SDG ColdFusion Training

Summer S. Wilson, AgriLife IT

Session 1: April 16, 2009

Part 1: Reintroduction to Web Dev and ColdFusion

Quick Reminder: Static versus Dynamic

A simple website delivers static pages. The content doesn't change unless manually modified by the developer, and it generally just contains text, images, and perhaps some embedded objects like JavaScripts, Flash, etc.

Static pages are processed as follows:

1. A user requests the page by typing its URL in a browser.
2. The browser requests the page from the web server.
3. The web server locates the HTML page and sends it to the browser.
4. The browser interprets the HTML and displays the page to the user.

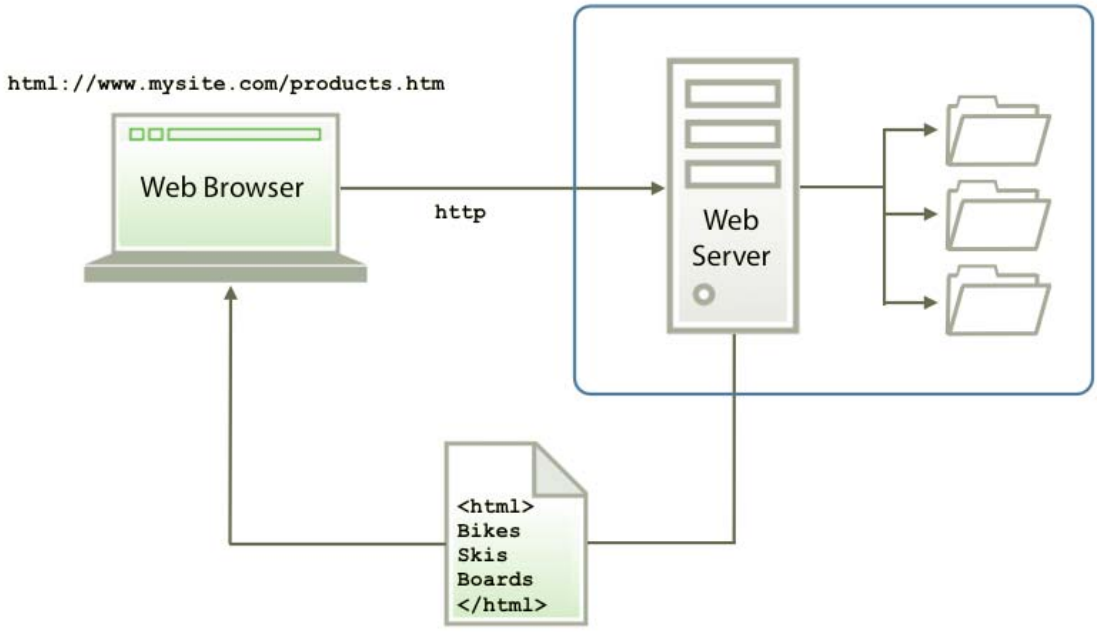

Figure 1: Serving up static web pages

A dynamic website, aka a web application, using programming to change what a user sees based on system- or user-derived input and commands. It is generally powered by one or more application servers, which act as middleware between the browser-based front-end display and the back-end databases. It is processed similar to static pages, with two extra

steps added between the browser requesting the page and it sending back the HTML to the viewer:

A user requests the page by typing its URL in a browser.

1. The browser requests the page from the web server.
2. **The web server, alerted by a unique file extension, passes the request on to an application server for processing.**
3. **The application server processes the request and returns HTML back to the web server.**
4. The web server sends the resulting HTML page to the browser.
5. The browser interprets the HTML and displays the page to the user.

