This article was published in Summer 2002 edition of SQL>Update_RMOUG
(the newsletter of the Rocky Mountain Oracle Users Group).

An Overview of Web Technologies for Oracle-Driven Web Sites

by
Dustin Marx

A number of technologies are available for building database-driven web pages and sites. Developers using an Oracle database have even more options available for using Oracle to drive their web sites. This article briefly discusses some of the web technologies currently available that enable developers to build Oracle-driven web sites. Some advantages and disadvantages of the approaches are also listed. Examples are provided to illustrate several of the approaches.

Examples Background

The examples are based on the scott/tiger user schema. Imagine that you are tasked with creating a web page that lists all of the employees who work for President King at the fictitious company represented in the scott/tiger schema. Some of the technologies you might use to create this web page are discussed and examples of using some of these technologies are presented.

The Query

The query used to obtain the employee information from the Oracle database is shown in Listing 1:

Listing 1 – Employee SQL Query

```
SELECT e.empno, e.ename AS name, e.job,
       b.ename AS boss, dname AS department
  FROM emp e, emp b, dept
 WHERE e.mgr = b.empno
   AND e.deptno = dept.deptno;
```

Note that the `SELECT` statement shown will not list President King as an employee since President King has no associated manager.

The Stylesheet

The simple Cascading Style Sheet (CSS) employed by these examples is shown in Listing 2:

Listing 2 – marxstyle.css

```
BODY
{
      font-size:12.0pt;
      background:white;
      color:black;
}
A:link { color: blue; }
A:visited { color:gray; }
A:active { color:red; }
p.TITLE
{
      font-size:20.0pt;
      color:blue;
      font-weight:bold;
      text-align:center;
}
```

The Environment

All the examples have been tested using Oracle products on a Windows XP platform. The database used in all examples was Personal Oracle 8.1.7. SQL*Plus was used for the `MARKUP` example and Oracle9*i*AS was used for the PL/SQL Web Toolkit (mod_plsql), PL/SQL Server Pages (mod_plsql), Java servlets (Apache JServ Java servlet engine), and JavaServer Pages (OracleJSP with JServ) examples.

Oracle supplies many Java tools to facilitate the Java-based web technologies. For example, although I used Borland JBuilder to build my Java code (servlet and JSP examples), I could have used Oracle's JDeveloper, a product that has its roots in an earlier version of JBuilder. The documentation also states that the Tomcat (Apache Jakarta) Web Server can also be used for serving up servlets and JSPs through Oracle9*i*AS. This can be advantageous since Tomcat is the reference implementation of the Servlet and JSP specifications and is widely used.

The Output

Each example shown provides nearly identical output on the web browser. However, there are some slight differences. These differences mainly result from the automatic HTML table construction done by various Oracle tools (SQL*Plus MARKUP as well as the Oracle JavaBean [DBBean] and custom tag library [sqltaglib] used in JSP examples). Small characteristics of the constructed table differ between these and the other examples. However, all examples list a table showing the 13 employees who work for President King. Each employee's information is listed in the same order as queried: number, name, job, boss, and department.

The Disclaimer

I did not follow my personal coding and style standards in these examples in an effort to conserve space. I usually employ a lot of white space to enhance readability. Also, in an effort to maintain as much simplicity as possible, I embedded some features in the code examples that I would normally have separated into other classes, components, or layers. I specifically mention some of the most offensive of these as they are encountered throughout the article.

SQL*Plus

SQL*Plus provides some basic web support that one might use to develop a simple web report like the one described in this example.

SQL*Plus Markup System Variable

A very basic way to generate a web page from a query is to use the SQL*Plus system variable `MARKUP`. This relatively new feature of Oracle SQL*Plus allows a developer to tell the SQL*Plus environment to put the output in HTML format and this output HTML source can be spooled to a file. Once the HTML output is spooled to a file, another user (such as President King) can access the generated HTML file with a web browser. Listing 3 shows how the employee query might be done using SQL*Plus's `MARKUP`:

Listing 3 – Using SQL*Plus `MARKUP`

```
SET markup html on spool on pre off
SET markup html head '<title>Our Employees</title><meta http-
equiv="Content-Style-Type" content="text/css"><link type="text/css"
rel="stylesheet" href="C:/webstuff/marxstyle.css">'
SET markup html body 'bgcolor="white" text="black"'
SET markup html entmap off
SET feedback off
SPOOL markupResults.html
PROMPT <p class=TITLE>Our Employees</p>
SELECT e.empno, e.ename AS name, e.job,
       b.ename AS boss, dname AS department
  FROM emp e, emp b, dept
 WHERE e.mgr = b.empno
   AND e.deptno = dept.deptno;
SPOOL off
SET markup html off spool off
SET feedback on
```

