Fast Track to Web / Database Applications for Programmers

A Comprehensive Guide to Web Development Using Microsoft IIS, SQL Server, and Macromedia ColdFusion MX

David M. Chandler

TurboManage Corp.

February 2000

David M. Chandler
109 Lanyard Bend
Peachtree City, GA 30269

Unit 1
Getting Started with Web Applications

Unit Overview

Web programming with modern tools is fun and productive. In this unit, you will get your feet wet in a typical Web applications environment using Microsoft Internet Information Server (IIS), Microsoft SQL Server, and ColdFusion.

Goals

Upon completion of this unit, you should be able to

- Make intelligent choices regarding Web tools and platforms
- Configure IIS for a typical Web application
- Create databases and run queries in SQL Server
- Feel comfortable working with ColdFusion Studio
- Write simple ColdFusion applications

Lesson 1
Web Development Background

Introduction

In five short years, Web technology has emerged as the leading paradigm for network applications development. A brief survey of current development methodologies will help you better understand which tools and languages are well-suited to various types of applications.

Objectives

By the end of this lesson, you should be able to

- Understand the nature and limitations of server-side Web applications
- Explain the advantages and disadvantages of the CGI, API, application server, and Java servlet models
- Understand the n-tier model of applications development
- Explain the distinctions between front-end and back-end Web programming
- Make informed choices regarding operating systems, application servers, and development tools

In the Beginning…

There was the HyperText Markup Language, or HTML. The key features of HTML were

1) content is separate from presentation
2) hyperlinks

HTML is static. In fact, to build an intranet of static HTML pages, you do not even need a Web server—just a network drive.

Web servers offer two principal advantages over a network drive:

1) speed
2) no requirement to log in
3) ability to process dynamic pages

How the Web Works

Web browsers and servers speak the HyperText Transfer Protocol (HTTP). HTTP is

1) lightweight
2) stateless (no permanent connection is established)
3) request-response oriented

HTTP supports various *methods*. The most familiar is the GET method, which retrieves a document. The POST method is commonly used to post form data, and the PUT method is used to place files on the server (Web servers and browsers have not implemented PUT until recently with the introduction of WebDAV in IE 5 and IIS 5).

Demonstration 1-1 HTTP GET

1. Telnet to localhost port 80
2. Type “`GET /`” and hit RETURN twice
3. Notice the MIME type

The CGI Model

At the heart of the Web's early interactive capabilities was the Common Gateway Interface, or CGI. The CGI mechanism is relatively simple: the browser encodes data from fill-in forms into an HTTP GET or POST request and sends it to the server. The server looks at the URL request and determines what to send back based on the filename.

If the file extension is HTML or another static document type, the server READS the requested file and sends it to the browser.

If the file extension is CGI or some other configured application type, the server EXECUTES the requested file and returns the output to the browser. In this case, the CGI program specifies the MIME type of the output so the browser will know what to do with it.

In the CGI model, the server makes the HTTP request available to the CGI program through server environment variables. Frequently-used environment variables include

SCRIPT_NAME
REMOTE_HOST
REMOTE_ADDR
REMOTE_USER
CONTENT_LENGTH
QUERY_STRING

For a complete list of environment variables available for CGI programming, check your Web server's documentation.

CGI programs are relatively hard to write because

1) CGI programs must parse environment variables to get form data from the browser (although there are libraries of CGI functions for common languages).
2) Every line of output must come from a print statement (perl's "here document" feature helps).
3) Connecting to databases from native languages is relatively complex (perl's DBI module eases some of the pain).

In addition, the CGI model has several disadvantages:

1) Every request forks a new process.
2) It is impossible to maintain browser state.
3) It is impossible to maintain database connections.

The API Model

In order to address these limitations, the major Web server vendors created APIs which allow a Web “program” to reside in process with the Web server. Thus, the overhead of forking a new process is replaced by the lower overhead of making a function call in a dynamically-linked library. Database connections remain open once initialized, and applications can store user information in memory.

However, the API model has one glaring problem: if your application crashes, the whole Web server crashes. Even without this risk, writing Web applications as functions in DLLs requires significant programming skill. The API model is used primarily

1) when you want to replace a Web server’s built-in function (say, browser authentication or logging) with your own
2) to implement scripting engines like Active Server Pages or ColdFusion

Server-Side Includes

One of the earliest methods of making Web pages more dynamic was called server-side includes. Server-side includes are special instructions to the server which are embedded in HTML pages. These instructions allow you to display the current date, include the contents of another file, or even include the output of a program executed on the Web server. For example, the following HTML snippet demonstrates a server-side include to display the current date on the page:

```
<HTML>
<TITLE>SSI Demonstration</TITLE>
<BODY>
This page demonstrates server-side includes.
<!-- #echo var="DATE_LOCAL" -->
</BODY>
</HTML>
```

When a document contains server-side includes, the server must read the document and insert the included information first. This is called *parsing*. When a document ending in a server-side include extension is requested, the server parses it before sending it back to the browser.

Typically, documents containing server-side includes are named .shtml, .shtm, or .stm (IIS). These extensions alert the Web server to parse the file first.

While server-side includes are simple to use, they only support a handful of operations and are not intended to be used as a programming language. They may be useful for simple tasks like including a standard footer in every document if no other dynamic features are required, but are otherwise rarely used.

Caution

Server-side includes are potentially dangerous because the `#exec` directive allows arbitrary programs to be executed on the server.

Server-side includes are important because they are the predecessor of the full-fledged scripting languages now implemented in application servers.

Application Servers

Web application servers solve all the aforementioned problems. The application server itself runs in a separate process and communicates with the Web server using the lightweight API model. The actual Web programs are written in the application server scripting language. Application servers provide capabilities not otherwise available such as debugging and database connection pooling.

Application servers support server-side scripting languages, which are typically easier to learn than native languages and provide a number of important features:

1) No URL parsing is required.
2) Database access is much simpler.
3) Scripts run in the context of an application framework with persistent variables to overcome statelessness.
4) Scripts can easily call pre-built routines to do common things like send mail or receive an uploaded file.

Microsoft Jscript and VBScript are object-oriented server-side scripting languages used to create Active Server Pages. ASP scripting is easier to learn than writing C functions in DLLs and not as likely to crash the Web server (although it is possible to introduce memory leaks).

The real power of application server scripts is that you embed scripts right in Web pages. For most people, this is easier than writing an entire program to create a dynamic Web page. Here is a typical ASP script to query a database and display a list of names:

```
<%
'Establish a connection with data source.
strConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Data\Employees.mdb"
Set cnn = Server.CreateObject("ADODB.Connection")
cnn.Open strConnectionString

'Instantiate a Recordset object.
Set rstCustomers =
Server.CreateObject("ADODB.Recordset")

'Open a recordset using the Open method
'and use the connection established by the Connection
object.
strSQL = "SELECT FirstName, LastName FROM Customers
WHERE LastName = 'Smith' "
rstCustomers.Open strSQL, cnn

'Cycle through record set and display the results
'and increment record position with MoveNext method.
Set objFirstName = rstCustomers("FirstName")
Set objLastName = rstCustomers("LastName")
Do Until rstCustomers.EOF
     Response.Write objFirstName & " " & objLastName &
     "<BR>" rstCustomers.MoveNext
Loop
%>
```

ColdFusion is tag-oriented rather than program-oriented. As a result, ColdFusion code is generally easier to create and maintain than ASP. Here is the same query in ColdFusion:

```
<CFQUERY name=”qListSmiths”
      datasource=”Employees”>
SELECT FirstName, LastName
      FROM Customers
      WHERE LastName = ‘Smith’
</CFQUERY>

<CFOUTPUT query=”qListSmiths”>
#qListSmiths.FirstName#
#qListSmiths.LastName#<BR>
</CFOUTPUT>
```

The n-Tier Model

The application server model is frequently referred to as the three-tier or n-tier model. The main components are

1) Database layer
2) Business logic layer (application server)
3) Presentation layer (browser)

The primary benefit of the tiered development approach is that it is possible to run each piece on a separate server, thereby harnessing the power of parallel computing. In addition, the n-tier model simplifies application development effort by separating the three layers.

Java Servlets and Java Server Pages (JSP)

The newest technology for developing Web applications is Java servlets. As the name implies, each servlet is a miniature server. The servlet stays in memory waiting for a request and then processes the request. Servlets are written in Java and offer high performance; however, by themselves they offer none of the Web application framework inherent in scripting languages like ASP and ColdFusion.

Writing Java servlets in native Java is even more difficult than writing CGI programs because servlets must handle the low-level HTTP request-response sequence. However, servlets can also be created with JavaServer Pages (JSP).

JavaServer Pages consist of snippets of Java code embedded between script tags in an HTML document. When a .jsp page is requested, a server such as Allaire's JRun compiles the JSP page into a Java servlet and executes the resulting code. JSP pages provide an application framework similar to ASP or ColdFusion. However, JSP, like ASP, still requires more lines of code to do any given task than a tag-oriented language like ColdFusion.

Enterprise Web Development Trends

Web application servers and scripting languages are ideal for Web application front-end programming. However, complex applications like e-commerce systems frequently require interaction with legacy databases or other running applications. In addition, some applications require native language features not available in higher-level scripting languages. Since back-end functionality will never see the light of day in a Web browser, it probably doesn't make sense to use a Web scripting language for these purposes.

Data-intensive tasks can be done in the database using stored procedures, as this approach offers performance advantages and programming constructs such as cursors and transactions which are most appropriate for data-oriented tasks.

Transaction-oriented applications typically require a transaction monitor or some sort of middleware which ensures end-to-end data integrity. A popular middleware object server is WebLogic, which hosts Enterprise JavaBeans (EJB) components. Among other things, object servers like WebLogic map relational database tables into Java objects, making it relatively easy to extract and process data from heterogeneous sources. Java servlets and JSP would not be appropriate as back-end tools because they are request-response oriented.

Because of the requirement to interface with middleware or legacy systems, back-end programming is typically done in a lower level language like Visual Basic, C++, or Java. Java has the most performance overhead, but offers a much richer set of basic data types (such as hashes and linked lists) than C++, is easier to use for TCP/IP-based communication, and its object orientation is less complex than that of C++. In addition, the cross-platform nature of Java means you won't have as much of the Windows programming learning curve. For this reason, Java is rapidly becoming the most popular language for Web back-end development.

Front-End and Back-End Integration

The Web front end will more than likely require a way to tap into certain back-end functionality. The ColdFusion Application Server provides a variety of ways to do this. ColdFusion programs can instantiate CORBA and COM objects. CORBA objects require a CORBA middleware server. COM objects are simply DLLs hosted on the server. With version 4.5, ColdFusion programs can also instantiate Enterprise JavaBeans (EJB) objects hosted by any major EJB server and call Java servlets hosted by a servlet engine such as Allaire's JRun. In addition, you can create custom ColdFusion tags in Java.

Large-scale applications typically wrap up core functionality into CORBA, COM, or EJB containers so that they can be called from front-end and back-end programs alike. Of these technologies, EJB is the most in vogue. Alternatively, Visual Basic and Visual C++ both make it relatively easy to create COM objects.

Choices, Choices

Which is the best platform for your Web application? The answer depends largely on

1) The nature of your application (data-intensive, transaction-based, real-time).
2) Your organization's experience with various languages and operating systems.
3) The need for scalability.

Operating Systems

Unix is a highly robust, scalable applications platform. However, as companies have adopted Windows NT as their network operating system, Windows NT has become more attractive as an applications platform for several reasons:

1) The Intel-based hardware required to run Windows NT is relatively inexpensive compared to Unix vendors' RISC-based solutions.
2) The software tools investment required to develop in the Windows NT environment is significantly less than that required in a Unix environment.
3) Unix administration expertise is hard to find, and is difficult to justify for a single application server.
4) With version 4.0, Windows NT has emerged as a robust platform for application servers.
5) With proper application design and load balancing software, NT is highly scalable for Web-based applications.

The preferred platform for high-end applications is still Unix, however, for a variety of reasons:

1) Unix servers can be run without the overhead of a windowing environment, thereby increasing performance.
2) Unix is famously robust.
3) Because much Unix development has taken place in the open source community, it is a more secure environment than NT for Web applications.

Web Servers

On NT, it’s IIS, Apache, or Netscape. Both Netscape and Microsoft have higher-end servers for catalog publishing and e-commerce applications. Microsoft Site Server is a nice tool for managing the process of authoring and deploying content to server farms.

On Unix, Apache and Netscape are running the show. Apache (www.apache.org) is open source, free, reliable, and very popular. For content management capabilities or GUI administration, though, you’ll have to look to Netscape.

Application Servers / Scripting Languages

The most popular commercial application server environments are ColdFusion and Active Server Pages. In the open source world, the most popular environments are PHP3, ZOPE (based on the Python language), and Apache’s mod_perl.

The only commercial cross-platform application server is ColdFusion, which runs on NT, Solaris, and Linux.

Development Tools

The leading Web applications development tools are Microsoft Visual Interdev, Allaire’s ColdFusion Studio, and Macromedia Drumbeat.

Visual Interdev and Drumbeat are used to create ASP applications, whereas ColdFusion Studio is used primarily to create ColdFusion applications.

As you will come to appreciate, ColdFusion Studio is an extremely rich and powerful Integrated Development Environment (IDE).

Where to Go from Here

Online Resources

Check out the MSDN Library at msdn.microsoft.com. This is an extremely useful site. The Web Workshop at msdn.microsoft.com contains a plethora of reference materials and tutorials on the Web a la Microsoft.

You can find Netscape's version of the world at developer.netscape.com.

The Web Building category at www.cnet.com is chock full of great tutorials and reviews. This is a great place to keep up with what's current.

Books

CGI Programming on the World-Wide Web (O'Reilly) is now out of print, but you can find it online at www.oreilly.com/openbook. This is still a fantastic resource for CGI programming, both as an excellent tutorial and a CGI cookbook. Most examples are written in perl.

Dynamic HTML: The Definitive Reference (O'Reilly) is an indispensable compendium for Web content developers. It contains complete reference material for all of the HTML tags, CSS style attributes, browser document objects, and JavaScript objects supported up to version 4 of Netscape Navigator and Microsoft Internet Explorer.

Apache: The Definitive Guide, 2nd Edition (O'Reilly). Like it sounds.

O'Reilly has a ton of other Web books in the Web & Internet Resource Center at www.oreilly.com.

Lesson 2
Introducing IIS

Introduction

Microsoft Internet Information Server is an easy-to-learn and powerful Web server. In this lesson, you will learn the basics of setting up Web sites in IIS.

Objectives

By the end of this lesson, you should be able to

- Understand the difference between Web sites and virtual directories
- Create a new Web site
- Create a new virtual directory
- Understand shares, redirects, and directory indexing
- Enable default documents
- Configure directory security
- Monitor performance

How to Get It

Microsoft Internet Information Server is a popular and free Web server for the Windows NT operating system. IIS 4 is part of the NT Option Pack, which ships with Windows NT 4. You can also download it from Microsoft's Web site.

IIS runs on Windows NT Workstation or Windows NT Server. On NT Workstation, it is called Personal Web Server, but is identical to the NT Server version with one important difference: Personal Web Server can handle only ten simultaneous connections. Microsoft built in this artificial limitation to prevent people from using the cheaper Windows NT Workstation as a production Web server.

There is also a stripped-down version for Windows 95/98 called Peer Web Services. Peer Web Services does not have the Internet Service Manager and is not intended for production use.

Tip

When you install NT Option Pack, make sure you select the custom installation and check the Internet Management Console. Some versions of Option Pack do not install this by default.

Getting Started

You can start and stop IIS from one of two places:

1) Control Panel | Services | World Wide Web Publishing Service. This is the place to configure automatic or manual startup.
2) Internet Services Manager. The IIS Admin service must be running in order to access the Internet Services Manager.

What Is a Web Site?

In Microsoft lingo, a Web site is a collection of resources which can be accessed through a single IP address and port. The default Web site runs on port 80. IIS allows you to configure multiple Web sites, each running on its own address-port combination.

Walkthrough 2-1 Exploring IIS

Setup

In this walkthrough, we'll explore the Internet Services Manager. You must have previously installed Internet Information Server from the NT Option Pack and chosen to install the Internet Services Manager during the installation.

Steps

1. Open Internet Service Manager.
2. Expand the icons down to the virtual directory level.
3. Right-click on the computer and select Properties.
4. Explore the MIME types and WWW Service properties.
5. Notice that the IP address and port boxes are grayed out.

What Is a Virtual Directory?

A virtual directory is a mapping between a URL root path (`http://127.0.0.1/somewhere`) and a physical directory (`C:\Inetpub\Wwwroot`). A virtual directory can also be a redirect mapping to point to another URL on the same or another Web server. A Web site can have multiple virtual directories, and there is no need for the directories to be related or even on the same server.

To access a subdirectory of a virtual directory, simply append the relative directory path to the URL. For example, if

http://localhost/myweb **è** `D:\MyWeb`, then

http://localhost/myweb/sandbox/`play.htm` **è** `D:\MyWeb\sandbox\play.htm`.

Tip

On Windows NT, URLs and filenames are not case-sensitive. However, on Unix, they are. On Unix, the above example would not work because myweb and MyWeb are of different case. Also note that virtual directories cannot contain spaces.

Understanding Hierarchical Properties

In IIS, properties are automatically inherited at each level from the parent container. If you set a property at the parent level which overrides a property further down in the hierarchy, you will get a warning which asks you how you want to resolve the conflict.

IIS lets you configure the properties of any virtual directory or subdirectory individually. This is very handy for restricting access to directories or directory trees on an individual basis.

Walkthrough 2-2 Create a New Virtual Directory in Internet Service Manager

Setup

In this walkthrough, we'll create a new virtual directory under the default Web site.

Steps

1. In Internet Service Manager, click on Default Web Site.
2. Right-click on New | Virtual Directory.
3. Type "wft" and click Next.
4. Find the WebFastTrack directory which was created when you installed the courseware. Click Next.
5. Check the Read and Script boxes (not Execute!) and click Finish.
6. Right-click on the newly-created virtual directory and explore Properties.

Walkthrough 2-3 Create a New Virtual Directory in Windows Explorer

Setup

One of the most convenient features of IIS is the ability to create a new virtual directory right from Windows Explorer.

Steps

1. Open Windows Explorer and navigate to the WebFastTrack directory.
2. Right-click on the directory and click Sharing.
3. Click the Web Sharing tab. You should see the alias (virtual directory) you just created in Internet Service Manager.

Directory Options

IIS provides a variety of powerful options for building production Web sites. These are discussed briefly in conjunction as we walk through the directory properties interface accessed by right-clicking on any virtual directory.

Virtual Directory Settings

Use this tab to set up a redirect URL, directory permissions (not to be confused with NT access control), and application settings (primarily ASP settings). Default directory permissions and application settings are also configurable at the site and computer level.

Directory permissions are straightforward. Note that you can turn off logging. This is especially useful for directories containing images, as there is no reason to log image hits.

The “Directory browsing allowed” checkbox allows users to see a listing of all the files in a directory. This is not recommended for production sites.

The “Index this directory” checkbox works with Microsoft Index Server to create a full-text index of the directory contents. This is especially useful for intranet or Internet search applications, and is only available on Windows NT Server.

Did You Know?

Besides HTML and PDF files, Microsoft Index Server will also work with most popular document formats (Word, Excel, PowerPoint, etc.)

Redirects

Redirects deserve a bit of attention because the language is somewhat confusing.

"The exact URL entered above"

This checkbox controls whether the redirect is to a specific URL or to a URL path.

For example, suppose you create a virtual directory named "stuff" which maps to http://localhost/newstuff. Now suppose the browser requests /stuff/mystuff.htm.

If the box is checked, the browser will be sent to http://localhost/newstuff/.

If the box is not checked, the browser will be sent to http://localhost/newstuff/mystuff.htm.

If you've moved a tree of files to another directory, don't check the box. If there is no longer any correlation between the old and new filenames, or you want to send browsers to another site, check the box.

"A directory below this one"

This checkbox is used in the special case where you want to redirect to a physical directory beneath the present physical directory (which is the directory you specified before you clicked the radio button to make this virtual directory a redirect). It's a little confusing because the physical directory doesn't show up anywhere, but it's still there.

Caution

If you don't check this box and you redirect to a subdirectory, you will generate an infinite loop of redirects. This can be an interesting way to test server load capacity.

"A permanent redirection for this resource"

HTTP makes a distinction between a permanent and temporary redirect. This checkbox controls which header gets sent; however, it is unlikely that many browsers or proxy servers actually use this information.

Documents

Use this tab to enable a default document and/or document footer.

A default document is the resource which will be returned if the URL specifies only a directory path.

A document footer is appended to each file as it is sent to the browser.

Design Tip

When using a directory path in a URL to reference a directory index or default document, you should always include the trailing slash. Otherwise, the Web server will look for a file with the directory name and send a redirect to the browser with the trailing slash. This causes an unnecessary round trip to the server.

Directory Security

Directory security can be configured three ways.

Anonymous Access and Authentication Control

The HTTP protocol allows the browser to supply authentication credentials with each request to the server. Typically, the first time the browser encounters a protected resource, it will pop up a password dialog box for the user. Thereafter, the browser automatically sends the same information with each request in the same URL tree.

The Anonymous Access and Authentication Control tab lets you specify users who can access content in the directory tree. Unfortunately, the only authentication choices available without the use of third-party software are anonymous and Windows NT authentication. For Internet applications which require password security, it is obviously impractical to use Windows NT authentication, so these applications must either

1) Write a custom authentication function for ISAPI,
2) Use a third-party solution like Authentix, or
3) Use a login screen which authenticates against a user database.

Anonymous access lets anyone connect to the resource. The Web server reads or executes the requested file as the user specified in this section.

Note

In order for anonymous access to work, the Web server must have permission to read files in the directory. Therefore, the Windows NT permission for the directory and files must include read access for the Web server user specified in the directory security properties. Normally, this is `IUSR_machinename`.

Basic authentication requires a username and password for a valid NT user. When a browser (IE or Netscape) encounters a resource protected by basic authentication, the browser will pop up a dialog box requesting a username and password. If no NT domain is specified in the basic authentication properties, the user must enter this as part of the username, like this:

```
NTDOMAIN\username
```

Windows NT Challenge / Response only works in Internet Explorer. When this box is checked, the browser will automatically send the domain and username of the user currently logged in. This is very handy for creating personalized intranet sites.

Secure Communications

This tab lets you create an SSL key for your Web site so you can encrypt all traffic and conduct secure transactions. You can enable or disable SSL at the directory level; however, an SSL key applies to an entire Web site.

Before you can actually use SSL, you must obtain a valid certificate from a certification authority. Leading certificate authorities are VeriSign and Thawte Consulting. The SSL Key Manager built into IIS makes it quite easy to apply for and obtain a certificate.

Tip

When setting up SSL, don't forget to configure the SSL port at the Web site level.

Designing an SSL-enabled application requires careful planning. Here are a couple things to remember:

1) SSL is resource-intensive due to the extensive key calculations required during each request. You should turn on SSL for only those parts of your application which actually need it.
2) You cannot mix SSL and non-SSL resources on the same page. Therefore, you must typically mirror files such as images which are common to both the secure and non-secure parts of your application.
3) SSL-enabled resources use the URL prefix https instead of http. It is always a good idea to use relative rather than absolute URLs, but this is especially true if you are building an application and plan to turn on SSL later.

IP Address and Domain Name Restrictions

This tab allows you to restrict access to a directory on the basis of an IP address range or domain name. This tab is only available with IIS running on Windows NT Server.

Caution

Protecting resources by domain name is not secure because the reverse DNS lookups used to map an IP address to a domain name are easily spoofed. IP addresses themselves can also be spoofed, so neither of these capabilities should be trusted 100%.

When authentication is required in a directory, the domain and username of the authenticated user are stored in the CGI environment variable REMOTE_USER. Programs in the protected directory can access this variable to determine the identity of the user making the request.

HTTP Headers

The HTTP headers tab allows you to configure various aspects of server operation.

Content Expiration

Normally, you want your dynamic pages to expire immediately, and this is the default. But static HTML and images can have longer expiration periods. Otherwise, the browser and any proxy servers along the way have to continually hit the server to see if there is a new version, even if the image is only two seconds old.

Performance Tip

Setting expiration dates on content is probably one of the most commonly overlooked ways to reduce network and server load. It requires a little planning, but the results are well worth it.

Setting a content expiration period even fifteen minutes into the future helps reduce load because most Web sites use many of the same images and static content (like Cascading Style Sheets) on every page.

A good application design will take into account content expiration right from the start. This will typically mean that you put your static content (images, HTML, CSS) in a different directory than your dynamic content. This makes it very easy to set appropriate expiration periods for each kind of content.

You can also programmatically set an expiration period for a document using the HTML META tag in the document HEAD. The format is

```
<META HTTP-EQUIV=”Expires”
      CONTENT=”Tuesday, 29-Feb-00 17:00:00 GMT”>
```

This simulates the HTTP Expires header.

Custom headers

HTTP supports a number of headers, most of which are more easily modified by other means. However, if you are working with a custom browser or proxy server and need to send special headers, this is the place to configure them. The standard HTTP headers are

Content-length	Set automatically by the server
Content-type	Set automatically by the server or CFCONTENT
Expires	Set in this tab or META tag
Location	For redirects; set in the directory properties tab
Pragma	Set in custom headers or META tag
Status	Set automatically by the server
Refresh	Can set in META tag to implement polling
Set-Cookie	Set with CFCOOKIE

Content Rating

Use this section to enable content rating for your site or directory. Some browsers are set to ignore all sites unless they have a rating, so you might want to rate your site appropriately to avoid inconveniencing these users.

MIME Map

The default MIME types for the server do not show up here. If you wish to add new types for this directory only, this is the place.

Custom Errors

The custom errors tab provides a way to create better-looking and more meaningful error pages than the default messages. This is a nice touch, although very few people judge a site by its error pages.

File Options

Many of the directory properties can also be set for individual files. To access these properties, simply right-click on a file in the right pane. You might use this capability to

- Indicate that a file has moved (redirect).
- Control read or write access to only one file in a directory.
- Not log hits for a cascading style sheet.
- Require authentication for a single file only.
- Set an expiration time for a file.
- Create a custom error page for a frequently-requested file which no longer exists on the server.

Server Monitoring

There are three primary ways to monitor your server. In the event you should ever need to restart IIS, you can do so either from the Internet Service Manager toolbar or the Windows NT Services Control Panel.

Log Files

Logging properties are configured at the Web site container level. You can use a third-party usage analysis utility like WebTrends to generate useful graphs and statistics from the logs.

Event Log

IIS writes many events into the Windows NT event log. You can quickly access the Event Log Viewer by clicking on its icon on the Internet Service Manager toolbar.

Performance Monitor

IIS defines many useful measurements for the Windows NT Performance Monitor. You can quickly access the Web-related parameters by clicking on the icon in the Internet Service Manager toolbar.

Programmatic Administration

If you’re developing a commercial Web-based application which will be installed on many Web servers, you may want to create an installation script which creates a Web site and/or virtual directories for your application. IIS provides a way to do this in the IISAdmin object, which you can access with any COM-capable language such as C++, VBScript, or JScript. Either of the scripting languages may be run in the Windows Scripting Host, which is part of the NT Option Pack.

You can find more information on the IISAdmin object in the MSDN Library under Platform SDK | Web Services | Internet Information Services SDK | Programmer’s Guide | Advanced Programmatic Administration.

Where to Go from Here

Application Help

The Help menu in Internet Services Manager is the starting point for all IIS issues. The online documentation contains a variety of articles on Web programming, security, etc.

Online Resources

Check out the MSDN Library, Web Workshop, and Microsoft Knowledge Base at http://msdn.microsoft.com. This is an extremely useful site.

IIS information is available in the MSDN Library under Platform SDK | Web Services.

Lesson 3
Introducing SQL Server

Introduction

Microsoft SQL Server is a powerful and easy-to-use relational database engine. Using the SQL Server Enterprise Manager, you can quickly and easily create high-performance databases for your applications.

Objectives

By the end of this lesson, you should be able to

- Create a new database using SQL Server Enterprise Manager
- Create new tables, fields, and relationships
- Understand SQL Server logins and database security
- Use the SQL Query Analyzer to interact with the database
- Import data from other sources using Data Transformation Services
- Optimize performance using the SQL Server Profiler

Why SQL Server?

1) Robust and scalable
2) SQL Server Enterprise Manager
3) Inexpensive
4) Data Transformation Services
5) Transact-SQL

Getting Started

You can start and stop SQL Server from one of four places:

1) Control Panel | Services | MSSQLServer
2) SQL Server Service Manager in the system tray
3) SQL Server Service Manager in Start | Programs | Microsoft SQL Server 7.0
4) SQL Server Enterprise Manager

SQL Server Enterprise Manager is home base for everything to do with SQL Server. We'll begin by walking through it to see what's available.

Walkthrough 3-1 Exploring Enterprise Manager

Setup

In order to complete this walkthrough, you must have previously installed Microsoft SQL Server 7.0

Steps

1. Open Start | Programs | Microsoft SQL Server 7.0 | Enterprise Manager.
2. Right-click on SQL Server Group and create a new SQL Server Registration for the local server.
3. Explore all the hierarchical folders.
4. Right-click on the local server and create a new database named WebFastTrack.

Tip

You can make the display less cluttered by editing the server registration properties and unchecking the box “Show system databases and system objects.”

Understanding SQL Server Security

SQL Server security is based on the concept of SQL Server *logins*. A login is a username and password pair necessary to establish a connection to the database.

Logins

SQL Server logins can be one of two types:

1) Windows NT user
2) SQL Server login

When you set up SQL Server, you can configure it to use Windows NT logins only or to also allow SQL server logins. For most Internet applications, you must allow SQL Server logins since Internet users will most likely not have NT accounts on your network. To see how SQL Server is currently configured, right-click on the computer name in Enterprise Manager and select Properties | Security.

A login is always required to establish a connection to the database, whether it's from Enterprise Manager, Query Analyzer, or another application via ODBC or OLE DB. However, a login by itself does not establish privileges to access any database.

Warning

The default login for the SQL Server system administrator is "sa" and the password is blank. Do change the password or anyone on your network can administer your database for you.

User Accounts

To obtain access to a database, a login must have a corresponding user account in each database, to which permissions are applied. When you create a new login in the SQL Server Security folder, Enterprise Manager will automatically create user accounts in the databases you specify in the Database Access tab.

Alternatively, you can create a new user account in the Users folder under each database. The login name and database user name do not have to be identical.

Roles

If you wish to create a group of user accounts with the same permissions, use a database *role*. Roles specify permissions for each object in the database and contain database user accounts.

Depending on the nature of your application, you may want to create one or more logins or roles to be used by the application. For example, you might define

- an application guest login or role with permissions to query tables only
- an application user login or role with permissions to query and update data
- an application administrator login or role with permissions to query, update, and delete data

Application logins are discussed more in Section 27, Database Security.

Walkthrough 3-2 Create a New Login and User Account

Setup

In this walkthrough, we'll create a login with permission to access the Northwind sample database.

Steps

1. In Enterprise Manager, click on the Logins folder under the Security folder.
2. Right-click to create a new login named "cfapp" with access to the Northwind database as the "public" role.
3. Click on the Users folder in the Northwind database.
4. Right-click on the newly-created user and select Properties.
5. Click the Permissions button. Notice that the user has no permissions.
6. Observe that the user is a member of the "public" role.
7. Click the Properties button to access the properties for the "public" role. Click the Permissions button to see what privileges are defined for role members.

Using Query Analyzer

SQL Server Query Analyzer is an interactive tool for running SQL statements and viewing results. In addition to executing queries, Query Analyzer can also show you the query execution plan with the relative cost for each operation.

There are two ways to launch Query Analyzer:

1) From Start | Programs | Microsoft SQL Server 7.0
2) From the Tools menu in Enterprise Manager

The latter is the easiest, as it will automatically establish a connection using the login information associated with the server registration in Enterprise Manager.

Walkthrough 3-3 Query Analyzer

Setup

Open SQL Server Enterprise Manager.

Steps

1. In Enterprise Manager, click on the Northwind database.
2. Click Tools | SQL Server Query Analyzer.
3. Select the Northwind database.
4. Turn on Results in Grid and Show Execution Plan.
5. In the query window, type in “SELECT * FROM Orders”.
6. Press F5 or the green triangle to execute the query.
7. Click the Execution Plan tab.
8. In the query window, add “ORDER BY CustomerID”.
9. Execute the query again.
10. Review the execution plan. Notice the sort.
11. Click in the query window, then click the Save button to save your query.
12. Click in the results window, then click the Save button to save the results.
13. Click on the Connection properties. In the General tab, turn on stats time and I/O stats.
14. Execute the query again.
15. Click on Connection properties. In the Advanced tab, change the results format to “Other delimited” and specify the vertical bar (|) as the delimiter.
16. Execute the query and save the results as WebFastTrack/Orders.rpt.

SQL Server Profiler

SQL Server Profile is a window into database operation and performance. Using Profiler, you can

- Identify and trace the worst performing queries
- Identify the cause of a deadlock
- Trace the performance of a stored procedure
- Trace activity by user

Programmatic Administration

If you're developing a commercial application which will be installed on many servers, you may want to write a program which automatically creates the database needed by your application.

The most straightforward way to do this is to use the subset of ANSI-standard SQL known as the Data Definition Language. DDL includes SQL commands such as CREATE TABLE and CREATE INDEX.

Fortunately, SQL Server has the capability to create DDL automatically. This is known as database scripting, and is available through Enterprise Manager. Simply right-click on a database and select All Tasks | Generate SQL Scripts.

Data Transformation Services

DTS is an extremely powerful tool for importing and exporting data from other sources. DTS gives you the ability to create import and export *packages* so you can use the same script many times. In addition, you can schedule import/export tasks to run at scheduled times.

Walkthrough 3-4 Data Transformation Services

Setup

Open SQL Server Enterprise Manager.

Steps

1. In Enterprise Manager, right-click on Northwind | All Tasks | Export Data. Click Next.
2. Choose the Northwind database and a valid login. Click Next.
3. On the Choose a Destination Screen, select Microsoft Excel 8.0 and the filename WebFastTrack\Lab\Northwind.xls. Click Next.
4. Select “Copy tables from the source database.” Click Next.
5. Select the Orders and Order_Details tables. Explore the Transform and Preview options. Click Next.
6. Check “Save DTS package” with the “SQL Server” option. Click Next.
7. Name the package “Export Orders.” Click Next.
8. Click Finish. The task will now run and display results. When all steps are marked complete, click Done.
9. Open WebFastTrack\Lab\Northwind.xls.
10. In Enterprise Manager, click on “Local Packages” in the Data Transformation Services folder.
11. Right-click on the “Export Orders” package and select “Design Package.”
12. Review the properties for each item in the diagram.
13. Click “Save As” on the Package menu to save the package as a file. This way, you can copy the package to other servers for later use.

Walkthrough 3-5 Import Text File

Setup

One of the most timesaving capabilities of Data Transformation Services is the ability to import from text files. In this walkthrough, we will import the results of the Orders query we saved in Walkthrough 3-3.

Steps

1. In Enterprise Manager, right-click on Data Transformation Services and select "New Package."
2. From the Data menu on the left side, click the "Text File (Source)" icon.
3. Name the data source "Northwind Orders" and find the file WebFastTrack\Lab\Orders.rpt. Click OK.
4. In the Text File Properties dialog box, select the "Delimited" option and check "First row has column names." Click Next.
5. Type in the vertical bar (|) in the field labeled "Other." This is the field delimiter. Click Finish, then OK. You should see the text source icon appear in the diagram window.
6. On the Data menu, click on the "OLE DB Provider for SQL Server" icon. Enter a valid username and password and select the WebFastTrack database. Click OK.
7. Click the "Northwind Orders" icon, hold down the CTRL key, and click the other icon. Now go to the Workflow menu and select "Add Transform."
8. Double-click the arrow and review the settings in all the tabs. Click OK when finished.
9. On the Package menu, select "Execute". You can also press F5 to do the same thing.
10. When the package finishes, close the window. You will be prompted to save the package. Click "Yes" and name it "Import Orders."
11. Click on "Tables" under the WebFastTrack database. You may need to press F5 to refresh the display. You should see the newly-created Orders table.
12. Open the Orders table to see that the import completed successfully.

Where to Go from Here

Application Help

Start | Programs | Microsoft SQL Server 7.0 | Books Online is the starting point for all local documentation.

The Query Analyzer Help menu has a useful shortcut to Transact-SQL Help, which is a pretty good SQL reference.

Online Resources

Check out the MSDN Library and Microsoft Knowledge Base at http://msdn.microsoft.com. This is an extremely useful site.

SQL Server information is available in the MSDN Library under Platform SDK | Data Access Services | Microsoft SQL Server Programmer's Guide.

Books

Inside SQL Server 7.0, Microsoft Press. This book comes with a 120-day evaluation copy of SQL Server 7.0. It is an advanced book, not an SQL Tutorial, and is very valuable for serious SQL Server development.

Other

The Microsoft BackOffice Resource Kit, which is included with an MSDN subscription, has additional SQL Server information.

Lesson 4
Getting Started with ColdFusion

Introduction

ColdFusion is simply the fastest way on earth to build dynamic, scalable production Web sites across multiple platforms. As you learn the ColdFusion environment, you will begin to appreciate the power and simplicity of the ColdFusion model.

Objectives

By the end of this lesson, you should be able to

- Understand the benefits of ColdFusion
- Explain the various components of ColdFusion development
- Configure ColdFusion with the ColdFusion Administrator
- Work with servers, data sources, and pages in ColdFusion Studio

Why ColdFusion?

Rapid Development

- Round-trip code editing
- Tag wizards, inspectors, and helpers save typing, increase accuracy, and save time looking up attributes
- Interactive debugger
- One-step deployment
- Total customization
- Reusable code (snippets, custom tags)
- Speedy query builder

Scalable Deployment

- Cross-platform
- Load balancing and clustering
- Superb caching and optimization support
- Web servers thread pooling
- Database server connection pooling

Open Integration

- Connect to servers running SMTP, POP3, FTP, HTTP, and LDAP
- Connect to COM, CORBA, and EJB objects
- ColdFusion Extensions (CFX) let you create your own tags in C/C++
- Connect to any ODBC database
- Native database drivers boost performance
- Schedule pages to run regularly
- Use built-in Verity full-text search engine

Total Security

- Authenticate users using NT users and groups, LDAP directories, or custom databases
- Authenticate developers before receiving access to protected resources
- Server Sandbox Deployment prevents multiple applications on the same server from accidentally stepping on each other
- Pass through user logins to database
- Restrict database operations

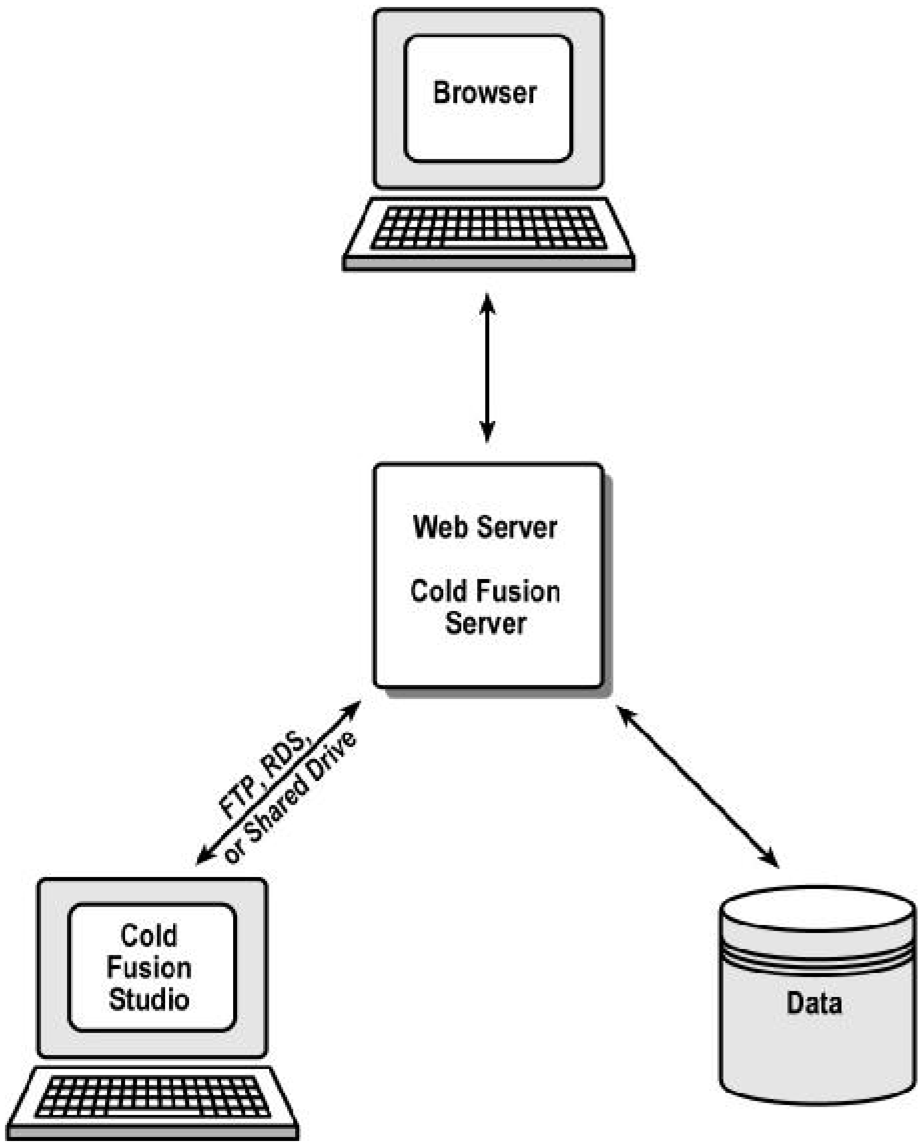
How Does ColdFusion Work?

ColdFusion applications run on the *ColdFusion Application Server*.

You administer the server using the *ColdFusion Administrator*.

ColdFusion Studio is the fastest way to create ColdFusion applications, although any text editor will do.

Figure 4-1 ColdFusion Server connects clients and servers.



Exploring ColdFusion Studio

Menus

File
New
Open from Web
Insert File
Convert from Text File

Edit
Indent
Bookmarks and Goto
Convert case

Search
Extended Find / Replace
Replace Special Characters

Tools
Tag Chooser & Expression Builder
Validation & verification
Document weight
Image map

Project
Reopen project

Options

Debug

Tags
Start Tag, End Tag
Matching Tag
Tag Tip F2
Inspect Tag F4
Edit Tag Ctrl+F4

View
Full-screen Ctrl+F12
Toggle Edit/Browse F12

Help
Tag Help F1

Main toolbar

Look – it floats!
Special characters
External browsers
CodeSweeper
Palette
Thumbnails
Style editor
Image map

Resource window

Explorer
Filter
Favorites
Link away
Allaire FTP & RDS (in Windows Explorer, too)
Web folders

Data Sources

Projects
New
Deploy

Site view

Snippets

Help
Search
Bookmarks

Tag Inspector

Editor toolbar

Numbers
Word wrap
Tag completion
Tag insight
Tag validation
Split editor

Browser toolbar

Size
Rulers

Quickbar

Look – it floats!
Right-click to customize

Common: Quick Start, Body, Image, Comment
Tables: Wizard, Quick
Frames: Wizard
CFML Basic: Comment
Script: ActiveX

Debugging toolbar

Results pane

Getting the Most from Studio

Templates

Templates can be great time-savers. They are especially useful when you will create many documents with a common structure, or when you want to include standard information in every document, such as revision history.