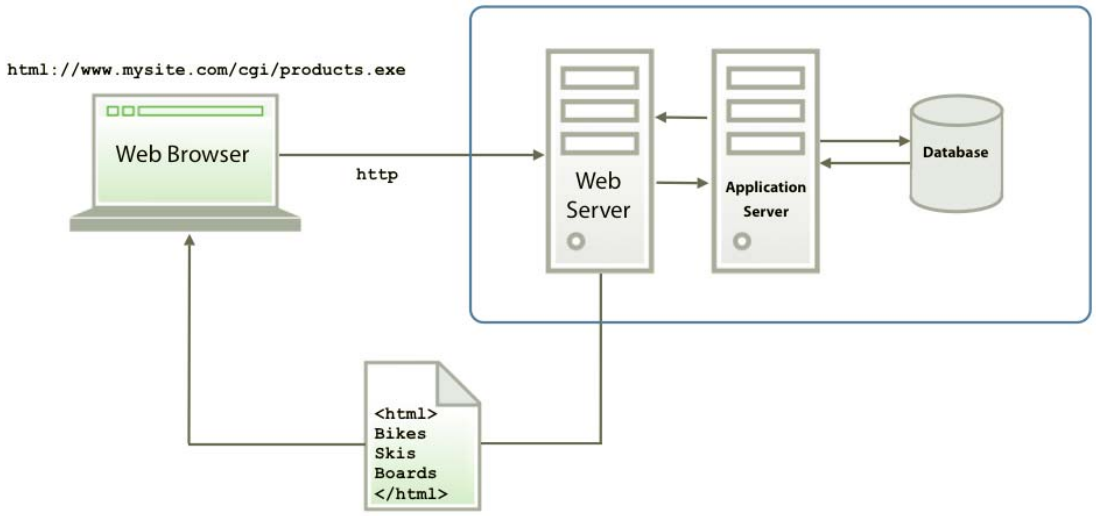

Figure 2: Serving up dynamic web pages

At its most basic, ColdFusion development consists of a single server that holds the ColdFusion application server, the web server, and the database. For best practices, however, the database should reside on a separate, standalone server. In a production environment, there should also be separate servers for development, testing, and production (i.e. up to six machines in all, or 1-2 using virtual machines).

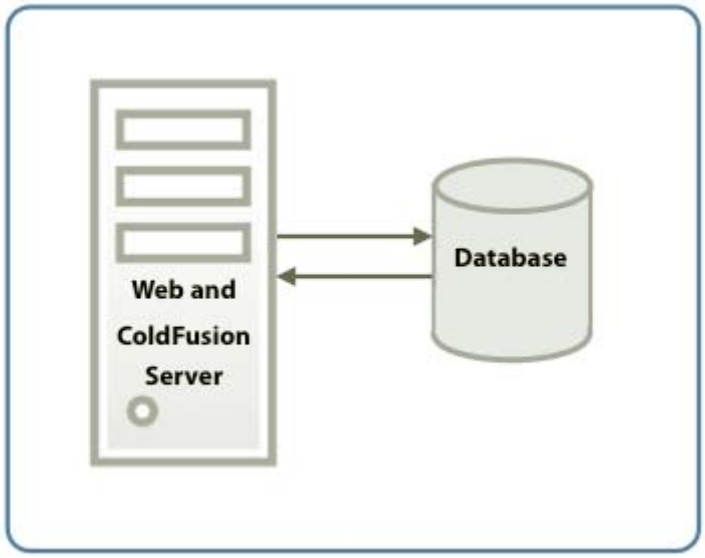

Figure 3: Ideal ColdFusion environment has separate database server

ColdFusion and CFML

ColdFusion, currently at version 8, is one such application server. Its pages have either a .cfm or a .cfc extension, which alerts the web server to send pages to it for processing. When the ColdFusion server is done processing, it sends the generated HTML back to the web server, which in turn sends it on to the browser.

In order to process pages, ColdFusion pages use ColdFusion Markup Language (CFML), a special tag-based markup language that can be embedded directly within HTML code to give the server commands and thus create dynamic pages. There is a large library of CFML tags available for you to use to give the server a variety of processing commands.

All CFML tags have a similar structure:

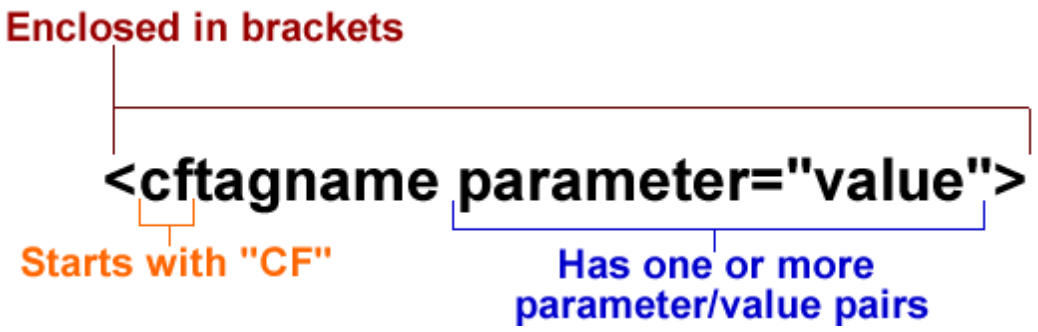

Figure 4: Features of a CFML tag

Combined with the ColdFusion server, CFML not only makes web application development faster and easier than with most other languages, but it also offers the developer a lot of power to perform a wide range of actions. Some features available in ColdFusion 8 include:

- Dynamic PDF generation
- Structured business reports
- Flash forms and controls
- XML forms
- AJAX forms and controls
- Charting engine
- Ready-to-use chart styles
- Improved text searching
- Image manipulation
- Interactive debugger for Eclipse users
- Form and parameter validation
- Integrated NT domain authentication
- Cross-site scripting attack prevention
- Application event traps
- Adobe Flex integration
- Read and create RSS and Atom feeds and XML
- SMS, Lotus Sametime, XMPP, file system and CFML asynchronous gateways Extensible gateway architecture
- .NET and Microsoft Exchange Server integration
- Strong encryption libraries
- Can invoke Java server-side constructs such as JSPs, servlets and tag libraries

As ColdFusion produces HTML pages and its tags are embedded within HTML, it can also still contain any normal client-side technology such as JavaScript, ActiveX, Flash, and Cascading Style Sheets (CSS).