There are several interesting things to note in the example shown in Listing 3. In the first line, `SET markup html` is used to turn HTML output mode on, to enable spooling of the HTML output to a file, and to turn off the preformatted (`<pre>`) tag. If the `pre` option had been set to `on`, the output HTML would appear on the browser just as it appears in SQL*Plus. With `pre` option set to `off`, however, the output of the query is placed in an HTML table. You could use the above approach without the `MARKUP` system variable, but `MARKUP` provides you with the HTML table and other tags necessary for a basic web page.

The `head` option of the `markup` variable (second line) allows for text to be inserted between `<head>` tags in the produced HTML. This is useful for things like title, style sheet inclusion, other `meta` tags, and anything else you would normally want between your head tags. The `body` option of the `markup` variable allows for attribute text to be specified for your opening `<body>` tag.

Notice that the output of this script file (which will be in basic HTML format) is spooled to a file with an `.html` extension. President King can then access that file through any web browser.

Note also that you can use the `PROMPT` SQL*Plus command as one method of including tags not produced by `MARKUP` in your HTML that is output to the specified `.html` spool file. In fact, you could produce all the HTML code with this method instead of using the `MARKUP` feature. This may be especially useful for developers using older versions of Oracle. After the query is executed in the above example, the remaining lines clear the options set earlier.

The contents of the produced file `markupResults.html` are shown in Listing 4 (with significant white space removed from the original to conserve space [this does not affect output on browser]):

Listing 4 – Spool output file (HTML format)

```
<html>
<head>
<title>Our Employees</title><meta http-equiv="Content-Style-Type"
content="text/css"><link type="text/css" rel="stylesheet"
href="C:/webstuff/marxstyle.css">
<meta name="generator" content="SQL*Plus 8.1.7">
</head>
<body bgcolor="white" text="black">
<p class=TITLE>Our Employees</p>
<br>
<p>
<table border="1" width="90%">
<tr>
<th>EMPNO</th>
<th>NAME</th>
<th>JOB</th>
<th>BOSS</th>
<th>DEPARTMENT</th>
</tr>
<tr>
<td align="right">7369</td>
```

```
<td>SMITH</td>
<td>CLERK</td>
<td>FORD</td>
<td>RESEARCH</td>
</tr>
 …
</table>
<p>
</body>
</html>
```

Advantages of `MARKUP` Approach

- Simple approach that requires only a little extra knowledge for regular SQL*Plus users
- Now comes standard with the database – no extra tools needed
- Cost: With very little extra training required and no extra licenses needed in most cases, this is perhaps the least expensive of the methods discussed in this article

Disadvantages of `MARKUP` Approach

- Best for only very simple pages (reports) [such as in our example]; in fact, only works for queries and not for DML statements
- Lacks flexibility of other methods discussed
- Requires individual to know SQL*Plus, SQL, and HTML – difficult to separate roles

My preference is to use `MARKUP` for simple web reporting needs that are internal, simple, and fairly static. This example is perhaps perfect for using the `MARKUP` approach since the data is fairly static (this small company likely does not change personnel information very often), the use of this generated page is primarily internal, and only a simple report is needed. I doubt that the `MARKUP` system variable, as part of SQL*Plus, was ever meant to be used for complex web application development. Other methods discussed here are more conducive for full-blown web applications. The intent of the `MARKUP` system variable seems to be to provide a simple and convenient way of generating web-ready database reports.

owa_util.showpage

The procedure `showpage` is part of the built-in PL/SQL package `owa_util.` It can be used in SQL*Plus to display PL/SQL stored procedure output if the stored procedure utilizes the `htp` or `htf` packages for HTML generation. The PL/SQL Web Toolkit (including the `owa_util`, `htp` and `htf` packages) is covered next.

PL/SQL Web Toolkit

The PL/SQL Web Toolkit facilitates the generation of web pages with PL/SQL code. PL/SQL packages can be written as they normally are and output can be rendered in HTML form using the `htp` (hypertext procedures) and `htf` (hypertext functions) built-in packages. This is made possible through a PL/SQL Web Gateway that comes with WebDB, Oracle Application Server, and Oracle9*i*AS.