ColdFusion Studio stores templates in Program Files\Allaire\ColdFusion Studio\Wizards. You can create your own directories here to show up in the File | New dialog.

You can also change the default template location in Options | Settings | Locations.

Snippets

Snippets are most useful for

1) Saving time re-entering code, and
2) Storing code segments for later reference.

Snippets can also be shared with other developers on a common network drive. To set this up, see Options | Settings | Locations.

Shortcuts

Keyboard shortcuts can be assigned to any tag or action in ColdFusion Studio. They can even be assigned to snippets. Some of the most useful shortcuts are

Ctrl+comma	Start tag
Ctrl+period	End tag
Ctrl+3	Surround with #
Ctrl+Shift+>	Indent
Ctrl+Shift+<	Unindent
Ctrl+Shift+M	HTML comment
Ctrl+M	Find matching tag
Ctrl+Shift+double-click	Select tag

There is no default shortcut for a CFML comment, but you can easily set one up.

If you select text in the editor window and right-click, there are many useful options.

Administering ColdFusion Server

Running ColdFusion Administrator

To run ColdFusion Administrator to administer the local server, use Start Menu | Programs | ColdFusion Server 4.5 | ColdFusion Administrator.

To run ColdFusion Administrator on a remote server, use this URL:

http://server_name/CFIDE/Administrator/index.cfm

You must know the ColdFusion Administrator password on the remote machine.

Exploring ColdFusion Administrator

Server

Data Sources

Extensions

Logging

Automated Tasks

Miscellaneous

Working with Servers in ColdFusion Studio

ColdFusion Studio allows you to work with files and databases on remote servers using ColdFusion Remote Development Services (RDS). The Remote Files and Databases tabs on the Studio resource pane let you set up remote servers.

In order for RDS to work, the ColdFusion server must be running the ColdFusion RDS service. By default, this is installed with ColdFusion Server. On production servers, it is recommended that you disable the ColdFusion RDS service in the Windows NT Services Control Panel in order to prevent someone from gaining access to files and data on the server.

Security Tip

The default installation of ColdFusion server installs basic security, which uses only a password to protect all RDS resources. To assign different access levels to resources by user, you must install Advanced Security Services with ColdFusion Server. To do this, simply rerun the ColdFusion Server installation program.

Walkthrough 4-1 Add a Remote Server

Setup

In this walkthrough, we'll add the local machine as an RDS server. ColdFusion Server must be running.

Steps

1. In ColdFusion Studio, click the Explorer resource tab.
2. From the pull-down menu, select Allaire FTP & RDS.
3. Right-click on the icon and choose "Add RDS Server."
4. In the Description and Hostname fields, type "localhost."
5. Enter the ColdFusion Studio password for the local ColdFusion server. If you are not using Advanced Security Services on the ColdFusion server, you do not need a username.
6. Uncheck the "Prompt for password" box. Click OK.
7. Expand the "localhost" icon to verify that you can see the directory tree. This is not terribly exciting on the local server, but very powerful (and dangerous) on development or production servers.

Working with Projects

What Is a Project?

A ColdFusion project is a collection of files in a single directory tree which are managed as a group. A project normally corresponds to an application or Web site.

Why use Projects?

- Deployment—upload a project to one or more remote server locations with a single command.
- Maintenance—conduct extended search and replace operations on an entire project, including files from different directory trees
- Source control—map a project into source control to better track your code
- Filtering—directory filters let you see just the project files without artifacts such as log files or source control files

Project Source Control

ColdFusion Studio supports very convenient integration with Microsoft Visual SourceSafe in ColdFusion projects. When this integration is enabled, you can check in and out files right from the ColdFusion resource pane.

To set up source control integration in a project,

1) Right-click on the project and select Source Control | Choose Source Control Provider.
2) Right-click on the project and select Source Control | Map Project to Source Control.

Once the project is mapped, you can perform most source control operations right in ColdFusion.

About Development Mappings

In order to “run” your code in the internal browser, you must create a development mapping which tells ColdFusion where to find files in URL space. Set up development mappings in Options | Settings | Browse | Development Mappings.

Walkthrough 4-2 Create a ColdFusion Project

Setup

In this walkthrough, you will create a new project to be used throughout the course. You should already have a directory named WebFastTrack.

Steps

1. In ColdFusion Studio, click the Projects resource tab.
2. Click the New Project Wizard icon.
3. Enter the name WebFastTrack and find the WebFastTrack directory. Click OK.
4. Create a new file from the default template. Change the title and enter some text between the BODY tags.
5. Save it as FirstPage.cfm in the WebFastTrack\Lab directory. Notice that it automatically appears in the project window.
6. Create a development mapping for the new project. Click Options | Settings | Browse | Development Mappings.
7. Find the WebFastTrack directory and use the URL http://127.0.0.1/wft.
8. Click Add, then OK.
9. Press F12 to run FirstPage.cfm.
10. In the Project pane, right-click on the WebFastTrack folder and select Properties. Explore the project folder and deployment options.

Starting and Stopping ColdFusion

From a Web Page

To start or stop the ColdFusion server from a remote machine, use the Start – Stop Web page in the ColdFusion Server program group on the Start menu. Replace localhost with the server name you wish to control. The URL is

http://server_name/CFIDE/Administrator/startstop.html

You must know the ColdFusion Administrator password on the remote machine.

Note
This feature seems to have disappeared in ColdFusion Server 4.5

In the Services Control Panel (Windows NT)

To start or stop the ColdFusion server on the local Windows NT machine, you can use the Windows NT Services Control Panel. If you are restarting the server in order to break database connections or re-initialize an application, you only need to stop and start the ColdFusion Application Server service.

If you need to refresh data sources appearing in the ColdFusion Studio resource pane, you must restart the ColdFusion RDS service.

On the Command Line (Windows NT)

```
NET STOP "Cold Fusion Application Server"
NET START "Cold Fusion Application Server"
```

Using Scripts (Unix)

To start or stop ColdFusion Server on Unix, use the supplied start and stop scripts located in the ColdFusion/bin directory (by default, /opt/coldfusion/bin). The Unix scripts will start and stop all ColdFusion processes, including the application server, ColdFusion RDS, and the WindU registry, which ColdFusion uses to emulate the Windows registry on Unix.

Where to Go from Here

Application Help

See *Using ColdFusion Studio*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

For more information on customizing ColdFusion Studio, see the following topics in the Studio Help resource pane:

Customizing the Development Environment
Scripting the Visual Tools Object Model

Online Resources

Allaire's Web site (www.allaire.com) is the jumping-off point for all things ColdFusion. The Developer area features helpful articles and the Allaire Tag Gallery, where you can find hundreds of third-party custom tags. The Reference Desk in the Developer area contains many helpful related Web references, as well as links to ColdFusion e-zines.

Allaire Alive (alive.allaire.com) contains online tutorials in a video format.

Cfmcentral.com is another ColdFusion development site.

Cnet.com has a few articles on ColdFusion.

Lesson 5
Querying Data

Introduction

ColdFusion offers powerful capabilities for querying and updating databases. ColdFusion queries work with any ODBC data source, and the Enterprise version has native drivers for Oracle, Informix, and Sybase.

Objectives

By the end of this lesson, you should be able to

- Create an ODBC data source using the ODBC Control Panel
- Create an ODBC data source using the ColdFusion Administrator
- View data sources in ColdFusion Studio
- Create a ColdFusion query using the graphical query builder
- Understand how database queries can be shared with other developers
- Use the basic CFQUERY tag
- Display and use query results

Understanding Data Sources

Before you can work with a database in ColdFusion, you must create an ODBC or OLE DB *data source*. A data source is not the database itself, but rather a collection of information needed to connect to the database. You can set up data sources in either the ODBC Control Panel or the ColdFusion Administrator.

An ODBC or OLE DB data source can be used to connect ColdFusion Server to a database anywhere on your network. ColdFusion Server and the database do not have to be running on the same platform. You can connect from ColdFusion Server on Windows NT to a database running on a Unix box or vice versa.

ODBC Data Sources

ODBC (Open Database Connectivity) is an open standard originally developed by Microsoft. ODBC drivers are available for every major platform. ODBC is by far the most universal database connectivity standard.

Note
Almost all databases require you to install database client software in addition to the ODBC drivers in order to connect to the database using ODBC. Microsoft SQL Server requires only the ODBC drivers.

OLE DB Data Sources (Windows NT Only)

OLE DB is a newer database connectivity standard developed by Microsoft. In general, OLE DB offers better performance than ODBC. Most major database vendors offer OLE DB drivers. In addition, OLE DB drivers are available for connecting to things other than relational databases, such as an LDAP server or Microsoft Exchange Server.

ColdFusion can use OLE DB data sources; however, they can only be configured in the ColdFusion Administrator.

Creating an ODBC Data Source on Windows NT

When running ColdFusion Server on Windows NT, you configure ODBC data sources using either the ODBC Control Panel or the ColdFusion Administrator.

ODBC Control Panel

Windows keeps track of ODBC data sources in the ODBC Control Panel. Data sources can be associated with individual users or the entire system. If you create a user data source, it will only be available when you are logged in. System data sources are available to all users and programs.

ColdFusion Administrator

ColdFusion Administrator lists the same ODBC data sources as the ODBC Control Panel System tab. You can manage data sources in either location. Only in the ColdFusion Administrator, however, can you configure ColdFusion settings such as connection pooling and the database login to use when connecting from ColdFusion.

A Word about Passwords

If you specify a default database login when creating a data source in the ODBC Control Panel, any user or program on the machine will be able to connect to the database. This is not recommended.

A less permissive approach is to specify a default username and password for the connection in the ColdFusion Administrator. This way, only ColdFusion programs will have access to the data source. For higher security applications, do not specify a default login at all. Instead, pass in the username and password in the CFQUERY tag.

Walkthrough 5-1 Create a Data Source in the ODBC Control Panel

Setup

In this walkthrough, we'll create a data source to the Northwind sample database included with SQL Server.

Steps

1. Open Control Panel | ODBC Data Sources.
2. Select the System DSN tab.
3. Click the Add button.
4. Select the SQL Server driver and click Finish.
5. In the Name field, type in "Northwind."
6. Enter a description if you wish.
7. Choose (local) from the Server pull-down menu.
8. Click Next.
9. Select the radio button for SQL Server authentication.
10. Uncheck the box "Connect to SQL Server…." We will specify a username and password in ColdFusion Administrator instead.
11. Click Next and then Finish.

Walkthrough 5-2 Modify a Data Source in ColdFusion Administrator

Setup

In this walkthrough, we will review the ColdFusion settings for the Northwind data source and specify a default username and password using the SQL Server login we created in Lesson 3.

Steps

1. Open the ColdFusion Administrator.
2. In the menu on the left, click ODBC.
3. Click on the Northwind data source.
4. Click the CF Settings button.
5. Enter “cfapp” in the username and password fields.
6. Click Update.
7. Note the status of the Northwind data source.

Creating an ODBC Data Source on Unix

When running ColdFusion Server on Unix, you configure ODBC data sources using both the ColdFusion Administrator and the odbc.ini file. Unix does not have an ODBC Control Panel like Windows NT.

In general, you must do the following:

1) Install the database client software which supports ODBC on the ColdFusion Server.
2) Edit odbc.ini in the ColdFusion odbc directory (by default, /opt/coldfusion/odbc/odbc.ini) to create a new data source name and specify driver information.
3) Configure ColdFusion settings for the new data source in the ColdFusion Adminstrator.

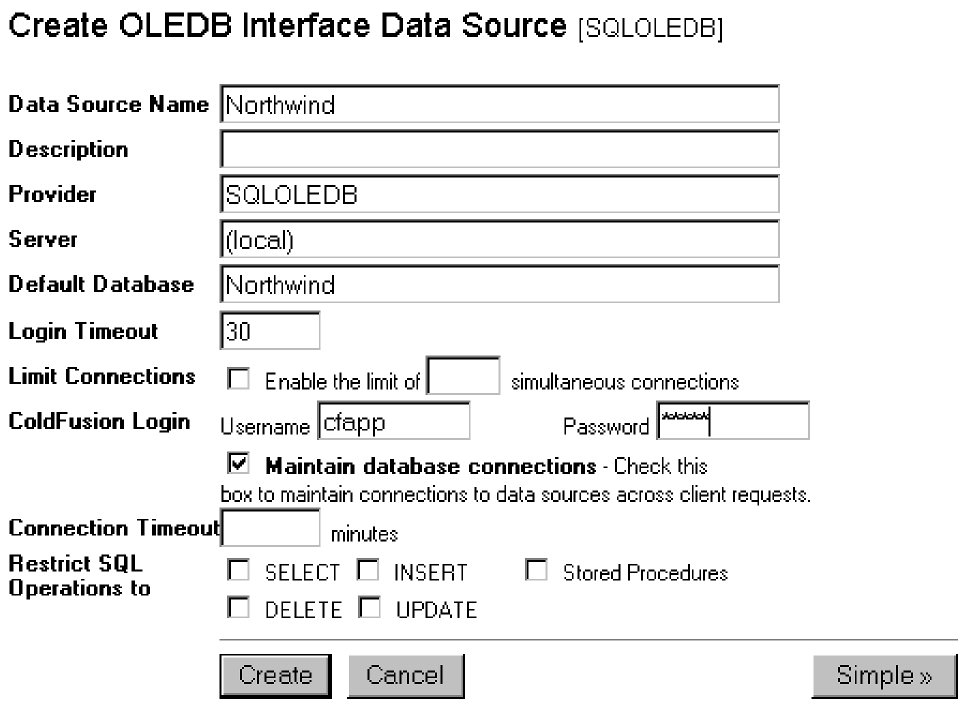
Creating an OLE DB Data Source (Windows NT Only)

On Windows NT, you can get a performance boost by using OLE DB instead of ODBC.

To create an OLE DB data source, you must first install the OLE DB *provider* (analogous to an ODBC driver) for your database. Normally, you obtain this from your database vendor. Microsoft also includes many OLE DB providers in the Microsoft Data Access Components available for free download at www.microsoft.com/data.

In the OLE DB data sources section of ColdFusion Administrator, simply reference the provider name. For Microsoft SQL Server, it is SQLOLEDB.

Figure 5-1 Create an OLE DB data source for SQL Server in ColdFusion Administrator.



Create OLEDB Interface Data Source [SQLOLEDB]

Data Source Name	Northwind
Description	
Provider	SQLOLEDB
Server	(local)
Default Database	Northwind
Login Timeout	30
Limit Connections	☐ Enable the limit of ___ simultaneous connections
ColdFusion Login	Username cfapp Password *****
	☑ **Maintain database connections** - Check this box to maintain connections to data sources across client requests.
Connection Timeout	___ minutes
Restrict SQL Operations to	☐ SELECT ☐ INSERT ☐ Stored Procedures ☐ DELETE ☐ UPDATE

Configuring Native Drivers

ColdFusion Enterprise Server 4.5 offers the following native drivers for even greater performance:

- Sybase System 11 and Sybase Adaptive Server 11.5
- Oracle 7.3, 8.0, and 8i
- Informix 7.3 and 9
- DB2 Universal Database 5.2 and 6.1

To configure a native driver, you must first install the database client software. You then configure the native driver using the ODBC section of ColdFusion Administrator. Be sure to select the native driver, not an ODBC driver.

Modifying a Data Source

You can make changes to a data source using any of the same methods you use to create the data source. On Windows NT, any changes you make in the ODBC Control Panel will also be reflected in the ColdFusion Administrator and vice versa.

If you have configured a data source to maintain database connections in ColdFusion Administrator, you may need to restart the ColdFusion Server before any later changes made to the data source will take effect in ColdFusion pages. In order for changes to show up in the database resource pane in ColdFusion Studio, you will have to restart the ColdFusion RDS Service.

Programmatically Creating a Data Source

On Windows NT, you can programmatically create a data source by directly modifying the registry (ColdFusion Server on Unix simulates the Windows registry, but there are no automated installation tools like InstallShield for Windows to help you make registry changes when installing software). The registry key containing information about ColdFusion data sources is

HKEY_LOCAL_MACHINE\SOFTWARE\Allaire\ColdFusion\
CurrentVersion\DataSources

For ODBC data sources, you must also update C:\WINNT\odbc.ini and the ODBC information in the registry at

HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\odbc.ini

Working with Databases in Studio

The database tab in the Studio resource pane offers a convenient way to work with databases. In the Database resource tab, you can

1) See the structure of a database, including tables, views, and fields
2) Build and run queries
3) Drag and drop from the database tab to the editor window

How Studio Database Access Works

ColdFusion Studio does not access data sources directly, but rather through a ColdFusion server via Remote Development Services (RDS). Using Studio, you can access data sources on any number of remote ColdFusion servers. Even on you local machine, Studio is actually contacting the local ColdFusion server through RDS.

RDS makes it possible to set up a ColdFusion server for workgroup development where all developers have access to the same database, queries, and common code.

In order to access a data source defined on a remote server, you only need to define the server as an RDS server in either the Remote Files or Databases tab. You do not need to define an ODBC connection on the local machine because ColdFusion will use RDS to access the remote server.

Note

If you specify a default username and password for a data source in the ColdFusion Administrator, that login will be used by ColdFusion applications on the server and also by developers connecting through ColdFusion Studio.

Walkthrough 5-3 Explore the Studio Database Resource Tab

Setup

In this walkthrough, we'll explore the Northwind database on your local server. You should have already set up the Northwind data source.

Steps

1. Click the Database resource tab and select the localhost server.
2. Expand the Northwind database.
3. Expand the Tables folder.
4. Right-click on the Employees sub-folder and select View Data.
5. Expand the Employees folder. Drag a table or field name from the Database resource tab to the edit window.
6. Expand the Views folder (note: in Microsoft Access, views are called queries, so Access queries show up as ColdFusion views).

Working with Queries

What is a Query?

A ColdFusion database query is any SQL statement. ColdFusion queries do not have to query data—they can also modify data or do other database operations.

Queries normally return one or more rows from a database, but they don't have to. SQL statements which modify data will normally not return any rows.

ColdFusion queries are not limited to database queries, although we are only interested in database queries for the moment. Other ColdFusion tags can return data in query objects, including CFDIRECTORY, CFFTP, CFHTTP, CFLDAP, and CFSEARCH. In addition, ColdFusion provides functions which let you build query objects from any type of data.

ColdFusion Studio provides a facility for easily creating and managing queries. Using the ColdFusion Query Builder, you can see the results of your query instantly and save your query for later use.

Where Are Queries Stored?

Queries which show up in the Database resource tab in ColdFusion Studio are stored on the ColdFusion server using ColdFusion RDS. This way, all developers on a project can share their queries as long as they are all pointing at the same ColdFusion server.

Displaying Query Output

To display the results of a query, simply reference the query fields as variables inside the CFOUTPUT tag with the QUERY attribute.

```
<CFOUTPUT QUERY="qContacts">
#qContacts.FirstName# #qContacts.LastName#<BR>
</CFOUTPUT>
```

This CFOUTPUT tag tells ColdFusion to loop over the query results one row at a time, printing the specified fields on each row.

Note

In a CFOUTPUT, it is not necessary to prefix field names with the query name; however, it makes your code much easier to read an maintain.

Walkthrough 5-4 Create a New Query with Query Builder

Setup

Open the Northwind database in the ColdFusion Database resource tab. In this example, we will create a query to show orders by customer.

Steps

1. Right-click on the Queries folder and select New Query…
2. Select the Customers table.
3. Click the Add button (+) and add the Orders table.
4. Join the Customers and Orders table by dragging CustomerID from one table to the other.
5. Double-click on the CompanyName and OrderID fields to add them to the query.
6. Click the exclamation point (!) icon to run the query.
7. Click the disk icon to save the query. Name it "OrdersByCustomer."
8. Create a new file from the default template.
9. Drag the OrdersByCustomer query into the editor window.
10. After the CFQUERY tag, type the following lines:

```
<CFOUTPUT query="OrdersByCustomer">
#CompanyName# #OrderID#<BR>
</CFOUTPUT>
```

11. Save the file as WebFastTrack\Lab\OrdersByCustomer.cfm
12. Press F12 to run the file.

Displaying Query Results

Special Formatting Functions

ColdFusion provides several functions useful for formatting query results.

See *CFML Language Reference*, Chapter 3: CFML Functions.

Date, Time, and Number Formatting

- DateFormat
- TimeFormat
- NumberFormat

Text and Other Formatting Functions

- HTMLCodeFormat
- HTMLEditFormat
- ParagraphFormat
- PreserveSingleQuotes
- StripCR
- URLEncodedFormat
- ValueList*
- QuotedValueList
- YesNoFormat

Presenting Query Results in a Table

ColdFusion Tables

ColdFusion provides its own tags, CFTABLE and CFCOL, for displaying tables. These are useful in that they spare you the “work” of CFOUTPUT and HTML table formatting. However, HTML formatting is much more flexible and not a whole lot more work.

HTML Tables

Here is the structure for your typical query results table. After you’ve done ColdFusion for a while, you’ll be able to write it in your sleep.

```
<CFQUERY NAME=”qListFolks” ...>
SELECT FirstName, LastName, Phone
      FROM People
</CFQUERY>

<TABLE ...>
<!--- The first row is the column headers. --->
<TR>
      <TH>First Name</TH>
      <TH>Last Name</TH>
      <TH>Phone Number</TH>
</TR>

<!--- Display table row for each query row --->
<CFOUTPUT QUERY=”qListFolks”>
<TR>
      <TD>#qListFolks.FirstName#</TD>
      <TD>#qListFolks.LastName#</TD>
      <TD>#qListFolks.Phone#</TD>
</TR>
</CFOUTPUT>
</TABLE>
```

Walkthrough 5-5 Build an HTML Table from a Query

Setup

In this example, we'll build an HTML table from the OrdersByCustomer query.

Steps

1. Edit OrdersByCustomer.cfm.
2. Surround the CFOUTPUT tags with TABLE tags:

```
<TABLE border="1">
<CFOUTPUT query="qOrdersByCustomer">
...
</CFOUTPUT>
</TABLE>
```

3. Replace the text inside the CFOUTPUT tags with the following:

```
<TR>
      <TD>#qOrdersByCustomer.CompanyName#</TD>
      <TD>#qOrdersByCustomer.OrderID#</TD>
</TR>
```

4. Save the file and browse it.
5. Try using the RecordCount, CurrentRow, and ColumnList query properties, as well as CFQUERY.ExecutionTime.

Introduction to Style Sheets

Version 4 and later browsers provide a way to create a consistent look for your application across all pages using Cascading Style Sheets (CSS).

What is a Cascading Style Sheet?

A style sheet is simply a list of HTML tag *selectors* and the associated style information. Style sheets have the extension .css. You can link to a style sheet from any number of pages in your site.

Using Style Sheets

To tell an HTML (or ColdFusion) page to use a style sheet, you include a link to the style sheet in the document's HEAD section, like this:

```
<HEAD>
      <TITLE>My Page</TITLE>
      <LINK rel="STYLESHEET"
type="text/css" href="Main.css">
</HEAD>
```

To quickly create this link in ColdFusion Studio, simply drag the style sheet from the resource pane into the HEAD section of any document. If you are using one style sheet for a hierarchy of pages, you can make things easier by using an absolute URL in the LINK HREF (as in "/Common/Main.css").

Be sure to configure your Web server to set an expiration time for style sheets. Otherwise, the browser will have to make an extra trip to the server for each page which references the style sheet.

Creating Style Sheets

Use the Allaire Style Editor to quickly and visually create a new style sheet. You can find it on the toolbar. Besides the basic HTML tags, you can use the following types of selectors to anchor styles:

- The BODY selector is the default style for a document
- The A selector lets you control the appearance of hyperlinks
- You can create your own style *classes* using .*class_name* and the class attribute of most HTML tags, including DIV and SPAN
- You can select a tag of a specific class using TAG.*class_name*
- You can nest selectors to select tag occurrences within other tags

Walkthrough 5-6 Creating a Style Sheet with the Allaire Style Editor

Setup

Open the WebFastTrack project in ColdFusion Studio.

Steps

1. Click on the Allaire Style Editor icon to create a new style sheet.
2. Add a TABLE selector and set a background color.
3. Add a TH selector to create a style for column headings.
4. Add a TD selector to defined a style for table cells.
5. Save the style sheet as Main.css.
6. Right-click on the project icon and select Synchronize Project to add the newly-created style sheet to the project.
7. Right-click on the project icon and select Refresh to refresh the display.
8. Edit OrdersByCustomer.cfm.
9. Drag Main.css from the project resource pane into the HEAD section of OrdersByCustomer.cfm.
10. Press F12 to test OrdersByCustomer.cfm with the style sheet.

Where to Go From Here

Application Help

For in-depth information on configuring data sources on Windows and Unix, consult the following chapter in *Administering ColdFusion Server*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 4: Managing Data Sources

See the following chapters in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 3: Querying a Database
Chapter 4: Retrieving and Formatting the Data You Want

For more information on style sheets, see the Style Sheets topic in the HTML Reference book in the Help resource tab.

Online Resources

A good place to learn more about style sheets is the Web Builder section of www.cnet.com.

Unit 2
Learning ColdFusion

Unit Overview

This unit will teach you the basics of constructing data-driven Web applications, including the use of forms and queries.

Goals

By the completion of this unit, you should be able to

- Create static and dynamic HTML forms
- Build queries from forms and display tabular results
- Validate forms for security and robustness
- Create SQL statements which modify data

Lesson 6
Creating Forms

Introduction

At the heart of the Web's interactive capabilities is support for forms. On the client side, users enter data through forms that contain familiar graphical elements such as text input fields, check boxes, and push buttons. The browser then sends the data to the server for processing.

Objectives

In this lesson, you learn

- The elements of HTML forms
- How form data is passed from client to server
- How to send data to a server without using forms
- How to access form data in ColdFusion programs
- How to embed state information in forms

Introduction to Forms

HTML's form support is very simple, and yet surprisingly complete. A handful of HTML tags can create the most popular elements of modern graphical interfaces, including text windows, check boxes and radio buttons, pull-down menus, and push buttons. In fact, using HTML forms in conjunction with server scripts is arguably the fastest and simplest way to create cross-platform graphical applications.

JavaScript extends HTML's form support by making it possible to change form elements dynamically on the client side only, avoiding frequent trips to the server. Also, JavaScript makes it possible to do data validation on the client (although it should also be done on the server for security reasons).

Java further extends forms by making it possible to use GUI elements which are not available in standard HTML forms. However, Java applets are an order of magnitude more difficult to write than HTML forms. In addition, browser support for Java varies widely, so it is generally safest to stick with standard HTML form elements.

How Forms Work

You construct HTML forms using tags similar to all other HTML constructs. Forms consist of three elements:

- Form container (<FORM …></FORM>)
- Named input fields (INPUT, SELECT, or TEXTAREA)
- One or more action buttons

A Web page can contain more than one form, each with its own submit buttons. However, when the user submits a form (typically by pressing the submit button), the browser only sends data from that form. Any information in other forms on the page is discarded.

The FORM Tag

When a user submits a form, the browser packages the form data and sends it to the URL embedded in the form header. The FORM tag specifies the location of the processing script and the HTTP method used to send the data. In addition, the FORM tag may assign a name to the form, which is not useful at all on the server, but can be useful to JavaScript programs running in the browser.

The FORM tag has the following basic structure:

```
<FORM ACTION="url" METHOD={GET | POST}>
...
</FORM>
```

The ACTION attribute specifies the path to the processing Web page (program). The tail end of a URL in an ACTION attribute can contain extra path information. This information is additional data that the browser sends to a program, but it is not included anywhere on the form. A URL to a ColdFusion program with extra path information looks like this:

http://address:port/filename.cfm/extra_path_
info

You can use extra path information in a URL to pass additional file name or directory information to a script.

The METHOD attribute determines how the browser sends form data to the server. The POST method causes form data to be sent as a separate data block within the request, whereas the GET method causes form data to be appended to the request URL.

How GET Works

The GET method encodes all form data in the URL in name-value pairs separated by the ampersand (&) like this:

http://127.0.0.1/wft/getdata.cfm?city=Cedar+Rapids&state=IA

Because URLs cannot contain spaces or special characters, form data in a URL must be *URL-encoded*. All spaces are converted to plus signs (+) All other special characters are converted to their hexadecimal representation after a percent sign like this:

http://127.0.0.1/wft/icecream.cfm?brand=Tom%26Jerry%27s

How POST Works

The POST method does not wrap form data in the URL, but rather sends a block of data containing a name-value pair on each line. Data sent with the POST method is still URL-encoded, however.

When to Use GET vs. POST

In general, forms use POST. If you are sending a lot of data (typically more than 1024 bytes), you should always use POST because the data appended to the URL may exceed the string length limitations of some operating systems. However, you must use GET when

1) You want users to be able to bookmark a page which results from a form submission, such as a query
2) You want ColdFusion to be able to automatically cache frequently-requested searches using the CFCACHE tag (more on this later)

Reading Form Data

ColdFusion makes the value(s) of each form input available in one of two variable classes, depending on the method specified in the FORM tag. The name of each variable is just the name specified in the HTML tag for the control. These names are never displayed anywhere. They are only used by your program to reference the form field.

GET

Because they are sent as part of the URL request, input fields in forms which use the GET method are named *URL.field_name*. ColdFusion server automatically parses and decodes the name-value pairs appended to the URL.

POST

Input fields in forms which use the POST method are name *Form.field_name*. ColdFusion automatically parses and decodes the name-value pairs sent with the POST request.

When you use the POST method, the variable Form.Fieldnames contains a comma-separated list of the names of all form fields passed in.

Form Elements

Forms consist of standard GUI controls such as text boxes, check boxes, and menus. You give each control a name that eventually becomes a variable name used by your ColdFusion program.

You can use many types of graphical controls in forms. These include

Type of Control	**HTML Tag**
Text box	<INPUT TYPE="text"…>
Password box	<INPUT TYPE="password"…>
Hidden field	<INPUT TYPE="hidden"…>
Check box	<INPUT TYPE="checkbox"…>
Radio button	<INPUT TYPE="radio"…>
File upload	<INPUT TYPE="file"…>
Reset button	<INPUT TYPE="reset"…>
Submit button	<INPUT TYPE="submit"…>
Graphical submit button	<INPUT TYPE="image"…>
Multi-line text window	<TEXTAREA…></TEXTAREA>
Drop-down list	<SELECT SIZE="1"…> <OPTION> … </SELECT>
Scrolling list	<SELECT SIZE="n"…> <OPTION> … </SELECT>

Text and Password Boxes

Text and password boxes are simple data entry fields. The only difference between the two is that all text typed in password boxes shows up as asterisks (*). The general form for a text or password field in HTML is

```
<INPUT TYPE=”{text | password}”
      NAME=”name”
      [VALUE=”default_text”]
      [SIZE=”width”]
      [MAXLENGTH=”number_of_characters”]>
```

Use the SIZE attribute to control the display width of the text box. Use the MAXLENGTH attribute to limit the number of characters the user can enter. If MAXLENGTH is smaller than SIZE, the display will scroll inside the box. Use VALUE to pre-load the text box with default text.

Tip

Because browsers ignore white space, it is difficult to line up the left edges of text input boxes on multiple lines because the text to the left of the boxes is of different lengths. This is a good application for HTML tables, with one label and text box on each row. Another solution is to put labels above or to the right of input boxes.

In your program, you access the text a user has entered by referring to the named URL or Form variable.

Warning

Even though password boxes display asterisks to the user, the value in the box is sent in the clear over the network. To avoid transmitting data in the clear, you must use SSL.

Multiline Text Windows

The <TEXTAREA> tag is a multiline version of a text input box, complete with scroll bars and word wrapping. The format for a text area is

```
<TEXTAREA NAME="name"
      ROWS="rows"
      COLS="columns"
      WRAP="{off | virtual | physical}">
...
</TEXTAREA>
```

You can place default text between the opening and closing <TEXTAREA> tags.

WRAP="off" causes no text wrapping to occur in the box and data is sent as entered.
WRAP="virtual" causes text wrapping to occur automatically, but text is sent with no line breaks.
WRAP="physical" causes text wrapping, and text is sent with line breaks.

Version 4.0 browser support the syntax WRAP="{off | soft | hard}."

In your program, you access the text a user has entered in a text area by referring to the named URL or Form variable.

Check Boxes

A check box is a yes/no indicator. The general form for a check box is

```
<INPUT TYPE=”checkbox”
      NAME=”name”
      VALUE=”value”
      [CHECKED]>label
```

The VALUE field is not displayed anywhere. It specifies the value which will be sent to the server for the check box or radio button.

Note

Check boxes only show up in the form data sent to the server if they are selected. If a check box is not selected and you attempt to refer to its variable name in your program, you will get an error like “Error resolving parameter **FORM.FIELDNAME**….”

Aggregating Check Boxes

If you have a group of unrelated yes/no questions, each one should have its own name. However, if you want to aggregate check boxes, you can give each one the same name. For example, to allow users to choose from among a number of states for which they want to see jobs, you can write

```
<INPUT TYPE=”checkbox” NAME=”state”
VALUE=”IA”>Iowa

<INPUT TYPE=”checkbox” NAME=”state”
VALUE=”KS”>Kansas

<INPUT TYPE=”checkbox” NAME=”state”
VALUE=”NE”>Nebraska
```

In this example, the ColdFusion variable Form.state will contain a comma-separated list of the user-selected values, like “IA,KS”. Note that this only works for forms submitted with the POST method.

Radio Buttons

Radio buttons differ from check boxes in that only one per group can be checked. All radio buttons having the same name are considered to be part of the same group. The format for a radio button is the same as a text box:

```
<INPUT TYPE=”radio”
      NAME=”name”
      VALUE=”value”
      [CHECKED]>label
```

You should specify one radio button to be checked by default.

Hidden Fields

Sometimes, you may want to pass information from one form to the next without displaying it for the user. For example, you might want to keep track of the database ID for a record that the user modifies on a series of screens. In this case, you would create a hidden field Database_ID containing the record ID.

Hidden fields are useful for keeping track of state information in applications which use a series of forms to process data. They can also be useful for security purposes. This will be discussed in a later lesson.

The syntax for a hidden field is

```
<INPUT TYPE=”hidden”
      NAME=”name”
      VALUE=”value”>
```

Warning

You should not use hidden fields to “hide” things like passwords which might be sensitive. The user can always view the HTML source for your page.

Buttons

HTML supports a variety of push button types.

Reset Buttons

A reset button allows the user to restore a form to its state when the page was loaded. When the user presses a reset button, the browser simply restores all form fields to their default values.

The syntax for a reset button is

```
<INPUT TYPE="reset"
      [NAME="name"]
      VALUE="value">
```

For a reset button, the VALUE will be displayed as button text. Since pressing a reset button does not initiate a request to the server, the NAME field is useful only for JavaScript programming on the client.

Submit Buttons

Every form must have at least one submit button (except in certain cases where JavaScript is used to active the form's submit method). When the submit button is pressed, the browser sends the form data to the server.

The syntax for a submit button is

```
<INPUT TYPE="submit"
      [NAME="name"]
      VALUE="value">
```

The VALUE will be displayed as the button text. NAME is optional.

Named Submit Buttons

In some cases, you may want to have more than one button on a form. For example, you might want an Update and a Delete button on a record display. To determine which button was pressed to submit the form, you can give each button a name. Your program can then test for the presence of the associated URL or Form variable with that name.

Alternatively, you can give each button the same name and look for its value in your program. For example, you might write

```
<INPUT TYPE="submit" NAME="action"
      VALUE="Update">
<INPUT TYPE="submit" NAME="action"
      VALUE="Delete">
```

Your program would then look at the value of Form.action to determine which button was pressed.

Image Buttons

An image button is a submit button which uses an image rather than the default push button shape.

The syntax for an image button is

```
<INPUT TYPE="image"
      [NAME="name"]
      SRC="path_to_image">
```

The NAME attribute is not useful except in JavaScript on the client. SRC specifies the path to the image file.

Note

Unfortunately, an image button cannot be a named submit button because the browser does not send the NAME attribute for an image button to the server.

Plain Old Buttons

A plain old button does nothing by itself. You would only use this type of button with a JavaScript event handler.

```
<INPUT TYPE="button" VALUE="value">
```

Drop-down and Scrolling Lists

The most compact way to present either single-valued (radio button) or multi-valued (check boxes) choices is in a drop-down or scrollable list. Both types of lists use the same HTML tag, but appear differently in the browser.

Drop-down Lists

The syntax for a drop-down list is

```
<SELECT NAME="name" SIZE="1">
      <OPTION VALUE="value1" [SELECTED]>Label 1
      <OPTION VALUE="value2" [SELECTED]>Label 2
      ...
</SELECT>
```

In a drop-down list, only one item can be selected at a time. The SELECTED attribute indicates whether the item is selected by default. The VALUE attribute specifies the value which is sent to the server for the selection. If VALUE is not specified, the browser will send the label text as the value.

Scrolling Lists

The syntax for a scrolling list is

```
<SELECT NAME="name" [SIZE="n"] [MULTIPLE]>
      <OPTION VALUE="value1" [SELECTED]>Label 1
      <OPTION VALUE="value2" [SELECTED]>Label 2
      ...
</SELECT>
```

The SIZE attribute controls the number of rows to display. If SIZE is smaller than the number of options listed, the browser will automatically include scroll bars.

The MULTIPLE attribute, if specified, allows the user to select more than one item simultaneously by holding down the Ctrl key while selecting items.

File Upload

One of the most useful controls in version 4 browsers and later is the file upload control. This control allows the user to activate the standard file chooser dialog and select a file to upload to the server.

The syntax for the file upload control is

```
<INPUT TYPE="file"
      NAME="name"
      [SIZE="width"]
      [MAXLENGTH="number_of_characters"]>
```

The SIZE and MAXLENGTH attributes control the size of the text window which displays the selected filename.

The file upload control does have some limitations:

1) You must use the POST method.
2) You must specify an additional attribute in the FORM tag to tell the server to send both form data and a file.
3) You can only select one file per upload control, and you can only have one upload control per form.

When using a file upload control, the FORM tag will look like this:

```
<FORM
      ACTION="URL"
      METHOD="POST"
      ENCTYPE="multipart/form-data">
```

Note

File uploads are still a bit quirky. For example, using Internet Explorer 4, the file will not be transmitted properly unless you use the Browse button in the upload control to select the file. You cannot type in the filename. In addition, if you press Enter rather than click the form's submit button, the file will not be sent.

Processing File Uploads

File uploads must be handled differently by your ColdFusion program than other types of form inputs. To handle an uploaded file, use the ColdFusion CFFILE tag as follows:

```
<CFFILE
      ACTION="upload"
      FILEFIELD="name_of_form_field"
      DESTINATION="path_to_file">
```

For a complete discussion on using CFFILE, see Chapter 15: Managing Files on the Server in *Developing Web Applications with ColdFusion*.

Note

Internet Explorer 4 and 5 do not always set the MIME type of the uploaded file correctly. If the file being uploaded is open on the client, IE will incorrectly set the MIME type to application/octet-stream.

Displaying Form Data

To display any ColdFusion variable, simply enclose it in CFOUPUT tags.

```
The name you entered was
<CFOUTPUT>
#Form.FirstName# #Form.LastName#
</CFOUTPUT>
```

You can use a Form or URL variable anywhere you can use any other type of variable, including

- CFQUERY
- CFOUTPUT
- CFIF
- CFSWITCH / CFCASE
- As an argument to a function

Lab 6-1 Create a Simple Form

Setup

Open the WebFastTrack project in ColdFusion Studio.

Steps

1. Create a new page using the default template and save it as TestForm.cfm.
2. Click on the Forms toolbar and create a form using the POST method with ACTION="ShowFormData.cfm."
3. Try out all the elements in the Forms toolbar in your form.
4. Save the page and browse it.
5. Create another new page using the default template. Save it as ShowFormData.cfm.
6. Inside CFOUTPUT tags, display the values of all the form variables you created.
7. Save the page, browse TestForm.cfm, and submit the form.

Using JavaScript with Forms

Client-side JavaScript is a powerful tool for adding interactivity and data validation to forms.

One of the major differences between HTML forms and traditional client-server applications is that all the data is sent at once. For example, suppose you want to alter the contents of a drop-down list box based on whether or not a certain check box is checked. With HTML, this requires a trip to the server and two screens: one with the checkbox followed by one with the dynamically-generated drop-down list. With JavaScript, however, you can dynamically alter the contents of the drop-down list box when the check box is checked or unchecked.

Objects

All HTML elements on a page are JavaScript objects, including the entire document and the document window. Objects have events, properties, and methods.

Events

JavaScript defines client-side events for every form element and HTML tag. When these events are triggered, you can call a JavaScript *event handler*. Commonly-used JavaScript events are

OnMouseOver and OnMouseOut (used for rollover images)
OnChange (fires when the user has changed a form value)
OnSubmit (fires when the form is submitted)

To invoke an event handler, simply define it in the HTML tag.

```
<INPUT TYPE=”text”
      NAME=”phone_no”
      onChange=”check_phone ()”>
```

This example will call the check_phone function defined elsewhere in the page when the value in the text field changes.

Normally, JavaScript event handlers are defined in the <HEAD> section of an HTML document.

Properties

Object properties allow you to read or dynamically alter the values of JavaScript objects. Properties allow you to do things like

- Check the values of form elements to validate data
- Dynamically update form elements to reflect user input
- Dynamically display or hide text
- Redirect the browser to another location

For example, the following JavaScript appearing in a page will redirect the browser to another location:

```
<SCRIPT>window.location.href =
‘new_location.html’</SCRIPT>
```

Methods

Use JavaScript methods to perform actions on objects. Methods are referenced the same way as properties.

For example, here’s a script to pop a dialog box when the user clicks on a link to Yahoo.

```
<A HREF=http://www.yahoo.com

onClick="alert ('Enjoy Yahoo!')">Yahoo!</A>
```

Cool Tip

You can call JavaScript functions most places where you would otherwise use a URL. The preceding example could have been written as <A HREF=”javascript: alert (‘Enjoy Yahoo!’)”>Do you Yahoo?</A>

The JavaScript form object has a submit () method which you can use to submit the form as if the user clicked the submit button. When you select an item in the drop-down list below, the form is automatically submitted.

```
<form action="choose_location.cfm"
method="post">

<SELECT name="location" size="1"
onChange="this.form.submit ()">

      <OPTION SELECTED>Choose a location

      <OPTION>Place 1

</select>

</form>
```

Passing Data Without Forms

Since the GET method just appends form data to the URL, you can use URL encoding any time you want to send data to the server without forms. For example, if you want to create a hyperlink which goes to the employee_detail.cfm screen and passes in employee_id=24, you would use the following syntax:

```
<A HREF=”employee_detail.cfm?employee_id=24”>
Jones, Jon
</A>
```

When the user clicks on the hyperlink, the variable URL.employee_id will be defined with a value of 24 in employee_detail.cfm.