ColdFusion does not require any particularly development tool. Pages could even be written in NotePad, were one so inclined. Dreamweaver is the general tool recommended by Adobe, for its easy to use interface for building both the visual web pages and SQL queries, abilities to tie into the ColdFusion server through RDS for customized interactions, and built-in libraries and features designed specifically for ColdFusion development. Its current version is CS4 (aka 9)

Some other popular editing tools for development include:

- CFEclipse, a plug-in for Europa's Eclipse IDE–"an open-source framework for creating Integrated Development Environments":
    - Launched in January 2004, Eclipse as a whole one of the most popular IDEs, heavily supported by Macromedia and now Adobe as well (which is, of course, building its own code-named Bolt and in Beta testing)
    - Free, open-source, though must also have Eclipse (also free/OS)
    - No visual/WYSIWYG capability, pure hand-coding with some tag completion help and has built in SVN capabilities - can be part of the "Eclipse Gestapo"
    - Known resource hog so may not work in any usable fashion on older systems
    - http://www.cfeclipse.org/
- HomeSite+
    - The "original" and though no longer actively developed, still in heavy use
    - Was bundled with ColdFusion 5 and earlier, and with Dreamweaver MX
    - Primarily hand-coding text editor, with mild (and generally bad) WYSIWYG capabilities
    - ColdFusion Studio, built on the same engine, also still popular as it had the CF Tag library in it

Part 2: The Basics

Setting Variables

Variables are initialized and set using the <cfset> tag. You set the value equal to an expression, which can contain literal values, other variables, functions, and operators. The basic syntax is:

<cfset variableName = expression>

Examples:

Expression Types	Example
Literal values *With pure numbers, the double quotes are optional but are good to include to more explicitly show it is a literal value*	<cfset MyName = "Summer"> <cfset HasError = 0>
Variable *You do not include quotes or other special marks on variable names; the normal double pound signs are also unneeded, unless you are using double quotes*	<cfset MyName = ThisName>
Function	<cfset ShoppingCart = ArrayNew()>
Operators/Expressions	<cfset GrandTotal = SubTotal + Tax + Shipping> <cfset FlagMe = True> <cfset FullName = FirstName & " " & LastName>

As with all languages, CFML has some basic naming rules regarding variable names:

1. Variable names can only contain letters, numbers, and underscore characters; no special characters or spaces are allowed
2. A variable name must always begin with a letter
3. Variable names are case insensitive and can be written in mixed case, two common conventions:
    a. Word case: each "word" in a variable is written with an upper case, such as FirstName or MyMiddleInitial
    b. Camel Case: a variable's name is started with a lower case with additional words started with an upper case, such as firstName or myMiddleInitial.

Variable Scoping

ColdFusion recognizes several types of variables, each with its own "scope." The scope determines how long a variable exists, where its definition is saved, and when and how it can be referenced. The scopes available in ColdFusion include:

- Variable (also known as local)
- Query
- Form

- URL
- Session
- CGI
- Application

Without a declared scope, the cfset tag will put a variable in the local scope. Such variables can only be accessed within the page in which it is defined. Once the page is done processing, the variable ceases to exist. For best practices, however, you should always scope every variable to improve the performance, maintainability, and reliability of your code. This is done including the appropriate prefix when you set it.

<cfset scope.variableName = expression>

Typeless State

All ColdFusion variables are typeless-you do not generally have to identify the type of data being stored. Instead, CFML will try to cast variables into the expected data type based on the context in with they are being used. For instance, if an ampersand (&) is between two variables, then the values will be treated as strings and be concatenated. If a plus sign (+) is between two variables, then the values will be seen as numeric values and added.

If ColdFusion cannot figure out how to cast a variable, it will display an error when run. For example, the following code would throw an error, as it cannot convert a text string to a number.

```
<cfset firstName="Sue">
<cfset lastName="Smith">
<cfset fullName=Variables.firstName + Variables.lastName>
```

This next example works fine, however. Though both numbers are set with the surrounded with double-quotes, indicating a literal string, they can be converted successfully to the numeric type for performing mathematic operations on. The new value can also be converted back to a string for concatenating with the text expression.

```
<cfset theNum="40">
<cfset newNum=Variables.theNum + "25">
<cfset theString="New number is: " & Variables.newNum>
```

If you were to output theString, you would see: *New number is: 65.*

A key point to remember is that a numeric value can always be converted to a string, but a string can only be converted to a number if it is composed solely of numerals. We will go into more detail on specific types of variables as we go along.

