag libraries. Developers can build highly cohesive and loosely coupled JavaBean components for the business logic and data access functions. These beans are then available for use in other applications, thereby promoting reusability.

Extensibility
JSP is an extension of Java servlets. The JSP specification provides for extending the tags. An extended tag library can be created and used on different systems. JSP's also have a close relationship with the Extensible Markup Language (XML). A JSP page can be written as an XML document instead of an HTML document.

2. SERVER-SIDE SCRIPTING USING JSP

Types of elements
JSP is composed of five types of elements. The first three (scriptlets, expressions, and declarations) are grouped together as scripting elements. The other two elements are action and directives.

The **Scriptlet** element allows embedding a Java code fragment directly into a JSP page. Scriptlets can, but do not have to, produce output to the page.
<% Java code fragment %>

An **Expression** element is a Java expression whose value is evaluated and returned as a string to the page.
<%= Java code expression %>

A **Declaration** element is used to declare methods and variables that are initialized and to make them available to other scripting elements. Declaration elements must be a complete Java statement that ends with a semicolon and do not produce output like expression and scriplet elements. Declarations made with this element are "global."
<%! Java code declaration; %>

We also have **Directives** that send messages to the JSP engine. Their only purpose is to provide information on the compilation of the JSP page. We can have include, page, and taglib directives.
<%@ directive_name %>

Action elements are predefined functions. The Action element can create and use existing objects. Each action element has attributes. Two common attributes are the ID and Scope. The ID uniquely identifies the Action element and the scope attribute identifies the lifecycle of the Action element.
< jsp:action_name />

Resource actions specify resources external to the current JSP page. They specify the interactions with other JSP, HTML, and XML pages.

Development Environment
The scripting, using Java, will be demonstrated by using examples developed in the following environment: Microsoft NT server, using Internet Information Server (IIS), Jakarta Java Server and Java Server Pages (JSP). Oracle is the database used in the examples. However, the same examples have been executed using an Access database. The only difference is the Data Source Name (DSN) that points to the database. All of the application scripts are embedded within HTML documents that are stored on the server with the .jsp extension.

Examples
JavaBeans have not been used in the examples. The reason for this was to keep the examples as simple as possible and to clearly show the interaction with the database. An example of querying from a database is shown first. The steps include connecting to the database, creating a result set using SQL, and then looping through the result set and placing the resulting data into an HTML table on the Java Server Page using Java as the scripting language. Next examples of adding, changing, and deleting of record occurrences within tables in an Oracle database are shown. In these examples the process of connecting to the database and creating result sets is identical to that of querying from the database.

The first step on the Java server page is to identify that the scripting language is Java. This is accomplished with the following code at the very beginning of the Web page.

```
<%@     page language="java"
        import="java.sql.*"
%>
```

The page language clause identifies the scripting language. The java.sql clause imports the classes necessary for the SQL interaction with the database.
As seen the above code the <% starts scripting and %> ends scripting. This embedded code is interspersed throughout the HTML code on the Web page as needed.

Next the embedded script that is necessary to connect and open a database is shown.

Connecting to the database
The example database is a catalog table that has the following fields: an id, description, and price. The first step in interacting with the catalog table in the Oracle database is to create a connection to the database, and then open the domain of the database that the catalog table is within.

First an instance of the connection object is created with the java.sql.Connection statement. The variable **cnn** is user defined. In this example a Java Database Connection (JDBC) to Open Database Connection (ODBC) is established to connect to the database. The statement Class.forName establishes the name of the driver for the ODBC to JDBC linkage. The next three statements identify and establish the access for a specific domain within the Oracle database. The props object created by the java.util.Properties class allows for the specification of the userid and password to the database. The actual connection is made by the getConnection method of the DriverManager. The argument supplied is a combination of the JDBC and ODBC linkage along with the Data Source Name (DSN), which identifies the Oracle database. In this case the DSN is named demo. The props clause in the getConnection statement references the props object, which contains the userid and password to the database.

```
<%
java.sql.Connection cnn;

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
java.util.Properties props = new java.util.Properties();
props.put("user","CIS435_DEMO");
props.put("password","DEMO");
cnn =
java.sql.DriverManager.getConnection("jdbc:odbc:demo
",props);
%>
```

Opening a resultset
Next a recordset must be opened. A resultset is collection of rows selected from one or more tables from the database. In this example only the Catalog table is used. First an instance of the java.sql.Statement object must be created, and also an instance of the java.sql.ResultSet must be created. In the following statements, stmt and results are user-defined names for the objects. Then a metaData instance is created for the purpose of identifying the number of columns within the resultset. This is accomplished with the ResultSetMetaData method of java.sql. The statement int numCols established a variable, which will contain the number of columns.

```
<%
java.sql.Statement stmt;
java.sql.ResultSet results;
java.sql.ResultSetMetaData metaData;
int numCols;
```

```
%>
```

Query from a table in the database
Now that the objects have been created the data can be extracted from the database.

```
<%
stmt = cnn.createStatement();
results = stmt.executeQuery("Select * from Catalog");
metaData = results.getMetaData();
numCols = metaData.getColumnCount();
%>
```

First an instance of the statement object, named stmt, is created. Next the heart of whole process is accomplished the executeQuery method. This executes the SQL select command, which selects all the rows and all the columns from the Catalog table and places the data into the resultset named results. The getMetaData gathers information about the resultset. The meta data of interest is the number of columns, which is then placed into the variable named numCols.