You can use this technique to simulate the input of any form element except file upload.

Tip

Remember that URLs cannot contain spaces or special characters. If you need to set a value containing spaces or special characters, use the ColdFusion URLEncoded function.

JavaScript Tips & Tricks

Force a reload of the current page

```
<SCRIPT>window.location.reload ();</SCRIPT>
```

Go back to the previous page

```
<A HREF=”javascript:history.back ()”>Back</A>
```

Go to a new location

```
<SCRIPT>location.href =
‘new_location.html’;</SCRIPT>
```

Pop a message window

```
<SCRIPT>alert (‘message’);</SCRIPT>
```

Ask the user to confirm with OK or Cancel

```
<SCRIPT>
if (confirm (‘message’))
      do_something ();
</SCRIPT>
```

Prompt the user to enter a value

```
<SCRIPT>
prompt (‘Please enter your password’, ‘’);
</SCRIPT>
```

Pop a new window

```
<SCRIPT>
window.open ('somewhere.html', 'newWindowName',
‘toolbar,scrollbars,resizable,width=300,height=
200')
</SCRIPT>
```

Call a function in the parent window

```
<SCRIPT>opener.some_function ();</SCRIPT>
```

Where to Go From Here

Application Help

See the following chapters in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 6: Updating Your Data
Chapter 10: Building Dynamic Forms

Use the tag editor, tag inspector and tag help to learn the expanded syntax for each form elements.

Online Resources

See DHTML References under the ESSENTIALS section of the Web Workshop at msdn.microsoft.com. This is the definitive online reference for IE objects, methods, properties, etc.

You can find Netscape's version of the world at developer.netscape.com.

Books

Dynamic HTML: The Definitive Reference (O'Reilly) contains a very handy cross-browser listing of HTML elements along with their events, properties, and methods.

JavaScript: The Definitive Guide, 3rd Edition (O'Reilly). Like it sounds.

JavaScript Application Cookbook (O'Reilly)

Lesson 7
Creating Dynamic Queries

Introduction

The lightning of ColdFusion is the ease with which you can create dynamic queries from forms. In this lesson, we'll explore techniques for using form data to generate queries on the fly.

Objectives

By the end of this lesson, you should be able to

- Create queries with dynamic parameters
- Create queries with dynamic structure
- Harness the power of CFQUERY

Using Dynamic Query Parameters

ColdFusion variables can be used almost anywhere—in CFOUTPUT sections, in control logic (CFIF, etc.), as arguments to functions, and in queries. In this section, we'll explore using ColdFusion variables to build dynamic SQL statements.

Kinds of Dynamic Parameters

Any ColdFusion variable can be used to build an SQL statement inside CFQUERY, including

- Form fields
- URL parameters
- Server variables
- CGI environment variables
- Other queries
- Application, session, and client variables

Working with Variables in CFQUERY

To use a ColdFusion variable in a query, simply enclose it in pound signs. It's that easy.

For example, here's a ColdFusion snippet to return all employees who live in a particular state entered on a form:

```
<CFQUERY datasource="Employees" name="qLocals">
SELECT *
      FROM Employees
      WHERE State = '#Form.State#'
</CFQUERY>
```

ColdFusion substitutes the value of the form variable before sending the query to the database. If a user selected the state of Iowa on the form, the SQL statement sent to the database would be

```
SELECT *
      FROM Employees
      WHERE State = 'IA'
```

Handling String and Numeric Variables in SQL

ColdFusion is not a strongly typed language, and it's pretty good at converting between integers and strings as needed within ColdFusion code. When using variables in SQL statements, however, you must follow these simple rules:

1) When comparing a ColdFusion variable with a numeric database field, do not enclose the ColdFusion variable in quotes.
2) When comparing a ColdFusion variable with a text database field, you must enclose the ColdFusion variable in single quotes (‘).

Explicitly Typing Variables with CFQUERYPARAM

ColdFusion 4.5 lets you explicitly control the variable type of query parameters. CFQUERYPARAM is useful for three reasons:

1) It eliminates a significant security problem when using variables in queries (see Lesson 9: Validating Form Data).
2) It improves performance.
3) It allows you to insert and update large text fields with CFQUERY.

Working with Character Large Objects (CLOBs)

ColdFusion 4.5 introduces CLOB support for most databases (but unfortunately, not yet BLOB support for binary objects). For example, the following code reads a 45kB log file, inserts it into a SQL Server column of type text, and then displays all rows in the table.

```
<cffile action="READ" file="C:\winzip.log"
      variable="bigFile">

<cfquery name="InsertCLOB" datasource="Dev1">
INSERT INTO tmTest (cfield) VALUES (
<cfqueryparam value="#Variables.bigFile#"
      cfsqltype="CF_SQL_LONGVARCHAR">
)
</cfquery>

<cfquery name="qReadCLOBs" datasource="Dev1">
SELECT ID, cfield FROM tmTest
</cfquery>
<cfoutput query="qReadCLOBs">
      #qReadCLOBs.cfield#
</cfoutput>
```

Handling Date and Time Fields in SQL

Dates are not so friendly because every database handles them somewhat differently. Most databases will do their best to interpret string literals as date fields so you can do things like

```
SELECT *
      FROM Employees
      WHERE BirthDate < ‘1970’
```

or

```
SELECT *
      FROM Employees
      WHERE BirthDate < ‘3/1/1970’
```

However, if you wanted to find all employees who were born in 1970, you would have to use the DATEPART or YEAR function to make the comparison, as in

```
SELECT *
      FROM Employees
      WHERE YEAR (BirthDate) = 1970
```

Microsoft SQL Server provides the following functions for extracting date and time information:

GETDATE	Returns the current date and time
DATEADD	Adds two datetime values
DATEDIFF	Subtracts two datetime values
DATENAME	Returns the specified part of a datetime value as a string
DATEPART	Returns the specified part of a datetime value as an integer
DAY	Returns the day of the month portion of a datetime value
MONTH	Returns the month portion of a datetime value as an integer
YEAR	Returns the year portion of a datetime value as a four-digit integer

Searching on Patterns

In order to perform a wildcard search using a ColdFusion variable, use SQL’s LIKE statement as follows:

```
SELECT *
      FROM Employees
      WHERE FirstName LIKE ‘Dav%’
```

This query will match all first names beginning with “Dav.”

If the search parameter were coming from a form, you would write

```
SELECT *
      FROM Employees
      WHERE FirstName LIKE ‘#Form.FirstName#%’
```

You can use a wildcard with the LIKE operator at the beginning of a string literal, at the end, in the middle, or some combination thereof.

Keep in mind that SQL wildcard searches are typically database-intensive because the database must read every row in the table. Indexing the table on the search field will help some, but even indexing won’t help if the search begins with a wildcard.

Note

SQL Server text comparisons are not case-sensitive by default. The only time you can change this is during SQL Server setup.

Walkthrough 7-1 Create a Search Form and Drill-Down Application

Setup

Open the WebFastTrack project in ColdFusion Studio. In this lab, you will create three pages: a customer search form, a query page displaying the search results, and a drill-down page showing all orders for a customer.

Steps

1. Create a new page with a search form. Use the POST method and ACTION=”CustomerSearch.cfm.”
2. Create a text input field named “CompanyName.”
3. Save the page as CustomerForm.cfm.
4. Create a new query to select all rows from the Northwind Customers table where the company name field matches the form input. Save the query as ListCustomers.
5. Create a new page which displays the query results in a table.
6. Save the query page as CustomerSearch.cfm.
7. Browse CustomerForm.cfm to try out the search.
8. Modify CustomerSearch.cfm to include a column containing the customer ID in a hyperlink like <A HREF=”CustomerOrders.cfm?ID=#ListCustomers.CustomerID#”>. This will be used to drill down to customer orders.
9. Create a new page to display all orders for the selected customer.
10. Create a new query which selects all rows from the Orders table where the CustomerID is equal to URL.ID. Save the query as CustomerOrders.
11. Display the query results in a table.
12. Save the page as CustomerOrders.cfm.
13. Browse the application, beginning with CustomerSearch.cfm.

Using Form Fields in Queries

Single-Valued Fields

Single-valued form fields include

- Radio buttons
- Individual check boxes (not multiple check boxes having the same name)
- Drop-down lists
- Text and password boxes
- Hidden fields

Since these types of Form variables can have only one value, it is easy to include them in a query comparison as demonstrated in the previous section.

Multi-Valued Fields

Multi-valued fields include

- Multiple check boxes with the same name
- Scrolling lists which allow multiple selections

For multi-valued fields, the data stream coming from the browser consists of multiple name-value pairs, like this:

```
http://127.0.0.1/select_state.cfm?state=ia&stat
e=ks&state=ne
```

ColdFusion parses this data and puts the values in a comma-separated list. In this example,

```
Form.state = “ia,ks,ne”
```

To test for multiple values, use the SQL IN operator. For example, to return the records for all customers who live in one of the selected states, you would write

```
SELECT *
      FROM Customers
      WHERE State IN (#Form.State#)
```

This query would work if the values in Form.State were numeric; however, they are strings, which means the database expects to see single quotes around each value. The best way to do this is to use the ListQualify function in the query:

```
SELECT *
      FROM Customers
      WHERE State IN
      (#ListQualify (Form.State, "'", ",")#)
```

ListQualify () adds the delimiter character (in this case a single quote) around each list item. When the query is sent to the database, it will look like:

```
SELECT *
      FROM Customers
      WHERE State IN (‘ia’,’ks’,’ne’)
```

What will happen if one of the items in the list contains an apostrophe (as in Joe’s Diner)? When used inside a CFQUERY, the ListQualify function is smart enough to escape any single quotes inside a list item, but retain the single quote delimiters between items. For this reason, ListQualify is superior and more secure than the PreserveSingleQuotes function, which cannot distinguish between single quotes in items and single quotes between items.

Warning

Some texts recommend that you put single quotes around each item in the form, then use the PreserveSingleQuotes function to build the SQL IN clause. This technique leaves the door wide open to SQL injection attacks (see Lesson 9: Validating Form Data) and should never be used. Use ListQualify instead to safely build an IN clause containing string items.

If a list item contains a comma, the ListQualify function will treat it as two list items in this example because the comma is interpreted as the item separator. For this reason, you should not use a comma in the value of a checkbox or multi-select list option unless you’re doing something especially clever.

Using Dynamic Query Structure

The ability to use variables inside queries gives you a lot of flexibility, but more sophisticated applications require you to alter the structure of a query as well.

For example, suppose you have an orders database and a search form which allows you to search on several criteria, including shipping method. You might have a drop-down list box which lists each method. It would be convenient to allow the user to select “All shipping methods” as well. Your drop-down list box might look like this:

```
<SELECT NAME=”ShipVia” SIZE=”1”>
      <OPTION SELECTED>All
      <OPTION>FedEx
      <OPTION>Airborne
</SELECT>
```

If the user selects a shipping method, your query would look like this:

```
SELECT *
      FROM Orders
      WHERE ShipVia = ‘#Form.ShipVia#’
```

If the user selects “All,” your query would look like this:

```
SELECT *
      FROM Orders
```

Fortunately, ColdFusion provides a way to create the structure of a query on the fly. Inside a CFQUERY tag, you can use CFIF / CFELSEIF / CFELSE. So you would write

```
<CFQUERY NAME=”GetOrders” ...>
SELECT *
      FROM Orders
<CFIF Form.ShipVia IS NOT “All”>
      WHERE ShipVia = ‘#Form.ShipVia#’
</CFIF>
</CFQUERY>
```

What happens if you have more than one search term, and you're not sure which of the terms will be included? You may not have a WHERE clause at all, in which case the structure of the query will result in a syntax error.

The solution is to use a cool SQL trick. It costs the database nothing, but makes it much easier to write queries involving many possible search terms. It looks like this:

```
<CFQUERY NAME="somequery" ...>
SELECT *
      FROM Orders
      WHERE 0=0
<CFIF Form.ShipVia IS NOT "All">
      AND ShipVia = '#Form.ShipVia#'
</CFIF>
</CFIF Form.Customer IS NOT "All">
      AND Customer = '#Form.Customer#'
<CFIF>
...
</CFQUERY>
```

Walkthrough 7-2 Expand the Customer Search Application

Setup

In ColdFusion Studio, open the WebFastTrack project.

Steps

1. Open CustomerForm.cfm and add a text input field named City.
2. Save the page.
3. Open CustomerSearch.cfm and introduce conditional logic to the query to add all non-empty fields to the WHERE clause.
4. Save the page.
5. Browse CustomerForm.cfm to try out the new field.
6. Merge CustomerForm.cfm into CustomerSearch.cfm by putting the form in a table row (after the column headers).
7. Use a named submit button to determine whether to display the form only or run the query also.
8. Display the values of the previous search criteria in the form.
9. Set default parameters for the form fields so the page loads correctly when there is no query.
10. Add a named submit button to clear the current search.

Using Query Results

CFQUERY Properties

RecordCount	The total number of records returned by the query.
CurrentRow	The current row of the query being processed.
ColumnList	Returns a comma-delimited list of the columns.

The ColumnList property is especially useful when you don't know the column names in advance, such as when you SELECT *.

After each query, you can also access the query's execution time in milliseconds with the variable CFQUERY.EXECUTIONTIME.

Grouping Query Output

The GROUP attribute is used when you want to generate multiple HTML tables from a single query (say, one table for each category by grouping on the Category column). Less commonly, GROUP is useful when it takes several query rows to make one HTML table row.

Note that the CFOUTPUT GROUP attribute is quite different than the SQL GROUP BY clause. The SQL GROUP BY clause is used for defining the group of data on which an aggregate function will operate, whereas CFOUTPUT GROUP is merely a convenient way to determine when you are moving to the next "group" of records in a query recordset.

To use the GROUP attribute, your query must include an ORDER BY clause which names the column you will group on.

```
<CFQUERY NAME="qProducts" ...>
SELECT Category, ProductName FROM Products
ORDER BY Category
</CFQUERY>


<CFOUTPUT   QUERY="qProducts"
            GROUP="Category">
Products in #qProducts.Category#
<TABLE>
      <CFOUTPUT>
      <TR><TD>#qProducts.ProductName#</TD></TR>
      </CFOUTPUT>
</TABLE>
</CFOUTPUT>
```

Returning Partial Recordsets

You can return a partial recordset with the CFOUPTUT STARTROW and MAXROWS attributes. This is useful for paging through a set of 500 results, say, 50 at a time. Your application can remember the last row it displayed (CurrentRow) and pass this value to the next page to use in the STARTROW attribute.

The problem with this approach is that the entire query is retrieved from the database and cached in memory on the ColdFusion server. For this reason, you should not use CFOUTPUT STARTROW and MAXROWS to page through a large query (more than 1000 records).

CFOUTPUT MAXROWS vs. CFQUERY MAXROWS

To more efficiently limit the size of a query, use the CFQUERY MAXROWS attribute. This tells ColdFusion to stop retrieving records from the database altogether after the limit has been reached and is therefore more efficient.

Practical applications which must page through large recordsets may use both techniques. Use CFQUERY MAXROWS to limit the total number of records to, say, 1000, then use CFOUTPUT STARTROW and MAXROWS to page through the resulting query a screen-full at a time.

Nesting Queries with CFLOOP

You cannot nest two CFOUTPUT statements unless the outer one specifies the GROUP attribute (see previous page). However, it is possible to nest query output or even queries using CFLOOP.

You can loop over a query variable (recordset) programmatically using the CFLOOP tag. The following example demonstrates how you can include output from two different queries within a single CFOUTPUT.

```
<CFOUTPUT QUERY=”qEmployees”>
...
      <CFLOOP QUERY=”qAddresses”>
      #qAddresses.Street1# #qAddresses.City#
      </CFLOOP>
...
</CFOUTPUT>
```

Note that whenever you use a query variable which is not specified in a CFOUTPUT, you must prefix the column name with the query name.

Where to Go From Here

Application Help

See the following chapters in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 3: Querying a Database
Chapter 4: Retrieving and Formatting the Data You Want

Lesson 8
Creating Dynamic Forms

Introduction

Just as you can use URL and Form variables as parameters in dynamic queries, so you can use queries to build dynamic forms.

Objectives

By the end of this lesson, you should be able to

- Build dynamic HTML forms using database queries
- Use special ColdFusion form elements

Building Dynamic HTML Forms

Many times, the values of form elements will themselves come from a database, such as a list of states or countries in a list box or group of checkboxes.

You can dynamically create any HTML form element using CFOUTPUT to read values from a query.

Text and Password Boxes

You may want to create a text or password box with a default value obtained from the database. For example, suppose you have a form which allows customers to modify their profile and you want to automatically fill in their last saved information.

```
<CFQUERY NAME="qCustomerInfo" ...>
SELECT *
      FROM Customers
      WHERE CustomerID = 338
</CFQUERY>

<FORM ACTION="UpdateCustomer.cfm"
      METHOD="POST">
<CFOUTPUT>
Your dog's name:

<INPUT TYPE="text"
      NAME="dog_name"
      VALUE="#qCustomerInfo.DogName#">
</CFOUTPUT>
</FORM>
```

Note

You only need to specify a QUERY attribute in CFOUTPUT when you will be looping over more than one row. When you want to include the results of several queries having only one row, it is more convenient and more readable to use a single CFOUTPUT and use the query name in the variable reference.

Radio Buttons and Checkboxes

To create a group of radio buttons or checkboxes from a database query, simply loop over the query using CFOUTPUT.

```
<CFQUERY NAME="qStates" ...>
SELECT StateCode, StateName
      FROM States
</CFQUERY>


<FORM ...>
<CFOUTPUT QUERY="qStates">
<INPUT TYPE="radio"
      NAME="State"
      VALUE="#qStates.StateCode#">
      #qStates.StateName#
</CFOUTPUT>
</FORM>
```

Oftentimes, it is useful to show as selected an item which a user has previously chosen. You can easily do this using conditional logic as shown here to create a group of checkboxes.

```
<FORM ...>
<CFOUTPUT QUERY="qStates">
<CFIF CurrentState = qStates.StateCode>
      <INPUT TYPE="checkbox"
            NAME="State"
            VALUE="#qStates.StateCode#"
            CHECKED>#qStates.StateName#
<CFELSE>
      <INPUT TYPE="checkbox"
            NAME="State"
            VALUE="#qStates.StateCode#">
            #qStates.StateName#
</CFIF>
</CFOUTPUT>
</FORM>
```

Drop-down Lists

Drop-down lists work the same as radio buttons and check boxes. Here is an example showing a dynamically-generated drop-down list.

```
<FORM ...>
<SELECT NAME=”State” SIZE=”1”>
<CFOUTPUT QUERY=”qStates”>
      <OPTION VALUE=”#qStates.StateCode#”>
            #qStates.StateName#
</CFOUTPUT>
</FORM>
```

Scrolling Lists

Scrolling lists differ slightly if there can be multiple values selected and you want to show which ones are selected currently. The easiest way to implement this is to create a comma-separated list of the currently selected values. Recall that if you reading from a multi-valued form variable, ColdFusion automatically builds this list. For each record in the query, you use the ListFind function to find out whether it's in the list of currently-selected items.

```
<CFSET SelectedStates = “IA,KS,NE”>
<FORM ...>
<SELECT NAME=”State” SIZE=”10”>
<CFOUTPUT QUERY=”qStates”>
<CFIF ListFind (“#Variables.SelectedStates#”,
“#qStates.StateCode#”)>
      <!--- Show state as selected --->
      <OPTION VALUE=”#qStates.StateCode#”
            SELECTED>#qStates.StateName#
<CFELSE>
      <!--- Do not show as selected --->
      <OPTION VALUE=”#qStates.StateCode#”>
            #qStates.StateName#
</CFIF>
</CFOUTPUT>
</FORM>
```

Tip

The CFSELECT tag offers a shortcut for building SELECT lists from a database query. However, if you need to add OPTION tags which are not in the database, or need to perform a complex comparison to determine which item is selected, then you must use the method above.

Building Dynamic ColdFusion Forms

ColdFusion forms offer an assortment of tags which use Java applets to build more dynamic controls. These include

- CFGRID
- CFSLIDER
- CFTREE and CFTREEITEM
- CFTEXTINPUT
- CFAPPLET

CFGRID is a mini-application in a tag. It creates a view of a database table which you can use to add, update, and delete records, and gives quite a bit of control over formatting.

CFTREE is a very cool tag. Besides creating a neat display, it automatically builds the tree structure from the relationships specified in the CFTREEITEM tag, regardless of the order of the CFTREEITEM tags.

Two tags are almost identical to their HTML counterparts, but allow additional client-side validation methods. These are

- CFINPUT (for radio buttons, check boxes, text, and password boxes)
- CFSELECT (shortcut way to build a select list from a query)

If you use CFINPUT or CFSELECT and look at the resulting HTML source code, you will see that ColdFusion uses the regular HTML tag, but includes a JavaScript event handler and JavaScript validation function for you.

Walkthrough 8-1 Create a Group of Checkboxes from a Query

Setup

In ColdFusion Studio, open the WebFastTrack project. In this walkthrough, you'll create a page to list products in selected categories. The list of categories will come from a database query.

Steps

1. Create a new page using the default template.
2. Create a form using the POST method and ACTION=”ListProducts.cfm.”
3. Save the page as ListProducts.cfm.
4. Create a new query to select all rows from the Northwind Categories table. Save the query as ListCategories.
5. In the form, build a group of checkboxes from the ListCategories query. Don't forget a submit button.
6. Save the page and browse it.
7. Now create a query to list all products in one of the checked categories. Save the query as ListProducts.
8. Add code to run the query and display results if at least one category is checked (hint: use the IsDefined function).
9. Save the page and browse it.
10. Now add logic to display a checkbox as checked if it was selected in the current query (hint: use the ListFind function).

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 10: Building Dynamic Forms

Lesson 9
Validating Form Data

Introduction

Every computer program should attempt to validate data input. However, this is especially important on the Web to prevent hackers from compromising security. Before JavaScript, Web applications could only use server-side validation. Now, however, you can use a combination of client-side and server-side validation. ColdFusion provides helpful capabilities for performing common validation tasks.

Objectives

By the end of this lesson, you should be able to

- Understand why it's important to validate user input
- Explain client-side vs. server-side techniques
- Use ColdFusion capabilities for validating input

Why Validate?

One of the great annoyances of Web programming is that users with temperaments ranging from ornery to malicious can view source and make their own versions of any page. This includes Web forms, which means you have no guarantee that a GET or POST request to URL came form your form at all. Worse, a programmer might purposefully write a utility to submit all kinds of bad data to various URLs on your Web site. In ColdFusion, this is almost a trivial task. Such a program is not a bad idea for automated testing, but malicious users are not likely to be in your corner when they find a bug.

Chandler's First Law of Web Applications Security

Never assume that form data came from your form.

Web application developers often forget that a page request doesn't necessarily come from where you expect it. Anyone can request pages out of order, remove form fields, add new ones, or change their contents, and alter every piece of information sent by the browser, including cookies and the referring page. Man-in-the-middle tools like Achilles make it very easy for hackers to intercept each browser request and to view and modify every piece of information being sent to the server. For this reason, you can never assume that your pages will be accessed only in the manner that you intended.

There are basically two reasons to validate all form fields:

1) Avoid errors
2) Protect data

Avoid Errors

Although ColdFusion provides a lot of useful ways to handle errors, it's still better to avoid them in the first place. Typical causes of program errors include

- Parameter not found. Your program attempts to use a URL or Form variable which has not been defined.
- Incorrect data type. Your program attempts to use do math on a Form variable which is actually a string.
- SQL syntax error. Your program has used a numeric variable in a database character field or vice versa.

ColdFusion provides a variety of functions useful for data validation. Some of the most common are listed below.

Function	**Description**
isDefined	Check to see if a variable exists.
isNumeric	Check to see if a variable is a number.
isDate	Check to see if a variable is a valid date.
Find	Find a sub-string in a string.
REFind	Find a regular expression in a string.
ListFind	Find an item in a list.
ListContains	Find a sub-string in a list.

Note

The ListContains function looks for a sub-string within an item in a list. If someList = "38472,40273,38127" and you write ListContains (someList, "12"), the function will return TRUE. Ordinarily, you will use ListFind instead, which would return FALSE.

Protect Data

Here's a quick case study to convince you to check your variables before using them in a query. Suppose you have a search form which lets the user type in an order number to retrieve the order from the database. In your ColdFusion program, you will have a statement that looks like this:

```
SELECT *
      FROM Orders
      WHERE OrderNum = #Form.OrderNum#
```

Let's say the order number is a text input box with MAXLENGTH=5. This limits the user to five characters, right? Wrong. The user can save their own version of the form and take out MAXLENGTH altogether. Then, in the order number text box, they might enter

```
24883; DELETE FROM Orders
```

ColdFusion will send this to the database:

```
SELECT *
      FROM Orders
      WHERE OrderNum = 24883
;
DELETE FROM Orders
```

Since SQL allows you to submit multiple statements with a semi-colon, this is a perfectly valid statement. Unless the database login used by your application prevents DELETE operations, you're in trouble. This type of attack is known as *SQL injection*, and is covered in depth in Lesson 23: Designing Secure Applications.

Numeric fields are especially dangerous. Character fields are safer because they are delimited by single quotes and ColdFusion automatically escape single quotes in variables when used inside a CFQUERY. Thus, an attacker cannot change the structure of the SQL statement. However, as an extra precaution, you may want to use Find and related functions to check for semi-colons.

You have three defenses against this type of attack:

1) Use the IsNumeric () function to verify that your form variable really is a number.
2) Use CFQUERYPARAM to check the data type.
3) Use a database stored procedure because arguments to stored procedures are strongly typed.

Client-Side Validation

Client-side validation is done using code (HTML or JavaScript) which runs in the browser. Use client-side validation to

1) Prevent typos and otherwise help the user to understand what data is valid
2) Save a trip to the server if invalid data is entered

Client-side validation takes basically one of two forms.

Form Elements Attributes

You can do limited validation by using form elements themselves. Radio buttons, check boxes, and lists restrict user input and therefore perform validation inherently.

For text and password boxes, you can use the MAXLENGTH attribute to limit input to a certain number of characters.

Scripting

For more complex validation, you must use a scripting language on the browser (Netscape's JavaScript or Microsoft's JScript or VBScript).

JavaScript and JScript validation techniques typically use one of the following event handlers:

OnChange – triggered when the user tabs or clicks off the form element and the value has changed
OnSubmit – triggered when the form is submitted

Validation scripts typically use the alert () and/or prompt () methods to inform the user that invalid data has been entered.

Server-Side Validation

Client-side validation is useful, but it can never be a substitute for server-side validation (refer to Chandler's First Rule of Defensive Web Programming). Even if you use client-side validation, you should server-side validation to absolutely, positively, guarantee the quality of your data.

Here are some common applications for server-side validation.

Check URL and Form Variables to Make Sure They're Defined

Before you reference any URL or Form variable for the first time, you should make sure it is defined. ColdFusion provides two functions which are particularly helpful for doing this.

The IsDefined () function lets you check to make a variable is defined before using it. Note that the argument to IsDefined is the name of a variable enclosed in quotes, not the variable reference itself. You can use an expression like this to set a default value for any variable:

```
<CFIF NOT IsDefined (“Form.State”)>
      <CFSET Form.State = “CA”>
```

ColdFusion also provides a shortcut for this expression in the CFPARAM tag. The equivalent CFPARAM expression is

```
<CFPARAM name=”Form.State” default=”CA”>
```

Check All Variables Used in Queries

As an earlier example demonstrated, dynamic parameters can be very dangerous in ColdFusion queries. It is a good idea to make sure that variables do not contain semi-colons or other expressions which could be used to modify the SQL statement.

Use the IsNumeric function to validate numeric variables. Text variables are trickier. If someone attempted to append or modify an SQL statement by including bogus data in a character string literal (WHERE State = ‘#Form.State#’), they would have to include at least one single quote in their input to avoid creating an SQL syntax error. Since ColdFusion automatically replaces single quotes in variables with double single quotes (‘’), there is no danger.

However, if you use the PreserveSingleQuotes function to handle comma-delimited character values, you should check the form variable to make sure it doesn't contain any semi-colons, and preferably to make sure each value is valid. You should perform the same check on each value in a list of numeric values.

Be Careful with Evaluate

Evaluate is a ColdFusion function which lets you write code and run it on the fly. It can be useful, but must be used cautiously. You should only use Evaluate when the range of input is well-understood and validated.

Warning

Never evaluate a URL or Form variable without first validating it. In almost every case, you can use structure notation instead of Evaluate to get the value of a dynamically-named variable (see example below).

You may have an application where you cannot know in advance the names of all the form variables. For example, you might generate a query page which generates a listing of customers. The listing might contain a checkbox allowing you to mark each customer as active or inactive. Since you don't know in advance which customers will be in the query, you cannot assign static names to the HTML checkboxes. Suppose you assign them dynamically from the query using the Customer_ID field.

When the form is submitted, the ColdFusion program which processes the form could get a list of all the form variables using Form.Fieldnames and build an SQL statement using this data. However, this approach might allow the user to modify the form and submit an invalid ID.

The better approach would be to query the database for a list of customers which could have been in the query, and then test for the presence of a corresponding form variable using the IsDefined function. If the form variable exists and is numeric, it can be safely included in a SQL statement. The example below shows how to correctly loop over a series of variables named ZIP1, ZIP2, and ZIP3 and test each to see if it is numeric.

```
<cfloop index="i" from="1" to="3">
      <cfset theValue = Form["ZIP" & i]>
      <cfif NOT isNumeric (Variables.theValue)>
            The field is not numeric.
      </cfif>
</cfloop>
```

Note

If you want to loop over all form variables, use CFLOOP with the attribute COLLECTION=#Form#.

Using ColdFusion Validation

In HTML Forms

You can use hidden form fields to indicate required fields and do server-side validation on types

- Integer
- Float
- Range
- Date
- Time
- Eurodate

Note

In order to use hidden form fields to indicate required and validated fields, you must use the POST method in your FORM tag.

In ColdFusion Forms

In a CFFORM, you can specify your own JavaScript function in the ONVALIDATE attribute of one of the ColdFusion form elements like CFINPUT, CFGRID, CFSLIDER, CFTEXTINPUT, and CFTREE.

You can also enable server-side validation using the VALIDATE attribute of CFINPUT and CFTEXTINPUT. ColdFusion forms offer more advanced validation types than HTML forms, including

- Date
- Eurodate
- Time
- Float
- Integer
- Telephone
- Zipcode
- Creditcard
- Social security number

Lab 9-1 Experiment with Data Validation

Setup

Open the WebFastTrack project in ColdFusion Studio. This is a free-form lab allowing you to experiment with different types of validation.

Steps

1. Create a new page containing a CFFORM and a new page to handle the form submission.
2. Create various types of form inputs and use hidden fields to indicate the required variables.
3. Use hidden fields to validate a text input field as an integer or date.
4. Create a CFINPUT and use the ONVALIDATE attribute to validate a telephone number or credit card number.

Where to Go from Here

Application Help

See the following chapters in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 6: Updating Your Data
Chapter 10: Building Dynamic Forms

Lesson 10
Beyond Queries: Modifying Data

Introduction

Thus far, we have used the CFQUERY tag for database queries only. However, in ColdFusion, you can use the CFQUERY tag to send any statement to the database, including inserts, updates, and deletes.

Objectives

By the end of this lesson, you should be able to

- Insert a record using CFQUERY
- Insert a record using CFINSERT
- Update a record using CFQUERY
- Update a record using CFUPDATE
- Delete a record using CFQUERY

Inserting Data

Inserting with CFQUERY

You can use dynamic parameters (ColdFusion variables) in an SQL INSERT statement just like you can use them in a SELECT statement.

The general form for an SQL INSERT is

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...)
```

For example, if you created an insert form with form inputs for each field in the database, you would put something like this in the page which processes the form:

```
<CFQUERY NAME="AddEmployee" ...>
INSERT INTO Employees (FirstName, LastName)
VALUES ('#Form.FirstName#', '#Form.LastName#')
</CFQUERY>
```

Note

When using an SQL INSERT statement, you must specify a value every field in the table except for fields which allow NULLs or fields which have default values.

Inserting with CFINSERT

ColdFusion provides a handy shortcut for adding records. To use CFINSERT, you simply name the form fields the same as the database fields. CFINSERT will automatically map between the two. The previous example using CFINSERT looks like this:

```
<CFINSERT DATASOURCE="EmployeeDB"
      TABLENAME="Employees">
```

That's it! Before you get too excited, however, keep in mind that CFINSERT does not do any data validation. Therefore, you should still validate the form variables before calling CFINSERT.

Walkthrough 10-1 Add a New Employee Using CFINSERT

Setup

Open the WebFastTrack project in ColdFusion Studio. In this walkthrough, you will create a form and form handler to add a new employee to the Northwind database.

Steps

1. In SQL Server Enterprise Manager, find out which fields are required in the Northwind Employees table.
2. Create a new page for the new employee form.
3. Create a form with a FirstName and LastName field. Make these fields required using hidden variables.
4. Save the form page as NewEmployee.cfm.
5. Create a page using CFINSERT to handle the form submission.
6. Save this page as AddEmployee.cfm.

Updating Data

Updating with CFQUERY

The general form for an SQL UPDATE statement is as follows:

```
UPDATE table_name
SET   column1 = value1,
      column2 = value2,
      ...
WHERE some_condition
```

Typically, the WHERE clause uses the primary key such as a record ID to update the correct record. You can emebed the record ID in an update form as a hidden variable so the user is not even aware of it. An update statement for the Employees table might look like:

```
<CFQUERY NAME="UpdateEmployee" ...>
UPDATE Employees
SET   LastName = '#Form.LastName#',
      FirstName = '#Form.FirstName#
WHERE EmployeeID = #Form.EmployeeID#
</CFQUERY>
```

Unlike INSERT, UPDATE does not require a value for every field, but updates only those fields specified.

Updating with CFUPDATE

ColdFusion provides a tag to make update brain-dead easy, too. The previous example would be written

```
<CFUPDATE DATASOURCE="EmployeeDB"
      TABLENAME="Employees">
```

ColdFusion automatically finds the primary key in the database, looks for a form field with the same name, and uses its value in the WHERE clause to update the correct record. As with CFINSERT, you should do your own data validation first.

Walkthrough 10-2 Create an Employee Update Form

Setup

Open the WebFastTrack project in ColdFusion Studio. In this walkthrough, you will create a page listing all employees, then an employee update form and form handler.

Steps

7. Create a new page with a query to list all employees in the Northwind Employees table.
8. Display the results of the query in a table.
9. Create a hyperlink column in the table. The hyperlink should go to the EmployeeDetail.cfm page and pass the Employee ID as a URL parameter.
10. Save the page as ListEmployees.cfm.
11. Create a new page to display all information for the selected employee in a form, where the first name and last name are editable fields.
12. Create a hidden field in the form containing EmployeeID.
13. Create a Modify button which links to UpdateEmployee.cfm.
14. Save this page as EmployeeDetail.cfm.
15. Create the update page using CFUPDATE.
16. After the update, use CFLOCATION to direct the browser back to ListEmployees.cfm.
17. Save this page as UpdateEmployee.cfm.

Deleting Data

ColdFusion does not provide a CFDELETE tag because it's pretty simple in CFQUERY. The general form for an SQL DELETE statement is

```
DELETE FROM table_name
WHERE some_condition
```

To delete a record from the Employees table, you would write

```
<CFQUERY NAME="DeleteEmployee" ...>
DELETE FROM Employees
WHERE EmployeeID = #Form.EmployeeID#
</CFQUERY>
```

Warning

Be especially careful to validate form variables used in DELETE statements!

It goes without saying that delete operations should be handled carefully. If you forget the WHERE clause, for example, you will delete every record in the table.

To revisit an earlier security example using the example above, suppose that a clever user modifies the hidden field containing EmployeeID and replaces "345" with "345 OR 0=0". The statement sent to the database will now be

```
DELETE FROM Employees
WHERE EmployeeID = 345 OR 0=0
```

Note that a malicious user can delete every record in the table without even using a semi-colon!

Walkthrough 10-3 Delete an Employee Record with Validation

Setup

Open the WebFastTrack project in ColdFusion Studio. In this walkthrough, you will add a Delete button to the employee detail page.

Steps

1. Open EmployeeDetail.cfm.
2. Add a Delete button to the form and save the page.
3. Open EmployeeUpdate.cfm.
4. Add a new section to delete a record if the Delete button has been pressed. Be sure to validate EmployeeID using the IsNumeric function.
5. After the delete, use CFLOCATION to redirect the browser to ListEmployees.cfm.
6. Save the page and test it out.

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 6: Updating Your Data

Unit 3
Developing Real-World Applications

Unit Overview

Now that you've learned the basic of ColdFusion programming, you're ready to explore to more advanced features of the ColdFusion programming environment. In this unit, you'll learn the ColdFusion programming language and environment.

Goals

By the completion of this unit, you should be able to

- Use ColdFusion expressions and data structures effectively
- Run the interactive and non-interactive debugging tools
- Build applications using the ColdFusion application framework
- Create reusable, modular custom tags
- Build a generic error handling framework

Lesson 11
The ColdFusion Language

Introduction

ColdFusion offers a rich programming environment with most all of the concepts from other high-level languages. The combination of these features with the simplicity of CFQUERY and other tags makes for a very flexible programming environment.

Objectives

By the end of this lesson, you should

- Understand the different kinds of ColdFusion variables
- Know how to control page flow using ColdFusion control structures
- Be able to use functions and expressions
- Understand the uses and pitfalls of dynamic expressions
- Know how to use regular expressions
- Be able to work with arrays and structures

ColdFusion Variables

You may want to refer to *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Creating and Using Variables

See Chapter 2: Writing Your First ColdFusion Application

- Create variables with CFSET
- Display variables with CFOUTPUT

See Chapter 5: Making Variables Dynamic

- Test for a variable's existence with IsDefined ("var_name")
- Create default variables in any scope with CFPARAM

Naming and Scoping Variables

Variable Names

- Must begin with a letter
- Can be followed by letters, digits, and _
- Not case-sensitive!

ColdFusion 5 Variable Scopes	
Application	Apply to all pages having same CFAPPLICATION tag
Arguments	Holds arguments passed to a user-defined function
Attributes	Used inside custom tags to get passed-in attributes
Caller	Used inside custom tag to refer to vars in calling page
CGI	Set by Web server. Turn on server debugging to see all.
Client	Long-term storage for individual clients (browsers)
Cookie	Browser cookies. Set with CFCOOKIE tag.
Form	Variables in an HTML form submitted via POST method
Query	Referenced by query name
Request	Apply to entire browser request, including custom tags
Server	ColdFusion Server info. Turn on debugging to see all.
Session	Apply to one browser session, subject to server timeout
URL	Variables in an HTML form submitted via GET method
Variables	Local variables used in a page. Prefix not necessary.

Tip

For clarity, performance, and security reasons, you should explicitly scope all variables, even those in the Variables scope.

Using Pound Signs
See *CFML Language Reference*, Chapter 5: Operators

Controlling Page Flow

Conditional Processing

CFIF / CFELSEIF / CFELSE

CFSWITCH / CFCASE / CFDEFAULTCASE

Note: Unlike C, ColdFusion does not require a break statement after each CFCASE.

Redirecting Application Pages with CFLOCATION

Tip: You can also redirect pages using client-side JavaScript, but doing it on the server will work with all browsers.

Stopping Application Pages with CFABORT

Note: Outside of custom tags, CFEXIT works exactly like CFABORT.

Including Application Files with CFINCLUDE

CFINCLUDE inserts another file into the current page. The important thing to remember about CFINCLUDE is that all included ColdFusion tags or scripts run as if they are in the main page. CFINCLUDE does not create a separate context for local variables like custom tags do.

Creating Loops

Index Loops (FOR)

Condition Loops (WHILE)

Query Loops

Looping over a List

Looping over a Structure or COM Collection

Nesting Loops

Note: you can use nested loops to nest queries, but you should avoid it.

Breaking out of a Loop with CFBREAK

Infinite Loops

Expression Terms

About ColdFusion Expressions

See *CFML Language Reference*, Chapter 5: Operators

Hint

Use the ColdFusion Studio Expression Builder to help you remember operators, functions, constants, and variable names.

Strings

The empty string is “”.

To use a single quote (‘) in a string, use double quotes around the string:

```
<CFSET Title=“John’s World”>
```

To use double quotes in a string, use single quotes around the string:

```
<CFSET Line=’She said, “Hello”’>
```

If you must use a single quote in a single-quoted string, escape it by using two single quotes. The same goes for double-quoted strings.

```
<CFSET Title=’John’’s World’>

<CFSET Line=“She said, ““Hello”””>
```

Boolean Values

1 = TRUE = “Yes”
0 = FALSE = “No”

Functions

Note that many functions have optional arguments.

Operators

Overview

See *CFML Language Reference*, Chapter 5: Operators.

It's FORTRAN! No, it's COBOL! No, it's ColdFusion!

Operator Precedence

Tip
As in all languages, use parentheses!

Functions

See *CFML Language Reference*, Chapter 3: ColdFusion Functions.

Using Pound Signs

In CFOUTPUT and Strings

In general, you only need pound signs around variables in a CFOUTPUT section or string. Inside a ColdFusion tag, you can use variables without pound signs.
To escape a pound sign (#) in a string or CFOUTPUT section, use ##.

```
<CFOUTPUT>Enter PIN, then press ##.</CFOUTPUT>
```

In Tag Attribute Values

If the value of a tag attribute is a variable, function, or array element, Allaire recommends the following syntax:

```
<CFCOOKIE Name="User" Value=#Form.Username#>
```

This is more efficient than using quotes around the attribute value.

Nested Pound Signs

Don't do it. There's always a better way to write it.

Dynamic Expressions

Evaluating Dynamic Expressions

Use the Evaluate function to dynamically evaluate any expression. It is useful when you want to do computation inside a ColdFusion tag or function, as in this simple example:

```
<CFOUTPUT>
2 + 2 = #Evaluate(2+2)#
</CFOUTPUT>
```

Use SetVariable to cleanly assign a value to a variable whose name is dynamic. This example code creates variables named Var1, Var2, and Var3.

```
<CFLOOP index="i" from="1" to="3">
<!--- Could also Evaluate("Var#i# = i*2"); --->
      <CFSET SetVariable ("Var#i#", i*2)>
</CFLOOP>
```

A better way to assign dynamically-named variables is to use structure notation (covered later in this lesson). This is safer and faster than using Evaluate, and allows you to delete all related variables with a single statement (StructDelete). In ColdFusion, the Form and URL scopes are actually structures, so you can easily reference dynamically-named form fields. This example code prints the name and value of all variables in a form. Note the special syntax used to pass the entire Form structure in the collection attribute of the CFLOOP tag.

