Web Site Development
AN INTRODUCTION

A Primer in Web Application Development

The purpose of this primer is to provide you with some concept of how web applications work. You will look at some database information, some application development code, and the resulting web page created by that code. Since databases are the core of dynamic web applications, we'll begin there.

Databases are used to store information in a logical manner that will allow a person to search (query) that information in a manner that will provide them with the results they desire.

Database software is provided by several different companies which will offer a variety of products based upon your data storage needs, ways you wish to access that data and the number of users that will be required to have access to the data. Microsoft for example has two main database products. The first is Access, which is included in the office products suite, and SQL Server, which comes in both a standard and Enterprise edition. Enterprise software is used for large scale database requirements where many users are reading, writing and searching data all at the same time. The standard edition of SQL Server can easily cost you thousands of dollars in licensing from Microsoft. MySQL is another database software solution that is free and in wide use. Since it is open source software, it is available for FREE, which makes it very attractive. The downside is that it is used through a command-line interface, which means no pretty user interface with buttons to click. However there are several products available that put a bit of a face on MySQL, such as phpMyAdmin.

Databases usually consist of a number of tables that hold various types of information.

Employee Table (EMP_TBL)

EmployeeID	FirstName	LastName	DeptID
1	Robert	Jones	56
2	Samantha	Udi	56
3	Peter	Barnes	71
4	Simi	Yee	71
5	Walter	Chovski	14

Department Table (DPT_TBL)

DeptID	DeptName	Extension
12	Human Resources	526
14	Transportation Services	528
16	Marketing	530
56	Administration	636
71	Information Technology	680

The tables above provide data on employees and departments within a sample organization. If we wanted to find all the employees who work in the Information Technology department, we could perform a search/query on the database and come up with that information.

We write the instructions for the search using an SQL (Structured Query Language) statement. SQL statements are quite standard from database system to database system. Once you learn the basic syntax of the language you'll fair quite well.

Here is the SQL statement that will give us the names of all employees in the database who work in Information Technology.

```
SELECT EMP_TBL.FirstName, EMP_TBL.LastName
FROM EMP_TBL
WHERE EMP_TBL.DeptID=71;
```

If you take a moment and read through the SQL statement, you should find it quite easy to read and understand. All it says it get the first name and last name of all employees who work in the department that has the id of 71. The query will yield a result such as:

```
Peter Barnes
Simi  Yee
```

Let's take a look at a more complex query that will involve both of our tables. This time we want to retrieve the names of all the employees within Information Technology along with the name and extension number of their department. Here is the SQL statement:

```
SELECT EMP_TBL.FirstName, EMP_TBL.LastName, DPT_TBL.DeptName,
DPT_TBL.Extension
```

(continued next page)

```
FROM DPT_TBL INNER JOIN EMP_TBL ON DPT_TBL.DeptID = EMP_TBL.DeptID
WHERE EMP_TBL.DeptID=71;
```

This statement builds on the previous one but joins the criteria across the two tables to yield a result such as:

```
Peter Barnes      Information Technology  680
Simi  Yee         Information Technology  680
```

Databases are in use everywhere. Computer systems rely on them for handling all the complex information they require to perform. Being able to access database information to be used on the web is extremely important. If the information is being added dynamically (in real time when the user of a web site requests the information) to a web page, it simply means that if the information in the database is current, so will the information displayed on the web page.

Often users of web pages do not even realize they are looking at dynamic information that has been queried from a database. If you have a customized start page that displays only news items you are interested in (Excite, My MSN), those news items are being taken from a database based upon your personal preferences you set-up on the site.

So how does this work? The HTML page on the server includes special instructions that get acted upon before the page is sent to the user who requested it. These instructions might include a database query written in SQL that will retrieve some information and automatically include it in the HTML document by actually writing additional HTML code to the page. These instructions can be written in a variety of web application languages and are often dependent upon the server (Windows, LINUX) and the database system (Access, Microsoft SQL Server, MySQL) being used. Common web application development languages in use include ASP, ASP.net, PHP, JSP and ColdFusion.

If you are looking at a web page, you can often tell whether the page is dynamic or not based on the extension.

Extension	Dynamic?	Language/Company	Typical Server Used	Typical Database Used
.htm	No	HTML none	Any web server	none
.asp	Yes	ASP Microsoft	IIS (Internet Information Server – Microsoft) running on Windows	MS SQL Server* MS Access**
.aspx	Yes	ASP.net Microsoft	IIS (Internet Information Server – Microsoft) running on Windows	MS SQL Server* MS Access**
.php	Yes	PHP None – Open Source	Typically Apache running on Linux Also available for Windows	MySQL***
.jsp	Yes	JSP Sun Microsystems	Sun	any of the above
.cfm	Yes	CFML Macromedia	ColdFusion Application Server	any of the above

* Microsoft professional server product. You are looking at around $7000 to get into this product at the most basic level.
** Provided as a part of Microsoft Office product suite.
*** Open Source Database – free, very powerful and widely used – huge user community for support

Other Notes: Windows XP users must have the professional version to run IIS. No additional costs to use IIS with ASP or ASP.net beyond the operating system price. The ColdFusion application server is available to run on IIS, LINUX as well as others. ColdFusion will run you about $1700 for a single server license. Development version limited to a single IP address is free.