WebAlchemy is a useful tool that will generate PL/SQL calls to `htp` procedures and `htf` functions based on supplied HTML code. A developer can write static HTML first, use WebAlchemy to convert that HTML to `htp` and `htf` calls, and then add other PL/SQL processing code where applicable.

Listing 5 shows the specification and Listing 6 shows the body of a small package that produces HTML very similar to that produced by the `MARKUP` method discussed previously.

Listing 5 – PL/SQL Package Specification

```
CREATE OR REPLACE PACKAGE toolkit_emp
```

```
IS
  PROCEDURE list_emps;
END toolkit_emp;
```

Listing 6 – PL/SQL Package Body

```
CREATE OR REPLACE PACKAGE BODY toolkit_emp
IS
  -- This procedure is "private" - not declared in spec
  PROCEDURE start_page
  IS
  BEGIN
    htp.htmlOpen;
    htp.headOpen;
    htp.title('Our Employees');
    htp.meta('Content-Style-Type', null, 'text/css');
    htp.linkrel('stylesheet','C:\webstuff\marxstyle.css','text/css');
    htp.headClose;
    htp.bodyOpen;
  END start_page;

  PROCEDURE list_emps
  IS
    CURSOR cur IS
      SELECT e.empno, e.ename AS name, e.job,
             b.ename AS boss, dname AS department
        FROM emp e, emp b, dept
       WHERE e.mgr = b.empno
         AND e.deptno = dept.deptno;
  BEGIN
    start_page;

    -- Print (htp.p) out HTML statements I want
    -- if I don't know of a PL/SQL Web Toolkit
    -- procedure for that functionality.
    htp.p('<p class=TITLE>Our Employees</p>');
    htp.tableOpen;
    htp.tableRowOpen;
    htp.tableHeader('EMPNO');
    htp.tableHeader('NAME');
    htp.tableHeader('JOB');
    htp.tableHeader('BOSS');
    htp.tableHeader('DEPARTMENT');
    htp.tableRowClose;
    FOR cur_rec IN cur LOOP
      htp.tableRowOpen;
      htp.tableData(cur_rec.empno);
      htp.tableData(cur_rec.name);
      htp.tableData(cur_rec.job);
      htp.tableData(cur_rec.boss);
      htp.tableData(cur_rec.department);
      htp.tableRowClose;
    END LOOP;
    htp.tableClose;
    htp.bodyClose;
    htp.htmlClose;
  END list_emps;
END toolkit_emp;
```

Advantages of PL/SQL Web Toolkit

- Enables mixing of PL/SQL for logic and data processing with HTML for presentation
- For most Oracle developers, there is no new language (such as Java) to learn – PL/SQL can accomplish all HTML generation needs since the `htp.p` procedure can be used to print out HTML tag text even when a toolkit procedure does not exist for that tag
- With WebAlchemy, HTML can be written separately from PL/SQL and PL/SQL data access and processing code can be added later

Disadvantages of PL/SQL Web Toolkit

- Proprietary solution – any change in underlying database would almost certainly require significant change to the presentation layer code
- Although WebAlchemy helps, writing PL/SQL code can still be tedious
- Does not completely separate the roles of database/back-end developers and GUI/front-end developers

owa_util.showpage Revisited

The `owa_util` package provides some useful procedures and functions, including the `showpage` procedure (which accepts no parameters). The `owa_util.showpage` procedure is useful for viewing (in SQL*Plus) the HTML code produced by `htp` and `htf`. Although this procedure will not actually render the HTML code like a browser does, it does enable one to see the generated HTML code that a browser would render when rendering the output of the PL/SQL. The SQL*Plus buffer limit (2,000 bytes by default; 1,000,000 bytes maximum) can be a limitation to using `owa_util.showpage`.

PL/SQL Server Pages (PSP)

PL/SQL Server Pages allow developers to mix HTML and PL/SQL in the same file. While the PL/SQL Toolkit approach requires all code to eventually be PL/SQL code (either written as PL/SQL directly or after being converted from HTML with WebAlchemy), PSP allows HTML and PL/SQL to be intermixed. Instead of making calls to PL/SQL stored procedures and functions that produce HTML, the HTML itself can reside alongside the PL/SQL. This same basic concept applies to all the “Server Pages” technologies; the major difference between the various “Server Pages” technologies is the type of language/syntax that can be mixed with HTML.