```
<CFLOOP collection=#Form# item="thisVarName">
      <CFSET thisValue = Form[thisVarName]>
      <CFOUTPUT>
      #thisVarName# = #thisValue#<BR>
      </CFOUTPUT>
</CFLOOP>
```

Caution

As in all languages, be very careful when using Evaluate (), especially to process user input of any kind. For security and performance reasons, you should use it only when absolutely necessary.

Regular Expressions (Pattern Matching)

See *Using ColdFusion Studio*, Chapter 12: Testing and Maintaining Web Pages

There are four ColdFusion functions which allow the use of regular expressions. These are

- REFind
- REFindNoCase
- REReplace
- REReplaceNoCase

ColdFusion offers most of the regular expressions functionality in Unix or perl, with one important difference: ColdFusion regular expressions have no notion of special characters like \n (newline), \t (tab), etc. To match a newline in ColdFusion regular expressions, you must use the ASCII equivalent of the newline as follows:

```
REFind (“[#Chr(10)##Chr(13)#]”, string, ...)
```

Tip

To reference a newline or other special character, use the Chr function to create it from its ASCII value.

This technique is useful for parsing files. Using the CFFILE tag, you can read a file into a ColdFusion variable. You then treat the file as a ColdFusion list, delimited by the newline sequence, Chr(10) & Chr(13). To read each line of the file one at a time, you simply loop over the list. This example reads a file and prints each line, one at a time.

```
<CFFILE action="READ"
      file="D:\TEMP\Active setup log.txt"
      variable="myFile">
<CFSET newline = Chr(10) & Chr(13)>
<PRE>
<CFLOOP index="oneLine"
      list="#myFile#"
      delimiters="#newline#">
      <CFOUTPUT>#oneLine#</CFOUTPUT>
</CFLOOP>
</PRE>
```

Lists

ColdFusion lists are surprisingly useful animals. A list is just a group of items in a string delimited by some character, typically a comma. Lists derive their usefulness from three sources:

1) You can loop over them
2) Multi-valued form values are represented as lists
3) ColdFusion provides a handful of list functions

A list is similar to an array, and can be converted to and from an array. However, there are some things you can easily do with a list which would require you to loop over an entire array. An example is finding out whether a list contains some value, which you can do with the ListFind function.

List Functions

Among the most useful list-related functions are:

ListFind	Find the position of an item in a list
ListContains	Find a sub-string within a list
ValueList	Convert a query column to a list
QuotedValueList	Convert a query column to a quote-delimited list

Using Lists

In Forms

Probably the most common application for lists is when processing forms since multi-valued form variables (grouped check boxes and multiple select lists) return the selected values as a comma-separated list.

With Files

As demonstrated in the previous example, it can be useful to think of a file as a list delimited by some character, typically a newline. In fact, this is the only way ColdFusion provides to read a file line by line.

Arrays

See *Developing Web Applications with ColdFusion*, Chapter 9: Handling Complex Data with Structures

Quick Take for Programmers

1) ColdFusion arrays may have up to three dimensions.
2) ColdFusion uses brackets to index into arrays [*n*].
3) ColdFusion array numbering starts at [1].
4) ColdFusion arrays are dynamic. You do not have to declare the range in advance.
5) Multi-dimensional arrays are indexed like Array[1][2][3].

Using Arrays

To create an array, use the ArrayNew function, which takes as an argument the array's *dimension* (not the range).

```
<CFSET months = ArrayNew (1)>
```

To initialize an array, use the ArraySet function. The syntax is

```
ArraySet (array_name, startrow, endrow, value)
```

To populate an array, loop over the array indexes with CFLOOP.

```
<CFLOOP INDEX="j" FROM="1" TO="12">
      <CFSET months[j] = MonthAsString(j)>
</CFLOOP>
```

Copying Arrays

To copy an array, simply use an assignment statement like <CFSET array1 = array2>. You should avoid copying large arrays.

Structures

See *Developing Web Applications with ColdFusion*, Chapter 9: Handling Complex Data with Structures

ColdFusion structure are useful for several reasons:

1) The dot notation syntax makes it easy to group related information.
2) You can pass an entire structure to a custom tag.
3) You can use a structure as an associative array.
4) Application, CGI, Cookie, Form, URL, and Session variables are all structures.

Quick Take for Programmers

It's C! No, it's perl! No, it's ColdFusion!

1) ColdFusion structure elements can be referenced with a dot notation similar to C.
2) ColdFusion structures can be referenced with associative array notation similar to perl.

Creating Structures

No matter how you use a structure, you must create it with the StructNew function.

```
<CFSET Employee = StructNew ()>
```

Structures can be used to help segregate data in an object-oriented way. Instead of creating all variables in the local Variables scope, put them in structures named after the object class. This gives you several benefits:

1) It makes it easier to understand to which objects variables belong.
2) You can delete related variables with a single StructDelete statement.
3) You can easily copy all object properties with a single StructCopy statement. You might do this to persist object data to the Session scope, for example.

Tip

When you create a structure in the Variables scope, always refer to it the same way you created it. If you create it without the Variables prefix but try to reference it with the Variables prefix, ColdFusion will complain that the variable is undefined. This is not logically consistent with the way ColdFusion handles simple variables, but in effect you've created your own scope.

Adding to Structures

You can add to a structure one of three ways.

1) Use dot notation (Employee.FirstName).
2) Use associative array notation (Employee[“FirstName”]).
3) Use StructInsert.

All three statements below do the same thing. If you create a structure element one way, you can reference it in any other.

```
<CFSCRIPT>
Employee.FirstName = "David";
Employee["LastName"] = "Chandler";
StructInsert (Employee, “Email”,
“nospam@turbomanage.com”);
</CFSCRIPT>
```

An advantage of dot notation or associative array notation over StructInsert is that you can use the same statement to assign a new structure key or reassign an existing key. With StructInsert, you must specify allowOverwrite = “TRUE” to do this or you will get an error.

Finding Information in Structures

To find a value in a structure, you can use dot notation, associative array notation, or the StructFind function.

Note

You cannot use the dot notation to access a structure element with a dynamic property (key), as in Employee.#field_name# = David. Instead, use the array notation Employee[field_name].

Before you reference an element in structure, you may want to find out if it exists. The following functions are helpful.

IsStruct	Returns TRUE if the variable is a structure
StructCount	Returns the number of name-value pairs
StructIsEmpty	Returns true if the structure exists, but is empty
IsDefined	Test to see if the key exists

If the name of a key is dynamic, you cannot use the IsDefined function. Instead, use StructKeyExists.

Copying Structures

To create a duplicate copy of a structure, use StructCopy or Duplicate. Use assignment to create a copy by reference (pointer), as in

```
<CFSET copy_of_struct = struct1>
```

Use the Duplicate function to create a copy of any variable in its own memory space. Suppose you want to copy all Session variables into the local Variables scope so you don't have to worry about locking. You can do this by writing

```
<cflock scope="SESSION"
        type="READONLY"
        timeout="5">

      <cfset sCopyOfSession =
             Duplicate (Session)>

</cflock>
```

If you use StructCopy instead of duplicate, ColdFusion will give you an error when you try to access the local structure without a lock.

Deleting Structures

To delete an individual name-value pair in a structure, use StructDelete. If you don't specify the key, the whole structure will be deleted.

Note

ColdFusion Application and Session variables are actually structures. To delete one of the variables, you must use StructDelete.

Using Structures as Associative Arrays

To get a list of all structure keys, use StructKeyList or StructKeyArray. To loop over all keys in a structure, you can either loop over the list or array returned by these functions, or use CFLOOP with the COLLECTION attribute. The following code displays the value of all Session variables:

```
<cfloop collection=#Session#
        item="thisVarName">

      <cfset thisValue = Session[thisVarName]>

      <cfoutput>

      #thisVarName# = #thisValue#<BR>

      </cfoutput>

</cfloop>
```

Nested Data Structures

In ColdFusion, you can create complex data types by nesting the elementary data types to create structures of structures, arrays of queries, etc. To create an array of structures, you could write:

```
<cfset matrix = ArrayNew (1)>
<cfset ArraySet (matrix, 1, 5, StructNew ())>
```

Using Array Notation with Queries

A ColdFusion query variable is simply a structure of arrays. Each column in the query is a property (or key) of a structure, which is itself an array containing the data for each row. Therefore, you can use structure and array notation to access a specific cell in a query. The syntax is

Query_name[*col_name*][*row_number*]

Getting a List of all Values for a Query Column

Inside a CFOUTPUT or CFLOOP, if you reference *Query_name.field_name*, you will get the value for the current row in the loop. Outside one of these tags, however, you will get the array containing the field values for all rows. If you want a list instead of an array of values for a column, you can use the ValueList function. In the example below, the ListFind function is used with ValueList to make sure that a form variable is a valid selection:

```
<CFSET validLocs = ValueList (qLocations.ID)>
<CFIF NOT ListFind (validLocs,
                    Form.LocationID)>
      <CFABORT>
</CFIF>
```

Building a Structure of all Values for a Row

In some cases, you may want to copy a query row into a structure. This is useful when you want to pre-fill HTML form values from a database record. To do this, you loop over the query columns as follows:

```
<cfset sLoc = StructNew()>
<cfloop  list=#qLocations.ColumnList#
         index="thisKey">
      <cfset sLoc[thisKey] =
             qLocations[thisKey][1]>
</cfloop>
```

Where to Go from Here

Application Help

References to ColdFusion documentation are included throughout this lesson.

See also the list of functions by group in the *CFML Language Reference.*

Books

Mastering Regular Expressions (O'Reilly)

Lesson 12 Debugging

Introduction

ColdFusion Server features the ability to include debugging information on every page. ColdFusion Studio features a powerful interactive symbolic debugger. Learning how to use these capabilities can save you countless hours of banging away in the dark.

Objectives

By the end of this lesson, you should be able to

- Configure ColdFusion Server for debugging
- Configure the Studio debugger
- Set breakpoints and watches
- Observe variables and expressions
- Step through lines of code
- Monitor recordsets

Debugging with ColdFusion Server

Server-Side Debugging for all Pages

When ColdFusion Administrator is configured for debugging, ColdFusion Server can display the following information after each page is processed:

- Variables (with ColdFusion MX, you can selectively display all scopes except local Variables)
- Query information, including the SQL statement, processing time, and number of records returned
- Page processing time, including a breakdown of how much time is spent in each include file and custom tag

Debugging Individual Pages

If you want to debug a single page, simply append to the URL "?mode=debug." This will display page processing time, all variables, and any query debug options configured in ColdFusion Administrator.

Restricting Debug Output

You can configure ColdFusion Server to display debugging information only to certain IP addresses. This is highly recommended.

Warning

For all Internet servers, you should restrict debug output to internal IP addresses only. Even if you have no debug options checked, a knowledgeable ColdFusion user can append "?mode=debug" to any page and potentially find exploitable weaknesses.

Debugging Tips

- Use <CFDUMP var=#var_name#> to display an entire array, structure, or query in a nested hierarchy you can collapse and expand by clicking with the mouse.
- Set a flag in Application.cfm or a config file to turn on and off your custom debug code globally.
- To debug one query only, include the DEBUG attribute in CFQUERY.
- In ColdFusion Administrator, select the Debugging option to log long-running queries. This will help you find problem pages.
- If a user has already moved past an error message on screen, you can look in \CFUSION\Logs\application.log to find the error.

Walkthrough 12-1 Enable Debugging in ColdFusion Server

Setup

Open Start | Programs | ColdFusion Server 4.0 | ColdFusion Administrator. In this walkthrough, we'll enable the server to display debugging information.

Steps

1. In ColdFusion Administrator, click on Debugging.
2. Check all boxes.
3. In a browser, open WebFastTrack\Solutions\ListEmployees.cfm.

Debugging with ColdFusion Studio

ColdFusion Studio's interactive symbolic debugger is a very powerful tool. With it, you can see the value of any expression or variable, set watches on variables, set breakpoints, step through code line by line, and monitor queries.

Once you have started the debugger in ColdFusion Studio, it remains active until you stop it. This way, you can debug applications which require a sequence of pages.

Getting Started

See *Using ColdFusion Studio*, Chapter 9: Debugging CFML Applications from Studio.

Enabling and Disabling Breakpoints

ColdFusion provides no documentation on breakpoints, so here goes.

If you click once in the gutter, it sets an enabled breakpoint. The debugger will always stop here. If you click again, it disables the breakpoint, but keeps the breakpoint there in case you want to come back to it. When you click the third time, ColdFusion removes the breakpoint.

In the Breakpoints tab in the Debug window, you can enable, disable, or remove breakpoints by right-clicking on the breakpoint.

Setting Conditional Breakpoints

If you right-click on a breakpoint in the Breakpoints tab of the Debug window, you can Edit Condition to create a conditional breakpoint. When the breakpoint is enabled, the debugger will only stop if the condition is true. The condition can be any expression like "myVar IS 4."

Wildcard breakpoints do not appear to work.

Walkthrough 12-2 Using ColdFusion Studio Debugger

Setup

In ColdFusion Studio, open the WebFastTrack Project.

Steps

1. In the WebFastTrack project, open OrdersByCustomer.cfm.
2. Set a breakpoint before <CFTABLE> by clicking on the left side of the gutter in the editor window.
3. In the Debug toolbar, click the Start / Continue tool.
4. Choose the localhost ColdFusion server and verify that the Start URL is correct (http://127.0.0.1/wft/Lab/OrdersByCustomer.cfm).
5. Click the Mappings tab to create a mapping for the WebFastTrack directory if one does not exist. Click Add and OK.
6. Check the box which says "Don't prompt for these settings…"
7. Click OK.
8. You should see a blue bar in the editor window when the debugger stops at the breakpoint. Explore the various tabs in the Debug window which pops up.
9. Click the Step In or Step Over buttons on the Debug toolbar to see how these work.
10. Click the Start / Continue button on the Debug toolbar to finish.
11. Click the Stop button on the Debug toolbar to end the debugging session.

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 8: Debugging and Error Handling

Also see the following chapter in *Using ColdFusion Studio*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 9: Debugging CFML Applications from Studio

Lesson 13
Using the Application Framework

Introduction

The ColdFusion application environment is far more than a programming language. Application-level settings, client state management, custom error handling, and Web server security integration make lighter work of many Web programming tasks.

Objectives

By the end of this lesson, you should be able to

- Use CFAPPLICATION to define application settings
- Set up client state management
- Understand Application, Session, and Client variables
- Define custom error handlers
- Understand Web server security integration

Creating the Application Environment

ColdFusion lets you group related pages together into an *application*. An application is any collection of pages having the same application name as defined in the CFAPPLICATION tag. ColdFusion provides a convenient way to include this tag on every page in a directory tree using the Application.cfm file.

Application.cfm

When ColdFusion processes any page, it first looks recursively up the directory tree all the way to the drive's root directory for a file named Application.cfm. When it finds one, it processes the Application.cfm file before processing the requested page. This makes Application.cfm a convenient place to define

- The application name
- Application, client, and session management options
- Default variables (constants) for the application
- Custom error pages
- Data sources
- Default style settings

Any local variables you define in Application.cfm will be available to all pages in the application since the code in Application.cfm runs first. However, this is not the most efficient method of defining default variables because the variable assignment block will run at the beginning of every page request. For sake of performance and clarity, it is more common to use Application.cfm to define variables in the Application scope.

OnRequestEnd.cfm

ColdFusion provides another file, OnRequestEnd.cfm, which works much the same way, but at the end of every request. However, ColdFusion will only look for OnRequestEnd.cfm in the same directory as the requested page. OnRequestEnd.cfm will not run if the requested page has exited via CFEXIT or CFABORT, or if the page has encountered an error due to an unhandled exception. OnRequestEnd.cfm is an ideal place to

- Mirror persistent variables created on the page
- Display debugging info with CFDUMP
- Log successful page requests

Persistent Variables

One of the problems of the Web paradigm is that because browser connections are not permanent, it is not directly possible for the server to keep track of browser state information. Variables defined on one page are not automatically available to future pages. ColdFusion provides several classes of variables which can be used to help manage persistent information better.

Server Variables

Server variables are available to any page running on the ColdFusion Application Server. The default server variables give you information about the Web server and operating system, but you can also define your own. Because their scope is so broad, server variables are probably the least useful for typical Web applications; however, you could use them to let all applications know about scheduled downtime or other conditions which would affect the entire server.

Application Variables

Application scope variables are available to any page having the same CFAPPLICATION tag. They are most useful for caching queries and dynamic parameters (like a system outage notification message) which may change on the fly and are used throughout the application. It is often useful to define Application scope variables in Application.cfm, but not necessary. There is no need to define Application variables every time a page runs because application variables are persistent, but they must be defined initially. This is typically handled using a flag variable.

```
<CFIF NOT IsDefined
(“Application.IsInitialized”)>

      <!--- This is the first time through --->

      <CFSCRIPT>

      Application.DSN = “Northwind”;

      Application.IsInitialized = TRUE;

      </CFSCRIPT>

</CFIF>
```

Note

Because of the locking overhead associated with checking to see if a variable is defined, Allaire recommends that you define application-wide “constants” by simply setting Request scope variables in Application.cfm.

Session Variables

Session variables are used to keep track of browser sessions. A session is all the connections that a single client might make while viewing pages in an application. ColdFusion keeps track of sessions by setting two cookies in the browser, CFID and CFTOKEN. On each successive page request, the browser sends these cookies, which enables ColdFusion to map session variables.

Session variables are stored in memory, which makes them very efficient.

You can set a maximum and default timeout for session variables in the ColdFusion Administrator, and the timeout for each application in the CFAPPLICATION tag.

Session variables are typically used to remember things between pages that would otherwise be a pain to keep passing in URL or Form variables. For example, you can remember the search terms which a user has entered on a search screen. The user may proceed to the search results, then to a detail screen or two, and finally back to the search screen for a new search. If you stored the previous search terms in session variables, you can display them by default when the user returns.

Note

By default, session variables do not correspond to browser sessions. If a user closes the browser and fires it back up within the session variable timeout, the session will still be active. On the flip side, if a user leaves the browser open, but walks away for longer than the session variable timeout, the ColdFusion session will have ended.

Working with Server, Application, and Session Variables

Server, Application, and Session variables are all registered as ColdFusion structures and can be manipulated with the standard structure functions.

To create a new variable or update it, use a simple assignment:

```
<CFSET Application.DSN = “Northwind”>
```

To get a list of session, application, or server variables, use StructKeyList:

```
<CFSET AppVars = StructKeyList (Application)>
```

To delete a Server, Application, or Session variable, use StructDelete ().

Client Variables

Client variables are very similar to session variables; however, they are intended for more permanent storage. Client variables can be stored in one of three places:

1) The server registry
2) Browser cookies
3) A database

Client variables are used when you want to remember user preferences or defaults next time they come back to your application.

Client variables are most important for one particular situation, and that is load balancing across multiple ColdFusion servers. Only client variables stored in a database will work successfully in this situation. Why?

1) Session variables are out because they are only stored in memory on one server.
2) The registry is out because it only exists on one server.
3) Browser cookies are not secure and limited in number and size.

Note

For load-balancing applications where the information in the cookies doesn't have to be secure, client variables stored in browser cookies will offer better performance than database storage.

When client variables are enabled, ColdFusion sets six default Client attributes: CFID, CFTOKEN, URLToken, HitCount, TimeCreated, and LastVisit

Working with Client Variables

Client variables have their own special functions.

To create a new variable or update it, use a simple assignment:

```
<CFSET Client.BGColor = “Red”>
```

To get a list of client variables, use GetClientVariablesList ():

```
<CFSET CVList = GetClientVariablesList ()>
```

To delete a Client variable, use DeleteClientVariable ().

Cookie Variables

There is little difference between cookie variables and client variables using browser cookie storage. In either case, anything you set in a cookie can be easily read, copied, and modified by the user. One useful property of cookies is that you can set a different expiration time for each.

Working with Cookie Variables

You create or update cookie variables with the CFCOOKIE tag.

```
<CFCOOKIE NAME="User_ID" VALUE="2344"
          EXPIRES="100">
```

The EXPIRES attribute specifies the expiration time in days. Alternatively, you can include a timestamp. If you leave out the EXPIRES attribute, the cookie will expire when the user closes the browser.

To delete a cookie, use CFCOOKIE with EXPIRES=”now”:

You access cookie variables with the Cookie prefix:

```
<CFSET thisUser = Cookie.User_ID>
```

Note
If your page uses a <CFLOCATION> tag, no cookies will be set.

Rewriting Session Cookies

By default, the CFID and CFTOKEN cookies that ColdFusion uses to map to Session and Client variables are persistent in the browser. This means a user session can persist even after closing the browser, which can sometimes be confusing. In addition, the cookies get written to disk, which may present a security concern. Fortunately, there are two ways around this dilemna:

1. In ColdFusion MX settings, you can check the box to use J2EE session cookies, which reside in memory only.
2. In earlier versions of ColdFusion, you can set setclientcookies="NO" in your CFAPPLICATION tag and create your own CFID and CFTOKEN cookies in Application.cfm:

```
<cfif not isDefined("COOKIE.CFID")>
      <cfcookie name="CFID"
                value="#CLIENT.cfid#">
      <cfcookie name="CFTOKEN"
                value="#CLIENT.cftoken#">
</cfif>
```

Walkthrough 13-1 Creating a Data Store for Client Variables

Setup

ColdFusion can automatically create a data store for client variables. In this walkthrough, we'll set one up using ColdFusion Administrator.

Steps

1. In SQL Server Enterprise Manager, create a new database "CFClients."
2. Create a new database login "cfclients" with access to the CFClients database in the database owner role. Also set CFClients as the default database for this login.
3. Open ColdFusion Administrator and click on ODBC.
4. Create a new datasource "CFClients" with access to the CFClients database. For servername, type in "(local)." Verify the connection.
5. In ColdFusion Administrator, click on Variables.
6. Under Client Variables Storage, select CFClients and click Add.
7. Click the Create button.
8. Under Default Client Variable storage, select CFClients. Review the other settings on the page and click Apply.
9. Go back to SQL Server Enterprise Manager and look at the tables in the CFClients database.
10. Change the permissions for the cfclients login to act in the public role only.

Enabling Persistent Variables in Application Pages

In order to use application, session, or client variable, you must include the CFAPPLICATION tag in each application page (or the Application.cfm file).

CFAPPLICATION allows you to

1) Set a timeout for application variables
2) Set a timeout for session variables
3) Specify the client variables storage method
4) Enable session variables
5) Enable client variables

A typical CFAPPLICATION tag looks like this:

```
<cfapplication

      name="dmc"

      clientmanagement="yes"

      sessionManagement="YES"

      applicationTimeout=
                     "#createTimeSpan(1,0,0,0)#"

      sessionTimeout=
                    "#createTimespan(0,0,30,0)#"

      setclientcookies="YES">
```

This tag will enable session management with a timeout of 1 day for Application variables and 30 minutes for Session variables. When you set the setclientcookies attribute equal to “YES”, ColdFusion automatically creates the CFID and CFTOKEN browser cookies used to track session and client variables.

As discussed earlier in the section on cookies, you can create these cookies yourself in order to use memory cookies instead of the default persistent cookies. To do this, set the setclientcookies attribute equal to “NO” and put the following code in your Application.cfm:

```
<cfif not isDefined("COOKIE.CFID")>

      <cfcookie name="CFID"
                value="#CLIENT.cfid#">

      <cfcookie name="CFTOKEN"
                value="#CLIENT.cftoken#">

</cfif>
```

Locking Application, Session, and Server Variables

With ColdFusion 4.0, whenever you create, update, or access an Application, Session, or Server variable, you should use CFLOCK to lock these variables first.

Note

ColdFusion Server 4.5 can perform automatic read locking of all shared variables or full checking, in which case the server warns you if you attempt to write to a shared variable without a lock. In either case, however, you must still include a CFLOCK section when writing to shared variables.

Lock Scopes

When you use CFLOCK, you must specify a lock scope, which allows ColdFusion to synchronize access to the locked regions. In ColdFusion 4.5, you use the SCOPE attribute (Server, Session, or Application). In ColdFusion 4.0, you must use the NAME attribute. These are the suggested lock names for the different classes of variables:

Server variables	“Server”
Application variables	“#Application.ApplicationName#”
Session variables	“#Session.SessionID#”

Application.ApplicationName is defined for every application. Its value is the name you specify in the CFAPPLICATION tag. Session.SessionID is defined and unique for every session, and consists of ApplicationName + CFID + CFTOKEN.

Locking Files and Custom CFX Tags

You can use CFLOCK to synchronize access to any portion of code. Whenever updating a file, you should use CFLOCK with the filename to synchronize access to the file. You should also use CFLOCK if you have created non-thread-safe custom CFX tags.

Advanced Session Data Management

Session variables, because they are stored in memory between page requests, are efficient and convenient. Their primary drawback is that if the ColdFusion server goes down, all Session variables are lost. Likewise, if you have multiple ColdFusion servers in a server farm, Session variables are not replicated across servers. To address these situations, you can use ColdFusion Client variables stored in a central database accessible by all ColdFusion servers in a server farm.

Client Variables Revisited

Unfortunately, Client variables also have several limitations:

1) Client variables can only store simple values (not arrays, structures, or queries).
2) There is a performance penalty associated with reading and writing Client variables from the database, especially if you use them many times in a page.

The first limitation can be easily overcome using ColdFusion WDDX (for more thorough treatment of WDDX, see Lesson 17: Building Distributed Applications). With WDDX, you can *serialize* any complex structure, including a query, into a simple text string, which you can then store in a Client variable. In the following example, the qLocations query is serialized and then stored as Client.savedQuery.

```
<cfwddx
        action="CFML2WDDX"
        input="#qLocations#"
        output="qLocWDDX">

<cfset Client.savedQuery = Variables.qLocWDDX>
```

When you retrieve a client variable serialized with WDDX, you *deserialize* it to get back the original structure as follows:

```
<cfset Variables.qLocWDDX = Client.savedQuery>

<cfwddx action="WDDX2CFML"
        input="#Variables.qLocWDDX#"
        output="qLocations">
```

Caution

In ColdFusion 5 (but not MX), client variable size is limited to 65,000 bytes. Keep this in mind when serializing large structures.

Mirroring Session and Client Variables

You can combine the use of Session and Client variables to get the performance of Session variables along with the fault tolerance of Client variables. To do this, you simply mirror all Session variables as Client variables when you create them. If a server has gone down so your Session variables are lost, you can read from the Client variables. With a little organization, you can handle this mirroring centrally. Instead of referring to Session and Client variables directly in your pages, use a structure in the Request scope to hold all variables you want to persist (we'll call it Request.Mirror).

In OnRequestEnd.cfm, you store the entire Request.Mirror structure as both a Session and Client variable like this:

```
<cflock scope="SESSION" type="EXCLUSIVE"
        timeout="15" throwontimeout="Yes">

    <cfset Session.Mirror =
           duplicate (Request.Mirror)>

</cflock>

<cfwddx action="CFML2WDDX"
        input="#Request.Mirror#"
        output="WDDXMirror">

<cfset Client.Mirror = Variables.WDDXMirror>
```

Then, in Application.cfm, you read Session.Mirror, and if it's not available, Client.Mirror:

```
<cfif IsDefined ("Session.Mirror")>

      <cflock scope="SESSION" type="READONLY"
             timeout="15" throwontimeout="Yes">

            <cfset Request.Mirror =
                   duplicate (Session.Mirror)>

      </cflock>

<cfelseif IsDefined ("Client.Mirror")>

      <cfset WDDXMirror = Client.Mirror>

      <cfwddx action="WDDX2CFML"
              input="#Variables.WDDXMirror#"
              output="Request.Mirror">

<cfelse>

      <cfset Request.Mirror = StructNew ()>

</cfif>
```

By centralizing the management of persistent variables, you minimize the overhead of locking Session variables as well as the overhead of writing Client variables because both are done only once per request.

Generating Custom Error Messages

By default, ColdFusion returns a standard page for all request and validation errors (if using one of the server-side validation methods). If you want to make error pages fit better within the style of your application, you can define a custom error page using the CFERROR tag to specify the custom error page. Typically, you place the CFERROR tag in the Application.cfm file.

The custom error page is a regular ColdFusion page, except that you cannot use any ColdFusion tags except CFOUTPUT with the names of special variables which contain information about the error and the circumstances around the error. For a list of these variables, see CFERROR in the *CFML Language Reference*.

All errors are logged in the ColdFusion application log. By default, all ColdFusion logs are stored in \CFUSION\Log.

Once an error has been generated, page processing is aborted. To trap an error and handle it before this occurs, you can use structured exception handling, which is described in a later chapter.

There are four types of custom error pages in ColdFusion 4.5.

Types of Custom Error Pages in ColdFusion 4.5	
Exception	Handles all unhandled exceptions. Can use all CFML tags, including CFERROR (but not CFCATCH) variables.
Monitor	Catches exceptions before they are handled by CFTRY or a CFERROR page. Useful for logging all exceptions.
Validation	Handles validation errors when submitting a form.
Request	Handles everything else. Cannot use any CFML tags except <CFOUTPUT> with the CFERROR variables.

With CFERROR Type=”Exception,” you can additionally specify an exception type. This way, you can define multiple custom error pages for different types of exceptions, including custom exception types (see Lesson 15: *Structured Exception Handling*).

Walkthrough 13-2 Use Persistent Variables for Authentication

Setup

In this walkthrough, you will create an authentication framework using client variables stored in cookies. The application framework will work as follows:

1) When a user tries to access any protected page, the user will be redirected to the login page.
2) Upon successful login, a client variable will be set and the user will be directed back to the attempted page.
3) On future attempts to access protected pages, the application will automatically detect the client variable indicating that the user is logged in.

Steps

1. Create separate App and Auth directories in order to have different versions of Application.cfm.

2. In both Application.cfm files, include the same CFAPPLICATION tag which enables cookie-based Client variables.

3. In Auth/Login.cfm, create a username and password form. When the form is submitted successfully, set a Client variable named AuthUser containing the username.

4. In App/Application.cfm, redirect the user to the login page if the login variable (Client.AuthUser) is not set.

5. To make the authentication system more secure, modify Auth/Login.cfm and App/Application.cfm to use another client variable, AuthToken, which is the username encrypted with the browser's IP address (CGI.REMOTE_ADDRESS).

6. To make the authentication system more convenient, modify App/Application.cfm to remember the page which the user tried to access (CGI.SCRIPT_NAME) before getting redirected to the login screen. Upon successful login, go to this page.

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 5: Using the Application Framework

Lesson 14
Creating Custom Tags and Functions

Introduction

ColdFusion provides two powerful ways to create modular and reusable code: custom tags and user-defined functions (UDFs). You can create custom tags in C++, Java, or ColdFusion Markup Language (CFML). In this lesson, we'll explore creating custom tags and UDFs with CFML.

Objectives

By the end of this lesson, you should be able to

- Create custom tags and know where ColdFusion looks for them
- Call custom tags using one of three addressing methods
- Pass parameters to custom tags, including structures
- Use local, Caller, and Request variables in custom tags
- Return results to the calling page
- Use nested tag architecture to create more dynamic pages
- Create and invoke user-defined functions
- Create a user-defined function with optional arguments

What is a Custom Tag?

ColdFusion allows you to create your own tags similar to CFML tags. Custom tags can be created in either CFML or C++ using the CFX API.

A custom tag essentially acts like a function or procedure, and may have attributes like any other ColdFusion tag. The simplest way to invoke a custom tag is with the CF_ prefix. In this example, ColdFusion will search for the MyTag.cfm file in the same directory as the calling page.

```
<CF_MyTag attr1=”value1” attr2=”value2” ...>
```

A custom tag is just like any other ColdFusion page. You can use any ColdFusion tag, write output, run queries, etc.

Writing Custom Tags

Custom tags have their own variable scope. This differs from CFINCLUDE, which runs included code in the same variable space as the main page. Custom tags do have access, however, to all variables pertaining to the HTTP request or current application, including Form, URL, CGI, Cookie, Server, Application, Session, and Client variables.

Passing Data with Tag Attributes

Tag attributes are accessed through the special Attributes variable in the custom tag. For example, you might write a custom tag to delete an employee record from a database. When you invoke the tag, you want to pass in the employee ID. Your tag might look like this:

```
<CF_DeleteEmployee EID=244>
```

DeleteEmployee.cfm would look like this:

```
<CFIF IsDefined (“Attributes.EID”)>
      <CFIF IsNumeric (“Attributes.EID”)>
            <CFQUERY ...>
            DELETE FROM Employees
            WHERE EmployeeID = #Attributes.EID#
            </CFQUERY>
      </CFIF>
</CFIF>
```

Using Request Variables

Request variables are a special class useful primarily for sharing data between a main page and any number of custom tags. Request variables are so named because they are associated with a single Web request, and available to any code that processes the request.

Request variables are essentially global variables, but they are slightly less ugly because they do require the Request prefix, which distinguishes them from local variables and therefore reduces the likelihood of conflict. For this reason, when it is necessary to return data to a calling page, Request variables are preferable to the next method.

Note

Request variables are only available in ColdFusion 4.01 and later. However, you can use Session variables to the same effect.

Avoiding Caller Variables

ColdFusion defeats one of the purposes of custom tags by allowing you to access variables local to the calling page with the special Caller variable. When used in a custom tag, Caller.X refers to the local variable X in the calling page.

The Caller construct can also be used to set variables in the calling page, which could really be a source of confusion, especially if the custom tag is intentionally developed by another group for the purpose of “information hiding.” Use a structure to return results instead.

Passing Scalar Data

Scalar objects (numbers, strings, date / time values, and Boolean values) are passed by value. If your custom tag modifies a string passed in as an attribute, this will not affect the value of the string in the calling program.

Passing Complex Data

There are four complex objects in ColdFusion: arrays, structure, queries, and COM objects. To pass any of these to a custom tag, use the variable name surrounded by pound signs (but no quotes). For example, to pass the Orders query to a custom tag, you would write

```
<CF_MyTag order_query=#Orders#>
```

In MyTag.cfm, you would reference the query as follows:

```
<CFOUTPUT QUERY=Attributes.order_query>
```

The same syntax applies to arrays, structures, and COM objects.

Note

Structures, Queries, and COM objects are passed by reference, so any changes you make in a custom tag will be visible in the calling page. Arrays, however, are copied. This can be expensive for large arrays. If you need to pass a large array, pass it by reference as an element of a structure.

Returning Results

The best way to return results from a custom tag is in a structure. This way, the custom tag does not interfere with local variables on the calling page. Structures can also be used to pass in scalar values by reference so that any changes to the values made in the custom tag will be available to the calling page.

```
<CFSET TagResults = StructNew()>
<CF_MyTag Results=#TagResults#>
<CFOUTPUT>#TagResults.myName#</CFOUTPUT>
```

Here is a corresponding MyTag.cfm:

```
<CFSET Attributes.Results.myName = “David”>
```

Calling Custom Tags

ColdFusion gives you three methods for invoking tags, depending on how you use them.

1) Shorthand: <CF_*MyTag*>
2) Shared tags: <CFMODULE NAME="*path.MyTag*">
3) Application-specific: <CFMODULE TEMPLATE="*path*/*MyTag.cfm*">

The shorthand invocation will look for MyTag.cfm in the same directory as the calling file, followed by \CFUSION\CustomTags.

The shared tag invocation will look in the \CFUSION\CustomTags directory. To invoke

```
C:\CFUSION\CustomTags\Chandler\Nav\Toolbar.cfm,
```

you would use

```
<CFMODULE NAME="Chandler.Nav.Toolbar">
```

To invoke tags used only by your application, specify the TEMPLATE in CFMODULE. You can use a relative or absolute path (beginning with "/").

Note

An absolute path as used by CFMODULE is not a Web URL, but rather a ColdFusion mapping. Mappings are set up in ColdFusion Administrator. You can also use mappings and absolute paths with CFINCLUDE.

You specify attributes the same way no matter which calling syntax you use. Note that custom tags normally do not require an end tag. If you need to use an end tag for nested tag processing, you can use the shorthand

```
<CF_MyTag>
...
</CF_MyTag)
```

or use

```
<CFMODULE TEMPLATE="/somewhere/MyTag.cfm">
...
</CFMODULE>.
```

Lab 14-1 Create a Custom Tag to Display a Table from a Query

Setup

This lab will put together several things we've learned to date. We will create a custom tag QueryTable to display an HTML table with query results. The custom tag will accept as attributes

TableWidth – the HTML TABLE WIDTH attribute
Query – the query from the calling page

To get started, create a new file from a blank template and save it as Lab\QueryTable.cfm.

Steps

1. Create the table header (hint: use TH and loop over the list Attributes.Query.ColumnList to get the column names).

2. Loop over the query and output the value of each field in a table cell (hint: this will require nested loops and the Evaluate function).

3. Replace the table in OrdersByCustomer.cfm with a call to the new custom tag.

Encoding Custom Tags

ColdFusion Server 4.5 provides a command-line utility you can use to encode your custom tags before distributing them. The *cfencode* utility (used to be cfcrypt) lives in /CFUSION/bin on the server.

See the section “Managing Custom Tags” in *Developing Web Applications with ColdFusion*, Chapter 7: Reusing Code.

Warning

Before you encode your custom tags or any other ColdFusion code, make sure you have a clear-text copy somewhere! Once you’ve encoded a file, it cannot be decoded (except using illegal hacker tools not supported by Allaire).

Nested Custom Tags

You can nest custom tags by calling a custom tag within a custom tag, although keeping track of the Caller variable scopes will accelerate your mind’s aging process. For example, assume the following order of custom tag calls:

```
Main page
        CF_BuildTable
                CF_BuildTableRow
```

If the main page sets a variable X, then CF_BuildTableRow can refer to it as Caller.Caller.X. But you shouldn’t use the Caller scope, anyway.

If you do need to share data among related nested custom tags, you can

1) Pass a structure as one of the attributes to each tag,
2) Use Request or Session variables, or
3) Use the advanced nesting method.

Advanced Nested Tags

As if nested tags aren't bad enough, ColdFusion 4.0 introduces syntax for nesting custom tags in which the custom tag calls are all made in the main module. When invoked this way, nested sub-tags inherit the Caller scope from the main module rather than the parent tag. The new syntax uses an end tag as well as a start tag, like this:

```
<CF_BuildTable>
      <CF_BuildTableRow>
      <CF_BuildTableRow>
      ...
</CF_BuildTable>
```

Processing Flow

When invoked with an end tag, ColdFusion processes the custom tag *twice*. The flow of processing works like this:

1) Process the custom tag in start mode.
2) Process all sub-tags.
3) Process the custom tag in end mode.

Because a custom tag is executed twice in this architecture, you must include a CFIF or CFSWITCH construct in the custom tag to take the appropriate action based on whether the tag is being executed in start mode or end mode. The CFSWITCH statement would look something like this:

```
<CFSWITCH expression=”#ThisTag.ExecutionMode#”>
      <CFCASE value=’start’>
      <!--- Start tag processing --->
      </CFCASE>
      <CFCASE value=’end’>
      <!--- End tag processing --->
      </CFCASE>
</CFSWITCH>
```

Usage Guidelines

Why would you use the more complex syntax? There are perhaps three reasons:

1) When you want to pass attributes directly from the main page to nested tags. Using the simple syntax, you would have to pass these attributes through the nested hierarchy. Allaire suggests that the main application for the more complex nested architecture is building user interfaces from a query on the main page.
2) When a custom tag might call different sub-tags, depending on the application. In the simpler syntax, the custom tag must always call the same sub-tags.
3) When you want to use CFASSOCIATE to aggregate data from sub-tags back up to the parent tag.

In all of these cases, it can be argued that it would be simpler to jettison custom tags altogether and just write the code inline in the main page. However, custom tags are useful for allowing developers to focus on their particular pieces of the puzzle.

Advanced Topics in Custom Tags

Tag instance data

When a ColdFusion custom tag is executing, you can access certain tag properties using the ThisTag variables scope. The following variables are available:

- ExecutionMode -- "start" or "end"
- HasEndTag -- used for code validation, it distinguishes between custom tags that have and don't have end tags for ExecutionMode=start.
- GeneratedContent – holds the output generated by this tag and its descendants. ThisTag.GeneratedContent is always empty during start mode.
- AssocAttribs -- holds the attributes of all nested tags which are associated with CFASSOCIATE.

CFEXIT

The CFEXIT statement behaves in many different ways inside custom tags, depending on the location of CFEXIT, the execution mode, and the optional METHOD attribute of CFEXIT. See the CFEXIT tag in the *CFML Language Reference* for more information.

User-Defined Functions (UDFs)

In ColdFusion 5 and later, you can create your own functions that work just like the built-in ColdFusion functions.

When to Use a UDF

Like custom tags, user-defined functions provide a way to reuse common code. The principal advantage of a UDF over a custom tag is that a UDF returns a value, whereas a custom tag does not (although you can pass a structure to hold a return value, as we saw earlier). In addition, UDFs are faster than custom tags because CFSCRIPT is faster than ColdFusion tags and CFINCLUDE is faster than calling a custom tag.

Creating a UDF

In ColdFusion 5, to create a user-defined function, you use CFSCRIPT, which is very much like JavaScript. The following example creates a function and puts it in the Request scope. Note the use of the Arguments scope, which is similar to the Attributes scope in custom tags.

```
<cfscript>
function isStringSafe (str1)
{
    if (REFind ("[\<\>]*", Arguments.str1))
        return "No";
    else
        return "Yes";
}
Request.isStringSafe = isStringSafe;
</cfscript>
```

You can create a UDF anywhere in a page. Typically, you put related UDFs in a single .cfm template, and then include the template using CFINCLUDE in pages where you need to use one of the functions. Note that a UDF is actually a variable. You can put it in any scope and even pass it as an argument to other functions (i.e., a callback function).

Calling a UDF

You call a user-defined function just like any other ColdFusion function. The function must be defined somewhere in the page or in an included page. As with custom tags, arguments of complex data types (queries, structures, and COM objects) are passed by reference, but all others are passed by value.

Rules for Defining Functions

- A function name cannot start with the letters “cf” and cannot be the same as any other variable, built-in function name, or UDF. If you accidentally include a function definition twice, ColdFusion will generate an exception.
- You can call a UDF inside its own definition (recursion).
- A function does not have to return a value. A shortcut way to invoke a function that doesn’t return a value is <CFSET *function (arg1, etc.)*>.
- With CF5, you must write UDFs in CFSCRIPT, so you cannot use ColdFusion tags in the body of a function. With ColdFusion MX, however, you can create UDFs with the more advanced <CFFUNCTION> syntax, which does allow ColdFusion tags in the body of a function.

Variable Scoping in UDFs

Variables in a ColdFusion function can have their own local scope just like they do in custom tags. Unfortunately, unlike custom tags, you must explicitly declare variables as local. Otherwise, ColdFusion puts them in the page’s Variables scope (boo hiss). Worse, inside a UDF, you can access the Variables scope in the calling page. Don’t do it! You declare a local variable with the **var** keyword:

```
function doSomething ()
{
      // myVar is visible only in the UDF
      var myVar = 1;
}
```

Creating a UDF with Optional Arguments

To create a UDF accepting optional arguments, list only the required arguments in the function declaration. Optional arguments passed in are available in the Arguments scope, which can behave as either a structure of argument names or an array indexed by argument position. For example, this function will bitwise OR together any number of arguments.