Displaying Variables

The <cfoutput> tag and double pound signs are used to output text, that is send it to the browser for display. Content to be outputted begins with an opening tag, <cfoutput> and ends with the closing tag </cfoutput>. The ColdFusion server will process everything between those two tags in search of commands to perform.

```
<cfoutput>
      [HTML or ColdFusion processing instructions]
</cfoutput>
```

Within the <cfoutput> block, you identify variables to be evaluated by surrounding the variable name with pound signs (#). Let's look a short example:

```
<cfset firstName="Teddy">
<cfset lastName="Bear">
<cfset fullName=Variables.firstName & " " & Variables.lastName>
<cfoutput>
      Welcome, #Variables.fullName#
</cfoutput>
```

When run, this code will display the message "*Welcome, Teddy Bear*". Notice that the text "Welcome," displays literally, as it is not surrounded by pound signs. Literal text can also be moved outside the <cfoutput> block as it doesn't need to be processed.

```
      Welcome, <cfoutput>#Variables.fullName#</ cfoutput>
```

The <cfoutput> block can contain any mix of text, HTML, CSS, JavaScript, etc. However, you should avoid including the pound sign within your literal text or within your HTML/CSS (such as color code declarations), as it is a special symbol to the server. If you must use a pound sign as a pound sign, you need to double it (##) so the CFML server will know it is just a regular pound sign and not the start of a variable.

In general, you should only put code that you actually want the server to process between the <cfoutput> tags, as the more code included, the more processing ColdFusion must do to find

variables it must process. Therefore, a large block of literal text that is purely for display, but is located within a cfoutput can reduce performance.

Including Files

Using the <cfinclude> tag, you can create common visual page elements in a separate template, and insert the content wherever you need it. This template might contain a single image or an entire document. This is useful for things like common site header/footers, toolbars/menus, and any other reusable content.

The <cfinclude> tag only takes one parameter, *template*, which tells the tag the path and name of the file it should include. When ColdFusion encounters this tag, it searches for the included page at the location specific, brings it inline in the processing page, and continues processing. The syntax for this tag is:

<cfinclude template="templateFileName">

The following diagram illustrates this process. Notice that all of the code of Template.cfm is pulled into MyPage.cfm's code, and then they are executed as if they were a single page.

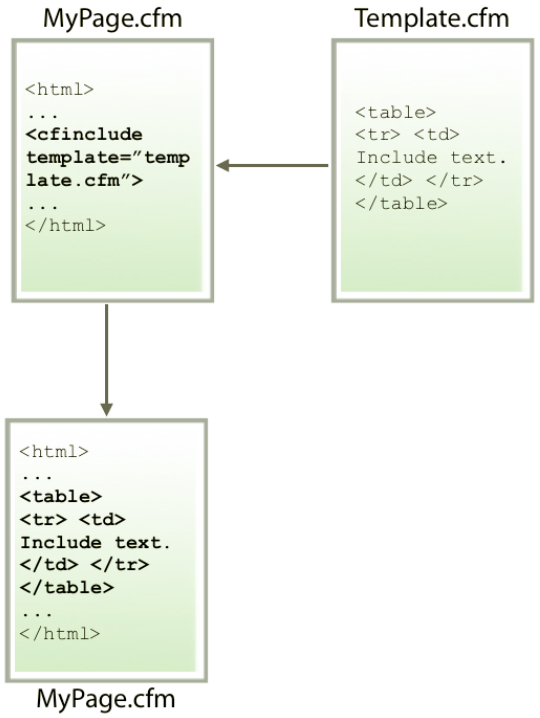

Figure 5: The including files process

Since a <cfinclude> tag includes one file inside another file, all local variables created within the calling page are also available to the included page and vice versa. With the <cfinclude> tag, the path to the file must be either a relative path or a ColdFusion mapping. Absolute paths

and URLs are not allowed. An include using a ColdFusion Mapping begins with a forward slash as in the following:

<cfinclude template="/Template_Mapping/File.cfm">

Introducing ColdFusion Functions

Functions manipulate variables in some way and produce a result, often returning a value as output. ColdFusion provides an extensive number of functions of the following types:

- Array functions
- Conversion functions
- Date and time functions
- Decision functions
- Display and formatting functions
- Dynamic evaluation functions
- Extensibility functions
- Full-text Search functions
- International functions
- List functions
- Mathematical functions
- Other functions
- Query functions
- String functions
- Structure functions
- System functions

There are many ways to call functions. When calling them within a cfoutput, notice that the pound signs surround the entire expression.

```
#Now()# - returns the current date and time as a timestamp
#Abs(-10)# - returns the absolute value of the value given
```

Functions can also be nested, like here to format the current date in a nicer format:

```
#DateFormat(Now())#
```

Some functions take multiple arguments. The following example takes a variable value and replaces every instance of the word California with CA.

```
#Replace(Variables.State_List, "CA", "California", "ALL")#
```

Code Commenting

It is always "Best Practices" to include comments in your code and to include an appropriate amount of detail in those comments. This helps ensure your applications are easy to maintain, and if you get hit by a bus tomorrow, makes it easier for someone to come behind you and figure out what you did. ColdFusion comments are very similar to HTML comments, except for the use of three dashes instead of two.

<!-- This is an HTML comment -->

<!--- This is a CFML comment --->

The benefit of using ColdFusion comments over HTML comments is that ColdFusion comments are not sent back to the browser, keeping them hidden from the end users (and hackers), but available to those who can access the code itself.