While PL/SQL Web Toolkit seems to be a more appropriate approach for a developer with high PL/SQL code needs and only small HTML code generation needs, PSP seems to be a more appropriate approach for the developer with high HTML needs and lesser PL/SQL needs.

Listing 7 shows PL/SQL Server Pages code that could be used to provide a browser with HTML similar to the previous examples.

Listing 7 – PL/SQL Server Pages Code

```
<%@ page language="PL/SQL" %>
<html>
<head>
<title>Our Employees</title>
<meta http-equiv="Content-Style-Type" content="text/css">
<link type="text/css" rel="stylesheet"
href="C:/webstuff/marxstyle.css">
</head>
<body bgcolor="white" text="black">
<p class=TITLE>Our Employees</p>
<table border="1" width="90%">
<tr>
```

```
<th>EMPNO</th>
<th>NAME</th>
<th>JOB</th>
<th>BOSS</th>
<th>DEPARTMENT</th>
</tr>
<% FOR emp IN (SELECT e.empno, e.ename AS name, e.job,
b.ename AS boss, dname AS department
                 FROM emp e, emp b, dept
                WHERE e.mgr = b.empno
                  AND e.deptno = dept.deptno) LOOP %>
<tr>
<td><%= emp.empno %></td>
<td><%= emp.name %></td>
<td><%= emp.job %></td>
<td><%= emp.boss %></td>
<td><%= emp.department %></td>
</tr>
<% END LOOP; %>
</table>
</body>
</html>
```

Note that while you compile PL/SQL procedures and functions as normal in SQL*Plus or other PL/SQL editing environment, PL/SQL Server Pages are actually loaded with an *operating system* command-line load instruction (`loadpsp`).

Advantages of PL/SQL Server Pages

- Easier mixing of HTML and PL/SQL than with PL/SQL Web Toolkit
- Need to know only HTML tags and do not need to learn PL/SQL procedure names and arguments as done with PL/SQL Web Toolkit
- PL/SQL scripting syntax is very similar to Java scripting syntax in JavaServer Pages

Disadvantages of PL/SQL Server Pages

- Like PL/SQL Web Toolkit, this is a highly proprietary approach to database-driven web pages
- Difficult to completely separate PL/SQL development from HTML development

Java Servlets

Java Servlets are written entirely in Java but have the ability to produce HTML code for rendering on a browser. In this way, Java Servlet technology does for Java (generation of HTML) what the PL/SQL Web Toolkit does for PL/SQL. Likewise, JavaServer Pages (discussed next) do for Java (direct mixing of Java and HTML) what PL/SQL Server Pages does for PL/SQL (direct mixing of PL/SQL and HTML). Note also that the Oracle documentation also talks about “PL/SQL Servlets.”

Listing 8 shows Java servlet code that produces an HTML page for the browser to render that is very similar to that produced by the previously discussed methods.

Listing 8 – Java Servlet code

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

/**
```

```
 * Servlet for producing list of employees working for President King.
 *
 * @author <a href="mailto:dustinmarx@yahoo.com">Dustin Marx</a>
 */

public class EmpServlet extends HttpServlet
{
   public static final String TITLE = "Our Employees";

   public void service (HttpServletRequest request,
                        HttpServletResponse response)
         throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String server =
        getServletConfig().getServletContext().getServerInfo();

      // Write HTML output
      out.println(  "<html>\n" + "<head>\n"
                  + "<title>" + TITLE + "</title>\n"
                  + "<meta http-equiv=\"Content-Style-Type\" "
                  + "content=\"text/css\">\n"
                  + "<link type=\"text/css\" rel=\"stylesheet\" "
                  + "href=\"C:/webstuff/marxstyle.css\">\n"
                  + "</head>\n"
                  + "<body bgcolor=\"white\" text=\"black\">\n"
                  + "<p class=TITLE>" + TITLE + "</p>\n");

      try
      {
         DriverManager.registerDriver(
           new oracle.jdbc.driver.OracleDriver());

