



founder & ceo

Fuat Kircaali fuat@sys-con.com

group publisher

Jeremy Geelan jeremy@sys-con.com

advertising

senior vp, sales & marketing

Carmen Gonzalez carmen@sys-con.com

vp, sales & marketing

Miles Silverman miles@sys-con.com

advertising sales director

Megan Mussa megan@sys-con.com

advertising sales manager

Andrew Peralta andrew@sys-con.com

associate sales manager

Corinna Melcon corinna@sys-con.com
Lauren Orsi lauren@sys-con.com

sys-con events

president, events

Carmen Gonzalez carmen@sys-con.com

events manager

Lauren Orsi lauren@sys-con.com

customer relations

circulation service coordinator

Edna Earle Russell edna@sys-con.com

sys-con.com

vp, information systems

Robert Diamond robert@sys-con.com

web designers

Stephen Kilmurray stephen@sys-con.com
Richard Walter richard@sys-con.com

accounting

financial analyst

Joan LaRose joan@sys-con.com

accounts payable

Betty White betty@sys-con.com

subscriptions

Subscribe@sys-con.com

Call 1-888-303-5282

For subscriptions and requests for bulk orders, please send your letters to Subscription Department

Cover Price: \$8.99/issue

Domestic: \$89.99/yr (12 issues)

Canada/Mexico: \$99.99/yr

All other countries \$129.99/yr

(U.S. Banks or Money Orders)

Back issues: \$12 U.S. \$15 all others

Data Access Objects

The mystery design pattern



By Jeffrey Houser

It seems that there's a lot of talk in the ColdFusion community about data access objects and data gateway

design patterns. Everyone talks about how great they are and why everyone should be using these patterns.

Unfortunately, there seems to be little talk about what exactly they are and how you would implement them in a ColdFusion application. At one point I did a lot of searching for resources on these two patterns and came up with nothing. These patterns aren't discussed in the famous "Gang of Four" book, nor are they covered in Head First Design Patterns. A Web search also came up empty. So where does one go to learn? Hopefully I can shed some light on these patterns. It was only after talking to different people about them that I started to understand their purpose and how you could implement them. In this article I'll explore the Data Access Object (DAO) in detail. Next month I'll examine the gateway pattern.

What Is a Design Pattern?

A design pattern is just another name for a best practice. There are usually certain tradeoffs you make for choosing one approach over another. I recommend you try to learn as many patterns as you can, and then you'll be in a better position to decide what will work best for your current situation.

When talking about design patterns, most ColdFusion developers are usually talking about ways to structure the code that makes

their business model. Design patterns in the model are a way to structure code in such a way that allows the most flexibility long-term. Software applications spend most of their life in a maintenance change as I'm sure you've already experienced.

Design patterns aren't limited to your business model, though. Model-View-Controller is a design pattern many CF developers are familiar with that's not related to how to structure your model; it's designed to help separate your model from your view code. Yahoo released a series of design patterns that are related to the user interface. You can read about them at <http://developer.yahoo.com/yypatterns/index.php>. When you implement DAOs or gateway objects you'll be doing so in your model.

What Is a Data Access Object?

Data Access Objects are a design pattern that lets you separate your data access from any business logic. This lets you easily change your data storage mechanism with minimal code changes. Perhaps you want to move from a MySQL database to a SQL Server database. Or perhaps you're ditching local data storage in favor of using a Web Service mechanism? Maybe you want to move some data out of the database and into XML files? Or perhaps you'll want to move them out of XML files into a database. If your application is implemented using DAOs then you'll just have to write a new DAO objects for the new data mechanism and then tell your application to use the new one instead of the old one.

In most cases, a DAO will have four methods in it, one for inserting data, one for updating data, one for selecting data, and one for deleting data. Your business model components will access the DAO objects as needed to maintain the data properly. This probably sounds harder than it is, so let me demonstrate with an example.