The costs of some of the web application development environments can be considerable, especially if you are setting up a server with these technologies for yourself or a client. Access to these technologies is considerably less in a shared hosting environment. Use of MS SQL server will run you at a minimum $40 per month. ColdFusion hosting accounts can be had for as little as $20 per month. Whereas PHP and MySQL are almost always included with any LINUX hosting account, same as ASP and often ASP.net with Windows based accounts.

(continued next page)

Now that you have had a chance to read about what databases are and what common web application languages are in use, it is time to get a bit more hands on. For this exercise, we will be using ColdFusion as our development environment. One of the nice things about ColdFusion is that through the administration settings of its application server, the connection to the database is done for us, which makes the reading and writing of data from our web pages quite simple.

1. In your **web_dev_exercises** folder you created (Exercise 3 – Step 1) create a new folder and name it **web_app**.

2. The examples you will be viewing are on the digitalvertebrae.com website. Please go to the following URL: http://www.digitalvertebrae.com/web_app_example/index.htm

   View Example 1:

   The page should display all the employees within the database. If you view the source for the page, you will not see any programming code beyond the normal HTML. The programming code was stripped from the document once its instructions were completed which resulted in a normal HTML document that was sent back to you. In order to understand this better, you will need to view the actual ColdFusion source code.

3. Return to the page you went to in Step 5 and **RIGHT-CLICK** the "**get it**" link for Example 1. From the menu that appears, select "**Save Target as**" and save the file (name it: **example1.cfm**) to the folder you created in Step 4.

   Now, open the file you just saved to your **web_app** folder using Notepad. What you should see now is a very similar page as you saw in the source code in Step 5, except now you should also see the ColdFusion Markup Language (CFML) code as well. Let's look closely at the code:

   ```
   <cfquery name="get_employees" datasource="digver">
   SELECT FirstName, LastName
   FROM EMP_TBL
   </cfquery>

   <html>
   <head>

   <title>Step 3 Example</title>

   </head>

   <body>

   <cfoutput query="get_employees">
   #FirstName# #LastName# <br />
   </cfoutput>

   </body>
   </html>
   ```

   The darker type is the CFML code whereas the lighter type is just regular HTML source code.

4. When you clicked the "view it" link to see Example 1, you sent a request for the document named example1.cfm across the Internet to the Digital Vertebrae web server. Since the document has a '.cfm' extension, the page gets handed off to the ColdFusion application server before getting sent back to you. The application server then reads the instructions on the page, completes those instructions (removing the code as it does so), renders the HTML page based on the instructions and sends the resulting page back to you.

(continued next page)

Specifically, the first block of instructions contains ColdFusion code wrapped around a SQL statement.

```
 <cfquery name="get_employees" datasource="digver">
SELECT FirstName, LastName
FROM EMP_TBL
</cfquery>
```

The first line provides instruction to generate a query on the database identified as **'digver'** (this is setup in the administration area of the ColdFusion server and identifies an Access database for these examples) and provide the results in a query record set named **'get_emloyees'**. A record set is the data that is returned by the database as a result of a query (search).

The next two lines are the SQL statement that instructs the application server to get all of the employee first and last names from the database table named **EMP_TBL**.

The final line closes the CFML instructions.

5. The next set of instructions located in the body of the document will write the data to the page. These instructions are placed where you want the data to be put.

```
<cfoutput query="get_employees">
#FirstName# #LastName# <br />
</cfoutput>
```

The first line states to output the record set from the query named "get_employees". The next line gets added to the page for each record returned. In other words, if there are 100 names in the database, it will write 100 first names, last names and break <br /> tags. If there are 0, it won't write any. The pound signs mean to replace the field name with the data from the database. So in the case of the first record returned #FirstName# #LastName# <br /> is replaced with Robert Jones <br />. The second would be Samantha Udi <br />.

6. Granted, the first example is not very sexy, just plain text on a white page. The great thing about being able to get the data from the database in the manner that we can is that we can create and format our page however we wish and have the data added to it.

Go to the example page from Step 5 and view example 2.

The page will display the same information, except this time it is structured nicely in a table. Still not that sexy I realize, but still better than Example 1.

7. Return to the page you went to in Step 9 and **RIGHT-CLICK** the "**get it**" link for Example 2. From the menu that appears, select "**Save Target as**" and save the file (name it: **example2.cfm**) to the folder you created in Step 4.