         Connection conn = DriverManager.getConnection(
            "jdbc:oracle:thin:@yourhost:1521:slacker","scott","tiger");
         Statement stmt = conn.createStatement();
         String queryStr = new String(
            "SELECT e.empno, e.ename AS name, e.job, b.ename AS boss, "
          + "dname AS department FROM emp e, emp b, dept "
          + "WHERE e.mgr = b.empno AND e.deptno = dept.deptno");
         ResultSet rs = stmt.executeQuery(queryStr);
         out.println("<table border=\"1\" width=\"90%\">");
         out.println("<tr>");
         out.println("<th>EMPNO</th><th>NAME</th>");
         out.println("<th>JOB</th><th>BOSS</th>");
         out.println("<th>DEPARTMENT</th>");
         out.println("</tr>");
         while (rs.next())
         {
            out.println(  "<tr><td>" + rs.getString("EMPNO")
                        + "</td><td>" + rs.getString("NAME")
                        + "</td><td>" + rs.getString("JOB")
                        + "</td><td>" + rs.getString("BOSS")
                        + "</td><td>" + rs.getString("DEPARTMENT")
                        + "</td></tr>");
         }
         out.println("</table>");
      }
      catch (Exception e)
      {
         out.println(e);  // print exception information to screen
      }

      out.println( "</body>\n" + "</html>");
    }
}
```

I used JDBC^{TM} in the above Java servlet for my database access. In real applications, I do not include the JDBC code in the actual servlet; instead, I have specific Java classes or EJBs as part of a database isolation layer that handles database access. JNDI is also useful for obtaining a connection. To keep this case simple, however, I have embedded the JDBC directly in the servlet and obtained the connection with the traditional method. SQLJ is another option for Java developers using Oracle and Oracle9*i* now supports dynamic SQL statements in SQLJ. On a side note, JDBC is often called "Java Database Connectivity," but it is actually trademarked just as it is (JDBC^{TM}) with no other meaning.

While I used `PrintWriter.println()` statements in the above code to produce HTML, I could also have used the `oracle.html` package to handle some of the HTML production. I do not recommend printing exceptions directly to the screen despite that being shown in the example above. Instead, exceptions should be handled and/or only key information printed to screen rather than possibly an entire stack trace.

Advantages of Servlets

- As part of J2EE (Java 2 Enterprise Edition), Java servlets provide convenient access to powerful Java APIs and other J2EE components (such as EJBs)
- As part of J2EE, promotes the "Write Once, Run Anywhere"^{TM} platform independence

Disadvantages of Servlets

- Producing HTML requires tedious `PrintWriter.println()` calls similar to PL/SQL Web Toolkit in this way
- Separation of web designer and business logic developer roles not easy to accomplish
- For database developers, learning Java and object-oriented concepts may be daunting

JavaServer Pages

You have probably seen the `.jsp` extension on files when you have been using the web. These are JavaServer Pages (JSP) that are closely related to Java servlets. In fact, JSPs are actually converted into servlets by the web container and then run as servlets. The major advantage to writing JSP pages instead of directly writing the servlets is that a well-written JSP page can consist entirely of HTML tags and HTML-like JSP tags (including custom-written tags).

If a web page is mostly a processing page, a servlet is usually more appropriate. However, many web pages are mostly presentation with just a little database access (or other processing) and these are great candidates for JavaServer Pages. In most designs, a mixture of servlets for processing and JSPs for appearance works best.

Listing 9 shows how easy it is to mix HTML and Java using JSP technology, but this example is also a good idea of the problems of Java Server Pages when used incorrectly.

Listing 9 – JSP with all Java embedded directly in HTML

```
<jsp:directive.page import="java.io.*" />
<jsp:directive.page import="javax.servlet.*" />
```