MyFriends RSSCategory Component

To illustrate a DAO object, I want to take a



look at the `RSSCategory.cfc` from the MyFriends RSS aggregator. You can download the aggregator code from the software pod on my Web site at www.jeffryhouser.com. RSS feeds that are entered into the system can be categorized. This component is used for creating or updating one of those categories. It was originally built in the interest of speed without much thought to database portability.

The `RSSCategory` component has two properties, a `CategoryId` and a `Category`. It has two methods, an `init` method and a `commit` method. The `init` method contains a `select` statement. The `commit` method contains an `update` statement and an `insert` statement. If we change our data storage mechanism, all the SQL statements would have to be changed inside the component. This could, potentially, mean changing every component in our Model. This is about as close to a full rewrite as you can get. The intent in creating a DAO is to move the SQL statements outside of the main component and into a separate component. That way, you only have to change the DAO, leaving the rest of your app untouched.

Writing the Data Access Object

The Data Access Object won't have any properties, just the methods for selecting, inserting, updating, and deleting. (For the sake of brevity, I won't provide the delete method.). A CFC without any local properties is like a function library, and that is exactly what we're using it for.

Listing 1 shows the `select` method. The method name is `select` and it returns a query. It accepts two arguments, the `CategoryId` and the `datasource` name. The `CategoryId` is the primary key for the category you want to retrieve. The query name is defined as a local variable then the `select` query is run. I chose `RSSCategory.cfc` because of the simplicity of the queries. The resultant query, even if no data is returned, is passed out of the function. Easy as pie.

Listing 2 shows the `insert` method. This method is named `insert`, but returns `void`. It accepts three arguments, the `CategoryId`, the `Category`, and the `datasource`. This is different than the `select` method, which only needs the primary key. Since we're creating a new category from scratch, we need all the associated data for the query. The query variable is defined as a local variable on the next line. Then comes the query. In the case of the MyFriends project, UUIDs are used as primary keys so they'll be created outside of the component and passed in. If you were using other methods for primary key creation, such as an auto-incrementing integer, then you may not want to pass that value into the method. You may want to get it from the database after the insert and return it out of the method. These decisions are application-specific decisions, though, not data storage-specific. The update method, shown in Listing 3 is identical in form to the insert method and I don't have anything else to add about the code.

Modifying the RSSCategory Component

With the Data Access Object created, the next step is to modify the `RSSCategory` component to use the methods in the DAO instead of directly executing SQL. First, we want to create an instance variable to contain the DAO object. We can add this line

as part of the pseudo constructor code:

```
variables.instance.DAO = "";
```

The modified `init` method is shown in Listing 4. It adds a third argument, which is the type of DAO you want to use. Most likely this will be a global setting somewhere in your app. So I don't have to change any already-written code, I made this argument optional and added a default value, which refers to the Data Access Object I just created. First I define a local query variable. Then the code creates an instance of the DAO object. In the old version of the component, the query was located next. Here we call the `select` method and assign the results to the query variable. The remainder of the method sets the instance variables based on the query data.


The modified `commit` method is shown in Listing 5. The `commit` method remains largely unchanged. The method signature is the same. There is one argument for the data source. A local query variable is defined. If the primary key is an empty string then the new primary key is created and the `insert` method executed. Otherwise, the `update` method is run.

Why Use a Data Access Object?

With the example behind you, you might ask yourself why would you care about DAOs? I can honestly say that I rarely use them. Most of my applications are custom built for clients. How often do companies change their database? Yes, it happens, but it's rare. In my early days I converted a lot of Microsoft Access databases to SQL Server, but it's rare to find an Access-built app. It'd be even rarer to see an Access-built Web app that uses DAOs or any advanced design concepts.

DAOs really start being a benefit if you're going to build an application that will be deployed in multiple environments and those environments are unknown. Perhaps you're building a killer CMS? Or free blogging software? If so then you're going to want to use DAOs in your development. You don't know if the next deployment will be on SQL Server, Oracle, or even XML files.