Now, open the file you just saved using Notepad. This page is basically the same as the page from Example 1, except you should see that the output information is placed within a table row structure. So instead of simply putting first name, last name and a break tag, the application server will write the table row and table data tags for each record in the record set. Here's the code for the table:

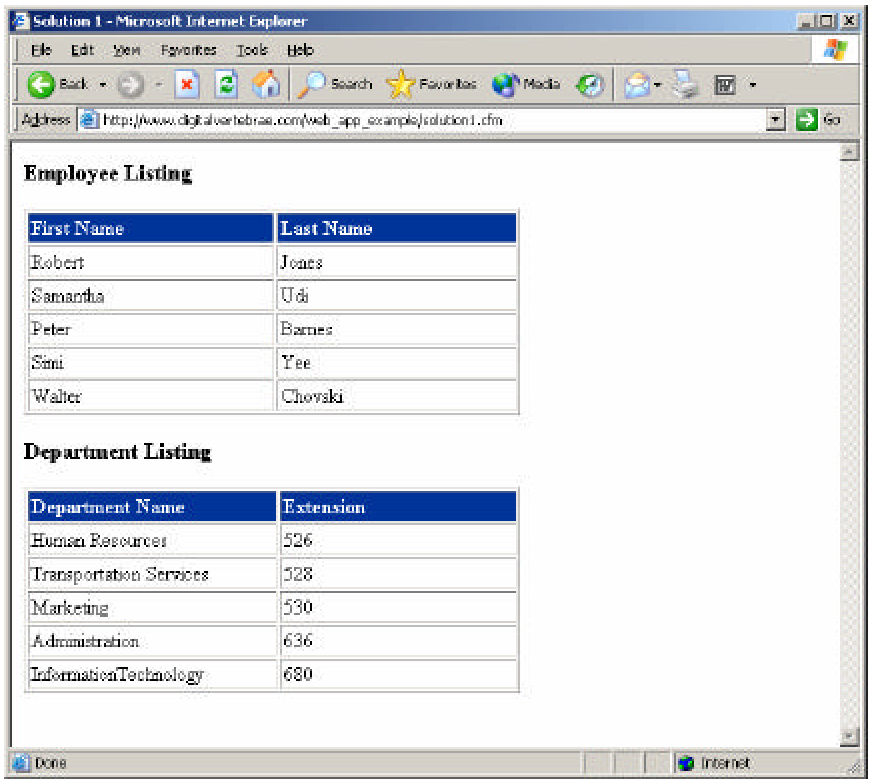
```
<table width="400" border="1" cellspacing="2" cellpadding="2">
  <tr bgcolor="#003399">
    <td width="142"><font color="#FFFFFF"><b>First Name</b></font></td>
    <td width="138"><font color="#FFFFFF"><b>Last Name</b></font></td>
  </tr>

  <cfoutput query="get_employees">
    <tr>
      <td>FirstName#</td> <td>#LastName#</td>
    </tr>
  </cfoutput>
</table>
```

The darker type is the CFML code whereas the lighter type is just regular HTML source code.

(continued next page)

8. Here is your chance to give it a try. Recall that we have two tables in our database, one for employees, the other for departments. Using Example 2 as a guide, you will add the additional CFQUERY information and CFOUTPUT tags to display a list of all the departments in the sample company. You will need to create a new CFQUERY for the top of the page. You will not be able to combine it with the CFQUERY for get_employees – that won't work. As well, create a separate table to display the departments. It can be similar to the employees table but should display the department name and extension.

   Give it a try. To test it, ensure you upload it to your folder on the Digital Vertebrae web server. Your page should look like this:


Solution 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.digitalvertebrae.com/web_app_example/solution1.cfm
Go
Done
Internet

Employee Listing

First Name	Last Name
Robert	Jones
Samantha	Udi
Peter	Barnes
Simi	Yee
Walter	Chovski

Department Listing

Department Name	Extension
Human Resources	526
Transportation Services	528
Marketing	530
Administration	636
InformationTechnology	680

   If you have problems the solution is available at: http://www.digitalvertebrae.com/web_app_example/index.htm

9. To truly illustrate the power of dynamic web pages though, it is better to give you an example of how you can enter data into a database from a web page and have it displayed back for you. The next example will follow this process through.

   What we will do is create a web page that allows you to add employees to the database and display the complete list of employees on that same page based upon the data in the database. The employee list will display first name, last name, department and extension. Text fields will be provided to enter in the first and last names as well as a drop down list to select which department the new employee will be in.

   We start by building a web page that has the required elements in it. We'll add the CFML code after. Please go to the following URL: http://www.digitalvertebrae.com/web_app_example/index.htm

   View Example 3:

   The page should display all the elements described above.