```
<jsp:directive.page import="javax.servlet.http.*" />
<jsp:directive.page import="java.sql.*" />
<html>
<head>
<title>
Our Employees
</title>
<meta http-equiv="Content-Style-Type" content="text/css">
<link type="text/css" rel="stylesheet"
href="C:/webstuff/marxstyle.css">
</head>
<body bgcolor="white" text="black">
<p class=TITLE>Our Employees</p>
<table border="1" width="90%">
<tr>
<th>EMPNO</th>
<th>NAME</th>
<th>JOB</th>
<th>BOSS</th>
<th>DEPARTMENT</th>
</tr>
<% try
{
   DriverManager.registerDriver(
      new oracle.jdbc.driver.OracleDriver());
   Connection conn = DriverManager.getConnection(
      "jdbc:oracle:thin:@yourhost:1521:slacker","scott","tiger");
   Statement stmt = conn.createStatement();
   String queryStr = new String(
      "SELECT e.empno, e.ename AS name, e.job, b.ename AS boss, "
    + "dname AS department FROM emp e, emp b, dept "
    + "WHERE e.mgr = b.empno AND e.deptno = dept.deptno");
   ResultSet rs = stmt.executeQuery(queryStr);
   while (rs.next())
   { %>
   <tr>
   <td><% out.println(rs.getString("EMPNO")); %></td>
   <td><% out.println(rs.getString("NAME")); %></td>
   <td><% out.println(rs.getString("JOB")); %></td>
   <td><% out.println(rs.getString("BOSS")); %></td>
   <td><% out.println(rs.getString("DEPARTMENT")); %></td>
   </tr>
<% }
}
catch (Exception e)
{
   System.out.println(e);
} %>
</table>
</body>
</html>
```

While the above works, it features many potential maintenance and reusability problems. As I discuss in my *JavaWorld* "JSP Best Practices" article (see "Resources" section), there are many approaches that can be taken to improve maintainability and reusability of JSPs. For example, Java beans are used in Listing 10 to improve the readability and maintainability of the code. The Oracle-supplied Java Bean (`oracle.jsp.dbutil.DBBean`) used in this example handles table construction related to the query's result set and reduces the amount of Java code in the JSP. The `DBBean` is an example of a "Portable OracleJSP Programming Extension." These extensions, which include other beans and custom tags supplied by Oracle, will work in any JSP environment. Oracle also supplies extensions that require OracleJSP and will not work in other JSP environments. If portability is a concern, you will want to stay away from the Oracle-specific extensions. There are many custom tag libraries available online for use without charge.

Listing 10 – JSP utilizing Java Beans (Oracle-supplied DBBean)

```
<jsp:useBean id="db" class="oracle.jsp.dbutil.DBBean" />
<html>
<head>
<title>
Our Employees
</title>
<meta http-equiv="Content-Style-Type" content="text/css">
<link type="text/css" rel="stylesheet"
href="C:/webstuff/marxstyle.css">
</head>
<body bgcolor="white" text="black">
<p class=TITLE>Our Employees</p>
<jsp:setProperty name="db" property="user" value="scott" />
<jsp:setProperty name="db" property="password" value="tiger" />
<jsp:setProperty name="db" property="URL"
                 value="jdbc:oracle:thin:@yourhost:1521:slacker" />
<% try
{
   db.connect();
   String queryStr = "SELECT e.empno, e.ename AS name, e.job, b.ename
AS boss, ";
   queryStr += "dname AS department FROM emp e, emp b, dept ";
   queryStr += "WHERE e.mgr = b.empno AND e.deptno = dept.deptno";
   out.println(db.getResultAsHTMLTable(queryStr));
   db.close();
}
catch (Exception e)
{
   System.out.println(e);
} %>
</body>
</html>
```

Even with Java Beans used above, there is still some Java in the JSP. Custom tags can be written (or existing custom tag libraries can be used) to remove more (even all) Java from the JSP.

Listing 11 shows a JSP that utilizes an Oracle-supplied custom tag to completely replace all Java in the two preceding JSP examples. I strive to remove all Java from my JSPs by using Java Beans and custom tags.

Listing 11 – JSP Utilizing Custom Tag (Oracle-supplied sqltaglib Tag Library)

```
<%@ taglib uri="/WEB-INF/sqltaglib.tld" prefix="sql" %>
<html>
<head>
<title>
Our Employees
</title>
<meta http-equiv="Content-Style-Type" content="text/css">
```