It is generally encouraged that you liberally comment your pages while developing, and that you include a comment block at the top of every page created that describes the purpose of the code, the author's name, the date created and any modification information or important notes. One recommended comment structure is shown
below:

```
<!---//----------------------------------------
Name: [page name here]
Author: [author name here]
Created: [today's date here]
Update:
Copyright: (c) 2009 Art Gallery
Purpose: [Purpose here]
----------------------------------------//--->
```

Note that after the <!--- the remaining dashes and slashes are seen purely as part of the comment.

Part 3: Interacting With the Database

One of ColdFusion's most powerful, and most used, features is its database connectivity. You can connect to and manipulate data from a wide range of databases using CFML. Best of all, unlike some other languages, the code is practically the same for almost all databases, as the unique database information is declared in the ColdFusion administrator when you set up a datasource rather than in your code.

Connecting to a database

A relational database is basically a structured collection of related data. This data is organized into tables with rows and columns of information. Each row of a table should have a unique identifier, referred to as a primary key, which is used to relate the tables to one another.

In order to access a database, you must first set up a data source in the ColdFusion administrator. Data sources are connection profiles and to tell ColdFusion how to find and access your databases. Each datasource will usually include the database server name, database name, and login information for the specific database user ColdFusion server should access the database as. For best practices, your database user should have read and update writes for tables, but should not the ability to create/delete tables or databases. Should your web application ever be hacked, this can at least keep hackers from destroying your database structure, though of course they would still be able to clear data.

ColdFusion communicates with these datasources through Java Database Connectivity (JDBC)–a standard application programming interface (API) for accessing information from different database systems and different storage formats. Once you have your datasource set up in the ColdFusion administrator, you can use Structured Query Language (SQL) queries to extract data from the database, or insert and manipulate data within it. The database driver converts the SQL to the appropriate native language of the database.

Querying the Database

To "talk" to a relational database, you must speak its language: Structured Query Language aka SQL ("see-quel") Using SQL you can SELECT data, INSERT data, UPDATE data, or DELETE data.

The SQL SELECT statement does just that, it "selects" or retrieves data from database tables. The basic syntax is:

```
SELECT {COLUMN LIST}
FROM {TABLE LIST}
WHERE {JOIN CONDITIONS AND FILTERS}
ORDER BY {COLUMN LIST}
```

Example:

```
SELECT ArtistID, FirstName, LastName
FROM Artists
ORDER BY LastNp0me
```

This statement returns three columns: ArtistID, FirstName and LastName, from a table called "Artists". As it does not have a where statement, it will pull all available rows, and then order them alphabetically by LastName.

To get ColdFusion to send SQL commands to the database, you wrap the query the <cfquery> tag:

```
<cfquery datasource="CF8introArtGallerySolution" name="qArtist">
      SELECT ArtistID, FirstName, LastName
      FROM Artists
      ORDER BY LastName
</cfquery>
```

In this instance we are using two of the most commonly used attributes of the cfquery tag. The *datasource* attribute, which tells the server which data source we are referring to, and the *name* attribute, which gives the result set returned by the query a name. Once the query has run, the result set is stored in the server's memory, and the name helps to identify it. You can choose any name you like for your query, within the same basic variable naming limitations noted earlier. Some feel it is a good practice to start your query names with a prefix of q.

When using SELECT statement, you may want to prefix the column names with the database table names for clarity, as relational databases often have the same column name used in multiple tables. For simple selects, it isn't required, however when we move to more complex selects that join multiple tables, it can be critical.

```
<cfquery datasource="CF8introArtGallerySolution" name="qArtist">
      SELECT Artists.ArtistID, Artists.FirstName, Artists.LastName
      FROM Artists
      ORDER BY LastName
</cfquery>
```

When you do this prefixing, the table names are stripped from the column names in the result sets. You only reference the column names themselves. The select statement here, run on our test database, would return this recordset:

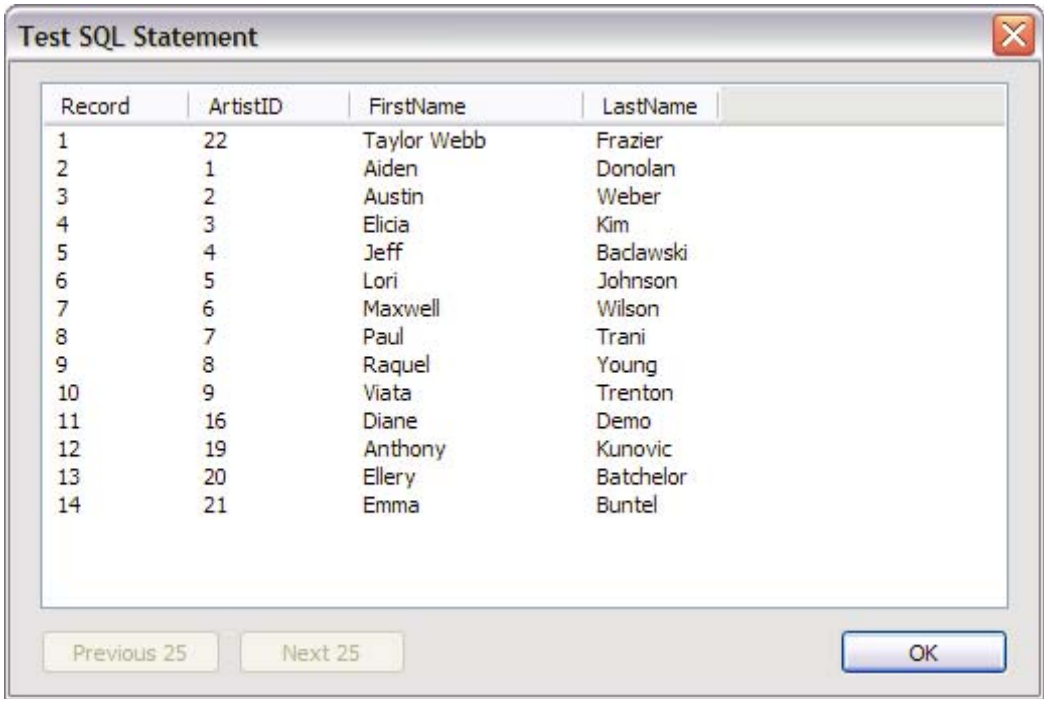

Figure 6: Example record set

To deal with possible duplicate column names, or to "correct" column names that might contain special symbols or spaces that could cause problems for ColdFusion, you can rename columns in a select statement using the AS keyword.

```
<cfquery datasource="CF8introArtGallerySolution" name="qArtist">
     SELECT ArtistID AS AID, FirstName, LastName
     FROM Artists
     ORDER BY LastName
</cfquery>
```

In this example, the ArtistID column is renamed to AID. In the resulting query set, the column would be referred to as AID. These are called column aliases.

Seeing the Query Results

The <cfdump> tag will let you quickly view the content of any variable of any time, including resultset data. Its syntax is very simple:

```
<cfdump var="#variableName#" />
```

Note that you MUST use pound signs around the variable name, or it will treat it as text. Let's look at the dump of our query above:

query

RESULTSET				
	query			
		CITY	FIRSTNAME	LASTNAME
	1	Santa Fe	Taylor Webb	Frazier
	2	Denver	Aiden	Donolan
	3	Berkeley	Austin	Weber
	4	Los Angeles	Elicia	Kim
	5	Hollywood	Jeff	Baclawski
	6	Pierre	Lori	Johnson
	7	Tulsa	Maxwell	Wilson
	8	New York	Paul	Trani
	9	Atlanta	Raquel	Young
	10	New York	Viata	Trenton
	11	Denver	Diane	Demo
	12	Aspen	Anthony	Kunovic
	13	Washington	Ellery	Batchelor
	14	Washington	Emma	Buntel
CACHED	false			
EXECUTIONTIME	16			
SQL	SELECT FirstName, LastName, Artists.City FROM Artists			

Figure 7: A Record Set Dump

The dump tag will automatically format the content in a structured method. Notice that it contains:

- the resultset
- . whether the query is cached
- . the execution time of the query
- . the SQL statement that was executed

However, beyond the results themselves, the only one of these items you can also access as a variable is ExecutionTime. This variable will display the amount of time spent executing and returning the data generated by the query. You can access it using #cfquery.ExecutionTime#. This will display the time for the last query executed on that page.

Once you've run your query and you have a record set, you can also print out the values within a <cfoutput> block. Remember, <cfoutput> is used to display any type of dynamic data, including query results. Displaying query variables is similar to displaying local variables. You surround the dynamic code with a <cfoutput> block and add pound signs around the individual

variable names. You must also make sure to let it know that you are outputting from a query. One way of doing this, if you only have one record to show, is simply scoping.

```
<cfquery datasource="CF8introArtGallerySolution" name="qArtist">
      SELECT Artist.ArtistID, Artist.FirstName, Artist.LastName
      FROM Artist
      ORDER BY LastName
</cfquery>
<cfoutput>
      #qArtist.ArtistID#
      #qArtist.FirstName#
      #qArtist.LastName#
</cfoutput>
```

For our query, this will display “22 Taylor Webb Frasier” which is the first row of the record set. To show all rows returned, add the query attribute to the <cfoutput> tag. The syntax is:

```
<cfoutput query="queryName">
...
</cfoutput>
```

Adding the query attribute lets ColdFusion known it needs to perform an iterative loop that executes once for every row in the named query. You can also add some HTML to break up the output some and avoid just having a long string of text, like this:

```
<cfoutput query="qArtist">
      #qArtists.ArtistID#
      #qArtists.FirstName#
      #qArtists.LastName#<br />
</cfoutput>
```

Running that code would give us this:

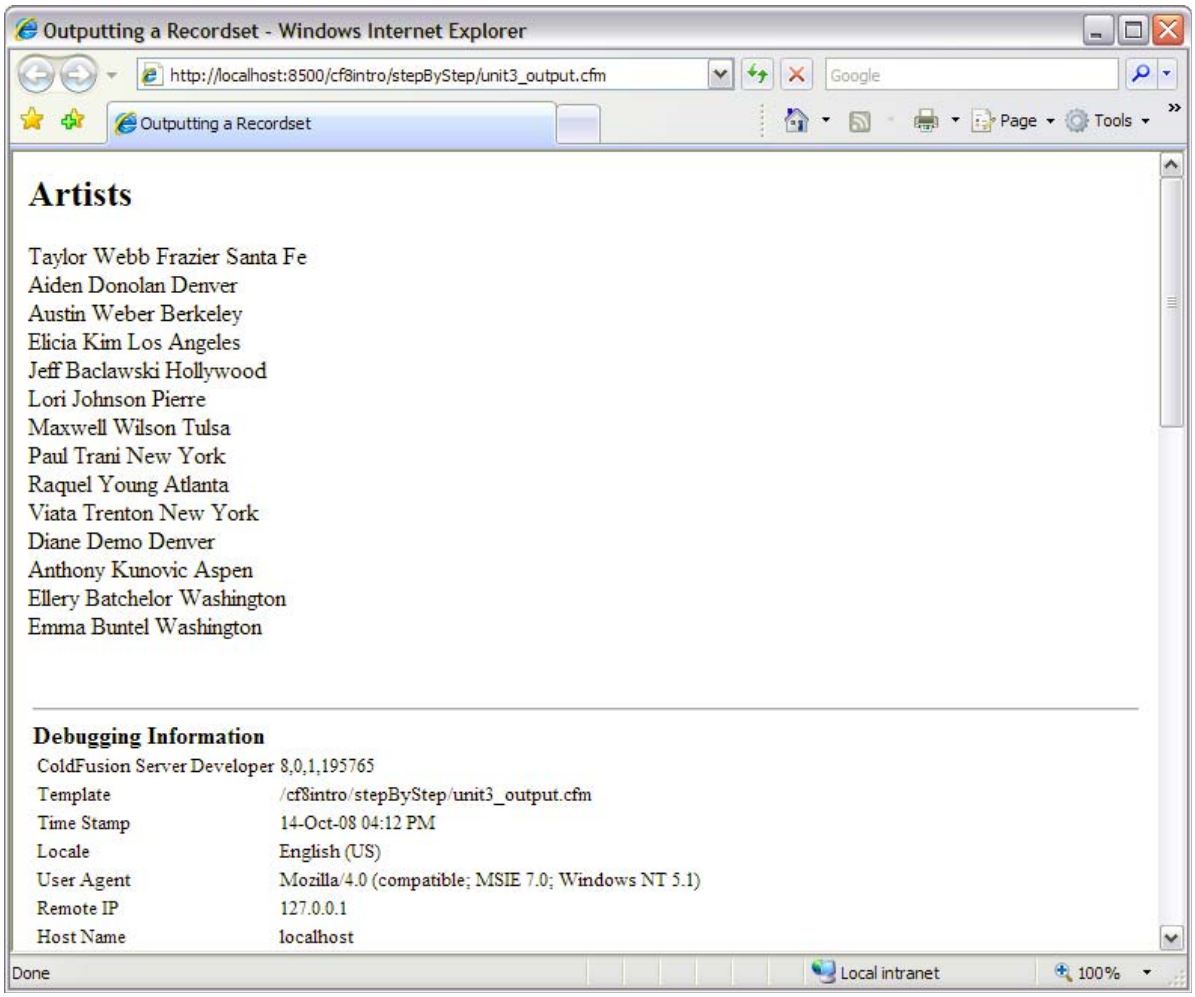

Figure 8: Displaying all qArtists data results

Of course, that isn't very pretty either, so how about really adding in the HTML to format our data into a nice table:

```
<table border="2" cellpadding="3">
<tr>
	<th>ArtistID</th>
	<th>FirstName</th>
	<th>LastName</th>
</tr>
<cfoutput query="qArtist">
	<tr>
		<td>#qArtist.ArtistID#</td>
```

```
            <td>#qArtist.FirstName#</td>
            <td>#qArtist.LastName#</td>
        </tr>
    </cfoutput>
    </table>
```