Now that the resultset contains the data from the database the contents of the resultset is next displayed on the Web page by looping through the resultset. This is accomplished by creating a row in the HTML table for each row in the recordset. The number of columns in the HTML table is determined by the numCols variable.

Creating an HTML table of data from the resultset
The following code shows the entire process of creating an HTML table of data from the resultset.

```
<table>
<%
while(results.next())              {
%>
<tr>
<%
for(int i=1; i <= numCols; i++)  {
%>
<td><%= results.getString(i) %></td>
<%
}  //  end of for loop
}  //  end of while loop
results.close();
%>
</table>
```

The first step in this process is to start the HTML table with the <table> command. Then the code for looping through the resultset is placed within the table. The while(results.next()) command tests to see if there are any more rows in the resultset. If there are then the looping process continues. For each row in the resultset a row in the HTML table is created with the <tr> statement. Then within each row of the HTML table the for loop steps through each column of the resultset row and creates a column within the HTML row with the <td> statement. Then the heart of this process is accomplished by placing the content of column from the resultset into the column within the row of the HTML table with the statement <%= results.getString(i) %>. The code <%= places the following value, in this case results.getString(i) onto the HTML Web page.

Adding a record to a table within the database
Now to the heart of the interaction with the database where records will be added, changed or deleted. In the following example a record will be added to the Catalog table.

The process of adding a record to a database table entails the use of two Web pages. The first page contains an HTML form where data is entered. The form also contains a submit button that when clicked submits the values of the objects on the form into a buffer which is available to the posted page. Below is the code for the HTML form.

```
<form action = "addUpdate.jsp" method="post">
  <table>
    <tr>
      <td><b>Stuff ID:</b></td>
      <td><input name="id" type="text" maxlength="8"
size="8"></td>
    </tr>
    <tr>
      <td><b>Description:</b></td>
      <td><input name="description" type="text"  max-
length="50" SIZE="50"></td>
    </tr>
    <tr>
      <td><b>Price:</b></td>
      <td><input name="price" type="text"  max-
length="8" SIZE="8"></td>
    </tr>
    </table>
<p><input type="submit" value="Continue"></p>
</form>
```

In this example, when the submit button is clicked the three fields – id, description, and prices are posted to the jsp page named addUpdate.jsp. The values actually go into a buffer, which is available to the jsp page.

The next step is then for the jsp page, in this case addUpdat.jsp, to get the values from the buffer and insert a record into the database with the values.

As in the case of the query from the database, a connection to the database must be made. Once that connection is made the values posted from the Web page with the form are then accessed and placed into variables. The below code explains that process.

```
<%
// Get the values from the form and place into variables
String tmpID = " ";
int cnvID = 0;
tmpID = request.getParameter("id");
cnvID = Integer.parseInt(tmpID);
```

```
String tmpDescr = " ";
tmpDescr = request.getParameter("description");
String tmpPrice = " ";
double cnvPrice = 0;
tmpPrice = request.getParameter("price");
Double tmpValue = Double.valueOf(tmpPrice);
cnvPrice =tmpValue.doubleValue();
%>
```

The three fields are accessed with the request.getParameter( ) method. The request object references the buffer area. Then content of each field is String. Therefore the number fields must be converted from string to either int or double, the explanation of which is outside the scope of this paper.
Now that the values are in variables on the Web page, and the database is connected, a record can now be inserted into the database table. First a determination is made as to whether a record with the same id is already in the table as follows.

```
<%
// Verify if the record is already on file
sql = "Select * From Catalog Where Cat_ID = " + tID;
stmt = cnn.createStatement();
results = stmt.executeQuery(sql);

// If the record is on file go to invalid page
if (results.next())     {
            results.close();
            %>
            <jsp:forward page = "onFile.jsp" />
            <%;
}  // end if
```

A resultset is created that will either contain one record or no records. If the resultset contains one record then the record is already in the table and the add process is terminated by closing the resultset and forwarding to a notification page.
If the record is not on file then a record is inserted into the table, and the resultset is closed.

```
<%
Statement insert = cnn.createStatement();
int rowInserted = 0;

// Insert the record into the table
sql = "Insert Into Catalog Values (";
sql = sql + tID + ", ";
sql = sql + "'" + tDescription + "', ";
sql = sql + tPrice + ")";
rowInserted = insert.executeUpdate(sql);
// Close the recordset
results.close();
%>
```

A create statement must be created, in this case named insert. Also, an int variable that will receive the result of the insert, which is named rowInserted, must be created. Then a String variable named sql is built with the SQL command that will insert the three fields into table. The row is actually inserted with the executeUpdate method using the sql variable as the argument. Since the absence of a duplicate record was already determined the use of rowInserted is not necessary.