```
function ORSum (b1, b2)
{
    var result = 0;
    var i = 1;
    for (; i LTE ArrayLen (Arguments); i=i+1)
        result = BitOR (result, Arguments[i]);
    return result;
}
```

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 6: Extending ColdFusion Pages with CFML Scripting
Chapter 9: Writing and Calling User-Defined Functions
Chapter 10: Creating and Using Custom CFML Tags

Online Resources

Creating Custom Tags, an online RealVideo at Allaire Alive (alive.allaire.com).

Before you write your own custom tag or application, be sure to check the Allaire Developer's Exchange at www.allaire.com/developer/gallery.cfm. Some of the tags you may find useful include:

CF_USPS_LookupCityFromZip	From the US Postal Service
CF_ShipTracker	Track FedEx, UPS, DHL packages
CF_InterShipper	Get shipping rates
CFX_CyberCash	Process CyberCash and credit cards
CFX_CyberSource	Process credit cards

In addition, be sure to check the open source library of ColdFusion functions at www.cflib.org, including such gems as

QueryToArrayOfStructures (inverts a query)
QueryToCSV (creates an Excel spreadsheet from a query)
GetHostAddress (does a reverse DNS lookup)

Many are excellent examples of how to wrap a Java function in ColdFusion.

Lesson 15
Structured Exception Handling

Introduction

Besides the default error behavior, ColdFusion allows you to build error handling routines to catch exceptions in problematic code.

Objectives

By the end of this lesson, you should be able to

- Identify applications for custom error handling
- Build an exception handler for any page or custom tag
- Construct a generic error handling framework

Exception Handling

By default, ColdFusion handles runtime errors by

1) Displaying an error page for the user, and
2) Logging the error in the ColdFusion application log.

Exception handling allows you to replace the default behavior with your own code. Exception handling allows you to do a great deal more than custom error pages defined with CFERROR. Unlike custom error pages, you can use exception handling to do things like

1) Retry or fix the exception (such as resolving a deadlock or retrying a file operation)
2) “Interpret” common database error messages for the user (like the cryptic integrity violation message when you add a duplicate record)
3) Log errors to a database
4) Notify a system administrator via e-mail
5) Display help text and resolution procedures from an application errors database
6) Display a customer support form asking the user for further information

Note

You can use <CFERROR Type=”Exception”> to do some of these things in ColdFusion 4.5, but you cannot continue page processing after the exception when using CFERROR.

Using CFTRY / CFCATCH

To handle exceptions in ColdFusion, you simply wrap code in the CFTRY/CFCATCH construct like this:

```
<CFTRY>
      Problem code
<CFCATCH TYPE="exception_type1">
      Exception processing code
</CFCATCH>
<CFCATCH TYPE="exception_type2">
      Exception processing code
</CFCATCH>
...
<CFCATCH TYPE="Any">
      Default exception handler
</CFCATCH>
</CFTRY>
```

The pre-defined ColdFusion exception types are

- Database
- Template
- MissingInclude
- Lock
- Security
- Expression
- Object
- Application
- Any (default)

You can also define custom exceptions using CFTHROW.

CFCATCH Variables

Code in a CFCATCH statement can reference several variables which provide information about the exception. The primary variables are

- CFCATCH.Type
- CFCATCH.Message
- CFCATCH.Detail
- CFCATCH.TagContext

Raising Errors with CFTHROW

The CFTHROW tag can be used anywhere to generate an exception. The most common use for this is to raise an exception which has been caught in a custom tag so that it will be visible to the calling page (and can likewise be caught there).

Note

In ColdFusion 4.5 and later, if a custom tag does not catch an exception which occurs in the custom tag, ColdFusion will automatically raise the exception to the calling page, where it can be caught by a CFTRY / CFCATCH around the custom tag call.

You can use CFTHROW anytime to generate an error message for the user. Used with a custom error page, this is a really simple way to inform the user of any error condition, whether caused by an exception or not.

Using Custom Exception Types

In ColdFusion 4.0 and later, you can create custom exception types simply by specifying them in the TYPE attribute of CFTHROW.

In ColdFusion 4.5, you can use a hierarchical naming convention for your custom exception types. For example, you might define these types:

ObjectProcessing
ObjectProcessing.Project
ObjectProcessing.Project.Folder

Using this method, an exception handler which catches Type=”ObjectProcessing” will also catch all the sub-types. This is exceptionally powerful when used with nested CFTRY / CFCATCH statements or when used with custom error pages to catch exceptions.

Processing Flow

Outside a Custom Tag

When an exception occurs, ColdFusion will first look to see if there is a CFTRY block. If there is, ColdFusion will process the first CFCATCH block whose Type attribute matches the exception type (this is why CFCATCH Type="Any" should always be last). Upon completion, ColdFusion will exit the CFTRY block and continue processing the page.

If there is no CFTRY block, ColdFusion will abort page processing and generate an error page (or custom error page, if you have defined one with CFERROR).

This is rather nice behavior, as it still allows you to display a nice-looking custom error page for those exceptions you happen to miss.

Inside a Nested CFTRY Block

If two CFTRY blocks are nested in the same page and the innermost block fails to catch the exception, it will bubble up until it is caught. This allows you to write specialized CFTRY blocks around small portions of code, while maintaining general exception protection for the whole. If you catch an exception in a nested CFTRY block, you can also force it to bubble up to be caught again in the next outer block by using CFRETHROW or CFTHROW in the inner CFCATCH.

Inside a Custom Tag

Inside a custom tag, exception handling works exactly the same way as in nested CFTRY blocks. If an exception in a custom tag is not caught by a CFTRY block in the custom tag, ColdFusion will "bubble up" the exception to the calling page, where it will be caught if there is a CFTRY block wrapping the call to the custom tag. If you catch an exception in a custom tag, you can force it to bubble up to be caught again in the calling page by using CFRETHROW or CFTHROW.

Tip

Remember that ColdFusion does not automatically abort page processing after catching an exception, but rather exits the CFTRY block and continues processing. If you want to abort page processing, use <CFABORT> as the last statement in your CFCATCH clause.

Error Handling Architecture

Good error handling should be a part of any application. Besides informing the user, a good error handler should

1) Log errors for later review
2) Notify an administrator of serious conditions
3) Provide additional feedback to the user such as a suggested remedy
4) Provide a convenient link to technical support

It's also handy to have a framework for reporting general errors (like validation or operator errors) which do not generate exceptions. A common way to do this is to create an application error database (using a database table or array). An elegant way to handle both general errors and exceptions is to build an exception handler framework, then use CFTHROW for general errors, too.

In ColdFusion 4.5, you can define a custom error page to handle all exceptions, but unfortunately, an exception-handling page cannot use the CFCATCH variables, so you're stuck with Error.Diagnostics (which includes the Message and Detail fields from CFTHROW). In addition, page processing aborts once the handler has been invoked.

To get around these limitations, you can wrap every page in CFTRY. The CFCATCH clause(s) are responsible for implementing the functionality listed above.

Creating a Centralized Exception Handling Framework

In order to wrap every page in CFTRY, you can use a *controller* page through which all application requests come. Instead of going directly to EmployeeUpdate.cfm, the application would go to controller.cfm?action=EmployeeUpdate. You can easily implement a controller using CFINCLUDE to run the specified page. This way, every page gets wrapped in a robust exception handler.

A controller can also be used to

1) Set application constants
2) Declare common functions
3) Display a common navigational framework on every page
4) Check to see that users are authorized for any given page
5) Manage session data
6) Simplify flow control

The concept of a controller is the basis of the popular Fusebox architecture, which is similar to the very powerful Model View Controller (MVC) design pattern. MVC architecture is covered in depth in Lesson 25: Model View Controller.

Lab 15-1 Practice Exception Handling

Setup

Open ColdFusion Studio and select the WebFastTrack project. In this lab, we'll practice handling exceptions with CFTRY/CFCATCH.

Steps

1. Open OrdersByCustomer.cfm.
2. Introduce an exception by trying to include a non-existent file. Run the page and see what happens.
3. Now wrap the include statement in a CFTRY / CFCATCH block to catch type Any and display CFCATCH.Message. Run the page and see what happens.
4. Move the include statement by itself into QueryTable.cfm, the custom tag you created earlier. Wrap the CFTRY around the call to QueryTable. Run OrdersByCustomer.cfm. What happens?
5. Now copy the CFTRY block around the include statement in QueryTable.cfm. Run OrdersByCustomer.cfm. Notice that the exception is caught in the custom tag.
6. In the CFCATCH block in QueryTable.cfm, throw an exception with Message="no include, dude." Run OrdersByCustomer.cfm. Notice that the exception is caught in the calling page.

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 8: Debugging and Error Handling

Unit 4
Data and Applications Architecture

Unit Overview

In this unit, you'll gain exposure to some of the more advanced capabilities of SQL and databases. You'll learn how to create applications in which ColdFusion must integrate with other systems, and you'll learn how to work with databases and ColdFusion to ensure maximum data integrity and database performance.

Goals

By the completion of this unit, you should

- Have a better understanding of what's possible with SQL
- Know how to integrate ColdFusion with remote services
- Understand techniques for working with data from legacy systems
- Be able to avoid common database design mistakes
- Understand database transactions, stored procedures, and triggers
- Know how to design your application with performance in mind
- Be able to use ColdFusion query and page caching

Lesson 16
Extending Your SQL Toolbox

Introduction

SQL is perhaps one of the highest-level languages every devised. In this lesson, we'll explore more of the power and flexibility of SQL.

Objectives

By the end of this lesson, you should be able to

- Write complex SELECT statements involving joins and subqueries
- Use temporary tables to simplify cursor-oriented applications
- Use views to restrict data access and simplify joins
- Insert, update and delete records using subqueries

The SELECT Statement

Selecting Columns

“SELECT * FROM Employees” will return all fields. To name specific fields, use

```
SELECT FirstName, LastName, ...
        FROM Employees
```

DISTINCT

To suppress rows with duplicate values, use the DISTINCT keyword.

```
SELECT DISTINCT FirstName
        FROM Employees
```

Using Functions and Expressions

You can use SQL functions and expressions in a SELECT list.

```
SELECT Price * 1.05, Qty, MONTH (OrderDate)
        FROM Orders
```

You can also name more than one column in an expression.

```
SELECT Price * Qty
        FROM Orders
```

Note

You can also select a constant, as in “SELECT Name, 33 AS Qty FROM …” This is most often in conjunction with an INSERT statement (See *Inserting Data* below).

Using the CASE Function with SELECT

SQL Server supports a very useful function which allows you to SELECT a value based on the value of some expression.

For example, if you want to categorize restaurants on the basis of price, you could write

```
SELECT Name, ‘Price Category’ =
        CASE
                WHEN price IS NULL THEN ‘Not Yet Priced’
                WHEN price < 10 ‘THEN ‘Very Reasonable’
                WHEN price >=10 AND price < 20 THEN ‘Moderate’
                ELSE ‘Expensive’
        END
        FROM Restaurants
        ORDER BY price
```

You can also use the CASE function in a WHERE clause.

Note

You can use SQL Server’s CASE statement where you would use Oracle’s DECODE statement.

Naming Derived Columns with Aliases

When you include a function or an expression, the result is called a derived column. You use the AS keyword to name derived columns. This is required if you want to use the derived column in the ORDER BY clause or in a ColdFusion expression using the query.

```
SELECT Price * Qty AS SubTotal
        FROM Orders
```

In ColdFusion, you would refer to the derived column as *QueryName*.SubTotal.

The FROM Clause

Selecting from a View

You can use a view name in place of a table name in the FROM cause to select from a view.

Selecting from Multiple Tables

You can SELECT data from multiple tables by creating a join.

If you select data from two tables without specifying a join condition, you create the *Cartesian product* of the tables. The Cartesian product consists of every possible combination of rows from the tables. Cartesian products are useful in rare circumstances such as when you want to create a new table with multiple rows for each existing table. The following statement generates the Cartesian product of the Orders and Customers tables. (Please don't try this on a production database.)

```
SELECT *
        FROM Orders, Customers
```

SQL-92 refers to the Cartesian product as a CROSS JOIN.

Inner Joins

The most familiar join is the inner join. Traditionally, join conditions were specified in WHERE clauses; however, the new ANSI SQL-92 syntax is easier to understand. An inner join includes only those rows from the Cartesian product which match the join conditions. The following statement will return all fields from both the Customers and Orders table for each order from CustomerID 1234.

```
SELECT *
        FROM Customers JOIN Orders
                ON (Customers.CustomerID = Orders.CustomerID)
        WHERE CustomerID = 1234
```

Outer Joins

An outer join contains the combination of rows from the tables that satisfy the join conditions. However, unlike an inner join, an outer join includes all rows from the *dominant* table, regardless of whether matching rows were found in the *subservient* table. The fields from the non-matching rows in the *subservient* table are filled with NULLs.

In the old SQL syntax, simply include the OUTER keyword before the table name in the SELECT statement.

```
SELECT Employee.*, pm.MemberID
        FROM OUTER Employees, PlanMembers pm
        WHERE EmployeeID = pm.EmployeeID
```

The preceding statement will include a row for every employee. For those employees in the PlanMembers table, the row will include their MemberID. For those employees not in the PlanMembers table, MemberID will be NULL.

The new SQL-92 syntax uses LEFT OUTER JOIN and RIGHT OUTER JOIN (or LEFT JOIN and RIGHT JOIN for short). LEFT JOIN specifies that all rows from the left table not meeting the join condition are included in the result set, and output columns from the other table are set to NULL in addition to all rows returned by the inner join. Likewise, RIGHT JOIN includes all rows from the right table. The previous example could be rewritten

```
SELECT Employee.*, pm.MemberID
        FROM Employees LEFT JOIN PlanMembers pm
                ON EmployeeID = pm.EmployeeID
```

A special type of join known as the FULL JOIN includes all rows from either table.

Join Conditions

The most understandable join condition is an equality, as shown in the previous example. However, join conditions can use any SQL operator. This is typically most useful with self joins. You can also specify multiple join conditions using Boolean operators (AND, OR, etc.)

Self Joins

A special type of inner join is the self-join. Self joins are most useful in conjunction with join conditions other than equality for operations such as ranking and pairing. For example, the following self join returns all pairs of employees for whom one salary is more than 5 times the other.

```
SELECT a.LastName AS Executive, b.LastName AS Grunt
        FROM Employees a JOIN Employees b
                ON (a.Salary > 5 * b.Salary)
```

Using Table Aliases

As illustrated in the previous example, table aliases are necessary for doing a self join. For joins involving multiple tables, they are a useful shortcut. Table aliases are also required to write correlated subqueries (see below).

Selecting from a Derived Table

A derived table is the result of another SELECT statement. A derived table is essentially a temporary view. Derived tables are useful when the derived table contains aggregate functions or subqueries which would unnecessarily complicate the main SELECT list. The following example returns the name of every company and the number of orders placed by that company.

```
SELECT CompanyName, NumOrders
        FROM Customers,
        (
                SELECT CustomerID, COUNT(*) AS NumOrders
                        FROM Orders
                        GROUP BY CustomerID
        ) CustOrders
        WHERE Customers.CustomerID = CustOrders.CustomerID
```

The WHERE Clause

Including and Excluding Rows

To include rows in a query, use the equality or inequality operators.

```
SELECT *
        FROM Employees
        WHERE FirstName = ‘David’
```

```
SELECT *
        FROM Orders
        WHERE ShipVia <> ‘UPS’
```

Pattern Matching

You can use the LIKE operator with one or more wildcards to match patterns in character fields.

```
SELECT *
        FROM Employees
        WHERE FirstName LIKE ‘Dav%’
```

Dealing with NULLs

To test for a NULL value, use the IS NULL and IS NOT NULL operators.

```
SELECT *
        FROM Employees
        WHERE TerminationDate IS NOT NULL
```

Note
NULL values are not tested when comparing for inequalities.

Allowing NULL columns is generally not recommended for two reasons:

1) It degrades performance, since every NULL value is stored separately from the rest of the table, and
2) NULLs can be tricky to work with.

For example, the following query will not return any records with a NULL fax number:

```
SELECT *
        FROM Employees
        WHERE Fax NOT LIKE ‘319%’
```

Finding a Subset of Values

Use the IN operator to find rows matching a set of values.

```
SELECT *
        FROM Employees
        WHERE LastName IN (‘Chandler’,’Chandelier’)
```

Finding a Range of Values

Use the BETWEEN operator to find rows matching a range of values.

```
SELECT *
        FROM Employees
        WHERE Salary BETWEEN 54000 AND 72000
```

Creating Compound Conditions

Use Boolean operators to create multiple conditions.

```
SELECT *
        FROM Employees
        WHERE Salary > 72000
                AND (HireDate > ‘3/12/2000’
                OR NumViolations > 1)
```

Using the CASE Function in a WHERE Clause

In SQL Server, you can use the CASE function in a WHERE clause to make a comparison more dynamic. For example, you could use this construct in a stored procedure to omit a comparison if a parameter is NULL:

```
SELECT * FROM Customers
WHERE City =
        CASE
                WHEN @City IS NULL THEN City
                ELSE @City
        END
```

You could also write this statement using the IsNull shortcut function in SQL Server:

```
SELECT * FROM Customers
WHERE City = IsNull (@City, City)
```

Using Derived Columns in WHERE

You cannot reference a derived column in a WHERE clause by its alias. Instead, you must duplicate the derived expression.

```
SELECT UnitPrice * 1.05 AS Taxed
        FROM Orders
        WHERE UnitPrice * 1.05 > 100
```

You can use any SQL expression in a WHERE clause, including those involving columns not named in the SELECT list.

```
SELECT UnitPrice * 1.05 AS Taxed
        FROM Orders
        WHERE UnitPrice * Quantity > 100
```

The ORDER BY Clause

Specifying a Sort Order

You can sort queries by any number of columns in ascending (ASC, the default) or descending (DESC) order.

```
SELECT *
        FROM Employees
        ORDER BY LastName ASC, FirstName DESC
```

Referencing Columns with Aliases

Unlike the WHERE clause, you can reference derived columns by their aliases in the ORDER BY clause. In fact, all columns in the ORDER BY clause must appear in the SELECT list.

```
SELECT ProductID, UnitPrice * 1.05 AS Taxed
        FROM Orders
        ORDER BY Taxed
```

Aggregating Records

Using Aggregate Functions

The aggregate functions are COUNT, AVG, MAX, MIN, and SUM. (SQL Server 7 adds STDEV and VAR for statistical functions).

Use aggregate functions to return a single value from a group of rows. For example, to find the number of employees whose last name is “Chandler,” you would write

```
SELECT COUNT(*)
        FROM Employees
        WHERE LastName = ‘Chandler’
```

The GROUP BY Clause

GROUP BY is one of the more powerful SQL constructs. It is always used to compute an aggregate function on related groups of records. For example, to find the number of employees having each last name, you would write

```
SELECT LastName, COUNT(*)
        FROM Employees
        GROUP BY LastName
```

This query will return one row for each last name and the number of employees who share that name.

To find the total of each order from an OrderDetails table, you would write

```
SELECT OrderID, SUM (UnitPrice * Quantity) AS OrderTotal
        FROM OrderDetails
        GROUP BY OrderID
```

The HAVING Clause

The HAVING clause complements a GROUP BY clause by applying one or more conditions to groups *after* they are formed, and only including the matching groups in the results. One advantage of HAVING over WHERE is that you can include aggregate functions in a HAVING clause. To list only those employees who have the same name as another employee, you would write

```
SELECT LastName, COUNT(*)
        FROM Employees
        GROUP BY LastName
        HAVING COUNT(*) > 1
```

Like the WHERE clause, you cannot reference a derived column by its alias, but you can recompute the expression in the HAVING clause.

Lab 16-1 Create a Complex Query Using ColdFusion Query Builder

Setup

In ColdFusion Studio, open the Northwind database. In this walkthrough, we'll build a query which returns a list of top customers by total sales volume. We'll begin by computing the total for each order.

Steps

1. Right-click on the Northwind database and select New Query.
2. Add the Orders and OrderDetails tables to the query. Join them on OrderID.
3. Double-click on the OrderID column to select it for output.
4. Create a column to compute the sum of UnitPrice * Quantity and call it Total.
5. Group by the OrderID column.
6. Run the query.
7. Add the Customers table to the query and join Customers and Orders on CustomerID.
8. Change the GROUP BY field to Customers.CompanyName.
9. Run the query.
10. Now limit the display to those customers having orders totaling $10,000 or more. Use the Criteria field in the Total column. Notice that this creates a HAVING clause.
11. Sort by the Total column in descending order. (Use a separate column named Total and do not select it for output.)
12. Run the query.
13. Select the Orders.OrderID column for output and use the Count function to find the number of orders for each customer. Call this column NumOrders.
14. Run the query.

Using Subqueries

A subquery is a SELECT statement nested in the WHERE clause of another SELECT, INSERT, UPDATE, or DELETE statement. Subqueries can be introduced with four different keywords.

ALL

Use the ALL keyword to determine whether a comparison is true for every value returned by the subquery. If the subquery returns no values, the search condition is *true*.

```
SELECT *
        FROM Employees
        WHERE Salary <= ALL
                ( SELECT Salary FROM Employees )
```

Note the less-than-or-equal-to operator. What would this query return with only the less-than operator?

ANY

Use the ANY keyword to determine whether a comparison is true for any value returned by the subquery. If the subquery returns no values, the search condition is *false*.

```
SELECT *
        FROM Employees
        WHERE Salary < ANY
                ( SELECT Salary FROM Employees
                        WHERE HireDate > ‘1993’ )
```

This example returns a row for every employee who is paid less than any employee hired after 1993.

IN

The IN keyword is a shortcut for =ANY. NOT IN is equivalent to !=ALL.

EXISTS

The EXISTS keyword tests to see if a subquery returns any rows.

```
SELECT *
        FROM Products
        WHERE NOT EXISTS
                ( SELECT *
                        FROM OrderDetails od
                        WHERE od.ProductID = Products.ProductID
                                AND od.Quantity > 100 )
```

This example finds all products which have never been ordered in quantity greater than 100. Note that you could write the same thing with an outer join as follows.

```
SELECT DISTINCT Products.*
        FROM Products LEFT JOIN OrderDetails od
                ON od.ProductID = Products.ProductID
        WHERE (od.Quantity <= 100)
                OR (od.Quantity IS NULL)
```

On SQL Server 7.0, the first query takes 10 ms, whereas the second takes 270 ms. This is because the outer join returns a row for each product every time it occurs in an order. If you move the quantity test into the join condition, the query only returns a row for each time the product is ordered in quantity > 100, which is equivalent to the query using NOT EXISTS. The resulting query runs in 20 ms.

```
SELECT DISTINCT Products.*
        FROM Products LEFT JOIN OrderDetails od
                ON od.ProductID = Products.ProductID
                        AND od.Quantity > 100
        WHERE od.Quantity IS NULL
```

In general, NOT EXISTS is a better solution than an outer join for any data which may contain NULLs.

Single-Valued Subqueries

If you know that a subquery will return exactly one value, you do not need to use ANY or ALL. This is always the case when the subquery uses an aggregate function. This query returns all the students who did better than average on a test.

```
SELECT StudentID
        FROM TestScores
        WHERE TestScore >
        ( SELECT AVG (TestScore)
                FROM TestScores )
```

Correlated Subqueries

In a *correlated subquery*, the value produced by the subquery depends on a value produced by the outer SELECT that contains it. The important thing to note about correlated subqueries is that they are relatively expensive because the subquery must be executed once for every value produced by the outer SELECT. An uncorrelated subquery, like the previous example using the EXISTS keyword, is executed only once.

Here is an example of a correlated subquery to select the ten highest-paid employees. The ranking problem is surprisingly hard in SQL because it is a declarative language. In essence, this query asks, “show me the employees for whom there are less than ten higher-paid employees.”

```
SELECT *
        FROM Employees main
        WHERE 10 >
                ( SELECT COUNT (DISTINCT (Salary))
                        FROM Employees sub
                        WHERE sub.Salary > main.Salary )
        ORDER BY Salary DESC
```

A much easier way to do this is

```
SET ROWCOUNT 10 (or use CFQUERY MAXROWS=10)
SELECT *
        FROM Employees
        ORDER BY Salary DESC
```

Set Operations

The hairier parts of SQL are no doubt thanks to its origin in the holy grail of computer science: set theory. Nevertheless, thinking about SQL in terms of set theory can save you thinking time when looking for the right keyword to introduce a subquery.

Computing the Union of Two Sets

This one's easy. To select all rows in both sets, use the SQL UNION operator, as in

```
SELECT SSN
        FROM Employees
UNION
SELECT SSN
        FROM Vendors
```

By default, UNION omits duplicate rows. To include duplicate rows, use UNION ALL.

Computing the Intersection of Two Sets

The intersection of two sets is all rows which are in both sets. Use the IN or EXISTS keyword to introduce subqueries showing rows which are contained in both tables.

```
SELECT *
        FROM Employees
        WHERE SSN IN
        ( SELECT SSN
                FROM Vendors )
```

You could also compute this intersection with a join.

```
SELECT *
        FROM Employees JOIN Vendors
                ON Employees.SSN = Vendors.SSN
```

Computing the Difference of Two Sets

The difference of two sets is the set of elements contained in one but not the other. Use the keywords NOT EXISTS or NOT IN to introduce subqueries showing the difference between two tables.

```
SELECT *
        FROM Employees
        WHERE SSN NOT IN
                ( SELECT SSN
                        FROM Vendors )
```

Temporary Tables

A temporary table is one which exists only during a group of statements (in SQL Server 7 parlance, a *batch*) or for the duration of the database connection. Temporary tables are useful primarily

- For staging updates to a large number of rows, especially when the updates must be done in a single transaction. In SQL Server, temporary tables can be up to four times faster than regular tables.
- For storing intermediate results of complicated operations. A two-stage approach using a temporary table is often faster and easier to understand than a behemoth SQL statement.

The implementation of temporary tables varies from database to database. SQL Server 7 supports *private* and *global* temporary tables.

Private temporary tables

- Can only be accessed by the connection which created the table
- Exist only for the life of the connection
- Are prefixed with a single pound sign (#*table_name*)

Global temporary tables

- Can be accessed by any connection
- Exist as long as any connection is still using them
- Are prefixed with a double pound sign (##*table_name*)

Creating Temporary Tables

The simplest way to create a temporary table is to select data into it. SQL Server automatically creates the table using the data types and column names from the corresponding table in the SELECT statement.

```
SELECT *
        INTO #NewHires
        FROM Employees
        WHERE HireData > ‘3/12/2000’
```

You can also create a temporary table using the CREATE TABLE statement.

Deleting Temporary Tables

Temporary tables will automatically be dropped when they are no longer in use. However, you can manually delete a temporary tables when you’re done with it using the DROP TABLE statement, as in

```
DROP TABLE #NewHires
```

Views

Views are synthetic tables. A view contains a combination of data that exists in real tables and other views. The basis of a view is just a SELECT statement. Views are useful in the following cases:

- To restrict users to particular columns of data. You name only permitted columns in the SELECT list in the view.
- To restrict users to particular rows of tables. You specify a WHERE clause to return only the permitted rows.
- To constrain inserted and updated values to certain ranges using WITH CHECK OPTION.
- To provide access to derived data without having to store redundant data in the database.
- To hide the details of a complicated SELECT statement such as a frequently-used join.

Creating Views

To create a view, simply prefix a SELECT statement with CREATE VIEW.

```
CREATE VIEW Employees_Public AS
SELECT EmployeeID, LastName, FirstName
        FROM Employees
```

Using Views

To reference a view, simply use it as you would a regular table.

```
SELECT *
        FROM Employees_Public
```

Limitations of Views

Because a view is not really a table, it cannot be indexed or modified with ALTER TABLE, RENAME TABLE, or RENAME COLUMN. In addition, because the SELECT statement underlying a view must be merged with user queries on the view, a view

- Cannot contain an INTO clause
- Cannot contain a UNION clause
- Cannot contain an ORDER BY clause

You cannot INSERT or UPDATE through a view containing a join unless all columns being updated are part of the same table. You cannot DELETE through a view containing a join because the database would not know from which underlying table to delete rows.

Deleting Rows

The syntax for a DELETE statement is simple:

```
DELETE FROM table_name
        WHERE conditions
```

You should (almost) always use a WHERE clause with your DELETE statement. On critical systems, you should first determine how many rows will be deleted by replacing the DELETE statement with “SELECT COUNT(*).”

Note

With SQL Server 7.0, if you need to delete every record from a table, it is much faster to use the TRUNCATE TABLE statement.

Using Subqueries with DELETE

The WHERE clause in a DELETE statement can be almost as complicated as in a SELECT. If, for example, you want to delete all products which have never been ordered, you would write

```
DELETE
        FROM Products
        WHERE NOT EXISTS
                ( SELECT *
                        FROM OrderDetails od
                        WHERE od.ProductID = Products.ProductID )
```

You cannot, however, name a table in the FROM clause of a subquery which is named in the DELETE FROM clause.

Limitations of DELETE

You cannot delete from multiple tables at the same time (a join).

Updating Rows

An UPDATE statement looks like this:

```
UPDATE table1, table2 …
        SET assignments
        WHERE conditions
```

Selecting Rows to Update

Like DELETE, the WHERE clause can be almost as complicated as in a SELECT, containing multiple conditions, subqueries, etc.

Using Subqueries with UPDATE

Subqueries with UPDATE work just like they do with DELETE. To flag all products as inactive which have never been ordered, you would write

```
UPDATE Products
        SET Status = ‘Inactive’
        WHERE NOT EXISTS
                ( SELECT *
                        FROM OrderDetails od
                        WHERE od.ProductID = Products.ProductID )
```

As with DELETE, you cannot name a table in the FROM clause of a subquery which is named in the UPDATE statement.

Updating with Derived Values

The right side of an assignment in an UPDATE statement can be an expression. For example, to raise prices on all products, you might write

```
UPDATE Products
        SET UnitPrice = UnitPrice * 1.03
```

You can also include a single-valued subquery in an assignment. For example, to set your prices equal to the lowest competitor’s, you would write

```
UPDATE OurPrices op
        SET UnitPrice =
                ( SELECT MIN (UnitPrice)
                        FROM CompetitorsPrices cp
                                WHERE cp.ProductID = op.ProductID )
```

Note: this is a good way to go out of business.

Updating with Selected Values

The assignments in an UPDATE statement can take one of two forms. The clearest form uses separate assignments:

```
UPDATE Customers
        SET FirstName = ‘David’, LastName = ‘Chandler’
        WHERE CustomerID = 103
```

Alternatively, you can write

```
UPDATE Customers
        SET (FirstName, LastName) = (‘David’, ‘Chandler’)
        WHERE CustomerID = 103
```

The latter form is useful for bulk assignments, such as this statement to update the Customers table from the NewAddress table.

```
UPDATE Customers c
        SET (Street1, City, ST, ZIP) =
                (( SELECT Street1, City, ST, ZIP
                        FROM NewAddress n
                        WHERE n.CustomerID = c.CustomerID ))
        WHERE CustomerID IN
                (SELECT CustomerID
                        FROM NewAddress)
```

UPDATE Limitations

You cannot query the table that is being modified. You can, however, refer to the present value of a column in an expression or in a WHERE clause in a subquery, as shown in the example to raise all prices.

You can not update through a join unless all columns to be updated are contained in a single table.

Inserting Rows

There are actually two SQL statements which can be used to insert data: INSERT and SELECT INTO.

Single Rows with INSERT

A single-row INSERT looks like this:

```
INSERT INTO Employees (FirstName, LastName, HireDate)
VALUES (‘David’, ‘Chandler’, ‘3/12/2000’)
```

You do not have to specify values for every column. The database supplies values for the remaining columns as follows:

- It generates a serial number for an unlisted serial column.
- It generates the default value for each unlisted column which specifies a default value.
- It generates a NULL value for any unlisted column that allows NULLs but does not specify a default value.

Note that you must include a value for every column which neither allows NULLs nor specifies a default value.

Multiple Rows with INSERT

You can use INSERT with a SELECT statement to include rows from another table.

```
INSERT INTO Employees (FirstName, LastName)
        SELECT FirstName, LastName
                FROM NewEmployees
                WHERE HireDate > ‘3/12/2000’
```

When using the INSERT … SELECT construct,

- The SELECT statement cannot contain an INTO clause.
- The SELECT statement cannot contain an ORDER BY clause.
- The SELECT statement cannot refer to the table into which you are inserting rows.

Multiple Rows with SELECT … INTO

The SELECT … INTO statement inserts new rows into a new table. SQL Server will automatically create the new table using the data types and column names from the SELECT list. If you want to insert rows into an existing table, however, you cannot use SELECT … INTO.

Online Analytical Processing (OLAP)

With SQL Server 7.0, Microsoft introduced OLAP capabilities. OLAP is essentially crosstabs on steroids. With OLAP, however, the raw data necessary for multidimensional analysis is pre-computed on the server.

Typical crosstab questions include things like

- How many white cars of any model were sold by year?
- How many red cars were sold in the last five years by model?
- How many cars were sold last year by color?

SQL Server provides OLAP capabilities through the CUBE and ROLLUP extensions to GROUP BY in Transact-SQL. In addition, conditional expressions such as CASE in SELECT and UPDATE statements are useful in implementing data warehouse solutions.

A datacube is a special kind of cross product which represents all possible permutations of attributes in the underlying data. A datacube for car sales data might look like

```
Units_sold      model       year   color
--------------  --------    -----  ------
62              Chevy       1990   Blue
5               Chevy       1990   Red
87              Chevy       1990   White
154             Chevy       1990   ALL
49              Chevy       1991   Blue
…
64              Ford        1992   Red
```

This data is extracted from various joins on the raw sales data (like an Orders table) and makes computations on the data much faster than it would be to compute all the sums from the underlying tables.

For example, to show sales of all cars by year, we need only write

```
SELECT year, SUM (Units_sold)
        GROUP BY year
```

Using a datacube, it is possible to slice and dice data many ways with simple aggregate functions and a WHERE clause. All sums are pre-computed, which saves future queries from having to do this step.

SQL Server 7.0 provides very powerful wizards for doing OLAP processing, and Excel 2000 pivot tables (crosstabs) can take advantage of SQL Server datacubes to significantly reduce processing time.

Where to Go from Here

Application Help

For help using SQL, consult the Transact-SQL Help available through the Help menu in Query Analyzer.

For more in-depth help with SQL and SQL Server, consult Start | Programs | Microsoft SQL Server 7.0 | Books Online.

Books

Believe it or not, *SQL for Dummies* is a pretty decent tutorial on even the finer points of SQL. Don't let the title fool you.

The best all-around tutorial on SQL is the *Informix Guide to SQL Tutorial*, published by Informix Press.

Inside SQL Server 7.0 is a superb reference for all things SQL. Written by a Microsoft insider, it is the definitive reference for SQL Server behavior and rationale. Chapter 12, Transact-SQL Examples and Brainteasers, is simply a blast. This book also includes a 120-day evaluation copy of SQL server 7.0 on CD.

Joel Celko's *SQL for Smarties* is a highly-technical reference for SQL programmers, devoting almost an entire chapter to NULLs. Unless you're really on the metal, you probably won't find it useful, but if you are, you need this book. Celko devotes a few pages to the interesting and challenging topic of non-recursive tree processing algorithms.

Lesson 17 Building Distributed Applications

Introduction

ColdFusion can connect to a variety of popular Internet services, including mail, directory, and other Web servers. In this lesson, you will learn how to use these capabilities to build distributed Web applications.

Objectives

By the end of this lesson, you should be able to

- Use CFMAIL and CFPOP to send and receive e-mail
- Understand how CFLDAP can be used to provide centralized authentication and authorization services
- Use CFFTP to connect to FTP servers
- Use CFHTTP to build Web agents and distributed applications

Sending Mail with CFMAIL

ColdFusion provides the capability to send mail from the ColdFusion server in the CFMAIL tag. Using CFMAIL, you can

- Send plain text or HTML-formatted messages
- Attach a file to the email
- Send to a group of recipients specified in a query

CFMAIL can be used to

- Notify an administrator of error conditions
- Send a username and password to someone registering for a free download
- Inform customers of order status

CFMAIL can also be used to annoy people. Don’t do it.

How CFMAIL Works

When you use the CFMAIL tag, ColdFusion connects to an SMTP server to send the message. ColdFusion sends a separate message to each recipient, even if recipient addresses are specified in a query. This protects the confidentiality of the recipients’ e-mail addresses.

Tip
Always test CFMAIL with a few known addresses before you turn it loose on live users.

CFMAIL can communicate with any standard SMTP server. You specify the default server in the ColdFusion Administrator. You can also specify a different server in the CFMAIL tag. In the ColdFusion Administrator, you can also specify whether to log failed messages, all messages, and/or the contents of each message.

ColdFusion uses the following directories for mail services. You should periodically check the mail logs to look for potential problems.

\CFUSION\Mail\Log	Log files
\CFUSION\Mail\Spool	Queued messages
\CFUSION\Mail\Undelivr	Undeliverable messages

Mail Security

Using CFMAIL, you specify both the sender and recipient addresses. The sender address can be any valid SMTP address. An unscrupulous person could use CFMAIL to send mail to people anonymously, or worse, as someone else. This is not a problem with CFMAIL, but rather with the SMTP protocol. Internet mail was never intended to be secure. Any SMTP mail client, including Netscape and Outlook Express, lets you specify any return address you like. "Forging" Internet mail is no different than mailing a letter using someone else's return address.

Also be aware that Internet mail is not encrypted in transit. You should not use CFMAIL to send sensitive information such as someone's credit card number or a highly-sensitive password.

Handling Returned Mail

There are basically two approaches for handling returned mail. For small amounts of mail, you can simply use someone's real address in the FROM field. If a message bounces, mail servers will usually send a notice to the sender.

Note

You can specify an Internet email address including both the address and a name. For example, to send a message from ACME Customer Service (service@acme.com), you would use FROM="service@acme.com (ACME Customer Service)"

For large amounts of mail, you will probably want some sort of automated processing to handle returned mail. One way to do this would be to specify a POP mail box as the return address and use the CFPOP tag to retrieve mail from the POP server on a regular basis.

Retrieving Mail with CFPOP

ColdFusion can connect to any POP3 (Post Office Protocol) server to retrieve e-mail. POP3 is the standard protocol used by almost all Internet Service Providers. In addition, most corporate e-mail systems, including Microsoft Exchange, offer the ability to retrieve mail via POP3.

Using CFPOP, you can

- Retrieve message headers only or all message details
- Automatically save attachments to a directory

Typical applications for CFPOP include

- Building a Web-based e-mail gateway to allow secure retrieval of e-mail over the Internet
- Automatically handling returned mail

How CFPOP Works

When you use the CFPOP tag, ColdFusion connects to the specified POP server with the supplied username and password. Depending on the attributes you specify, ColdFusion will return a query containing the message headers only or all message details. If you specify a path for attachments, ColdFusion will also save attachments and include the saved filenames in the query results.

The query result set consists of one row for each message and the message details in the query columns. You can treat this query like any other ColdFusion query variable.

Deleting Messages

By default, messages are not deleted when you retrieve them. To delete messages which have already been retrieved, use CFPOP with ACTION=Delete.

Using Directory Services with CFLDAP

LDAP (Lightweight Directory Access Protocol) is the Internet standard for directories. The major Internet people search engines like BigFoot and Yahoo use LDAP. But more importantly for most ColdFusion developers, enterprises are rapidly moving toward LDAP to enable a single sign-on for all operating systems and applications. Using ColdFusion, you can build applications which use the LDAP infrastructure to authenticate and authorize users.

A popular LDAP server is Netscape Directory Server. In addition, Microsoft Exchange 5.5 running on Windows NT 4.0 can act as an LDAP server, and Windows 2000 exposes its ActiveDirectory services using LDAP.

How CFLDAP Works

When you use the CFLDAP tag, ColdFusion connects to the specified LDAP server with the supplied username and password. Depending on whether you are requesting information or modifying data, ColdFusion will either return information in a query or update the data as requested.

Authenticating and Authorizing Users

A primary use of LDAP is to validate a username and password, and to find out what resources the user is authorized to use. Your ColdFusion application can use LDAP in one of three ways.

1) You can connect directly to an LDAP server using the CFLDAP tag.
2) If you have installed Advanced Security Services, you can configure ColdFusion Server to automatically use an LDAP server to authenticate your application users.
3) You can perform authentication at the Web server level using an LDAP-compatible product like Netegrity SiteMinder.

Obtaining User Information

Another popular use of LDAP is to obtain user information. When a user logs in, your ColdFusion can query all information associated with the user in the LDAP directory, including e-mail address, phone number, location, etc.

Modifying Directories

CFLDAP can also be used to add new records and modify records on an LDAP server. This enables you to write a powerful administrative tool in ColdFusion.

Transferring Files with CFFTP

CFFTP implements the Internet standard File Transfer Protocol and enables ColdFusion Server to communicate with FTP servers as an FTP client.

Note

CFFTP does not enable ColdFusion Server to communicate with a browser via FTP. To send a file to the browser, use CFCONTENT. To receive a file from a browser, use CFFILE with ACTION=Upload.

CFFTP provides full-featured FTP support, including

- The ability to connect through a proxy server
- The ability to cache connections for improved performance
- The ability to send or receive files in ASCII or binary mode

You may wish to use CFFTP

- To connect to legacy and back-end systems
- To transfer files to a content staging area outside a firewall
- To transfer files to business partners on a regular basis

Almost every computing platform can run an FTP server, including mainframes. For this reason, CFFTP is often the simplest way to communicate with heterogeneous systems.

Building Distributed Web Applications with CFHTTP

One of the more powerful capabilities of ColdFusion is the ability to communicate with any Web server. Using CFHTTP, you can do anything a browser can do, including

- Retrieve information and save it to a variable or file
- Post form data
- Upload files
- Send cookies
- Use SSL to encrypt data

Note

You cannot use SSL support in CFHTTP on Window NT when running ColdFusion as a service. For more details, see *Developing Web Applications with ColdFusion*, Chapter 15: Interacting with Remote Servers.

You can use CFHTTP to do many useful things:

- Build a meta search engine to aggregate search results or news
- Maintain competitive intelligence by automatically parsing competitors' Web sites
- Automatically retrieve data from a subscription service and make it available on your intranet
- Build an interface to a transaction processing system
- Connect to Web-enabled devices
- Connect to legacy or back-end systems
- Build an automated load tester for Web applications
- Build an automated Web site monitoring tool

Using the CFHTTP GET Method

The GET method sends a URL to a Web server and retrieves the resulting data. For example, to retrieve the Yahoo home page, you would write

```
<CFHTTP METHOD=”GET”
      URL=http://www.yahoo.com/
      RESOLVEURL=”Yes”>
```

If you want to retrieve images or follow links on the page, you must use additional CFHTTP requests just like a browser does. The RESOLVEURL attribute tells ColdFusion to automatically convert relative paths to absolute paths, which makes subsequent retrievals easier.

After a CFHTTP request, the following variables contain information about the request:

CFHTTP Variables Defined After a Request	
CFHTTP.Header	The raw response header
CFHTTP.ResponseHeader[key]	A structure containing each field in the response header
CFHTTP.MimeType	The MIME type of the returned content.
CFHTTP.FileContent	The body of the response.
CFHTTP.StatusCode	The HTTP error code if the THROWONERROR attribute was specified.

Using CFHTTP to Retrieve a File

By default, ColdFusion puts the resulting data stream into the CFHTTP.FileContent variable. You can save it to a file using the PATH and FILE attributes. ColdFusion saves the MIME type of the file as CFHTTP.MimeType, so you can use this method with any text or binary file.

Using a Proxy Server

You can retrieve Web pages through a proxy server by specifying the optional PROXYSERVER and PROXYPORT attributes in CFHTTP.

Using the CFHTTP POST Method

Use the CFHTTP Post method to send form data, URL parameters, cookies, HTTP headers (CGI variables), or files. Use CFHTTPPARAM to specify each variable to send.