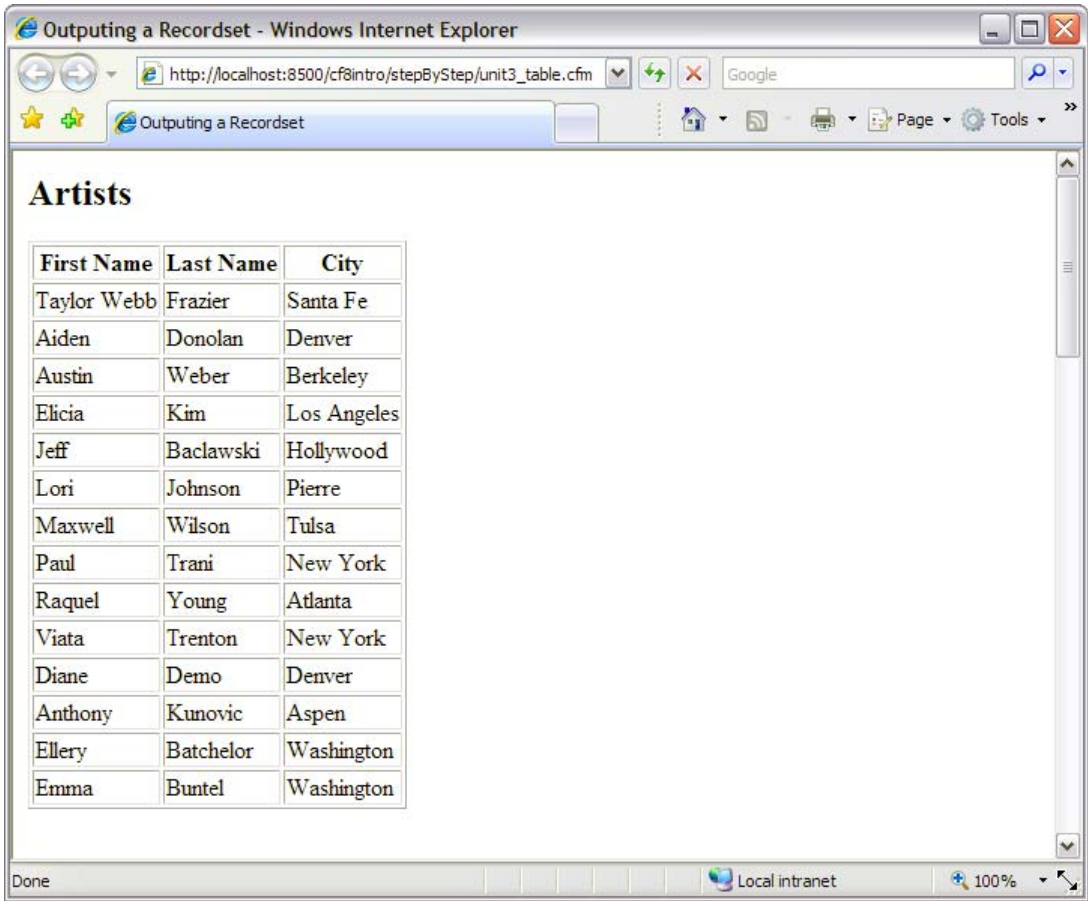

Figure 9: Dynamic table set

When you create a query using the <cfquery> tag, you can also define the result attribute, which names a predefined result set structure.

```
    <cfquery name="qArtist" datasource="CF8introArtGallerySolution"
    result="resultInfo">
        SELECT FirstName, LastName, Artists.City
        FROM Artists
    </cfquery>
```

Use the <cfdump> tag to view the contents of this structure.

```
    <cfdump var="#resultInfo#" />
```

struct	
CACHED	false
COLUMNLIST	CITY,FIRSTNAME,LASTNAME
EXECUTIONTIME	16
RECORDCOUNT	14
SQL	SELECT FirstName, LastName, Artists.City FROM Artists

Figure 10: cfdump of resultinfo

The record set structure contains several variables, each of which can be accessed using #resultinfo.VariableName#:

Cached

This is a simple Boolean value that denotes whether the query is cached or not.

ColumnList

Lists the column names in the result set in a comma-separated list

ExecutionTime

This is the same as the cfquery.ExecutionTime variable noted earlier; it displays the amount of time spent executing and returning the data generated by the query. Unlike the cfquery value, however, this one is specific to the result set rather than being just the one for the last query run.

RecordCount

Contains the total number of rows, or records, returned by the query

SQL

Gives you the actual SQL statement that was run by the query

Another variable that is available for result sets is CurrentRow. When ColdFusion runs a query, is assigns a sequential number to each row in that query, a value stored in CurrentRow. When you are using cfoutput to loop over a result set, you can access it just like any other column in the query:

```
<cfoutput query="qName">
     Row #qName.CurrentRow#<br />
</cfoutput>
```

Figure 11: Shows RecordCount and CurrentRow being used

Best Practices Reminders

1. Your database should reside on a separate, standalone server. In a production environment, there should be separate servers for development, testing, and production
2. All of your ColdFusion variables must follow the following guidelines.
    a. Use only letters, numbers, and underscores
    b. Must begin with a letter
    c. Do not use special characters or spaces
    d. Use consistent case/format no matter which method you decide to do
    e. Use names descriptive enough for maintenance and clarity
3. Always scope every variable using the appropriate prefix
4. Always comment your code for easier maintenance.
5. When creating your databases, never use spaces or special characters in the names of tables, columns, etc; if you do have problematic column names, use SQL aliases to rename them for use in your application
6. Determine a naming convention for your variables, including query names, and use it consistently