Final Thoughts

You should now have an understanding of what DAOs are and how to use them in your ColdFusion applications. I wish I could give you a list of resources to learn about gateways, but my research has always come up empty. I can think of many alternate implementations of this pattern that could achieve the same affect. I'd love to hear how you implement DAOs. Let me know! 

About the Author

Jeffrey Houser has been working with computers for over 20 years and in Web development for over 8 years. He owns a consulting company and has authored three separate books on ColdFusion, most recently ColdFusion MX: The Complete Reference (McGraw-Hill Osborne Media).

jeff@instantcoldfusion.com

Listing 1 The select method

```
<cffunction name="Select" access="public" returnType="Query">
<cfargument name="CategoryID" type="string" required="true">
<cfargument name="dsn" type="string" required="true">

<cfset var qGetCat = "">

<cfquery name="qGetCat" datasource="#arguments.dsn#">
select *
from RSSCategories
where RSSCategories.CategoryID =
<cfqueryparam value="#arguments.CategoryID#" cfsqltype="cf_sql_var-
char">
</cfquery>

<cfreturn qGetCat>
</cffunction>
```

Listing 2 The insert method

```
<cffunction name="Insert" access="public" returnType="Void">
<cfargument name="CategoryID" type="string" required="true">
<cfargument name="Category" type="string" required="true">
<cfargument name="dsn" type="string" required="true">

<cfset var qUpdateCat = "">

<cfquery name="qUpdateCat" datasource="#arguments.dsn#">
insert into RSSCategories(CategoryID, Category)
values (
<cfqueryparam value="#arguments.CategoryID#" cfsqltype="cf_sql_var-
char">,
<cfqueryparam value="#arguments.Category#" cfsqltype="cf_sql_var-
char"> )
</cfquery>

</cffunction>
```

Listing 3 The update method

```
<cffunction name="Update" access="public" returnType="void">
<cfargument name="CategoryID" type="string" required="true">
<cfargument name="Category" type="string" required="true">
<cfargument name="dsn" type="string" required="true">

<cfset var qUpdateCat = "">

<cfquery name="qUpdateCat" datasource="#arguments.dsn#">
update RSSCategories
set Category =
<cfqueryparam value="#arguments.Category#" cfsqltype="cf_sql_
varchar">
where CategoryID =
<cfqueryparam value="#arguments.CategoryID#" cfsqltype="cf_
sql_varchar">
</cfquery>

</cffunction>
```

Listing 4 The RSSCategory init method

```
<cffunction name="Init" access="public" returnType="Boolean">
<cfargument name="CategoryID" type="string" required="true">
<cfargument name="dsn" type="string" required="true">
<cfargument name="DAOType" type="String" required="false"
default="RSSCategory_DAO_SQLServer">

<cfset var qGetCat = "">

<cfset variables.instance.DAO = createobject('component', '#arguments.
DAOType#')>

<cfset qGetCat = variables.instance.DAO.select(arguments.CategoryID ,
arguments.dsn)>

<cff qGetCat.recordcount is 0>
<cfreturn false>
</cff>

<cfscript>
variables.instance.CategoryID = qGetCat.CategoryID;
variables.instance.Category = qGetCat.Category;
</cfscript>

<cfreturn true>

</cffunction>
```

Listing 5 The RSSCategory commit method

```
<cffunction name="Commit" access="public" returnType="void">
<cfargument name="dsn" type="string" required="true">

<cfset var qUpdateCat = "">

<cff variables.instance.CategoryID is "">
<cfset variables.instance.CategoryID = createuuid()>
<cfset variables.instance.DAO.insert(variables.instance.CategoryID,
variables.instance.Category,
arguments.dsn)>

<cfelse>
<cfset variables.instance.DAO.update(variables.instance.CategoryID,
variables.instance.Category,
arguments.dsn)>
</cff>

</cffunction>
```

Download the Code...
Go to <http://coldfusion.sys-con.com>