Changing a record in a table within the database
The process of changing a record in a database table is similar to that of adding a record. The values, which are posted from the Web page with the form, are in a buffer available to the jsp page. The example for the add process would apply to the change except the jsp page will now be changeUpdate.jsp.

Again a connection to the database must be made, and the values from the form are then accessed and placed into variables. As in the case of the add process, a determination is made as to whether the record exists or not. In the case of the change if the record does not exist the process is terminated. If the record is on file then the record is modified and the resultset is closed.

```
<%
int rowModified = 0;
Statement modify = cnn.createStatement();
// Modify the record in the table
sql = "Update Catalog Set Description = '" + tDescription + "'";
sql = sql + Price = " + tPrice;
sql = sql + " Where Cat_ID = " + tID;
rowModified = modify.executeUpdate(sql);
%>
```

A create statement must be created, in this case named modify. Also, an int variable that will receive the result of the update, which is named rowModified, must be created. Then a String variable named sql is built with the SQL command that will modify the three fields in the row of the table. The row is actually modified with the executeUpdate method using the sql variable as the argument. Since the presence of the record was already determined the use of rowModified is not necessary.

Deleting a record from a table within the database
The process of deleting a record from a database table is almost identical to that of changing a record. The most notable exception is the sql command that either updates or deletes a record. The other exception is that the only value from the buffer is the id field, which is the primary key in the database table. The name of the jsp page will now be deleteUpdate.jsp.

Once again a connection to the database must be made. As in the case of the change process, a determination is made as to whether the record exists or not. If the record does not exist the process is terminated. If the record is on file then the record is deleted and the resultset is closed.

```
<%
int rowDeleted = 0;
Statement delete = cnn.createStatement();
// Delete the record from the table
```

```
sql = "Delete From Catalog Where Cat_ID = " + tID;
rowDeleted = delete.executeUpdate(sql);
%>
```

A create statement must be created, in this case named delete. And an int variable that will receive the result of the delete, which is named rowDeleted, must also be created. Then a String variable named sql is built with the SQL command that will delete the row from the table. The row is actually deleted with the executeUpdate method using the sql variable as the argument. Since the presence of the record was already determined the use of rowDeleted is not necessary.

3. CONCLUSIONS

When developing interactive Web page applications several options are available. One of those options is Java Server Pages. This paper has given examples of the fundamental processes for interactive Web page development - querying from a database, adding records to a database, and changing or deleting records. In each of these processes the type of scripting must be identified, in this case Java. A connection must also be made to the database.

In the case of the query, data is extracted from the database and placed into a resultset. The content of this result set is then displayed on the Web page. Although not necessary, for clarity of display the placement of the data into an HTML table is preferable.

For add, change, and delete processes two Web pages are necessary. The first page contains a form, which is used for data entry, and the values from the objects on the form are then posted into a buffer area, which is available to the JSP page that will interact with the database. On the JSP page the values from the buffer area is placed into variables on the Web page. These values are then used to add a record, change a record, or delete a record.

4. REFERENCES

Avila, John, 2001, *Server-Side Java Programming for Web Developers*. Scott/Jones Inc., El Granada, CA.

Annunziato, Jose, & Stephanie Fesler Kaminaris, 2001, *Sams Teach Yourself JavaServer Pages in 24 Hours*. Sams Publishing, Indianapolis.

Ben-Natan, Ron, & Ori Sasson, 2000, *WebSphere Starter Kit*. McGraw-Hill, New York.

Bhamidipati, Kishore, 1998, *SQL, Programmer's Reference*. McGraw-Hill, New York.

Fields, Duane K. & Mark A. Kolb, 2000, *Web Development with Java Server Pages, A practical guide for designing and building dynamic Web services*. Manning, Greenwich.

Gittleman, Art, 2001, *Internet Applications with the Java 2 Platform*. Scott/Jones Inc., El Granada, CA.

Hougland, Damon, & Aaron Tavistock, 2001, *Core JSP*. Prentice Hall, Upper Saddle River, NJ.

Powell, Thomas A., 1998, *HTML, the Complete Reference*. McGraw-Hill, New York.

Sharma, Vivek and Rajiv Sharma, 2000, Developing e-Commerce Sites, an Integrated Approach. Addison Wesley, Boston.

Staugaard, Andrew C., Jr., 1999, Java for Computer Information Systems. Prentice-Hall, Upper Saddle River, NJ.

Tremblett, Paul, 2000, Instant JavaServer Pages. McGraw-Hill, New York.