```
<CFHTTP METHOD="POST"
      URL="http://tx.domain.com/post_tx.cfm"
      USERNAME="matilda"
      PASSWORD="#Application.TxPassword#">
<CFHTTPPARAM TYPE="Cookie"
      NAME="AuthUser"
      VALUE="#Application.TxAuthToken#">
<CFHTTPPARAM TYPE="URL"
      NAME="action"
      VALUE="post_charge">
<CFHTTPPARAM TYPE="Formfield"
      NAME="CreditCardNo"
      VALUE="#Form.CreditCardNo#">
<CFHTTPPARAM TYPE="Formfield"
      NAME="Amount"
      VALUE="#OrderTotal#">
<CFHTTPPARAM TYPE="CGI"
      NAME="USER_AGENT"
      VALUE="My Application">
<CFHTTPPARAM TYPE="File"
      NAME="Packing List"
      FILE="c:\temp\packlist.txt">
</CFHTTP>
```

If posting to another ColdFusion page, the CFHTTP parameters of type URL, Formfield, and Cookie are available as URL, Form, and Cookie variables, respectively. The processing page should receive posted files using CFFILE ACTION=Upload.

CFHTTPPARAM TYPE="CGI" allows you to send HTTP headers. The resulting CGI variable in the processing will be prefixed with "HTTP_". For example, if you send a CGI parameter named USER_AGENT, the processing page will see a CGI variable named HTTP_USER_AGENT. This particular example demonstrates how you can send the name of your "browser" as part of the request.

Creating a Query from a Text File

A real time-saving capability of CFHTTP is the ability to create a query from a delimited text file retrieved with CFHTTP. This is a very convenient mechanism for retrieving "back-end" pages created for the purpose of sharing data. For example, to create a query from a comma-delimited file where each value is contained in double quotes, you would write

```
<CFHTTP METHOD="GET"
      URL="http://dserv/orders/july/orders.txt"
      NAME="JulyOrders"
      DELIMITER=","
      TEXTQUALIFIER="""""">

<CFOUTPUT QUERY="JulyOrders">
      OrderID: #OrderID#
      Order Number: #OrderNumber#
</CFOUTPUT>
```

Note

There are six quotes altogether in TEXTQUALIFIER. The outer two surround the attribute value. The value itself is two double quotes (""), but each quote must be escaped with an additional quote because it occurs in a quoted string.

The real power of CFHTTP is demonstrated when you retrieve not static files, but other dynamic Web pages. For example, you might write a script on a mainframe Web server to accept a URL query parameter and produce a comma-delimited text file. Your ColdFusion application can then use CFHTTP to post the required query parameter and retrieve the data as a query.

Using CFHTTP to Create Agents

You can use the CFHTTP tag to create Web "agents" which go out and collect data from other Web sites. However, you should make sure that the Web sites you are visiting allow this kind of use. Ideally, you should build cooperative agents, in which you and the other Web sites agree on a convenient data format for exchange. This significantly simplifies parsing the data stream, and helps ensure that what you are doing is legal.

Exchanging Data with WDDX

Web Distributed Data Exchange (WDDX) lets you easily exchange structured data among a growing variety of applications. WDDX is an XML DTD created by Allaire Corporation, and is transparently supported by a variety of applications, including ColdFusion, ASP, JavaScript, Python, perl, Java, or any COM-enabled application.

Using WDDX, you can “wrap up” variables, including structures and queries, and send them to another application. WDDX is the preferred method for exchanging data between two ColdFusion servers because ColdFusion handles all the required parsing.

Tip

Besides exchanging data between servers, WDDX is very useful for converting a complex data type (array, structure, or query) into a simple string variable that can be stored in a database field or Client variable. We used this technique for advanced session data management in Lesson 13: Using the Application Framework.

How WDDX Works

WDDX is typically used to return data from a back-end page. The steps are as follows:

1) The application template (a ColdFusion page) uses CFHTTP to send parameters to a back-end page.
2) The back-end page processes the form variables, etc., and runs a query or performs some other action.
3) The back-end page calls CFWDDX to serialize the query or other data structures and place the results in a local variable.
4) The back-end page uses CFOUTPUT to send back the serialized data structures.
5) The application page reads CFHTTP.FileContent and calls CFWDDX to deserialize it into native ColdFusion data structures.

Creating a Back-end Page with WDDX

A back-end page created with WDDX should do the following:

1) Read any Form variables, URL parameters, etc.
2) Perform some actions based on this input.
3) Use <CFSETTING> to turn off debug output, which would corrupt the WDDX data stream.
4) Call CFWDDX and send the results using CFOUTPUT.

```
<CFWDDX ACTION="CFML2WDDX"
      INPUT="#VariableToExport#"
      OUTPUT="VariableNameToHoldWDDXStream">


<CFOUTPUT>
#VariableNameToHoldWDDXStream#
</CFOUTPUT>
```

The back-end page should not send any HTML or text, only the serialized WDDX stream.

Creating an Application Page with WDDX

An application page created with WDDX should do the following:

1) Send a request to the back-end page using CFHTTP.
2) Call CFWDDX to deserialize CFHTTP.FileContent.

```
<CFWDDX ACTION="WDDX2CFML"
      INPUT="#LocalVariableName#"
      OUTPUT="VariableNameToHoldData">
```

After calling CFWDDX, the application page will have access to the sent data through the ColdFusion variable named in the CFWDDX OUTPUT attribute.

Server-Client Data Exchange

WDDX can also be used to send data to JavaScript running in a browser. However, this is not for the faint of heart. For most application, it is probably more straightforward to send data to JavaScript simply by writing JavaScript assignment statements in your ColdFusion application page. The ColdFusion methods for serializing and deserializing data to JavaScript are

- CFWDDX ACTION="CFML2JS"
- CFWDDX ACTION="WDDX2JS"

Where to Go from Here

Application Help

See the following chapters in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 13: Sending and Receiving Email
Chapter 14: Managing Files on the Server
Chapter 15: Interacting with Remote Servers
Chapter 16: Connecting to LDAP Directories

For more information on WDDX, see the following book in the Studio Help resource pane:

Exchanging Data via XML

Online Resources

See the Allaire Alive video “Creating Web Agents” at http://alive.allaire.com.

You can find WDDX drivers for other languages and platforms at www.wddx.org.

Lesson 18
Working with Legacy Data

Introduction

Today's enterprise Web applications rarely stand alone. ColdFusion provides a variety of tools for working with data across the enterprise.

Objectives

By the end of this lesson, you should be able to

- Understand the various methods available for connecting to legacy data
- Use CFFILE and CFDIRECTORY to work with files and directories
- Understand techniques for parsing and creating text files
- Understand how to use Data Transformation Services to import and export data

The ColdFusion Connectivity Toolbox

As we saw in the last chapter, ColdFusion gives you a wide variety of methods which you can use to send and receive data from other servers.

File-Oriented Methods

For simplicity and interoperability between databases, there's still nothing like the good old delimited text file. In a typical ColdFusion environment, you can

- Use SQL Server Data Transformation Services to import and export data from other sources, including text files
- Use CFFTP to transfer files with FTP
- Use CFHTTP to retrieve files from Web servers
- Use CFFILE to read and write files on network drives
- Create a query from a text file with CFHTTP
- Create a query from a text file using an ODBC driver
- Parse a text file using ColdFusion lists
- Create a delimited text file from a query using CFOUTPUT

In previous lessons, we looked at the first three methods. In this lesson, we'll look at the remaining methods.

Data-Oriented Methods

Applications which require real-time connectivity to remote data may not be able to use intermediate files. For these types of applications, you will be better off with one of the following methods:

- Use CFWDDX to exchange data between ColdFusion and other CFWDDX-enabled applications
- Use CFHTTP to post and retrieve dynamic queries to any Web server
- Use CFQUERY to connect to any ODBC-compatible data source
- Use CFOBJECT to connect to CORBA and COM objects
- Create custom CFX tags to connect to anything at all
- Connect to remote data sources from within SQL Server stored procedures using OpenQuery and OpenRowSet

Working with Files and Directories

Using the File System to Exchange Data

When you need to exchange files with another server on the same network, it is often possible to use a shared drive for data exchange. Any configuration can work as long as both ColdFusion and the other application have access to the shared drive.

The problem with any file-oriented method of data exchange is that one or both applications must continually poll a directory to see whether any new files have been placed there. In addition, you must be careful to consider

- locking (is the other application writing to the file?)
- data verification (did the file transmit successfully?), and
- synchronization (has the other application seen this file yet?).

Where a network shared drive cannot be set up, you may be able to create a shared data space by running an FTP or HTTP server on the ColdFusion server, the remote server, or somewhere in between. Between CFFILE, CFFTP, and CFHTTP, there has to be some way to connect!

Creating a Locking Mechanism

There are no cookie-cutter approaches to locking problems, but these are some typical solutions:

1) Use a flag file to keep track of which files are in use
2) Read from or write to a temporary copy rather than directly from a file which may be updated
3) Use CFLOCK to facilitate file sharing with other ColdFusion applications

Verifying Data in Shared Files

ColdFusion does not provide pre-built functions for computing and verifying file checksums. However, you can check to see if another application is done writing a file by using a special end-of-file flag like “99999999.” If your application reads the file but does not detect the end-of-file flag, you know the other application is still writing to it.

Synchronizing Shared Files

Synchronization methods typically fall along these lines:

1) Use flag files to indicate when your application has seen a file or made modifications to it, or
2) Simply delete each file when you’re done with it.

Working with Files

The CFFILE tags lets you read, write, append, and delete files, as well as receive file uploads. The only trick to using CFFILE is that you can only create a file in an existing directory. To see if a file already exists, use the FileExists () function.

Working with Directories

ColdFusion lets you read the contents of a directory using the CFDIRECTORY tag. Naturally, ColdFusion returns the directory listing as a query. To see if a directory exists, use the DirectoryExists () function.

Polling for Updates

In order for a remote server to tell your ColdFusion application that new data is available in a shared directory, the remote server can simply send a Web request to your application. If this method is not available, however, your ColdFusion application will have to regularly poll the shared directory for updates.

Fortunately, this is not difficult. In the ColdFusion Administrator, you can set up ColdFusion pages to run at a regular interval. You can also add new tasks to the ColdFusion Scheduler programmatically with the CFSCHEDULE tag. To poll a directory, you simply write a ColdFusion page which looks for files in a directory and add it to the schedule.

Creating a Query from a Text File using ODBC

In the last lesson, you learned how to use CFHTTP to create a query from a text file on another Web server. But what if the text file is on the ColdFusion server already? Of course, you could write a back-end page just to serve up the file to CFHTTP, but there is also another way.

How It Works

One of the standard ODBC drivers is the text file driver. This driver allows you to query a delimited text file using SQL. The text file ODBC driver will not support the full range of SQL statements, but you can at least use it to "convert" a text file to a ColdFusion query object. To use this method,

1) Create the ODBC data source using ColdFusion Administrator or the ODBC Control Panel.
2) Use the CFQUERY tag to issue a SELECT statement retrieving rows from the file.
3) Use CFOUTPUT or CFLOOP to display or loop over the results as usual.

Comparison with Other Techniques

The main advantage of this approach is that it saves you, the application programmer, all the work involved in reading a text file line by line and parsing out the columns, etc. However, if the file is not organized into rows and columns, this method will not work.

Parsing a Text File Using ColdFusion Lists

Unlike every procedural language, ColdFusion's declarative syntax gives you no easy way to read a text file line by line. However, you can use the following technique.

1) Read a file into a ColdFusion variable using CFFILE ACTION=Read.
2) Treat the file as a ColdFusion list delimited by the newline sequence, Chr(10) & Chr(13).
3) To read each line of the file one at a time, you simply loop over the list.

To further parse a delimited file into fields (data columns), you simply treat each line as a list and use a nested loop to iterate over each field. This is demonstrated in the example.

```
<CFFILE action="READ"
      file="D:\TEMP\Active setup log.txt"
      variable="myFile">
<CFSET newline = Chr(10) & Chr(13)>
<!--- Outer loop looks at each line --->
<CFLOOP index="thisLine"
      list="#myFile#"
      delimiters="#newline#">
      <!--- Inner loop looks at each field --->
      <CFLOOP index=”oneField”>
            list=”#thisLine#”
            delimiters=”,”>
      <!--- Process each field here --->
      </CFLOOP>
</CFLOOP>
```

Programmatically Creating a Query Object

If neither the CFHTTP nor ODBC methods of creating a query object from a text file are suitable for your application, you can programmatically create a query object using the ColdFusion query functions QueryNew, QueryAddColumn, QueryAddRow, and QuerySetCell. Any query you create with these functions can be used like a regular ColdFusion query object.

Creating a Delimited Text File with CFOUTPUT

If you wish to create a back-end page which produces a delimited text file for other applications, simply use CFQUERY and CFOUTPUT.

You should also make sure that your application page does not accidentally send any HTML (don’t forget about Application.cfm) and you should set the MIME content type to text/plain.

Here is some sample code to create a comma-delimited file containing employee information.

```
<CFQUERY NAME=”ListEmployees” ...>
SELECT FirstName, LastName, EmployeeID
      FROM Employees
</CFQUERY>

<CFCONTENT TYPE=”text/plain”>

EmployeeID,First Name,Last Name
<CFOUTPUT
QUERY=”ListEmployees”>#EmployeeID#,#FirstName#,
#LastName#
</CFOUTPUT>
```

Note that the opening <CFOUTPUT> tag and the employee data are on the same line. If you break them up, ColdFusion insert an extra return after each line.

When run against the Northwind sample database which comes with Microsoft SQL Server 7.0, this example produces

```
EmployeeID,First Name,Last Name
1,Nancy,Davolio
2,Andrew,Fuller
3,Janet,Leverling
4,Margaret,Peacock
5,Steven,Buchanan
6,Michael,Suyama
7,Robert,King
8,Laura,Callahan
9,Anne,Dodsworth
```

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 14: Managing Files on the Server

Lesson 19
Database Design and Integrity

Introduction

One of the curious facts of life in the Web Age is that Web developers with no previous training or experience with databases are expected to perform the role of database administrators. If that's you, then this lesson is for you—a crash course to help you design databases and maintain database integrity.

Objectives

By the end of this lesson, you should be able to

- Identify and avoid common database design mistakes
- Understand database relationships, constraints, and referential integrity
- Understand the purpose of triggers and when to use them
- Use indexes to boost performance
- Write simple stored procedures

Databases in a Nutshell

In a nutshell, a database is a collection of tables and the relationships between them. By following a few simple rules, you can make your life a lot easier.

Tables

1) Each table should represent one and only one kind of thing, be it a person, order, item, etc.
2) Tables should not duplicate information found in other tables.
3) Tables should not contain information which may be derived from other information.

There are occasions when each of these rules may be broken for performance reasons, but life is generally simpler if you stick to these rules.

Rows

1) Rows must stand alone. A row should not depend on any other row. The order of rows should not be significant in the data model.
2) Rows must be unique. In every row, some column or combination of columns must contain a unique value.

Columns

1) Columns must stand alone. The order of columns is immaterial.
2) Columns must contain only single values, not lists or repeating groups. Composite values should be broken into separate groups (like FirstName, LastName).
3) Each column must have a unique name and a single data type.

Database Constraints

When you define a table, you may create column or table *constraints*. You may constrain the following characteristics:

- Data type (integer, char, datetime, etc.)
- Format (yy/mm/dd)
- Range (100-500)
- Allowable values (only grades A, B, C, or D)
- Uniqueness
- Whether NULLs are allowed
- A default value
- Referential constraints

Using constraints is part of good defensive programming. Constraints prevent an application from accidentally corrupting data with bad values. Constraints like the default value constraint and referential constraints help ensure data consistency. The only downside to constraints is that the database burns CPU cycles checking them anytime modifications are made to the constrained columns.

Designating Keys

Primary Keys

The *primary key* of a table is the column or columns whose values are different in every row. The column(s) you choose for the primary key should always be unique. Once you establish the primary key, the database will not let you insert a record with a duplicate key. NULL values are not allowed in a primary key column.

Composite Keys

Sometimes, no single column uniquely identifies every record. In these cases, the combination of columns which guarantees uniqueness is called the *composite key*. You can easily create a composite key in most relational databases. Composite keys are, however, a little trickier to work with than single-column keys for two reasons:

1) Every column in the composite key must be duplicated in each table which references the composite key.
2) For most applications, it's very handy to be able to uniquely identify any record using a single number.

Surrogate Keys

In general, databases work much faster with numbers than with text. For this reason, it usually makes sense to assign a unique numeric key to each record, regardless of whether it is the logical key for the table. This is called a *surrogate key*. You can still enforce the uniqueness of the logical key using constraints.

The most convenient surrogate key is typically an auto-increment column. Informix calls this a serial data type. SQL Server calls it an identity type. Either way, when you insert a new record into the table, the database automatically generates the next available number for the column.

Foreign Keys

A foreign key is a column or columns that contain values that match the primary key in another table. A column can be a foreign key and part of a primary key at the same time, as is the case in a typical OrderDetails table, where the OrderID is part of the primary key (OrderID, ProductID) and is also a foreign key referencing the Orders table.

E/R Diagrams

An Entity/Relationship diagram simply shows primary keys and the relationships between tables. There are many tools for creating E/R diagrams. Some make a distinction between logical diagrams and physical diagrams, but most people relate most easily to the physical data model.

Using the diagrams feature of SQL Server Enterprise Manager, it is trivial to create keys and key relationships by dragging fields from one table to another.

Modeling Relationships

There are basically three types of relationships which can be represented in a database.

One-to-One

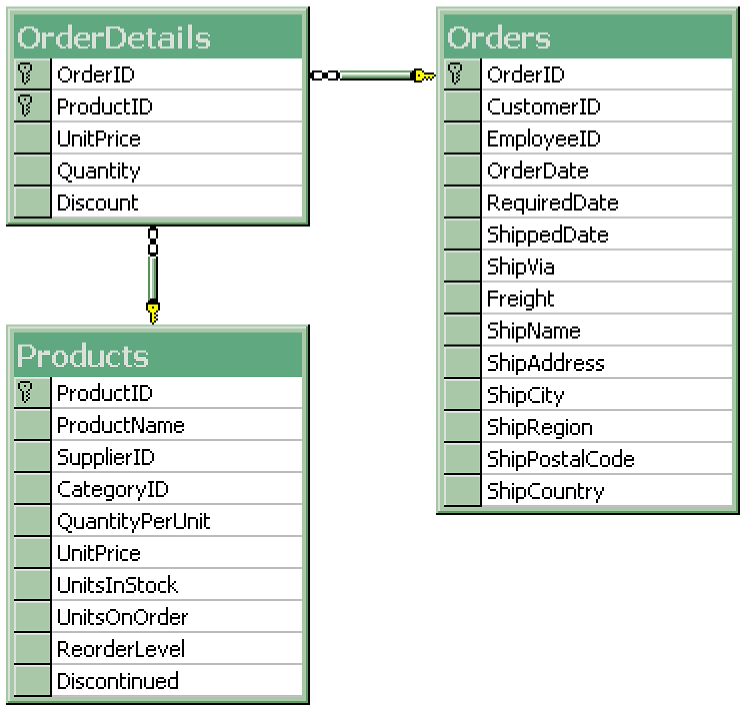
In a one-to-one relationship, every row in Table A corresponds to exactly one row in Table B and vice versa. One-to-one relationships are useful when it is desirable to store certain attributes in a separate table (for example, Social Security numbers).

One-to-Many

One-to-many relationships are the most common. In a one-to-many relationship, Table A may contain many rows which refer to a single row in Table B. An example of a one-to-many relationship is that between Customers and Orders. A customer may place many orders, and every order is placed by exactly one customer.

Many-to-Many

Many-to-many relationships cannot be implemented directly in relational databases. Instead, they must always use an intersection table. An example of a many-to-many relationship is that between Products and Orders. The join table is OrderDetails, which contains a row for each product in an order.



Principles of Good Design

How do you know when you've reached a good design? For the large part, a good design is one that follows good design principles, but there are a few clues that can tip you off to a sub-optimal design:

- Poor performance
- Frequent data corruption or inconsistency
- Difficulty creating meaningful queries

Reduce Redundancy

Every table should represent only one logical entity.

Ensure Integrity

There are three types of integrity to evaluate.

Domain Integrity

Domain integrity is concerned with ensuring the validity of data in every column. To ensure domain integrity, you can

1) Set the correct data type (including a custom type, if necessary)
2) Set default values for each column
3) Use CHECK constraints to ensure the correctness of column data

Entity Integrity

Entity integrity is concerned with the uniqueness of each row in the table. To ensure entity integrity, you can

1) Assign a unique primary key (it can be a composite key)
2) If using a surrogate key, create constraints which enforce the uniqueness of the logical key
3) Never update the primary key

Referential Integrity

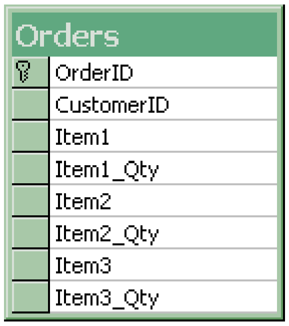
Referential integrity is concerned with enforcing the relationships between tables. To ensure referential integrity, you can

1) Use primary-to-foreign key relationships
2) Use stored procedures and transactions to ensure that updates to related tables occur together
3) Use triggers to automatically update related tables

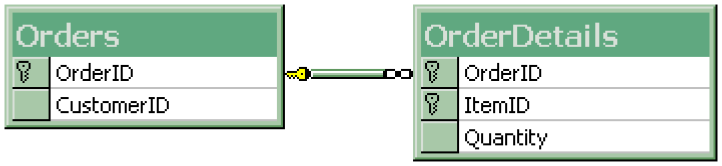
Common Design Mistakes

Repeating Groups of Information

The Orders table in the example below holds up to three items in each order. In this design, you cannot order more than three items, and if you order less, the unused space is wasted.



A much better design uses an OrderDetails table to hold the additional groups of information. Note that OrderDetails.OrderID is both a foreign key and part of a primary key.



Mixing Entities

Another common design mistake is to put a column in the wrong table, such as putting DatePaid in the OrderDetails table when it really belongs in the Orders table. Remember that each table should contain only one entity.

Duplicating Columns

A third common mistake is to include redundant information in a table. Anytime you find yourself using the same column in more than one table, it’s a good sign that you don’t have your entities entirely separate yet, and you probably need to create a new table.

Using Transactions

Database transactions ensure that changes are committed *together*. Transaction processing assures that all transactions are performed as a single unit of work, even in the presence of a hardware or system failure. Transactions have the *ACID* properties:

Atomicity means that each transaction is treated as all or nothing. If a transaction commits, all of its effects remain. If it aborts, all of its effects are undone.
Consistency ensures that data will always be logically correct. (For example, every debit must be accompanied by a credit). Constraints and rules are honored, even in a system failure.
Isolation separates transactions from each other. In the debit / credit example, another transaction can't see the work in progress, only the committed result.
Durability ensures that the effects of the transaction will persist even after a system failure. If a system failure occurs while a transaction is in progress, the transaction will be completely undone, leaving no partial effects.

Transactions are the single most important tool for maintaining data integrity.

Isolation Levels

Transaction isolation levels represent a tradeoff between concurrency and consistency. There are four transaction isolation levels. Your transactions will behave differently based on which isolation level is set, so the saying "Not to decide *is* to decide" applies here. These are listed in order of increasing consistency and decreasing concurrency.

READ UNCOMMITTED (dirty read) lets your transaction read data which may not yet have been committed by another transaction.

READ COMMITTED (SQL Server default) ensures that your transaction will never read data which has not been committed. However, if your transaction subsequently revisits the same data, that data might have changed or new rows might have been added (*phantoms*) which meet the criteria of the original query.

REPEATABLE READ ensures that if your transaction revisits data or reissues a query, no rows will have been updated or deleted. However, this level does not protect against phantoms.

SERIALIZABLE ensures that if your transaction reissues a query, the results will be exactly the same as before. No rows will have been updated, deleted, or added. Note that this requires the database server to lock rows which don't exist!

Working with Transactions

To use a transaction in SQL, you simply set the isolation level and wrap the related SQL statements in BEGIN TRANSACTION / COMMIT TRANSACTION.

```
SET TRANSACTION ISOLATIONLEVEL READ COMMITTED
BEGIN TRANSACTION
-- Update address in Employees table
-- Update address in Payroll table
COMMIT TRANSACTION
```

If one of the updates returns an error, then you *roll back* the transaction with the statement ROLLBACK TRANSACTION.

You can use transactions anywhere you can use SQL, including stored procedures and in CFQUERY. The following is perfectly legitimate in ColdFusion:

```
<CFQUERY ...>
BEGIN TRANSACTION
UPDATE Employees ...
UPDATE Payroll ...
COMMIT TRANSACTION
</CFQUERY>
```

Performance Tip

Keep in mind that the database implements transactions using locking mechanisms. Therefore, you want to keep each transaction as small as possible. If you have a large number of rows to update, you can stage the updates in a temporary table, then issue a single UPDATE inside a transaction to update the real table from the temporary table. Use this technique to avoid looping over SQL statements or nested queries inside a transaction.

Working with Transactions in ColdFusion

ColdFusion allows you to group any statements on a page together using the CFTRANSACTION tag. In order for this to work, the ODBC driver for the database must support transactions. (You will get an error if it doesn't).

```
<CFTRANSACTION ISOLATION=”read_committed”>
<CFQUERY ...>
      <!--- Update Employee address --->
</CFQUERY>
<CFQUERY ...>
      <!--- Update Payroll address --->
</CFQUERY>
</CFTRANSACTION>
```

If a statement in a ColdFusion transaction returns an error, ColdFusion will automatically roll back the transaction.

In ColdFusion 4.5, you have explicit control over transaction commits and rollbacks using <CFTRANSACTION Action=”Commit”/> or <CFTRANSACTION Action=”Rollback”/>. Complicated database operations should wrap each CFQUERY in a CFTRY block, then roll back if an exception occurs.

Note that you cannot nest transactions in ColdFusion (you can in SQL).

Caution

You should not mix CFTRANSACTION with transaction statements inside CFQUERY or a stored procedure, as this will result in undefined behavior. If you're using stored procedures, build the transaction support into the stored procedure.

Within one transaction block, you can write queries to more than one database; however, you must commit or rollback the transaction to a particular database prior to writing a query to another database.

True distributed transactions across databases, where all databases commit together, are not possible in native CFML. This could possibly be achieved using COM objects with CFOBJECT and a *transaction monitor* like Microsoft Distributed Transaction Coordinator.

Using Stored Procedures

A stored procedure is a group of SQL statements stored on the database server. Because stored procedures are parsed when they are created, they execute much faster than statements submitted on the fly.

Stored procedures help maintain database integrity by ensuring that applications interact with the database only in prescribed ways.

In addition, Microsoft Transact-SQL stored procedures give you access to more powerful programming constructs such as

- The ability to use database cursors
- The ability to connect to remote data sources using OpenQuery and OpenRowSet
- Nested stored procedure calls and recursion

Variables Useful in Stored Procedures

Local variables in stored procedures are distinguished by prefixing the variable name with an at sign (*@variable_name*).

In addition, Microsoft SQL Server provides a number of global system variables which can be used in any SQL statement. These are prefixed with two at signs. Some of the most commonly used are

@@ERROR	The status code from the last statement
@@ROWCOUNT	The number of rows affected by the last statement
@@IDENTITY	The identity value created by the last INSERT
CURRENT_USER	The database user associated with the connection

Note that CURRENT_USER (and CURRENT_TIMESTAMP) are actually keywords rather than system variables.

Because these variables are global, their values change after each applicable statement.

Creating Stored Procedures

Stored procedures are not hard at all to write. In the simplest case, all you have to do is prefix a group of statements with the CREATE PROCEDURE statement. This example creates a stored procedure to return all rows in the Employees table. The statement first checks to see if the procedure already exists. If you already know the procedure exists, you could drop the IF EXISTS statement and use ALTER PROCEDURE instead of CREATE PROCEDURE.

```
IF EXISTS (SELECT name FROM sysobjects
WHERE name = 'ListEmployees' AND type = 'P')
      DROP PROCEDURE ListEmployees
CREATE PROCEDURE ListEmployees
AS
SELECT *
      FROM Employees
RETURN @@ERROR
```

The @@ERROR symbol is a special variable in SQL Server which always contains the status code returned by the most recent statement. It is useful to return the value of @@ERROR to the calling program in the event that an error occurred.

Executing Stored Procedures

To execute this stored procedure in Query Analyzer or in a CFQUERY tag, you simply use

```
EXEC ListEmployees
```

The result of executing ListEmployees in CFQUERY would be a regular ColdFusion query variable containing the query results. A stored procedure returns a result set for every SELECT statement encountered. If you want to be able to use each of the result sets in a ColdFusion query variable, you must call the stored procedure with CFSTOREDPROC.

Executing Stored Procedures with CFSTOREDPROC

CFSTOREDPROC is the most flexible way to call a stored procedure. This methods lets you

- Pass explicitly typed parameters
- Read multiple return parameters
- Assign each result set to a ColdFusion query variable
- Read the return code

```
<CFSTOREDPROC datasource="Northwind"
      procedure="SalesByCategory"
      returncode="Yes">
      <CFPROCPARAM type="in"
            cfsqltype="cf_sql_varchar"
            maxlength="15"
            dbvarname="CategoryName"
            value="Beverages">
      <CFPROCPARAM type="in"
            cfsqltype="cf_sql_varchar"
            maxlength="4"
            dbvarname="OrdYear"
            value="1998">
      <CFPROCRESULT resultset="1"
            name="BeverageSales">
</CFSTOREDPROC>
```

This example passes two parameters to the stored procedure which returns a ColdFusion query variable named BeverageSales.

Getting the Return Value

If you called ListEmployees inside CFQUERY and @@ERROR was non-zero, ColdFusion would generate a database exception. In this case, your exception handler could find the value of the return code in CFCATCH.SQLSTATE.

Alternatively, if you use CFSTOREDPROC, you can find the value of the return code in CFSTOREDPROC.STATUSCODE.

Note

In order to access the return code, you must set RETURNCODE="Yes" in the CFSTOREDPROC tag. By default, ColdFusion will not make available the exit status.

Passing Parameters to Stored Procedures

Stored procedures can accept input parameters. You declare these parameters right after the CREATE PROCEDURE statement. The example illustrates a stored procedure to return authors having a given first name and last name.

```
CREATE PROCEDURE ListAuthorsByName
@LastName varchar(20),
@FirstName varchar(20)
AS
SELECT *
      FROM Authors
      WHERE LastName = @LastName
            AND FirstName = @FirstName
```

Note that you do not need quotes around a string parameter in a stored procedure.

Executing Stored Procedures with Parameters

To execute this procedure in Query Analyzer or in a CFQUERY tag, you simply use

```
EXEC ListAuthorsByName
@LastName='Chandler',
@FirstName='David'
```

The parameter names are optional. If you do not specify them, SQL Server will use them in the order they are declared in the stored procedure. Alternatively, you can call the procedure with CFSTOREDPROC and use CFPROCPARAM to specify each parameter individually.

Tip

It's a good idea to keep CFPROCPARAM tags in the same order as the parameters are declared in the stored procedure even though you're using named parameters.

To call the ListAuthorsByName procedure with CFSTOREDPROC, you would write

```
<CFSTOREDPROC datasource="Northwind"
      procedure="ListAuthorsByName"
      returncode="Yes">
      <CFPROCPARAM type="In"
            cfsqltype="CF_SQL_VARCHAR"
            dbvarname="LastName"
            value="Chandler">
      <CFPROCPARAM type="In"
            cfsqltype="CF_SQL_VARCHAR"
            dbvarname="FirstName"
            value="David">
</CFSTOREDPROC>
```

Returning Parameters

Stored procedures can also return values to the calling program. These are called output parameters. This example inserts a new record into the Authors table and returns the new AuthorID.

```
CREATE PROCEDURE NewAuthor
@FirstName  varchar(20),
@LastName varchar(20),
@NewID int = 0 OUTPUT
AS
-- Declare a local variable to save the error
code
DECLARE @ErrorSave int
INSERT INTO Authors (FirstName, LastName)
VALUES (@FirstName, @LastName)
-- Save the error code now or the next
statement
-- will replace it
SET @ErrorSave = @@ERROR
SET @NewID = @@IDENTITY
RETURN @ErrorSave
```

This example assumes that the Authors table has an identity (auto-increment) column. Whenever a procedure inserts a record into a table with an identity column, SQL Server sets @@IDENTITY equal to the new value in the identity column. This turns out to be quite useful. Without a stored procedure, you have to SELECT the newly-inserted value after doing the insert and hope that no one else inserted a record in the mean time (or use a transaction around the INSERT and SELECT).

Executing Stored Procedures Which Return Parameters

In order to get the values of return parameters, you must execute the stored procedure with CFSTOREDPROC. Declare the parameter using CFPROCPARAM with TYPE=”OUT” or “INOUT.” As far as SQL Server is concerned, all parameters are input parameters and can also be output parameters if so declared. If you do not want an output parameter to be required as an input parameter also, set a default value in the declaration.

Handling Errors in Stored Procedures

One of the advantages of using stored procedures is that you have more control over error handling than with statements in a CFQUERY. Your stored procedure can return the value of @@ERROR from the most recent statement or set a custom return code. Since standard SQL error codes start at 0 and go up, you should return custom error codes which are negative so that there is no confusion. Alternatively, you can store a custom status code in a regular output parameter so as to avoid confusion with SQL Server status codes.

This example rewrites the previous example in a way that leaves room for more processing after the INSERT. In addition, the procedure returns an error code of –1 if the NewID parameter is non-zero.

```
CREATE PROCEDURE NewAuthor
@FirstName  varchar(20), @LastName varchar(20),
@NewID int = 0 OUTPUT
AS
DECLARE @ErrorSave int
IF (@NewID <> 0)
      RETURN (-1)
INSERT INTO Authors (FirstName, LastName)
VALUES (@FirstName, @LastName)
SET @ErrorSave = @@ERROR
IF (@ErrorSave <> 0)
      GOTO on_error
SET @NewID = @@IDENTITY
-- Continue processing
-- Return success
RETURN (0)

on_error:
RETURN @ErrorSave
```

Note

Microsoft SQL Server stored procedures can also return error codes and/or messages with the RAISERROR statement; however, this does not appear to provide any useful information back to ColdFusion, at least not with SQL Server 7 and an OLE DB data source.

Handling Stored Procedure Errors in ColdFusion

The beauty of the error-handling approach in the previous example is that your ColdFusion program can use a single CFSWITCH / CFCASE statement to handle SQL errors as well as custom return codes.

```
<CFSET thisError = CFSTOREDPROC.STATUSCODE>
<CFSWITCH expression="#thisError#">
<CFCASE value="0">
      <!--- Success. Do nothing --->
</CFCASE>
<CFCASE value="-1">
      <!--- Oops. We passed in a bogus
parameter --->
      <CFTHROW MESSAGE="Sorry, I goofed.">
</CFCASE>
<CFDEFAULTCASE>
      <!--- The error is an SQL error or we
forgot one --->
      <CFTHROW MESSAGE="Error #thisError#">
</CFDEFAULTCASE>
</CFSWITCH>
```

Securing Stored Procedures

You control access rights to stored procedures like any other database object. Before anyone can execute the stored procedure, you must grant execute permission. To do this in SQL Server Enterprise Manager, right-click on a stored procedure, select Properties, and click Permissions. It is also easy to append the following lines to your SQL when you create the stored procedure (the GO keyword is the Transact-SQL batch separator):

```
GO
GRANT EXECUTE ON procedure_name TO username
```

Encrypting Stored Procedures

Normally, stored procedures are stored as plain text on the server and anyone can see them. Anyone with permission to connect to the database can view the stored procedure. If you are shipping commercial software and wish to keep prying eyes away, you can specify WITH ENCRYPTION in the CREATE PROCEDURE statement. Before you encrypt a stored procedure, however, make sure you have a copy of the unencrypted text somewhere!

Limitations of Stored Procedures vs. CFQUERY

Unfortunately, the pre-parsing which makes stored procedures faster and more secure than CFQUERY also makes stored procedures slightly less flexible in two primary ways. However, there are relatively simple workarounds.

Stored Procedures Cannot Have Dynamic Structure

One of the neatest things about CFQUERY is the ability to use conditional statements (CFIF) to build the SQL statement at run time. This is typically used when building a query for a search form where you do not know in advance which search parameters the user will choose. In this type of application, you use CFIF inside CFQUERY to include in the WHERE clause only those parameters which the user has specified. Stored procedures cannot use dynamic structure in the same way because they are parsed when you create them.

The workaround is to use the CASE function (or the IsNull shortcut function) in the WHERE clause to ignore parameters which the user has not entered. The following example demonstrates this technique. This procedure looks for a wildcard match of both the City parameter and the CompanyName parameter. The LIKE operator and the PatIndex function as shown here both have the same result.

```
CREATE PROCEDURE ListCustomers

@CompanyName varchar(20) = “”,
@City varchar(20) = “”

AS

SELECT      *
FROM        dbo.Customers
WHERE City LIKE
        CASE
                WHEN @City = "" THEN City
                ELSE @City + “%”
        END
  AND 0 <
        CASE
                WHEN @CompanyName = "" THEN 1
                ELSE PatIndex (@CompanyName + "%", CompanyName)
        END
```

Stored Procedures Cannot Aggregate Parameters

With CFQUERY, you can include in a WHERE clause the SQL IN operator with a single ColdFusion variable containing a comma-separated list, as in the following statement:

```
<CFQUERY ...>

SELECT *

FROM Employees

WHERE State IN
(#ListQualify(Form.SelectedState,"'",",")#)

</CFQUERY>
```

Because ColdFusion is doing simple text substitution, you can actually pass multiple SQL parameters using a single ColdFusion variable. However, if you pass in Form.SelectedState as a parameter to a stored procedure, the database will treat it as a single parameter, not as a comma-separated list of parameters.

There are two potential workarounds for this. The simplest way is to pass in one parameter for each possible choice. Your ColdFusion program must loop over the list variable and create a stored procedure parameter for each selection. This technique only works, however, when you know in the maximum number of parameters which the user can select.

Alternatively, if the selections correspond to rows in a database table, you may be able to use bit masking to create a single parameter which is the sum of all the IDs for all selections. This technique works with IDs in the range 1 to 32 (the number of bits in a SQL Server integer). If your data has IDs outside this range, you can insert the potential selections (up to 32) into a temporary table with an IDENTITY field starting at 1. This ColdFusion code snippet creates a bit-masked parameter from the numeric IDs of all selected product categories.

```
<cfset CatMask = 0>
<CFLOOP index="cat" list="#Form.Categories#">
      <cfset CatMask = CatMask + 2^(cat – 1)>
</cfloop>
```

The corresponding stored procedure uses the POWER function and the logical AND operator (&) to retrieve products in the selected categories.

```
CREATE PROCEDURE ListProducts
@ProductMask int = 0
AS
SELECT ProductName, CategoryName
FROM Products JOIN Categories
        ON (Products.CategoryID = Categories.CategoryID)
WHERE @ProductMask & POWER (2, Categories.CategoryID - 1) <> 0
```

Using Triggers

A trigger is a special type of stored procedure which is executed on an event-driven basis rather than by a direct call. You might use a trigger to

- Maintain data integrity rules beyond simple referential integrity
- Keep running totals updated
- Implement a referential action such as a cascading delete
- Maintain an audit record of changes by updating a shadow table
- Invoke an external action using xp_cmdshell, xp_sendmail, or another extended stored procedure

A trigger can be set up to fire when data has been modified via an INSERT, UPDATE, or DELETE statement. You can define multiple triggers on a table for each event; however, you cannot control the order of firing. Triggers have the following properties:

- A trigger is executed *once* for each UPDATE, DELETE, or INSERT, regardless of how many rows were affected.
- A trigger fires *after* the data modification statement has performed its work but *before* that work is committed to the database. This allows the trigger to roll back the data modifications if necessary.
- A trigger has access to the before image and after image of the data via the pseudotables *inserted* and *deleted*, which have the same set of columns as the data being changed.
- Triggers can be recursive, either directly or indirectly. Nested triggers can be turned on or off in the database.

You create a trigger just like a stored procedure, but you also specify the event for which the trigger will fire. This trigger returns an EmployeeID when a new employee is inserted into the Employees table.