10. Return to the page you went to above and **RIGHT-CLICK** the "**get it**" link for Example 3. From the menu that appears, select "**Save Target as**" and save the file (name it: **employee_system.cfm**) to the folder you created in Step 4.

(continued next page)

11. Now, open the file you just saved to your **web_app** folder using Notepad. We will begin to add the code pieces required to make this page a functioning web application. At the very top of the page, add the following code to generate the query that will return the record set of employee information:

```
<cfquery name="get_employees" datasource="digver">
SELECT EMP_TBL.FirstName, EMP_TBL.LastName, DPT_TBL.DeptName, DPT_TBL.Extension
FROM EMP_TBL INNER JOIN DPT_TBL ON EMP_TBL.DeptID = DPT_TBL.DeptID
</cfquery>
```

12. Next, add to the table code so the results of the above record set will be displayed:

```
<table width="700" border="1" cellspacing="2" cellpadding="2">
  <tr bgcolor="#003399">
    <td width="142"><font color="#FFFFFF"><b>First Name</b></font></td>
    <td width="138"><font color="#FFFFFF"><b>Last Name</b></font></td>
    <td width="142"><font color="#FFFFFF"><b>Department</b></font></td>
    <td width="138"><font color="#FFFFFF"><b>Extension</b></font></td>
  </tr>
<cfoutput query="get_employees">
  <tr>
    <td>#FirstName#</td>
    <td>#LastName#</td>
    <td>#DeptName#</td>
    <td>#Extension#</td>
  </tr>
</cfoutput>
</table>
```

The darker type represents where you should make the additions to your existing page.

13. Upload the **employee_system.cfm** file to your folder on the web server and test it. You should see all of the employee data displayed in the table.

14. Next, we will add another query to generate a record set that will automatically be used to fill in the values of our drop-down list. Add the code below beneath the query you added in Step 14.

```
<cfquery name="get_departments" datasource="digver">
SELECT DPT_TBL.DeptID, DPT_TBL.DeptName
FROM DPT_TBL
</cfquery>
```

15. Next we'll adjust the select menu by adding the output code to add the departments to the list. Locate the select tag on the form and make the following additions as shown below:

```
<td width="139"><select name="DeptID">
   <option value="0" selected>select department</option><br>
                    <cfoutput query="get_departments">
                    <option value="#DeptID#">#DeptName#</option>
                    </cfoutput>
  </select>
</td>
```

The darker type represents where you should make the additions to your existing page.

16. Upload the **employee_system.cfm** file to your folder on the web server and test it. You should see when you click the drop-down list that the departments from the database are displayed.

(continued next page)

17. Next, we'll enable the web form so the data will be submitted to the web server to be added to the database when the user clicks add. In the next step, we'll be creating a separate CFML page that will have the SQL statement to insert the data to the database. This page will be called **employee_insert.cfm** and it will be the action page we define in the form tag. Make the following addition to your page:

    <form name="addemployee" method="post" action=" **employee_insert.cfm**">

    The darker type represents where you should make the additions to your existing page.

18. In Notepad, create a new blank document and save it as **employee_insert.cfm**

19. We'll now add code to the page that will insert the information received from the submitted form into the database. At the top of the page, add the following:

```
<cfquery name="insert_employee" datasource="digver">
    INSERT INTO EMP_TBL (FirstName, LastName, EmpID)
    VALUES ('#form.FirstName#', '#form.LastName#', '#form.EmpID#')
</cfquery>
```

    You will notice that the above SQL statement is different from what you've seen before. Basically, all it says is to take the values from our form and place them into the EMP_TBL corresponding fields.

20. We will add one more line of CFML instructions. These will redirect back to the **employee_system.cfm file** and send that back to the user. This way then user should be immediately able to see that the employee has been added. Add this line:

    <cflocation url="**employee_system.cfm** ">

21. Upload the **employee_ insert.cfm** file to your folder on the web server. Navigate to the **employee_system.cfm** document and try entering and submitting an employee. If all works properly you should see the addition added to the table when the page reloads. If not, and you receive an error – go back and review the steps until your application works.

    You can get the completed finished solutions at:

    http://www.digitalvertebrae.com/web_app_example/index.htm

22. What we have not attempted to do here is set any criteria for our query (such as display only employees in the Marketing Department) or sort criteria (by last name or first name or by department). We have also not covered the ability to delete records from the database.

    As well, we have not added any server-side or client-side validation to the submission form. Meaning that whatever is in the input boxes when the form is submitted will get added to the database, which in a real situation would not be acceptable.

    What you should have gained from this primer is a better understanding of what databases are, how web applications work, and how HTML forms are an integral part of allowing users to submit data to a web server.

    All of the web application languages discussed in this primer have commonalities with each other. Once you learn one well, understanding the syntactical differences of another is not a huge stretch. All of them can be used to build powerful and secure web based application provided the care is taken to develop them properly.