```
<link type="text/css" rel="stylesheet"
href="C:/webstuff/marxstyle.css">
</head>
<body bgcolor="white" text="black">
<p class=TITLE>Our Employees</p>
<sql:dbOpen URL="jdbc:oracle:thin:@yourhost:1521:slacker"
   user="scott" password="tiger" connId="myConn1">
</sql:dbOpen>
<sql:dbQuery connId="myConn1">
  SELECT e.empno, e.ename AS name, e.job, b.ename AS boss,
         dname AS department FROM emp e, emp b, dept
   WHERE e.mgr = b.empno AND e.deptno = dept.deptno
</sql:dbQuery>
<sql:dbClose connId="myConn1" />
</body>
</html>
```

In Listing 11, there is no Java code. Instead there are HTML tags, JSP tags (such as the very first line of the example that tells the JSP about a tag library being used), and custom tags (`dbOpen`, `dbQuery`, and `dbClose` – all from the same tag library [`sqltaglib`]). The functionality written in Java in the previous two JSP examples still needs to be done, but it is done in Java classes connected to the `sqltaglib` tag library. The association between the tag library and the Java tag handler classes that perform this now-hidden Java functionality occurs in XML deployment files that the web container accesses.

Note that all of these JSP examples (even the custom tag example) have the database connection information (including the password) hard-coded into the JSP. Besides reducing maintainability and reusability, this is a security risk. Therefore, I point out once again that this is only done in these examples for illustration purposes. In most cases, you'll want to use a JNDI lookup of your database source and/or move all database access code to its own layer.

Advantages of JavaServer Pages

- Provide for separation of roles – web designer uses HTML and HTML-like custom and JSP tags while Java developer writes logic behind tags and JavaBeans used by JSP
- As part of J2EE, JSPs provide convenient access to powerful Java APIs and other J2EE components (such as EJBs)
- As part of J2EE, promotes the "Write Once, Run Anywhere"^{TM} platform independence
- Often useful to use JSPs in conjunction with Java servlets

Disadvantages of JavaServer Pages

- For database developers, learning Java and object-oriented concepts may be daunting
- Bad practice and style can make JSPs difficult to maintain and reuse (see "JSP Best Practices" article listed in "Resources" section for ways to combat this abuse)

Other Web Technologies

There are many other technologies that can be and often are used with Oracle to create Oracle-driven web sites. Due to space considerations, I briefly touch on only a few of these and do not include any examples or bullets on advantages and disadvantages.

ActiveServer Pages (ASP)

This Microsoft technology is very similar to JSP and PSP (hence the words "Server Pages" in all three technologies' names). During your web browsing, you have probably seen ASP files that have `.asp` extensions. With ASP, script (such as VBScript) can be mixed easily with HTML. Like the J2EE solutions discussed earlier (servlets and JSPs), this is a very widespread and well-documented technology. There are platform concerns, however, outside of an NT/Windows environment.

ASP.NET

Similar to Active Server Pages, ASP.NET is part of Microsoft's .NET platform and adds functionality to the capabilities of traditional ASP. ASP.NET and .NET offer many great features, but platform dependence is still a serious drawback.

Template Engines

Template engines are often viewed as substitutes or replacements for server pages. For example, Java-based template engines are often compared and contrasted with the better-known JSP technology. Just as JSPs provide custom tags and JavaBean support to link Java implementation to the JSP without placing actual Java in the JSP, web engines also provide tags that allow external Java functionality to be used in the page. Popular template engines that are Java-based and open source include Velocity (part of Apache Jakarta project), Freemarker, and WebMacro. See the "Resources" section for more information on these template engines. Regular users of template engines are adamant in their claims about the superiority of template engines compared to JavaServer Pages. A disadvantage of using template engines is that there is less attention and literature devoted to them. Also, developers who favor more general approaches for their own career flexibility may be a little wary of using a specific template engine exclusively.

PHP: PHP Hypertext Processor

PHP is a server-side embedded scripting language. PHP files have a `.php` or similar extension. PHP provides several functions to be specifically used with Oracle. These include functions such as Ora_Logon (connect), Ora_Logoff (disconnect), Ora_Commit, and Ora_Do (parse SQL statement, execute that statement, fetch results). PHP also provides ODBC support. Like the template engines, PHP users are typically devoted. However, also like the template engines, PHP does not seem to get the same attention as the J2EE and .NET solutions.

CGI, Perl, Python, and C