```
CREATE TRIGGER GetEmployeeID

ON Employees

FOR INSERT

AS

SELECT EmployeeID AS NewID FROM inserted
```

Upon inserting a new record using CFQUERY, your ColdFusion program could refer to *queryname*.NewID to get the value of the new ID. This example demonstrates how a trigger can communicate back with ColdFusion. Way cool!

Another great use for triggers is to prevent mass updates or deletes to a table. If more than row is affected, your trigger can call RAISERROR (*message*, 16, 1), which will result in a ColdFusion exception.

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 6: Updating Your Data

For help creating stored procedures in Microsoft SQL Server, consult SQL Server Books Online. This is an excellent reference, complete with many examples.

Books

Inside SQL Server 7.0 (Microsoft Press) contains more than you will ever want to know about transactions, stored procedures, and triggers. This is the definitive reference for Microsoft SQL Server.

Other

The following video on Allaire Alive (alive.allaire.com) may be of interest.

Relational Database Design

Lesson 20
Optimizing Performance

Introduction

In this lesson, you'll learn a variety of techniques for increasing the performance of your ColdFusion database application, from database design to special ColdFusion features.

Objectives

By the end of this lesson, you should be able to

- Identify database design techniques which may help boost performance
- Understand how indexes, stored procedures, and triggers can boost performance
- Use CFQUERY BLOCKFACTOR to increase performance
- Use ColdFusion query, page, and template caching

Performance by Design

You should begin taking database performance into account right from the start with good database design. There are a number of design techniques which you can use to make frequently-used data more accessible, and thereby increase performance.

Introduce Controlled Redundancy

Although design rules would normally prohibit them, it is sometimes useful to create table columns which contain information stored in another table.

For example, a customer may have many different addresses, including mailing, shipping, and billing addresses for many locations. In order to allow the greatest flexibility, you would probably use a Customers and an Addresses table with a one-to-many relationship between them. This way, you can store an unlimited number of customer addresses.

However, if you frequently run queries which access a customer's main address, you may also wish to store the primary address in the Customers table. This increases performance, but reduces consistency because your application must now update the main address in both places (this would be a good application for a trigger).

Store Derived Values

In general, it is not a good idea to store values which are calculated from other columns. However, there is one particular application where this saves a lot of time, and that is keeping track of a running total.

For example, suppose you want to compute the account balance associated with each entry in a check register. If you don't store the balance with each entry, you must compute it for each entry by summing all previous entries in the register. As your register grows, this becomes impractical. It is much faster to keep a running total by storing the balance after each entry. Ideally, you should update the running total using a trigger to ensure consistency.

Use Surrogate Keys

Whenever the logical primary key is a character string or a composite key, you can boost performance by using a surrogate numeric key instead. You can still enforce the uniqueness requirements of the logical key using constraints.

Use Indexes

An index provides logical pointers into physical locations. Rows in your table can only be stored in one order; however, you can use indexes to allow the table to be accessed quickly in any order.

Indexes, like stored procedures, are something you must think about because the database creates them using default rules unless you specify otherwise. Not to decide *is* to decide.

Indexes can be based on a single column or on a combination of columns (a *composite* index).

An index can allow duplicate rows or it can be *unique*.

Finally, an index can be *clustered* or *unclustered*. A clustered index orders the physical table on disk in the order of the index. You can only have one clustered index per table. An unclustered index does not affect the order of the physical table.

When to Index

Unless specified otherwise, Microsoft SQL Server automatically creates a unique, clustered index on the primary key for every table. In addition, SQL Server creates a unique, unclustered index for every UNIQUE constraint.

However, you may wish to create additional indexes in order to speed up commonly-used queries. Good column index candidate are:

- Columns that are frequently used in joins, such as foreign keys
- Columns that are often used in an ORDER BY clause
- Columns that are frequently used as search criteria

You should use clustered indexes on all tables, except where the primary key is an identity column.

Indexing Drawbacks

Indexes speed up data retrieval but slow down data modifications because the database server must update the index every time data is modified. It is generally not beneficial to index

- Columns that are rarely referenced in queries
- Columns that can have only a few possible values
- Tables with only a few rows

Database Server Considerations

One simple way to boost database performance is to put the database on its own server. Otherwise, the database is competing with ColdFusion and the Web server for CPU cycles. By moving the database to a separate server, you can get two CPUs to work on a problem in parallel.

Network Considerations

Ideally, you want to keep the network connection between your database and your ColdFusion server free of other traffic. Avoid placing the servers on a heavily loaded subnet or on two different subnets which communicate through a slow router. Here are some optimal configurations.

1) Create a private connection between the servers by installing a second network card in each machine.
2) Use Ethernet switches instead of hubs. In switched Ethernet, every network card sees only its own traffic.

Using either a private connection or switched technology (which is a virtual private connection), you can take advantage of full-duplex Ethernet cards to double data rates. In addition, many servers now come with dual- or quad-port Ethernet cards.

Of course, network considerations are not terribly important if your Web server is only feeding a T1 to the Internet, since even 10 Mb Ethernet is nominally eight times faster than a maxed-out T1.

Hardware Considerations

For high-performance database applications, you should put some thought into your server.

- SQL Server can store database files and transaction log files separately.
- A single database can span multiple disks.
- You can get parallel I/O processing by spanning a database across multiple disks, and even across different SCSI controllers.
- I/O-intensive databases should use the faster 10,000 RPM SCSI disks because of the faster seek times.
- Most major databases, including SQL Server, can take advantage of multiple processors.

For redundancy, database servers often use disk mirroring (RAID 0) rather than the more space-efficient RAID 5 because mirroring is slightly faster.

Programming for Performance

A few simple programming techniques can go a long way toward creating high-performance applications.

Use Stored Procedures

Stored procedures offer two main performance benefits.

1) When you dynamically execute a query using CFQUERY, the database must first parse the SQL statements. Often, parsing time is as long as execution time. Stored procedures are parsed when they are created, which saves time during execution.
2) Stored procedures typically group related statements together. This is faster than sending small groups of statements using CFQUERY because of the overhead associated with CFQUERY.

Use Triggers

Triggers offer the same benefits as stored procedures. A trigger has much lower overhead than sending an SQL statement via CFQUERY.

Minimize Trips to the Database

In general, you should write a database application to retrieve groups of rows rather than one row at a time. For the most part, you don't need to worry about this since CFQUERY returns rows all at once. However, you should avoid queries inside loops. There is almost always a better way, whether it be a revised SQL statement or a stored procedure.

The next several topics focus on ColdFusion caching capabilities which help minimize trips to the database.

Maintain Database Connections

One of the reasons for using ColdFusion Application Server in the first place is that you can maintain open database connections. You configure the number of open connections and the timeout for each data source in the ColdFusion Administrator.

Note

You must purchase a concurrent database license for each open connection. For most applications, five is plenty.

Keep in mind that when you pass in a database login and password in CFQUERY, ColdFusion must establish a new connection if a connection for that user does not exist already. This makes it more desirable to use a single application login to the database rather than individual user logins; however, user logins are more secure.

Use Block Factor

CFQUERY and CFSTOREDPROC allow you to specify a *block factor* for the query. The block factor is the number of rows which will be retrieved at a time. The default is one (!), which means that by default, ColdFusion has to request each row in a query individually. For queries returning more than a handful of rows, you should set the block factor to something like 100. When using block factor, keep in mind that

- ColdFusion automatically downgrades the block factor as needed
- Errors are not returned if block factoring is not supported
- Setting too high a block factor forces the database driver to allocate more memory for buffering, which can reduce overall system performance

To specify a block factor in a CFQUERY or CFSTOREDPROC tag, use the BLOCKFACTOR attribute.

```
<CFQUERY NAME=”ListEmployees” ...
      BLOCKFACTOR=”100”>
```

Cache Queries in Persistent Variables

One way to minimize trips to the database is to cache frequently-used queries. You can easily force a query result set to persist in memory by assigning it to a ColdFusion server, session, or application variable.

```
<CFQUERY NAME=”Session.Prefs” ...>
```

You can also assign a query variable to a server, application, or session variable as follows:

```
<CFQUERY NAME=”PrefsQuery” ...>
...
</CFQUERY>
<CFSET Session.Prefs = PrefsQuery>
```

This technique is ideal for relatively static data which must be referred to frequently. For example, you might cache

- Application parameters in an Application variable
- A list of scheduled outages in a Server variable
- User preferences in a Session variable

Whenever the data in the query changes, you simply re-run the query to refresh the persistent variable.

Variable-Based Caching and Memory Consumption

Keep in mind that every cached query is using up RAM on ColdFusion server. For queries cached in Application or Server variables, this is probably not much of an issue. If you’re caching queries in Session variables, however, you must take into consideration the number of sessions which might be active simultaneously. Remember that a session does not time out when the browser closes, but rather after a period of inactivity defined in the ColdFusion Administrator and/or your CFAPPLICATION tag.

Variable-Based Caching and Load Balancing

One drawback to variable-based caching is that application pages on other ColdFusion servers in a cluster cannot access the cached queries. For Application and Server variables, it may be practical to create a couple of initialization and update routines which trigger updates on the other servers through CFHTTP whenever the underlying data changes. It is less practical to duplicate Session variables this way. Variable-based query caching is so attractive that it presents a strong argument for using the “single affinity” capability of load balancing devices.

Use Query-Based Caching

Query-based caching can be useful when you want to cache a query for a specific period of time. You can turn on query-based caching for each ColdFusion query by specifying the CACHEDWITHIN or CACHEDAFTER attributes in CFQUERY.

```
<CFQUERY NAME="GetStates"
      DATASOURCE="Northwind"
      CACHEDWITHIN="#CreateTimeSpan(1,0,0,0)#">
```

This query will use the same data for a day before it is refreshed.

A query can be retrieved from cache only when it uses all the same query information, including

- All tag attributes (Name, Datasource, Username, etc.)
- The same SQL statement (including the values of dynamic parameters)

The main drawback to query-based caching is that you cannot manually refresh the query like you can with variable-based caching. Query-based caching requires less programming effort than variable-based caching; however, it is only appropriate when it really doesn't matter if you might miss a modification to the underlying data made within the cache interval.

Query-Based Caching and Memory Consumption

ColdFusion can cache up to 100 queries. You may adjust this downward in ColdFusion Administrator to conserve memory if you anticipate caching large queries.

Query-Based Caching and Load Balancing

Query-based caching is somewhat easier to work with than variable-based caching in a load-balanced environment. With variable-based caching, you must either write agents to update the cached queries on each server, or write your application to hit the database if it can't find the cache variable. Query-based caching does the latter automatically. As users hit all the servers in a cluster, each server will build up its own (possibly different) group of cached queries. As with variable-based caching, the whole problem goes away if you use "single affinity" load balancing, but that decreases availability.

Page-Based Caching

Page-based caching is a very powerful ColdFusion feature which lets you bypass ColdFusion processing altogether for frequently-requested pages.

How it Works

You turn on page-based caching using the CFCACHE tag in a page. The first time ColdFusion processes the page, it caches the resulting HTML. Thereafter, whenever ColdFusion receives the same page request, it will retrieve the page from cache.

When using CFCACHE, ColdFusion saves the HTML to disk with a .tmp extension in the same directory as the ColdFusion page being cached (or the directory specified in the optional CACHEDIRECTORY attribute) . Also in the directory, ColdFusion creates a cache map named cfcache.map. This file is similar to a Windows INI file and stores

- The path to each .tmp file in the directory
- The time each .tmp file was written
- The URL for each cached page, including URL parameters

The real power of CFCACHE is that it caches a separate file for each unique URL request. The following URLs would result in two different cached files:

http://localhost/products/product_info.cfm?id=5
http://localhost/products/product_info.cfm?id=21

Note

All URL parameters must be in the same order for subsequent requests to find the cached page.

Everything above the CFCACHE tag is executed for each request, but everything below is cached. Only CFML tags are allowed above the CFCACHE tag.

Don’t Forget to Flush

After you make changes to cached pages, you need to flush the cache. There are four ways to do this. First, you can manually or programmatically delete the .tmp file and the entry in the .map file for each cached page.

Flush the Cache with ACTION="Flush"

Use ACTION="Flush" to force the flushing of cached pages. When you flush, you can use the EXPIREURL parameter to selectively flush pages created with specific URL parameters. You can use a wildcard in place of the URL parameter to flush a page with any parameters.

To flush all pages in the current directory, use

```
<CFCACHE ACTION="Flush">
```

To flush all product.cfm pages in the current directory with any URL parameters, use

```
<CFCACHE ACTION="Flush"
      EXPIREURL="product.cfm?*">
```

Use the TIMEOUT Attribute to Auto-Flush

To set cached pages to timeout in four hours, use

```
<CFCACHE TIMEOUT="#DateAdd ("h", "4", Now())#">
```

Use Cache URL Parameters to Flush Without Coding

You can use two special URL parameters to control caching behavior.

- "CFNoCache=Yes" forces ColdFusion to process the page, even if a cached page exists.
- "CFReCache=Yes" forces ColdFusion to recompile and cache the page.

When Not to Use Page Caching

Page caching is not a good fit for pages which

- Include user-specific information or preferences (unless you pass a user ID in the URL, and then you could generate a lot of cached pages!)
- Are frequently updated
- Are based on data from forms using the POST method
- May produce errors, as the error pages will be cached

Notes on Using CFCACHE

- CFCACHE requires that "simultaneous requests" be > 1
- CFCACHE ignores debug settings unless the page turns debugging on. CFCACHE uses <CFSETTING ShowDebugOutput="No">

Template Caching

ColdFusion Server automatically caches the P-code for every template (.cfm file) when it is first processed. Subsequent requests check to see if the file has been modified on disk. If the file has not been modified, ColdFusion retrieves it from memory.

You can control template caching to a limited extent in ColdFusion Administrator.

Template Cache Size

In ColdFusion Administrator under Server | Settings, you can specify how much memory to reserve for template caching. Allaire recommends that you set this equal to five times the size of all the ColdFusion pages on the server because the generated P-code is larger than the underlying .cfm files.

- If you set the size too small, you will end up reprocessing pages a lot.
- If you set it too big, you will waste memory on the server.

Trusted Cache

Check this box in ColdFusion Administrator to tell ColdFusion to skip the step of checking the timestamp on disk. This further enhances performance, but it is only appropriate for production environments where the pages don't change frequently.

To flush the trusted cache, you must

1) Disable trusted cache
2) Access the modified page(s)
3) Re-enable trusted cache

You do not need to restart the ColdFusion Application Server service to clear the cache, but this is the most practical method if you have cached a large number of pages. Unfortunately, this means the server has to rebuild all other types of cache from scratch.

Where to Go from Here

Application Help

Other than the caching-related tags in the *CFML Language Reference*, ColdFusion does not come with a lot of documentation regarding performance techniques.

Books

Inside SQL Server 7.0 (Microsoft Press) covers the nitty-gritty details of database performance if you really want to know!

Other

The following videos on Allaire Alive (alive.allaire.com) may be of interest.

Load Balancing with ColdFusion
Maximizing Performance Through Administration
Relational Database Design

Unit 5
Constructing Robust Applications

Unit Overview

This unit will explore some of the commonly encountered problems in Web development and their solutions, including a comprehensive review of security practices. The unit concludes with an overview of ColdFusion capabilities which haven't been covered earlier.

Goals

By the completion of this unit, you should be able to

- Take into account scalability and fault tolerance when designing your application
- Choose the authentication method most appropriate for your application
- Avoid common application and database security pitfalls
- Secure your Web, ColdFusion, and database server
- Understand additional ColdFusion capabilities

Lesson 21
Web Applications Challenges

Introduction

Web applications pose unique opportunities and unique challenges. The same things that make the Web so flexible and scalable also make it harder to write application. In this lesson, you'll explore some of the problems associated with large database queries, multi-user updates, browser behavior, and load balancing.

Objectives

By the end of this lesson, you should be able to

- Identify which type of load balancing solution is best-suited to your application
- Understand several techniques for dealing with large queries
- Take into consideration browser bookmarks and reloads when designing your application
- Design an application to correctly handle database updates by multiple users

Load Balancing

Because the Web is stateless, it is possible to build “server farms” where Web content is duplicated on each server. There are various ways to distribute requests amoung servers.

Separate Static from Dynamic Content

One of the simplest load balancing techniques is to keep your static HTML and images on a different server than your dynamic content. This way, each server can be optimized for its task and you don’t bog down your ColdFusion server with a bunch of image requests.

Note

With ColdFusion Enterprise Server, you can put your Web server and ColdFusion Application Server on different machines. This automatically separates static from dynamic content. In addition, it can help secure the ColdFusion Server.

Round Robin DNS

DNS servers are normally configured to return one IP address for each name requested. Round robin DNS will cycle through a list of IP addresses so that incoming requests are distributed among Web servers. The primary drawback to round robin DNS is that you have to modify DNS if you want to take a server off-line for maintenance. Even then, browsers and proxy servers may have cached an old address.

Hardware Solutions

A number of vendors have come up with hardware solutions for intelligently directing traffic destined for a single IP address to any number of servers. Typical hardware solutions can be used with almost any IP-based server (not just HTTP) and can also do some security things like

- Network address translation
- Port mapping
- Packet filtering
- Resisting various denial-of-service attacks

Major load balancing hardware vendors include

Cisco Systems (www.cisco.com)
F5 Networks (www.bigip.com)

Software Solutions

The most notable load balancing software solution is Microsoft's Windows Load Balancing Services, which is freely available for Windows NT and ships as part of Windows 2000 Advanced Server. With WLBS, all servers in the farm share load information with each other and jointly determine which one will handle the request. Unlike round robin DNS, WLBS makes it very easy to take a server off-line. You just unplug it from the network and the rest of the servers absorb the load.

Tip

For load balancing purposes, it's a good idea to have at least one more server than you really need so you can take one down for maintenance now and then.

Designing Applications for Load Balancing

Designing applications for load balancing is a tradeoff between

1. The fault-tolerance offered by a truly stateless application, and
2. The performance advantages offered by variable, query, and dynamic page caching.

For purposes of fault tolerance, you want the browser to be able to come back to any server, but for purposes of performance, you want the browser to come back to the one that served it.

Fortunately, with a little intelligence in your application, you can have the best of both worlds.

Use Single Affinity

Hardware and software load balancing solutions offer the capability to always direct traffic from a particular browser to the same server. This way, you get the benefits of caching. WLBS calls this the “single affinity” setting. The BIG/ip device from F5 networks can use a number of things to identify a single browser, including IP address and cookies. The latter is preferable; otherwise, all sessions coming from behind a proxy server (like AOL!) will be directed to a single server, thus defeating the point of load balancing.

… But Make Your Application Smart Enough to Do Without It

When a server fails or you have to pull it out for maintenance, you don’t want your application to fail. This requires you to build a little intelligence into your application. Here are a few simple rules.

1) Server and Application variables should derive their values from a central database so that they can easily by synchronized across multiple servers. When you update the underlying database, use CFHTTP and/or WDDX to sync up all servers.
2) If your application uses query-based caching, sit back and relax. ColdFusion automatically hits the database when there is no cache.
3) Use the advanced session data management technique presented in Lesson 11 to store session preferences, state info, and data. To tolerate a server outage, your application must mirror session data in the database (Client variables), cookies, JavaScript, hidden form variables, or URL parameters.
4) Do use Application, Server, or Session variables to cache queries. But also store the query parameters in some type of client variable so you can rebuild the query if you lose the persistent variable. There is no point in mirroring an entire query to the Client scope when you can easily run it again, and it will help you avoid exceeding the 65 kilobyte limit for Client variables in CF5.

Dealing with Large Queries

What do you do when your query returns 10,000 rows and you want users to be able to page through the results twenty at a time?

The basic idea is simple: you keep track of what row you're on and embed it in some type of client variable. Hidden form variables and URL parameters are easy to work with. When the user clicks on the "Next" button or hyperlink, the browser will transmit the row number and you can continue from there.

Which Type of Cache?

How you cache the query depends on a number of factors.

First, decide how many rows users are really willing to page through before they narrow their search. There is no need to cache any more than this number. A hundred is plenty for most people. We'll call this number QueryMax. The number of rows on a page we'll call PageMax.

Second, ask yourself whether the data has to be real-time. If users can live with data a couple hours old, you can use query-based caching or page caching to great effect. Otherwise, you'll have to use variable-based caching or no caching at all.

Third, decide whether the query is particular to a given session (user) or common to all users. If it is user-specific and you have a large number of simultaneous users, you probably don't want your directories filling up with page cache files and ColdFusion can only cache up to 100 queries using query-based caching, so you'll have to use variable-based caching or no caching instead.

The following matrix identifies the appropriate solution based on these parameters.

	Must be real-time	Can be somewhat stale
Not user-specific	Application variable or no cache	Query-based caching or page caching
User-specific with few users	Session variable or no cache	Query-based caching
User-specific with many users	Session variable or no cache	Session variable or no cache

Page caching only applies to queries which are not user-specific. Otherwise, each page of results is typically going to be viewed only once so there's no value in page caching.

The Variable-Based Caching Approach

1. When you first run the query, use CFQUERY with MAXROWS=QueryMax and assign it to a persistent variable.
2. Use CFOUTPUT STARTROW=1 MAXROWS=PageMax to display the first page of results.
3. After the CFOUTPUT, create a hidden form variable StartRow = *QueryName*.CurrentRow.

On successive requests to the page,

1. Use CFOUTPUT STARTROW=Form.StartRow MAXROWS=PageMax.
2. After the CFOUTPUT, create a hidden form variable StartRow = *QueryName*.CurrentRow.

You can test for the presence of the Form.CurrentRow to determine whether this is the first time through the page or not.

The Query-Based Caching Approach

1. The first time through, use CFQUERY MAXROWS=QueryMax CACHEDWITHIN=*some_time*.
2. Use CFOUTPUT STARTROW=1 MAXROWS=PageMax to display the first page of results.
3. After the CFOUTPUT, create a hidden form variable StartRow = *QueryName.*CurrentRow.

On successive requests to the page,

1. Use the same CFQUERY. ColdFusion will return it from cache.
2. Use CFOUTPUT STARTROW=Form.StartRow MAXROWS=PageMax.
3. After the CFOUTPUT, create a hidden form variable StartRow = *QueryName.*CurrentRow.

The Page-Based Caching Approach

1. Run CFQUERY below CFCACHE.
2. If URL.StartRow is not defined, set it to 1.
3. Use CFOUTPUT STARTROW=URL.StartRow MAXROWS=PageMax.
4. After the CFOUPUT, create a hidden form variable (GET method only) or hyperlink with StartRow = *QueryName*.CurrentRow.

Paging Through Results Without Caching

If you don't have the luxury of caching any part of the query, you'll have to run it each time. However, instead of using a row number to remember where you were in the query, you can store the value of one or more columns from the last row displayed. In order for this to work,

- The column(s) must be unique for each record, and
- The results must be sorted by the column(s)

When you run the query on subsequent pages, you add a WHERE clause containing the value from the last row on the previous page. Use CFQUERY MAXROWS=PageMax to limit the number of rows returned.

Hint

Better yet, use CFQUERY MAXROWS=PageMax + 1. Then you can use Query.RecordCount to tell you whether you've reached the last row in the query and thus, whether you need a Next button.

To implement a Previous button, you simply use the first row on the current page and reverse the sense of the comparisons and the ORDER BY clause. Then you can use query array notation to loop through the results backwards so they'll be sorted correctly on the screen.

Here's how you would get the next *n* records from the last row on the current page:

```
SELECT *
FROM Employees
WHERE search_conditions
AND (LastName > #Form.PreviousLastName#)
OR
(
      LastName = #Form.PreviousLastName#
      AND FirstName > #Form.PreviousFirstName#
)
ORDER BY LastName, FirstName
```

In order for this technique to work properly with multiple columns, the ORDER BY clause must contain enough columns to resolve any ambiguities in sort order. A good way to ensure this is to create a UNIQUE constraint on the sort columns. Since all UNIQUE constraints use an index, this will have the happy side effect of making queries faster.

Other Database Caching Options

Roll-Your-Own Caching

You can always create your own caching system by writing the initial query out to a file or to a global temporary table in SQL Server.

DBA Nightmare Caching

One other “caching” option is worth mentioning. You can simply re-run the entire query on every page, but only display the rows for the current results page using CFOUTPUT STARTROW and MAXROWS. Since the database has its own cache, this may actually be a practical way to go.

Bookmarks and Reload

One particularly challenging problem for stateful Web applications is that you cannot force users to visit pages in sequence. They can always go back to a previous page using a bookmark or the back button.

Bookmarks

Bookmarks are generally very useful. If there is no reason a user shouldn’t be able to go right to their favorite project page, for example, then the project page should be designed to read the project ID from the URL. This means that any forms used to get to the project page initially should use the GET method so the data will be stored in the bookmark. A clever design could even direct the user visiting the bookmark to the login page first, then to the requested page.

In some cases, however, bookmarks are not desirable. You probably don’t want users to be able to bookmark a page in the checkout sequence at your online store, for example. You can’t prevent users from setting bookmarks, but you can prevent them from visiting them successfully. The simplest way to do this is to require some variable which cannot be stored in a bookmark, such as a hidden variable in a form or a cookie which is only active for one browser session. If your page doesn’t find the variable, it probably means the user bookmarked the page, and you can return an error message.

Using MVC architecture with an event-driven stateful controller, it is possible to handle bookmarks securely and robustly. For further discussion, see Lesson 25: Model View Controller.

Preventing Reloads

Reloads are particularly bothersome. Suppose you have clicked “Order This” for the final time to purchase a product and nothing comes back. You are impatient, so you hit the Reload button. What happens? Now you’ve sent a second request to the server. Depending on the sophistication of the particular application, you may actually order the product twice.

How can you prevent this in your application? High-end Business transaction servers like WebLogic have built-in support for preventing reloads. The approach used is to generate a random *token* for each form. Once the form has been processed successfully, the token is destroyed. If a user attempts a reload of a page already submitted, the token transmitted will be invalid. Note that this also prevents users from bookmarking the page. The event-driven stateful controller in Lesson 25 uses a random *page nonce* as a token for this purpose. If the same page nonce is sent twice, the controller is able to detect this.

Tokens can be stored either in session data or in the database. For maximum robustness, the token should be stored in the database and deleted as part of the same transaction that writes to the database. This way, if you click the submit button twice, the action will be done only once, but if it was unsuccessful the first time, you can click it again to resubmit the form. If you store a token in session data only, there is a small possibility that a token could remain in the database if a submit page is interrupted after the database write but before the token has been deleted.

Managing Business Transactions

The most sophisticated applications may actually require two types of tokens to prevent reloads and double submissions.

1) Form tokens help ensure that each form is submitted successfully and only once.
2) Business transaction tokens help ensure that an entire sequence of forms is submitted successfully. This is larger in scope than a database transaction, but similar in concept.

The easiest way to manage a business transaction spanning several Web requests (forms, etc.) is to store all form data in the session data space until reaching the final submit page for the business transaction. In this model, the final page in the business transaction is the only one that writes to the database, and thus, you can use a single form token as discussed in the preceding section to manage both the database and business transactions. The event-driven stateful controller discussed in Lesson 25 can be used this way. For user convenience, check to see that the database operation was successful before disabling future submits for that business transaction. For maximum robustness, store the business transaction token in the database so you can delete it as part of the same transaction in which the database write occurs.

If your application is such that you must make modifications to the database in the intermediate pages, you will have to build your own commit / rollback logic for the entire business transaction. One way to do this would be to store a token for the entire business transaction in the database and in the session data space. On the final successful submit, you delete the token from the database. If the final submit fails, you would roll back the changes you made in the intermediate screens.

Building your own token-based business transaction monitor is not as hard as it sounds at first. A good design is to use a central page through which all form submissions are directed. This is the application controller concept presented in Lesson 25: Model View Controller. The controller can check the form and/or business transaction tokens before processing each request.

Browser Challenges

Displaying Progress as the Page Loads

One disappointing limitation of the Web is that it is very hard to display page results progressively as they come in, which would be very useful for large queries or showing users the progress of their order. There are two facets to this limitation:

1) Due to the complexity of HTML pages, particularly those involving tables, the browser usually has to wait for the entire page before it has enough information to begin rendering the page.
2) Web servers normally buffer all page output before sending anything back to the browser. One reason for this is that the size of the page in bytes is part of the HTTP headers.

In ColdFusion 5 and later, you can manually flush the output buffer at any time using the <CFFLUSH> tag. This sends all HTML generated up to that point in the page to the browser immediately. This won't help you display progress for single operations like a monster query or loading a large file, but it's a good candidate anytime you're looping over long-running code. Using <CFFLUSH> and a little Javascript, you can easily create a graphical progress meter.

Note

If an exception occurs after a <CFFLUSH>, the user will see all the output prior to the <CFFLUSH>, even if you're using custom error pages. Once output has been sent, there's no way to take it back.

If you have a relatively simple page (no tables), there is a way to stream output to the browser without any buffering at all. You can write an *nph* script (no parsed headers) in any language that supports CGI (like perl or C). You write the program like any other CGI program, but prefix the name with "nph-". When you do this, you must output the standard HTTP headers in your script (see any CGI book for details). However, this technique does not work with ColdFusion because ColdFusion does not use CGI.

Renaming Downloaded Files

Many Web sites use an application page to authenticate users before allowing them to download files. Instead of linking directly to secret_document.doc, you might link to download.cfm, which returns the document only for authorized users. The down side to this approach is that the browser will think that the name of the retrieved file is the name of the download program (say, download.cfm), so the browser will prompt the user to save the file download.cfm.

You can overcome this problem using the Content-disposition header as follows:

```
<cfheader name="Content-disposition"
          value="attachment; filename=fname.ext">
```

A related problem is that, depending on the file type, the browser may display the file content in the browser window itself, or it may launch the "Save or Open" dialog. If you want to launch the dialog, set the Content-disposition to "attachment" as shown. If you don't want to launch it, set it to "inline," like this:

```
<cfheader name="Content-disposition"
          value="inline; filename=fname.ext">
```

An alternative way of renaming downloaded files is to trick the browser by including the real filename after the download script in the URL. For example, you could use a URL like this in the hyperlink (or FORM ACTION) which specifies the path to the download program:

http://mysite/download.cfm/mysoft_22.exe

When the download is complete, the user will be prompted to save the file as mysoft_22.exe. This is one of the many uses of the URL extra path information field, which can follow any script name.

Note

Unfortunately, some releases of IIS 4 and 5 do not correctly implement the W3C HTTP protocol specification with respect to extra path information in the URL. Instead, IIS will try to interpret the extra path information as a real path and will return error 404, File Not Found. This has been discussed with much angst on Usenet, but so far nobody has determined why some installations work and others don't. Use the Content-disposition header instead.

Avoiding the Repost Question

A most annoying feature of most browsers is the repost question. When you reload a page containing a form, the browser will ask you, “Do you want to repost the form?” Normally, this makes sense. Sometimes, however, you want to programmatically reload a page (typically in response to some action in another frame). In this case, you don’t want the user to see the repost question.

The only sure-fire way to get the browser to shut up is to send a slightly different URL so the browser will not think it has loaded the page before. You can do this easily by appending a timestamp or sequence number to the URL, as in

```
<CFSET ts=URLEncodedFormat(Now())>
<SCRIPT>
<CFOUTPUT>
location.href = ‘Query.cfm?ts=#ts#’
</CFOUTPUT>
</SCRIPT>
```

The bogus parameter will not bother your application page, but will trick the browser into thinking it’s a new page.

The event-driven stateful controller presented in Lesson 25 appends a page nonce to every URL, which has the effect that the browser thinks every page is new. If you want to force the repost question, you have to include the no-cache headers discussed in the next topic.

Browser Caching Revisited

The flip side of the reload coin is making sure that your dynamic pages are never cached in the browser. There are two ways to do this:

1) Use a random string in each URL as discussed in the previous topic. The event-driven stateful controller in Lesson 25 uses this approach.
2) Use HTTP headers to tell the Web browser not to cache any pages.

You can set HTTP headers in IIS or in your code. If you do it in IIS, you will probably want to set this for an entire directory containing your dynamic pages. Alternatively, you can set the HTTP Expires and/or Pragma headers yourself in ColdFusion pages using CFHEADER or the HTML META tag.

CFHEADER tells the Web server to send the specified HTTP header to the browser. Some combination of the following headers should work for all browsers and proxy servers. Note the use of GetHTTPTimeString to produce a correctly-formatted date / time for HTTP headers.

```
<cfheader name="Expires"
          value="#GetHTTPTimeString ("1/1/2000")#>

<cfheader name="Pragma" value="no-cache">

<cfheader name="cache-control"
          value="no-cache,no-store,must-revalidate">

<cfheader name="Last-Modified"
          value="#GetHTTPTimeString (Now())#">
```

The downside to using these headers is that when you use the back button, the browser will always warn you that the page has expired.

Note

For dynamic pages like form postings, it is a good idea to include a CFHEADER Expires statement in your Application.cfm. Some people configure their browsers to never visit cached pages, which means they could always be getting stale data, especially when you use CFLOCATION. Setting the Expires header will override this configuration setting, at least in Internet Explorer.

The HTML META HTTP-EQUIV tag is processed by the browser and simulates an HTTP header.

```
<META HTTP-EQUIV=”Expires”
      CONTENT=”#GetHTTPTimeString (Now ())#”>
```

Sharing Data in Multi-user Applications

One of the challenges of the Web's stateless model is the problem of two users modifying the same data at the same time. Suppose, for example, that John loads the record update form to change the street address for a record. Sheila loads the same form to correct a typo for the same record. At this point, both see the same data. John changes the address and saves the data. Because the Web is stateless, Sheila's screen cannot automatically reflect the changes just made by John. Therefore, when Sheila saves the data, the information in the address fields will undo the changes made by John because Sheila's form contained the old address.

This is called the "lost updates" problem and introduces the notion of a business transaction. The lost updates problem is similar to the concurrent update problem that is easily solved through database locks and transactions, but is complicated by the fact that the updates occur on separate requests to the database, so they cannot be part of the same database transaction.

In general, users should be able to edit only the data which applies to them. By doing updates on the smallest pieces possible, you minimize the possibility of users "stepping on" data which does not pertain to them. However, it is not always possible to achieve complete separation.

In order to solve this problem, we can learn from how databases implement transactions and implement some kind of locking. Depending on the need for concurrency vs. scalability, you can choose either optimistic or pessimistic offline locking (see [Fowler] for excellent discussions of these patterns).

The Read Uncommitted Analogy

What has happened in this example is analogous to the Read Uncommitted isolation level for database transactions. John is in the middle of a Business transaction to update a record, and Sheila is able to read the record about to be modified. There are a couple ways to deal with this.

One approach is to create a version column in the table(s) you are updating. When someone pulls up the form, store the current version number of the record in the session data space. When the form is submitted, check the version number in the session data. If it is different than the version currently in the database, this means someone else has modified the record in the mean time, and you can warn the user or prevent the operation altogether. If the version numbers match, do the update and increment the version field. This would be a good application for a database trigger.

The version number technique is called optimistic offline locking, and allows the greatest degree of concurrency. It is analogous to source control systems that let you check out a file concurrently with other users. A really sophisticated optimistic locking strategy might even let you merge in changes made by another user when a conflicting update has been detected, just as some source control systems do.

Moving to Read Committed

Another approach is to take a cue from database transactions and support the next isolation level through record locking. In database land, the Read Committed isolation level prevents anyone from reading data that is in the middle of a transaction. It does this by locking the records involved in the transaction.

You can do the same thing for Web applications by creating a WebLock column in your data. When someone pulls up the update form, check the WebLock field. If it is not locked, then lock the record by assigning a username or form token to the WebLock field. When the form is submitted, verify that the username or form token matches the WebLock. You may also want to create a database trigger that prevents any modification on a locked record.

Implementing record locking for business transactions does not necessarily require application code. You can do the lock checking in the same stored procedure that you use to SELECT the record fields for the update form. In fact, it is preferable do it this way as part of a singe transaction in the stored procedure. Likewise, you should update the record through a stored procedure that checks the locks as part of a single database transaction. The only ColdFusion required is the code to return HTML in response to the various stored procedure return codes. If you define standard return codes like “Record Locked,” you can use the same CFINCLUDE or custom tag in every Web form to handle these cases.

Locking records to prevent simultaneous updates is called pessimistic offline locking, and allows the least concurrency. But when the consequences of lost updates are really severe, it may be your only choice.

Lock up, but Don’t Throw away the Key

Don’t forget that any locking solution must include a way to expire locks. Otherwise, a Web user could retrieve a record, walk away for the afternoon, and keep everyone else from working in the mean time. You should write a stored procedure to periodically expire locks and/or build this intelligence into all locking routines. Schedule the procedure to run regularly using SQL Server’s scheduling facility or wrap it in a ColdFusion page and use ColdFusion Scheduler.

The Repeatable Read Analogy

In database land, the Repeatable Read isolation level guarantees that records will remain the same if they are visited again in the same transaction. To accomplish this, the database must lock the records for the duration of the transaction. The same principle applies to the Web. If you need to prevent records from changing during the course of a business transaction which takes place over several screens, you simply maintain the Web lock for the duration of the business transaction. If you're checking and setting the Web lock inside stored procedures, this will happen anyway since the actual database update (stored procedure call) is typically accomplished all at once on the final form submission in the business transaction.

The Serializeable Analogy

Recall that the Serializeable isolation level guarantees that no new rows will appear if a transaction visits the same query again. Unless you want to handle range locking like the database, the best way to deal with this is to make sure your application would be able to recognize an additional row and know what to do with it. One way to identify new rows would be to look for those without a matching WebLock token.

Collecting Usage Statistics

Web servers produce a raw log file containing a record for each “hit.” There are numerous tools to digest this raw data into something meaningful. One popular tool, WebTrends, is smart enough to recognize application cookies in requests or set its own when a new browser visits the site. This allows you to track the number of *visitors*, not just the number of hits. In addition, you can learn how people are using the site.

Using any scripting language, you can also write your own tools to parse log files and answer specific questions like “How many NT users visit a certain section of my site?”

Here are some of the more useful reports you can run with WebTrends.

Resources Accessed

Most requested pages
Least requested pages
Top entry pages
Least-requested entry pages
Top exit pages
Single access pages
Top paths through site
Most downloaded files
URL parameter analysis
Error pages (very useful for security analysis)

Activity Statistics

Activity by day, day of week, hour, etc.
Activity by length of visit
Activity by number of views

Referrers & Keywords

Top referring sites
Top referring URLs
Top search engines
Top search phrases
Top search keywords

Browsers & Platforms

Browsers by vendor
Netscape browsers
Internet Explorer browsers
Platforms

Where to Go from Here

Online Resources

Cisco Systems (www.cisco.com)
F5 Networks (www.bigip.com)

Books

Web Performance Tuning (O'Reilly)
Web Security & Commerce (O'Reilly)

Martin Fowler's *Patterns of Enterprise Application Architecture* (Addison-Wesley) has excellent discussion of business transactions and offline locking patterns for Web applications.

Other

Windows Load Balancing Services (WLBS) software for Windows NT is included with MSDN Professional or Universal subscriptions

Lesson 22
User Authentication

Introduction

The Web model allows a variety of ways to authenticate and authorize users. In this lesson, you'll explore the possibilities.

Objectives

By the end of this lesson, you should be able to

- Understand the various methods of authenticating users
- Avoid the pitfalls of Web-based user authentication
- Evaluate the merits of various techniques for your application

Authenticating Users

Client, session, and cookie variables don't actually track users. They track browsers. If you want user preferences to be available regardless of where a user is working, you must require some sort of user authentication and store the user's preferences in a database so they will be available wherever the user logs in.

There are three methods you can use to authenticate Web users: integrated browser authentication, client certificates, and an application login screen.

Integrated Browser Authentication

When you protect a Web resource on the server, the Web server will request authorization from the browser and the browser will display a dialog box requesting authorization. When authorization is successful, the Web server will make the username available as the environment variable CGI.REMOTE_USER.

Integrated browser authentication is by far the easiest way to authenticate users to your application because you can easily configure it in the Web server. By protecting your application at this level, your application pages don't have to worry about whether the user is authorized. To get the username logged in, the application just has to look at the CGI.REMOTE_USER variable. If you're running IIS, the hardest thing about this method is stripping out the domain name from the username. This is easily done using regular expressions. Typically, you would include this line in your Application.cfm.

```
<!--- Strip domain name and backslash
from username, as in "TSS\dchandler" --->
<CFSET AuthenticatedUser =
REReplace ("#CGI.REMOTE_USER#", "^.*\\", "")>
```

The drawback of integrated browser authentication is that the database of authorization credentials is entirely dependent on which Web server you're using. For IIS, you can only use NT domain accounts, which is obviously not practical on the Internet. It is possible to get around this limitation by writing a custom DLL using ISAPI or by purchasing a 3rd-party product like Authentix which solves the problem. Microsoft Visual C++ has a wizard for creating ISAPI functions.

Application Login Screen

If none of the other methods are practical, you can create a login screen for your application. To implement an application login, you must do the following:

1) Set a cookie, client, or session variable when the user logs in.
2) In Application.cfm, check to make sure the required variable is defined.
3) If you want the user to be "logged out" automatically when the user closes the browser, you must set your own memory-only cookie or rewrite the standard CFID and CFTOKEN cookies as memory-only cookies as discussed in Lesson 13.

If you use your own cookie for authentication, you don't want to set a cookie equal just to the username of the logged in user or anyone could log in by setting a cookie in their browser. Ideally, you want some combination of

1) User ID
2) The browser's IP address (OK, but proxy servers could foul up)
3) CFID/CFTOKEN or SessionID (better than IP) to uniquely identify the browser.

You could concatenate all these strings, encrypt the result with CFEncrypt or Hash, and use the result to set a cookie. To prevent someone from snooping, you should also use SSL to protect your application.

Note

At least in Internet Explorer, cookies that expire when the user closes the browser (the default) are never written to a cookies file. However, if the user has configured the browser to prompt before accepting cookies, the user can still see (and therefore copy) any cookie set by your application.

Two-Factor Authentication

For really high security applications, you must also prevent someone from logging in even if they have obtained (by permission or otherwise) a user's password. This requires some form of two-factor authentication (something you have and something you know). The something you have is normally some sort of biometric device or dynamic password generator like a SecureID token.

You can achieve almost the same effect without special hardware by requiring two logins. Initially, the user must log in with a one-time password which sets a permanent cookie containing some secret and the encrypted IP address. Future logins use only the regular password, but also check for the permanent cookie (unfortunately, a sophisticated user could still send this permanent cookie to someone else).

The Authentication Database

Using an application login screen, you can authenticate against

1) Your own application user database
2) An LDAP directory server with the CFLDAP tag (Windows 2000 is an LDAP server)
3) ColdFusion Advanced Security Services with the CFAUTHENTICATE tag to hit an NT domain, LDAP server, or ODBC database

Client Certificates

Like integrated browser authentication, client certificates are handy in that the Web server does most of the work; however, they are not yet widely used and require some setup on the browser. For that reason, they are probably best suited for extranet apps. If client certificates are in use, ColdFusion makes various certificate parameters available as CGI environment variables. To see what they are, turn on ColdFusion Server debugging and show all variables.

Some companies first looking at client certificates believe they need to set up their own certification authority (CA) in order to positively identify certificate holders. This is not the case. A better approach is to have your authorized users register their certificates with you by going to a one-time registration page with a special one-time password you have mailed to them at a known address. When the user logs in, simply record the certificate ID that was presented in your user database.

Real-Money Examples

It may be helpful to examine the authentication practices of various real-money Web sites. For what it's worth, E*Trade uses an application login screen, whereas Datek uses integrated browser authentication. Either way can work.

Password Security

When handling user passwords, keep in mind that most authentication methods send passwords in the clear. These include

- Integrated Browser Authentication (HTTP Basic Auth)
- Password entered on a login screen

Remember that a password input box (<INPUT TYPE="Password">) displays asterisks to the user, but still transmits the password in the clear.

If you are using IIS and NT Challenge / Response authentication, passwords are encrypted.

Tip

When using HTTP Basic Authentication with IIS, the authenticated user's NT password is stored in the variable CGI.AUTH_PASSWORD!

In order to protect user passwords in transit, you must use SSL. This works with any authentication method. With integrated browser authentication, the browser must visit at least one page on your site before authorizing in order to establish the SSL session.

If you authenticate against your own database, you should use a one-way encryption scheme like the ColdFusion Hash function and store the resulting hash, not the clear-text password. When the user logs in, you hash the password they entered and compare with the stored hash.

Changing User Passwords

For high-security applications, users should not be allowed to change their own passwords because they may choose a password which they are already using elsewhere, either on their network or the Web. Thus, the password may be well-known or easy to discover by sniffing network traffic as they log into other network services with the same password.

In addition, if a user is using the same password at, say, E*Trade, and something happens to his or her E*Trade account, they may blame you because "you knew their password." Your application should provide some way for an administrator or customer service representative to reset a user's password; however, you must be extremely careful to determine that the user on the phone is who they say they are.

Some sites use two passwords, like E*Trade's login password and trading password.

Where to Go from Here

Application Help

See the following chapter in *Developing Web Applications with ColdFusion*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 17: Application Security

See the following chapters in *Administering ColdFusion Server*, which is part of the standard printed and online documentation. You will also find it in the Help resource pane in ColdFusion Studio.

Chapter 8: ColdFusion Security
Chapter 9: Configuring Basic Security
Chapter 10: Configuring Advanced Security

Books

Web Security & Commerce (O'Reilly) contains an extensive discussion of authentication techniques, including digital certificates

Lesson 23
Designing Secure Applications

Introduction

Designing a secure Web application involves much more than just authentication. In many cases, people using your application won’t be required to authenticate at all. The most likely security “holes” are typically application bugs or oversights, not sophisticated IP-based attacks. In this lesson, you’ll also learn strategies for securing your database.

Objectives

By the end of this lesson, you should be able to

- Identify common attacks like SQL insertion, cross-site scripting, and session hijacking
- Understand how to program defensively
- Secure your application at the database level as well as in code

Common Attacks

The most common and dangerous attack is SQL injection (also called SQL insertion). All languages that dynamically generate SQL statements (vs. prepared statements with typed parameters) are vulnerable, including ColdFusion, ASP, PHP, Java, and C++, depending on how you use them.

In a SQL insertion attack, an (ab)user can modify the contents of your CFQUERY by modifying your form variables. For example, a typical DELETE query might look like

```
<CFQUERY...>
DELETE
FROM t1
WHERE ID = #Form.ID#
</CFQUERY>
```

SQL Insertion with Numeric Fields

Let's say the ID comes form a drop-down list of account numbers. To run a SQL insertion attack, the abuser can save the form and modify one of the account numbers to be "1; DROP TABLE t1"

When ColdFusion substitutes the value of Form.ID in the query, it will look like

```
<CFQUERY...>
DELETE
FROM t1
WHERE ID = 1; DROP TABLE t1
</CFQUERY>
```

So the abuser drops your table. Note that semi-colons aren't the only dangerous thing. For a delete query like this one, an abuser could simply supply a value of "ID" to delete all records (DELETE FROM t1 WHERE ID = ID). Neat, huh?

SQL Insertion Rule #1
Anywhere you're using a numeric field in a CFQUERY, validate it with IsNumeric.

Numeric Validation Example

```
<cfset safeData = StructNew ()>
<cfif isNumeric (Form.ID)>
      <cfset safeData.ID = Form.ID>
<cfelse>
      <!--- Report entitlement error so a
sysadmin can look into the hacking attempt --->
      <cf_appendErrorStruct
            errorStruct="VARIABLES.validation"
            formField="ID"
            formLabel="ID"
            errorMessage="You attempted to pass
                          a bogus account ID"
            isEntitlementError="TRUE">
</cfif>


<!--- Display validation messages --->
<cf_writeErrorHeader
      errorStruct="VARIABLES.validation">


<CFIF StructIsEmpty (Variables.validation)>
      <!--- Continue only if all validation
passed --->
      <CFQUERY...>
      DELETE
      FROM t1
      WHERE ID = #safeData.ID#
      </CFQUERY>
      .....
</cfif>
```

Instead of using the raw Form variable in the query, I like to use a structure to indicate the variable has been validated. This forces you to use a variable scope in the query, which is a good thing.

SQL Insertion with Character Fields

Now, let's look at queries with character fields. SQL always requires single quotes around character fields, so your query might look like:

```
<CFQUERY...>
SELECT
FROM accounts
WHERE LastName = '#Form.LastName#'
</CFQUERY>
```

An abuser could potentially supply a form field containing single quotes and a semi-colon. However, ColdFusion automatically escapes all single quotes inside queries so as not to cause SQL syntax errors when the form variable contains an apostrophe (i.e., last name = O'Malley). So character fields are safe, UNLESS you've wrapped a variable in the widely-misunderstood PreserveSingleQuotes function, like this:

```
<CFQUERY...>
SELECT
FROM accounts
WHERE LastName =
      '#PreserveSingleQuotes(Form.LastName)#'
</CFQUERY>
```

The PreserveSingleQuotes function causes ColdFusion to preserve the quotes instead of escaping them as it normally would. So, an abuser can supply a string like Chandler'; DELETE FROM t1 WHERE 'a' = 'a

Then your query will look like

```
<CFQUERY...>
SELECT
FROM accounts
WHERE LastName = 'Chandler'
;
DELETE FROM t1 WHERE 'a' = 'a'
</CFQUERY>
```

SQL Insertion Rule #2
Don't use PreserveSingleQuotes () in a CFQUERY!

SQL Insertion with the IN Clause (Character Fields)

One case where the use of PreserveSingleQuotes is commonly and mistakenly used is where you are building a list of terms for an IN clause.

For example, suppose you have a group of checkboxes in a search form:

```
<input type=”checkbox” name=”state” value=”IA”>
<input type=”checkbox” name=”state” value=”GA”>
<input type=”checkbox” name=”state” value=”KS”>
```

If all three are checked, the value of the Form.state variable will be “IA,GA,KS.” You might use this in an IN clause to retrieve all records with a matching state.

```
SELECT * FROM data
WHERE state IN (#Form.state#)
```

However, since state is a char field, your list of search terms must be quote-delimited. You might be tempted to put single quotes around each state abbreviation in the form, then use PreserveSingleQuotes (Form.State) to build your IN clause. Don’t do it! Instead, use the ListQualify function to do this. When used in a query, the ListQualify function is smart enough to escape any quotes in a list term, but not the term delimiter quotes themselves. So you can handle a list of character strings as follows:

```
<cfquery…>
SELECT *
FROM table
WHERE state IN (#ListQualify
      (Form.SelectedStates, “’”, “,”, “ALL”)#)
</cfquery>
```

If your list contains a term with an embedded apostrophe, like “IA, G’A, KS”, ColdFusion will correctly send this to the database:

Note

The code above uses the default list delimiter (comma) as will all data coming from grouped form fields. If one of the field values contains a comma, then you may have problems. So don’t give your grouped checkboxes or multi-selects values that might contain commas. If you must, you can put your own delimiter in the field values and do your own pre-parsing. Test it well!

SQL Insertion with the IN Clause (Numeric Fields)

If the database field in an IN clause is a numeric type, then validate each term with isNumeric before the CFQUERY, as in

```
<cfset SafeData.IDList = StructNew ()>

<cfloop index="thisTerm"
        list="#Form.selectedIDs#">

      <cfif isNumeric (Variables.thisTerm)>

            <cfset SafeData.IDList = ListAppend
                         SafeData.IDList,
                         Variables.thisTerm)>

      <cfelse>

            <cf_appendErrorStruct
                  errorStruct="VARIABLES.validation"
                  formField="ID"
                  formLabel="ID"
                  errorMessage="You have selected an
                               invalid account ID."
                  isEntitlementError="TRUE">

      </cfif>

</cfloop>


<cfquery…>

SELECT *

FROM table

WHERE ID IN (#SafeData.IDList#)

</cfquery>
```

SQL Insertion Rule #3

When building an IN clause from a list for a character field, construct it with ListQualify instead of PreserveSingleQuotes. If the database field is a numeric type, validate each term in the IN clause with isNumeric.

Defensive Programming Practices

The central challenge of Web application security is being able to anticipate the anticipated. People using your application in the ways you intend are not likely to break it. People using it in ways you never imagined can cause serious problems. Since you cannot anticipate every possible wayward user, you must *program defensively*.

What does it mean to write defensive programs?

Never Assume

There are several important things to remember about Web requests:

1) Requests may not be in sequence. Just because your application gets a request for the second form in a sequence doesn't mean the user has already been to the first.
2) Requests may not contain the data you expect. A user can modify any form or URL to include more data, less data, or different data.
3) Requests may be sent more than one once. Reload is evil.
4) Cookies can come from anywhere. Since cookies are stored in text files, it's pretty easy for someone to set a cookie for your site in their browser, change the cookie, copy a cookie from someone else, or delete a cookie. Don't rely on cookies for security.
5) Requests may not come from a browser at all. Requests can come from proxy servers, robots, and automated "security scanners" specifically designed to find security holes in your application.
6) Requests that look like they come from a specific browser (say, Netscape 4.0) may in fact come from one of these other sources disguised as Netscape.

In other words, clients are fickle. Every application page must expect the unexpected.

Use a Bozo Filter

One strategy for making it more difficult to submit bogus form data is to look at the HTTP referrer in the script which accepts the form submission. If the HTTP referrer does not match the URL of the page with the form as sent by the application, then you know someone has modified the form. However, this technique is easily circumvented using man-in-the-middle tools such as Achilles.

A related technique is to send a form token with every form as discussed in the previous chapter. When the form is submitted, it must contain a current and correct token. This does not prevent a hacker form modifying a form, but does make it more difficult for someone to write a script which submits the form multiple times.

Validate, Validate, Validate

Chandler's First Law of Web Applications Security
Never assume that form data came from your form.

A well-designed Web application must do extensive data validation to prevent errors and potential security problems. One of the difficulties of Web design is that a knowledgeable user can save any HTML page containing a form and edit the form fields and data before submitting the form. For example, someone might expand the maximum size of text input fields or change the options or values in a drop-down list box. Because anyone can save and modify a page, client-side data validation using JavaScript cannot be relied upon to protect the application from bogus form submissions. You MUST validate data on the server, even if you also do it on the client.

Always Validate on the Server

Client- and server-side validation is discussed extensively in Lesson 9, *Form Validation*. Here is a quick recap:

1) Check URL and Form variables to make sure they're defined
2) Check all variables used in queries, especially numbers
3) Avoid Evaluate ()
4) Do not use PreserveSingleQuotes () in a CFQUERY

When validating form and URL variables, consider the following:

1) Check all text input fields for proper length and content. For maximum security, text fields that are used to update the database should be stripped of special characters like the apostrophe or semicolon, which can be used to submit multiple statements to the database.
2) Check all multiple-choice elements (drop-down list boxes, check boxes, radio buttons) to make sure that the selected option is a valid choice. Store valid choices in the session data space or, on the form action page, rerun the query used to populate the list.
3) If you are using named submit buttons, code a default behavior in case a hacker creates his own button name and value.
4) If you use looping constructs to handle a large number of form variables, make sure that you loop over the list of expected variables and not the list of user-supplied variables (From.Formfields). This way, you will not get tripped up by unexpected input. Use structure notation instead of Evaluate () to parse variables names or values. You do not want to allow a user to execute arbitrary code merely by sending you an expected form variable name or value!

Don’t Store Sensitive Variables on the Client

Ideally, the only information you should ever store in the browser is the CFID and CFTOKEN cookie pair automatically created by ColdFusion Server to map Session and Client variables. Hidden variables and cookies should not be used to store

1. Information that users would want to hide from potential snoopers, such as passwords and credit card numbers.
2. Information that you do not want users to see or modify, such as their login privileges and back-end passwords.

Instead, store sensitive items in session data space on the server. And of course, always protect sensitive applications with SSL.

Validate Program Output (Cross-Site Scripting)

In addition to validating all input for potentially harmful effects on the Web server or database, your Web programs should also validate input and/or output to prevent users from writing HTML in places where it doesn’t belong. For example, a bulletin board application typically allows users to post messages that will be read by other users. The application must check to make sure that no user posts a message containing malicious HTML, since the message will be displayed in many other users’ browsers. Potentially malicious tags include

- <OBJECT>
- <EMBED>
- <APPLET>
- <SCRIPT>
- <IMG>

Any time your application displays the value of a form variable, URL parameter, cookie, or stored value which originally came from one of these sources, you should check to make sure that the display variable does not contain malicious HTML. One way to neutralize potentially harmful HTML code is to wrap all output with HTMLEditFormat (), which converts the opening and closing tags to their HTML representations.

For more information, see Allaire Security Bulletin ASB00-05, Cross-Site Scripting Vulnerability Information for Allaire Customers. Note that this issue is not a weakness in any particular language or application server, but affects all Web applications.

Cross-site scripting is a relatively minor concern because only a logged-in user would be able to store malicious code for other logged-in users, and that is unlikely. In addition, it only has the potential to disrupt individual browser computers, not the server or any data on the server.

Always Specify Variable Scope

If you don't specify variable scope, ColdFusion looks in the following order:

1. Local variables created with CFSET and CFQUERY
2. CGI variables
3. File variables
4. URL variables
5. Form variables
6. Cookie variables
7. Client variables

Suppose that you set Client.AuthUser to the username when a user logs in. As long as you are storing client variables in the server or database, you feel secure because you know that Client.AuthUser cannot be modified in the browser. Your Application.cfm would check for Client.AuthUser to make sure the user is logged in. What happens if you get lazy and omit the Client prefix? Someone can submit a form with a variable named AuthUser and trick your application into letting them in!

Inside a CFOUTPUT which loops over a query, it is a good idea to use the query name with the database field names.

Besides the security benefits, explicitly scoping variables

- Makes your code more readable
- Improves performance

Just do it!

You should specify the local "Variables" scope even though ColdFusion looks for these first. If you forget to set the local variable and reference it without the scope, then a hacker can submit his own variable in the URL or form.

Use an Input Filter

A new class of product called *application firewalls* can sit on your Web server and check all URL and Form data for potentially malicious attacks like SQL injection. You can easily write your own input filter for ColdFusion by putting some code in Application.cfm that loops over the Form and URL structures and uses regular expressions to match potentially dangerous keywords and characters like SELECT, the semi-colon, etc. An input filter should not be strictly necessary if you've properly validated query parameters, but can be very useful as an intrustion detection system nonetheless.

Manage Sessions and Authentication Securely

Server-side session variables are a great way to prevent sensitive data from crossing the Internet. However, their security hinges entirely upon the strength of your session management and authentication scheme.

If a hacker can snoop or guess a valid session ID (CFID and CFTOKEN), he simply has to set those browser cookies and voila! He becomes the user associated with that session. For this reason, session IDs should be carefully guarded in the following ways:

1. Never send session tokens in the clear. Do not create the tokens in your application until after an SSL session has been established. This way, snoopers will not be able to learn any valid session IDs.
2. Do not generate session IDs sequentially or else an attacker can easily guess a valid session ID. By default, this is what ColdFusion does (although the requirement for a matching CFTOKEN makes it harder to fake CFID). In ColdFusion MX Administrator Settings, check the box to use a UUID for CFTOKEN. A UUID is a much stronger random string. In CF5, you can edit a registry setting to use UUIDs for CFIDs instead of the standard sequential numbering.
3. Never put a session ID in a URL. Otherwise, another user on a machine might be able to look through the browser history and re-establish a valid session. In addition, if a user leaves your site, all URL parameters will be sent in the Referer header to the next site.

Make Sure Users Can't Circumvent Access Control

Broken access control is another common weak spot in Web applications. If any page in your application fails to check for valid authentication or session credentials, your entire application can potentially be compromised. A common mistake is to allow users to browse directly to files in a URL (such as a link to a Word doc or PDF that requires authentication). Remember that Application.cfm only runs for ColdFusion pages. If a user can link directly to for_users_only.pdf, your authentication logic in Application.cfm won't run. The correct way to do this is to create a ColdFusion page that retrieves the file using CFCONTENT. Your hyperlink would then contain

```
http://localhost/getFile.cfm?FileID=22
```

Because getFile.cfm is a ColdFusion page, your authentication logic in Application.cfm will run and protect against unauthorized file access.

Files and Directories

How you arrange the files in your application also has security implications. Here are some general things to think about.

Don't Rely on Security by Obscurity

Some developers are in the habit of leaving back doors unlocked, like administration scripts where the URL is known only by a few people. Don't do this. It's quite easy to sniff URLs on the Internet or intranet, so anyone who really wants to can probably discover your secret. Not to mention that someone can write a script to try various likely filenames.

Keep Data, Configuration, and Log Files out of the Web Root

Don't store your application's Microsoft Access database in a Web-accessible directory. Ditto for log files. Applications frequently use configuration files containing the names of servers, required connection passwords, etc. Make sure these are stored outside the Web root.

Don't Store Passwords in Application.cfm

Here's one that's not even obscure. Even though the Web server *should* always execute Application.cfm rather than return its contents, there are well-known exploits to display the contents of a script file (see the next lesson for more information). In addition, there are probably some exploits still lurking out there, so reduce your risk now.

Separate Application and Administration Pages

Keep application and administration pages in separate directories. This makes it easier to require a different type of authentication for the administration pages, such as NT challenge / response. This way, you can also limit the range of IP addresses which are allowed to connect to the administration pages. Also, it makes it more convenient to set stricter file system permissions.

Be Careful with File Uploads

Applications which allow file uploads must be scrutinized very carefully.

Store Uploaded Files Outside the Web Root

Uploaded files should always be stored outside the Web root or else someone could upload an executable file (or ColdFusion script) and then call the URL to execute it! If any application on a server allows uploaded files, the entire server is at risk.

…But Not Above the Application Root

Ideally, uploaded files should be stored in a sister directory to the application directory, where only the application directory is visible to the Web server. The danger of storing files above the application directory in the physical file system is that someone could upload a file named Application.cfm file. If ColdFusion Server does not find an Application.cfm file in the same directory as the requested page, ColdFusion searches all the way up to the root of the file system, so the uploaded Application.cfm could potentially be executed for certain requests.

Disable Execute Permission on Uploaded Files

As an extra precaution, you should disable execute permission both in the file system and in IIS. Execute permission must be granted in both places in order for a file to be executed in response to a Web request. ColdFusion pages only require script permission in IIS.

On Windows NT, you can disable execute permission for all uploaded files by disabling it for the upload directory. All uploaded files will inherit the directory permissions. On Unix, you can specify the *mode* (permissions) for an uploaded file in the CFFILE tag.

Store Include Files Outside the Web Root

Keep your custom tags, UDFs, and other include files outside the Web root and instead set up a ColdFusion mapping to call these from your application pages. You don’t want users to be call your include files directly because they likely won’t have all the validation code in them.

Consider Disabling CFFILE

ColdFusion Server allows you to disable the CFFILE tag in the ColdFusion Administrator. This disables all file operations, including file uploads. You may also want to disable (or secure) CFDIRECTORY and CFCONTENT, especially in a shared hosting environment.

Encode ColdFusion Pages with cfencode

ColdFusion provides a utility for encoding your ColdFusion pages. The ColdFusion server can read these pages in their encoded form. This is useful for several reasons:

1) It makes it hard for someone to modify your pages by editing them in a text editor.
2) It prevents your code from being discovered in the event that someone does break into your server.
3) If you are selling a commercial software application, it helps protect your intellectual property.

Warning

Encoded ColdFusion pages can be decoded using illegal decryption tools, so you cannot rely on page encoding alone for security.

The cfencode utility is located in the \Bin directory where ColdFusion Server is installed. Here is a sample batch file used to encode all ColdFusion pages in a directory tree. Be sure to specify only *.cfm files. Your images and HTML won't be very useable after ColdFusion encoding! This example first saves a copy of Error.cfm out of the way because ColdFusion custom error pages will not work if the custom error template has been encrypted.

```
copy Code\App\Error.cfm .
c:\cfusion\bin\cfencode Code\*.cfm /r /v 2
copy Error.cfm Code\App
del Error.cfm
```

Important

Always copy your ColdFusion pages to a new directory before encoding them. Once they are encoded, you cannot get back the clear text for editing!

Typical Application Layout

Code/*	
Common/	No authorization required
Main.css	Main CSS style sheet
Error.cfm	Custom error page
Images/	All images
Auth/	Login handling
Application.cfm	Set default variables, link in style
Flowctl.cfm	Application controller
Terms.cfm	Display terms and conditions
App/	All pages requiring authentication
Application.cfm	Verify user logged in
Flowctl.cfm	Application controller
application pages	
Admin/	Admin pages
Application.cfm	Verify admin user logged in
Flowctl.cfm	Application controller
admin pages	
Config/	Configuration files
Defaults.cfm	Default variables
Cron/	Scheduled pages
Include/	Custom tags, UDFs, etc.
Data/	Text files or Access databases
Logs/	Log files
Upload/	Uploaded files

Important

The Code directory is the Web root. All other files are outside the Web root for security purposes.

Without a separate Auth directory, the Application.cfm in the App directory would require user authentication just to get to the login page. Obviously, that wouldn't work.

Some variables will be common to all three Application.cfm files, like the name of the data source. If you want to be able to change these in one place, you can write each Application.cfm to include Config/Defaults.cfm. This is also a good place to create a link to the CSS style sheet in the document HEAD section.

The controller file is discussed further in Lesson 25.

For better performance, set expiration times on the Common/ and Images/ directories as discussed in Lesson 2.

Database Security

An integral part of application security is database security. By correctly configuring your database, you add another layer of defense against would-be attackers.

Use Individual Database Logins

The most secure way to build a Web / database application is to pass through the Web login directly to the database. This requires that you have a database user for each application user. This is more secure because you can set database privileges on an individual basis. For example, you can use views or stored procedures containing a WHERE clause with CURRENT_USER to prevent users from seeing any data other than their own.

However, this approach has several disadvantages:

1) It requires more administrative effort
2) It reduces the benefits of ColdFusion connection caching since ColdFusion will have to create a new connection for every new user

Tip

With ColdFusion 4.5, you can wrap CFIMPERSONATE around a query to make a trusted connection to Microsoft SQL Server as the authenticated NT user. Be sure to check the “Use trusted connection” box in your ODBC data source.

Use Role Logins

The alternative approach is to use a group of database logins for all application users. For example, you might have an administrator login with all privileges and an application user login with permission to run only certain stored procedures.

By passing the username as a parameter to stored procedures, you can still ensure that users see only their own data.

The primary disadvantage to this approach is that you must store the database username and password somewhere accessible to the ColdFusion Server. You can store it in the ColdFusion data source parameters and set up one data source for each database login, or you can always included login information in CFQUERY. If you store passwords in a configuration file loaded by Application.cfm, use CFEncrypt and CFDecrypt to avoid storing the password in clear text.

Use Stored Procedures

Stored procedures make it much easier to develop secure applications for several reasons:

1) Because parameters are strongly typed and parsed beforehand, it is impossible for someone to send their own SQL statements through a stored procedure.
2) In ColdFusion Administrator, you can configure a data source to allow access to the database only through stored procedures.
3) You can grant execute permissions to logins and roles for each procedure. This gives you much finer-grained security control than giving broad access to tables.

In addition, stored procedures help ensure that all developers access data in a consistent manner, not to mention the performance advantages.

Use Views

If you don't use stored procedures, at least use views. Views still give you finer-grained security access than tables. Using views, you can deny users access to the underlying table(s) while still giving them permission to work with a subset of the data. Views can be used to

1) Restrict display and update of certain columns
2) Make only certain rows available

In addition, because views hide the complexity of joins and complicated queries, they also help ensure that all developers access data in a consistent manner.

Views do not, however improve performance like stored procedures. When you run a query against a view, the database has to mesh the query which defines the view with the query you supply, which takes just as long as if you had written the resulting query on the underlying tables. For this reason, you should avoid relying heavily on complicated views.

Where to Go from Here

Online Resources

See the Allaire Security Zone at www.allaire.com/security/.
See the Microsoft Security Zone at www.microsoft.com/security/.

The Open Web Application Security Project has produced an excellent guide called The Ten Most Critical Web Application Security Vulnerabilities. You can find it at www.owasp.org.

Books

Web Security & Commerce (O'Reilly)
Hack Proofing ColdFusion (Syngress)

Lesson 24
Securing Your Servers

Introduction

This lesson will teach you the basics of Web security, including network, Web server, ColdFusion server, and database security.

Goals

By the end of this lesson, you should be able to

- Understand network security for a typical Internet application
- Prevent common exploits against your Web server
- Lock down ColdFusion Server
- Protect your database from unauthorized access

Network Security

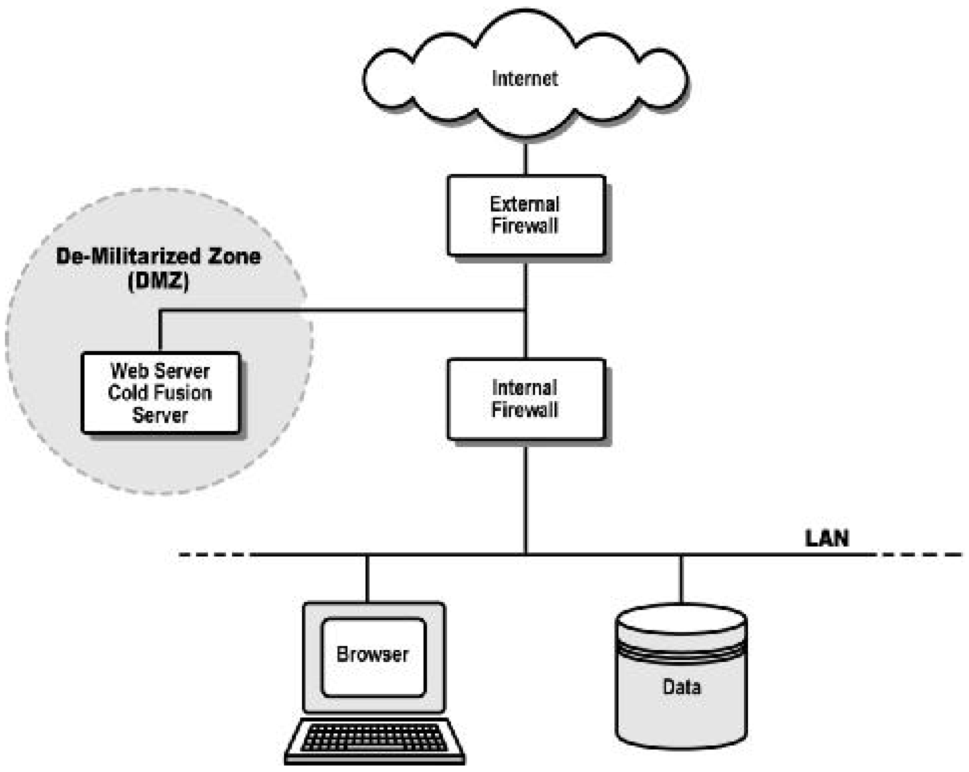
Internet applications typically use two firewalls as shown in Figure 24-1. The land between the firewalls is called the demilitarized zone, or DMZ.

This approach is very flexible. It allows you to safely allow many kinds of services between the intranet and the servers in the DMZ while restricting external services.

Typically, the intranet firewall is configured to deny all inbound connections except connections from the Web server to a database on the intranet. The Internet firewall is configured to allow only Web (port 80) and SSL (port 443) connections to the DMZ.

A properly-configured Internet firewall is an important first line of defense against attacks. By limiting your exposure to ports 80 and 443, you eliminate many of the potential exploits against services running on other ports.

Figure 24-1 A DMZ allows great flexibility.



Web Server Security

Secure the Machine

You should follow the practices recommended by your vendor to prevent dangerous network attacks. Typical steps include

1) Make you have applied all the latest security patches and service packs.
2) Disable all services which are not necessary. On both Windows NT and Unix, you can use `netstat -a` to find open ports.
3) Configure packet filtering to deny everything except TCP (IP protocol 6) ports 80 and 443 (if you're using SSL).

Secure the Web Server

It is *imperative* that you check your vendor's security guidelines and mailing lists to keep up to date. Web server security starts with understanding your configuration. In general, you should

1) Turn off directory browsing in the Web virtual directory in which the application runs.
2) Disable unnecessary script mappings (.cgi, .shtm, .htr, etc.).
3) Make files in the Web root read-only, both at the Web server and file system level. ColdFusion pages require IIS scripting permission, but not execute permission.
4) Encrypt all script files (.cfm) except for custom error templates in order to prevent hackers from discovering the application code.
5) Store all configuration information (usernames, passwords, etc.) and logs outside the Web server root to make them inaccessible.

For IIS, go through the security checklist at www.microsoft.com/security. At a minimum, you should do the following:

1) Remove IIS (well, most of it, including the sample applications).
2) Be sure to patch IIS 4.0 for the gaping ::$DATA security hole. This is fixed in NT Service Pack 4.
3) Patch IIS for the "Malformed Hit Highlighting Request" vulnerability (see Allaire Security Bulletin ASB00-07).
4) Disable all hidden administrative shares (C$, ADMIN$) using Net Share /d (see the IIS Security Checklist for more information).
5) Disable Microsoft RDS (this is different than ColdFusion RDS). Details on how to do this are available in the IIS Security Checklist.

Secure All Applications

Unfortunately, your Web server is only as secure as the weakest application. Evaluate all applications using the principles in the previous lesson.

Secure Your ColdFusion Server

Secure the Machine

You should follow the practices recommended by your vendor to prevent dangerous network attacks. Typical steps include

1) Make you have applied all the latest security patches and service packs.
2) Disable all unnecessary services.
3) Allow traffic from only those TCP ports which are necessary to connect to ColdFusion (and/or other core operating system services if the machine participates on a Unix or Windows network).

Secure ColdFusion Server

You cannot treat a production ColdFusion server the same as a development server. Many of ColdFusion's features, when not configured properly, become gaping holes. Do the following:

1) Disable ColdFusion RDS. RDS is an intentional security hole, allowing developers to remotely access databases and files on the server. On NT, disable the ColdFusion RDS service in Control Panel | Services.
2) In ColdFusion Administrator, restrict debug output to selected IP addresses. If you just turn it off without restricting the IP addresses, anyone can see debug output by simply appending "?mode=debug" to a URL.
3) Secure the ColdFusion Administrator. In the Web server, allow only internal IP addresses to connect to \CFIDE\Administrator and require some form of Web authentication. You may also want to change permissions on the files so that only the Web server and network administrators can use them. If you're really paranoid, move the directory off the server and just install it when you need it.
4) Set a strong password for the ColdFusion Administrator.
5) Remove the ColdFusion Server documentation. Some of the sample applications and examples contain vulnerabilities. You can easily remove the documentation by running the installation again.
6) Use the robots.txt file to prevent search engines from indexing your dynamic pages. See article 14719 in the Allaire Security Zone for more information.
7) In the ColdFusion Administrator, secure or disable CFEXECUTE (ooh), CFFILE, CFDIRECTORY, CFCONTENT, CFREGISTRY, CFOBJECT, and CFADMINSECURITY. This is especially important in a shared hosting environment.

Secure All Applications

There's a reason we keep repeating this. Your applications themselves are the most likely source of security vulnerabilities.

Secure Your Database Server

Secure the Machine

You should follow the practices recommended by your vendor to prevent dangerous network attacks. Typical steps include

1) Make you have applied all the latest security patches and service packs.
2) Disable all services which are not necessary.
3) For SQL Server, use the Server Network Utility to disable all Net-Libraries which you are not using. Typically, the only one you need is TCP/IP. You might consider changing the default port from 1433 to make the service more obscure in the event of an attack.
4) Allow traffic from only those TCP ports which are necessary to connect to the database (and/or other core operating system services if the machine participates on a Unix or Windows network).

Secure the Database

Important
If you're still using the default administrator account (sa), change the password!

How you go about securing your database will depend largely on how you're using it, but in general you want to

1) Restrict user access rights to the minimum necessary.
2) Use individual logins for each user (or application role).
3) Review your end-user authentication mechanisms.

Secure All Applications

Since you are probably running the database behind the intranet firewall, application security is likely to be a bigger concern than machine security. See the guidelines in the previous lesson.

Where to Go from Here

Online Resources

Get the IIS Security Checklist at www.microsoft.com/security.

Drop by the Allaire Security Zone at www.allaire.com/security. There are many bulletins, best practices articles, and alerts. In addition, you can subscribe to the Allaire Security mailing list.

Now and then, stop in at www.ntbugtraq.com. This is the archive site for a well-respected NT security mailing list. NTBugtraq usually has information on exploits days or weeks before Microsoft issues an official bulletin.

For security-related news on the lighter side, hit rootshell.com.

Books

There are half a dozen good books on security, but security books go out of date quickly. Your best bet is to check the latest titles from O'Reilly in their security resource center at www.oreilly.com.

Lesson 25
Model View Controller

Introduction

By now, you should have an understanding of the tools in your ColdFusion toolbox. In this lesson, we'll discuss the popular MVC pattern for Web application architecture.

Goal

By the end of this lesson, you should feel comfortable that you have enough knowledge to go and build a successful Web application!

Fundamentals of MVC

Model View Controller is perhaps the most quoted (and misquoted) design pattern around. It was discovered (the patterns community insists that patterns are discovered, not invented) by Trygve Reenskaug in the 1970s for use with Smalltalk. Only recently, however, has it gained widespread popularity, largely as a result of the interest in user interface programming that the Web has made possible.

Benefits of MVC

When applied correctly, MVC should help to improve

- Readability (especially flow control)
- Maintainability (loosely-coupled code minimizes code duplication)
- Security (through better flow control, centralization)

In general, the pattern is a highly structured way of organizing your code so as to minimize functional duplication and thereby promote reuse.

How is MVC Different?

Most applications are organized initially according to the Transaction Script pattern [Fowler]. In the transaction script model, an application is page-centric, and each page executes a prescribed series of database actions. As an application grows more complex, there tends to be a growing amount of duplication in the scripts. The transaction script model tends to mix HTML, domain logic (business rules), and flow control all together in a single template.

By contrast, MVC emphasizes loosely coupled code by factoring the code into three logical parts:

Model	**View**	**Controller**
• Encapsulates state (including database) • Responds to queries • Enforces domain logic	• Renders the model • Sends user actions to controller	• Controls application flow • Sends updates to model • Selects view

MVC Principles for Web Applications

MVC always recognizes the separation between model and view. For example, the same business rules apply to a new account whether it is entered by a user or an administrator and whether the interface is a green screen or the Web. In this example, there might be four different views, but the same model code should be used for all. The controller selects the appropriate model and view code for each request. In practice, it is quite hard to completely separate domain logic from presentation, but it is worth striving for.

Ideally, MVC calls for separation of controller and view also, but this is less important and not always found in practice. For example, a form may have an editable and non-editable mode (display vs. edit). MVC teaches the use of a single controller and two views to implement this. However, in Web design, it is common to find two templates, each containing both view and controller code. As with model and view, complete separation between view and controller is difficult to achieve.

Model

One of the difficulties in discussing MVC is the confusion of terms. Model code in MVC should not be confused with a *domain model*, which is a logical representation of all objects in a system. The model in MVC encompasses the domain model and data source, but also session variables and all the code that enforces domain logic and interacts with the data source. Examples of model code include

- Queries
- Code that reads or writes to the session data space
- Conditional logic that sets object properties based on other properties

A component-based model features collections of EJB, CORBA, and CFMX Components, all of which are containers for properties and methods associated with objects in the domain model.

A transactional model makes more extensive use of CFQUERY and stored procedures.

Ideally, model code should be view-independent. That is, it shouldn't care whether requests are coming from a Web browser, a WAP phone, or a user vs. an administrator screen for the same underlying method.

View

View code renders the model, typically as a form or table. Views

- Are action-independent. A well-designed view can combine add and update forms within a single view (but not read-only and editable modes).
- May call “view helpers” to assemble data for the view (like queries for a drop-down list box)
- May call other views to create a composite view, such as a data entry screen within the context of a standard navigation framework.

Controller

Here again, there is some confusion of terms. An MVC controller consists only of the logic that separates model and view. However, many applications also use a *page controller* (like Fusebox), which is a central page that dispatches requests to the appropriate handler page, or an *application controller*, which is similar in concept but also enforces the sequence through which users move through pages. We will discuss the application controller in more depth shortly.

In ColdFusion, a controller is likely to call model and view code using CFINCLUDE. Only in metadata-driven architectures is it really practical to separate controller and view code entirely, but it is at least good practice to keep controller logic in a view separate from HTML below.

Application Controller

An application controller is a design pattern that uses a single page to handle all Web requests and dispatch them to the appropriate model and view. An application controller shares much in common with a page controller, but is more useful when users are required to move through pages in a defined sequence. An application controller is typically used to

- Set module-wide or service-wide constants
- Declare common functions
- Implement standard navigation (menu, back button)
- Manage session data space
- Simplify flow control
- Handle all errors and exceptions robustly
- Check user permissions for every action
- Sequence page flow

We will now consider a particularly useful application controller called an event-driven stateful controller.

Event-Driven Stateful Controller

In page-driven architecture, every page does more or less the same things:

1. Read and validate form and URL parameters
2. Persist variables needed later
3. Modify something in the database
4. Run queries to populate the next screen
5. Display the next screen

This works fine when all the pages are relatively independent of each other. That is, they can be called in any order and do not depend on data structures created by other pages. However, problems begin to arise when there are many entry and exit paths to and from a page:

- How do you know which page you came from? This leads to the creation of flag variables like “what” and “origWhat.” Every page develops numerous branches based on the preceding page, and these multiple paths through every file become very difficult to maintain.
- There is duplication of code on many pages to check for prerequisite data structures.
- In addition, if there are four possible action pages for a given page, the action code must now be duplicated in four places.

In order to address these problems, the code must be factored another way. An event-driven controller does this by maintaining information about the current and previous pages at all times (a simple state machine). The request flow becomes

1. Look up previous state (= view = form) in session
2. Persist data from previous view (independently of where we’re going to)
3. Look up new event in URL
4. Include model to validate variables and insert / update / delete
5. If no errors, move to next state and view
6. Otherwise, return to previous state and include previous view

In the event-driven flow controller, the URL no longer contains a different page address for every possible page, but rather only an event name indicating which button or link was clicked. The controller itself keeps track of the previous page and selects the next page based on the event name. Model and view code is then dynamically included based on the previous state, current event, and validation results.

When using an event-driven flow controller, be sure to select event names that are unique for each state.

Stateful Controller Design

In addition to cleaning up page flow logic considerably, the stateful controller accomplishes an important security objective: it prevents users from modifying data after it has been entered in cases where later forms depend on earlier values.

For example, suppose Salli is assigning account privileges with a multi-form wizard. She selects an administrator account for which all privileges are possible, then proceeds to a screen where she can enable one or more of the privileges. She enables all privileges, then clicks the back button to go back to the first form in the sequence and selects an unprivileged account. She then uses the forward button to go to the final page in the wizard and submits it. At this point, the application has in session data space an unprivileged account with selected permissions that apply only to a privileged account. The application code must be fairly sophisticated to prevent this happening.

A stateful controller would prevent the preceding scenario because when Salli goes back to select a different user account, it would recognize that the event (selectUser) is invalid for the current state on the server, which is permSelect. Salli would get an error message and would have to continue with the next page in sequence or start over.

Unfortunately, this design effectively prohibits use of the back and refresh buttons altogether because when the user click these buttons, the browser does not inform the server what's happening. Thus, the browser and the server are on in two different states. With a little extra intelligence, we can securely keep browser and server in sync.

Handling the Back Button

In order to allow the back button in a stateful controller, we must detect any repeated request and roll back all session data to where it was during the previous request. This simulates what the browser does when you go back and resubmit a page.

To do this, we will use a random *page nonce* in each URL. The controller stores the associated state and the names of all session variables created during the request. At the beginning of each new request, the controller looks at the incoming nonce in the URL. If it's been seen before, then the user must have pressed the back or refresh button, so:

- Revert to the state associated with the prior occurrence
- Delete all state history after the previous occurrence
- Delete all session variables created after the previous occurrence

In this way, we allow the use of the back button, but we ensure that when Salli backs up to select a different account, all selections associated with the privileged account will be deleted from memory.

Final Steps for the Stateful Controller

A typical application consists of many different types of flows: single-page forms, multi-form wizards, frame-based forms, etc. Rather than write a custom controller for each module or service, you can use a set of application-wide controllers to implement the different kinds of flows. The key to this is to separate the flow control logic for each module from the logic of the controller itself (which handles the back button, etc.) The application-wide controller can dynamically include the module-specific logic based on hierarchical state names, like mts.mtinput.wireentry1. The dot notation corresponds to a directory path where the controller will find the controller definitions file. This hierarchical approach has several advantages:

- You can include module-specific settings and functions along with the module-specific flow logic
- A state in one module can transition to a state in another module
- The back button will work securely system-wide

Using MVC

We conclude with a few tips for designing pages consistent with the MVC pattern.

Model Tips

- Persist form variables on exit. For each form (view), write an include file to save all form variables in the session data space. In the flow control logic, always include this code for the previous state. Where possible, loop over a list of variables names rather than saving them all individually.
- Include database actions as separate files. This promotes reuse as well as readability.
- Model code should contain ColdFusion tags only (no HTML).

View Tips

Use the Value Object pattern to make the view code independent from model code. Instead of displaying form or query variables directly in the view, have the model code build a structure from the appropriate source (form, query, or session data, usually). Then the view can display the contents of the structure without any conditional logic required. Here is some pseudo-code for the model code to build the structure:

```
Initialize a local structure and set default properties
If edit or display mode,
    Get record key from URL or form and populate
    the structure from the query
Further populate the structure from
    Form (if defined) or Session space (if defined)
    Init properties to empty string if not defined (CFPARAM)
```

Ideally, views should not contain any ColdFusion tags except loops to populate drop-downs and other types of lists.

Debugging Tips

Use CFDUMP in OnRequestEnd.cfm to display your entire session data structure. This makes it easier to see what's going on behind the scenes.

```
<CFDUMP var=#Request.Mirror#>
```

You may want to create separate structures for user-entered session data and the variables the controller uses to keep track of state. There are cases where you might want to delete all user data, but no state data.

Beware nested structure syntax!

```
Request["var"]["prop"] ≠ Request["var.prop"]
```

A Word about Metadata Mapping

Although it's not necessarily related to our discussion of MVC, the metadata mapping pattern is an extremely useful complement to MVC architecture. Metadata is just data about data. In a metadata-driven application, you put information in a database about all the objects, properties, tables, and fields in your application and how they are related. You can then create much of your MVC model code dynamically rather than writing SQL queries to insert, update, and delete records for each table. Further, you can create only a handful of standard view templates corresponding to the fundamental database operations. Like model code, each view can then be constructed dynamically from the view templates and the metadata.

Metadata-driven architecture presents many advantages:

- Code becomes less much less error-prone. When you change your data model and the corresponding metadata, all corresponding model code and views are dynamically updated to reflect it.
- It becomes much easier to identify and organize custom code where necessary.
- Standard data validation methods can be applied automatically to every from parameter based on the data type specified in the metadata. This is much more secure than relying on hand coding everywhere.

Where to Go from Here

Books

Martin Fowler's *Patterns of Enterprise Application Architecture* (Addison-Wesley) has excellent discussion of the MVC and Application Controller patterns as well as many others.