The CGI approach has been around as long as web pages. Using this approach, developers can write routines in languages such as C and Perl that are executed by CGI calls from web pages. Now though, Perl can even be mixed with HTML through PSP (Perl Server Pages – not same as PL/SQL Server Pages). Oracle 9*i*AS uses the Apache HTTP Server and so the Apache/Perl mod_perl capability is available (see "Resources"). Jython is a Python implementation written in Java that runs on the Java platform.

As with PHP and the template engines and perhaps all the other technologies discussed, the developers who are intimately familiar with these technologies intensely defend them as the solution for all programming problems. Web sites on Perl tout its "simplicity" and other virtues.

Java Applets

While Java applets are no longer as popular as they used to be, I still think of them fondly since they introduced me and many others to Java and to the idea of more elegant solutions than CGI for web development. A Java applet is written in Java separately from the HTML. In other words, the Java is not mixed with the HTML in any way. Instead, inclusion statements in the HTML similar to image inclusions are used to include the Java applet. Applets are relatively easy to write and deploy.

Unlike JavaServer Pages and Java servlets, applets are executed on the client machine rather than the server. This means that there is some time associated with downloading the applet. It also puts more requirements on the client machine to be able to execute the applets. Applets are also limited in what they can do because of security concerns related to them running on the client machine.

Still Other Web Technologies

There are still many more web technologies that can be used in conjunction with Oracle to produce database-driven web sites. There are far too many to list here. The technologies listed above, however, are among the most popular of web development technologies. I do list a few more web-enabling technologies in the "Resources" section.

Conclusion

There are many useful web technologies that can be used in conjunction with Oracle to build web-driven web pages and sites. This article has attempted to summarize some of these technologies and provide examples of several of the discussed technologies. A good source of information on web technologies and Oracle is the *Oracle9i Application Server Overview Guide* (see "Resources"). I specifically recommend reading Chapter 2 of this document ("Oracle9*i* Application Server Services") to get a feel for the web technologies supported by Oracle9*i*AS. The textual descriptions, examples, lists of advantages and disadvantages, and the resources listed below are all intended to help you decide which web technologies you might want to learn more about as you build Oracle-driven web sites. All of these technologies have advantages and disadvantages. Sometimes, the decision of which technology to use may be more political (developer, managerial, and customer pressure) than technical.

Bio

Dustin is a software engineer and architect at Raytheon in Aurora, Colorado. While much of his experience with Oracle has been in conjunction with C++ and the Pro*C/C++ precompiler, his recent and current work has been heavily based on Java and Oracle's Java-related products (such as using JDBC, SQLJ, and Oracle9i Application Server).

Resources

Apache/Perl Integration Project http://perl.apache.org/	ASP.Net http://www.asp.net/
"Java Server Pages" by Bradley D. Brown, *SQL>UPDATE_RMOUG* (Fall 2001)	JavaServer Pages http://java.sun.com/products/jsp/
"JSP Best Practices" by Dustin Marx, *JavaWorld* (November 30, 2001) http://www.javaworld.com/javaworld/jw-11-2001/jw-1130-jsp.html	*JSP*: "Working With Databases" (Chapter 9) http://developer.java.sun.com/developer/Books/jsp_2ed/chapter9.pdf
Oracle® JavaServer Pages Developers Guide and Reference by Brian Wright and others http://technet.oracle.com/docs/products/oracle8i/doc_library/817_doc/java.817/a83726/title.htm	
Database Access from JavaServer Pages *by Julie Basu, Oracle Corporation*	
Velocity Template Engine http://jakarta.apache.org/velocity/	ColdFusion Server Technology http://www.macromedia.com/software/coldfusion/
Freemarker http://freemarker.sourceforge.net/	WebCream http://www.creamtec.com/webcream/
Developing Web Applications with PL/SQL (PL/SQL Web Toolkit and PSP) http://download-west.oracle.com/otndoc/oracle9i/901_doc/appdev.901/a88876/adgweb.htm	
WebAlchemy – converts HTML into PL/SQL code (calls to PL/SQL Web Toolkit) http://www.users.bigpond.com/ahobbs/	
Java Applets http://java.sun.com/applets/	Java Servlet Technology http://java.sun.com/products/servlet/index.html
Oracle9*i*AS http://www.oracle.com/ip/deploy/ias/	Oracle 9*i*AS Http Server FAQ http://otn.oracle.com/products/ias/http/ohs-faq-v1022.htm
Oracle9*i* Application Server Documentation Library http://otn.oracle.com/docs/products/ias/doc_library/1022doc_otn/index.htm	
Oracle9i Application Server Overview Guide http://otn.oracle.com/docs/products/ias/doc_library/1022doc_otn/ias.102/a87353/toc.htm	
Oracle 9*i* Developer Suite http://www.oracle.com/ip/develop/ids/	Tomcat (Apache Jakarta) http://jakarta.apache.org/tomcat/
Perl Server Pages http://psp.sourceforge.net/	PHP HyperText Preprocessor http://www.php.net/
Python http://www.python.org/	WebMacro http://www.webmacro.org/

Zope Page Templates http://www.zope.org/	