Radder Rad With Cfquery and Cut and Paste

Posted At : January 23, 2006 11:33 PM | Posted By : Mark Kruger
Related Categories: Podcasts, Coldfusion & Databases, Coldfusion Tips and Techniques

When I first heard of RAD my immediate thought was the wonderful folks of Virginia and the Cumberland Gap - where I met my wife (a nurse from Minnesota, what are the odds). In the blue green mountains of Appalachia, everyone knows about Rad. It's the opposite of Blue. If you mix a little yeller into it you get arnge. When I started studying IT and Technology. It didn't take me long to learn that RAD stood for "Rapid Application Development". Now if you've been using Coldfusion for any length of time you will know that "RAD" is a word often used in to describe the usability and accessibility of the language. Here one reason why....

Listen Here

Copy and Pasteability

Rather than force a user to wrangle with string concatenation, Coldfusion leverages it's status as a "tag-based" language to make it *conceptually* easier. In other words, I have an easier time picking out certain things of note and *seeing them apart from the code*. Nowhere is this more evident than when working with queries. Using CF your can cut and paste human readable queries with less grunt work than virtually any other language. Let's say you have the following query that you have worked out in your query analyzer:

```
SELECT   *
FROM   Users
WHERE   username = 'sally'
AND      password = 'theCamel'
```

If I where to move this query to ASP and replace the 2 "where" clause items with variables I would end up with this:

```
<%
    conn = server.CreateObject("ADODB.Connection")
   conn.Open "*string with username and password*"
   sql = "select * from users where " & _
       "username = '" & request("username") & "' AND Password='" & _
       request("password") & "'"
   qry    = conn.execute(sql)
   %>
```

Note that the end result bears little resemblance my initial query (so nicely formatted in Query analyzer). It's possible that I cut and pasted from Query Analyzer to begin with, but I had to seriously mangle the string to add the concatenation. You see, in ASP (and most other languages) you must concatenate a long string, taking care with the single quotes, and pass it along to a database object. The more difficult the query, the longer the string, the more confusing it becomes.

Now here's the same code in Coldfusion:

```
<cfquery name="qry" datasource="#mydsn#">
      select * from users
      where   username = '#form.username#'
      and      password = '#form.password#'
   </cfquery>
```

It should be noted that under the hood Coldfusion is parsing through the string that is held between the tags and putting it into an SQL Prepare statement and executing it using JDBC. So there is little *functional* difference here. Both snippets return a result set. But the first snippet simply take longer to write and it will be *much* more difficult to change and maintain.

This problem is even more pronounced when conditions are introduced into the string. Let's say you have a search form with 2 free text boxes - one for "search string" and the other for "type". You want to always search against the first box, but you want the second box to be "optional". That is, you only want to include it in the search if someone actually enters something in the second box. In Coldfusion:

```
<cfquery name="qry" datasource="#mydsn#">
      select * from news
      WHERE title LIKE '%#form.searchString#%'
      <!--- check the type --->
      <cfif len(trim(form.type))>
         AND    type LIKE '%form.type#%'
      </cfif>
   </cfquery>
```

So the conditional goes right into the query - easy. How about ASP?

```
<%
    conn = server.CreateObject("ADODB.Connection")
   conn.Open "*string with username and password*"
   sql = "select * from news where " & _
       "title LIKE = '%" & request("searchString") & "%'
   if request("type") <> "" then
      sql = sql & " Type LIKE '%" & request("type") & "%' "
   end if

   qry    = conn.execute(sql)
   %>
```

In my view this is just harder to read, and harder to copy back into QA and re-work. ASP is just one example. We could easily produce examples in just about every language. This easy encapsulation of Database code is one of the most precious and valued aspects of Coldfusion - and why it retains its popularity among programmers creating data management utilities on the intranet.

In fact, this "readability" results in a tremendous advantage for database code - namely, that it *need not be developed "within" the Coldfusion script*. I typically develop database code in Query Analyzer (when working with MS SQL). The code is immediately portable to Coldfusion. I don't have to parse it into a long string or encapsulate it into a stored procedure (unless other requirements demand it). Usually, all I have to do is "variableize" or "conditionalize" the where clause using CFQueryparam (and a handy snippet in Homesite or CF Eclipse or DW). I can copy and paste *from* my Coldfusion code into my analyzer and retain most of it's form - allowing me to run it and change it with little editing. I can then copy *back* to Coldfusion from query analyzer and re-variableize my where clause.

Coldfusion Users Know More SQL

I used to think that Coldfusion made database work *too easy* - resulting in CF coders who were not knowledgeable enough about SQL. I still believe that learning advance SQL is the *single most important thing* you can do to increase your skill set if you are already an intermediate Coldfusion programmer. But now that I've worked with a lot of *other* applications written in *other* languages I've come to the conclusion that Coldfusion greatly assists the CF coder become a better SQL programmer by making SQL as accessible as CFML itself. I think that CF coders can write *better* SQL code because they can focus on SQL as the locus of their efforts - rather than on the syntax of the parent coding language. Coldfusion programmers don't have to struggle over UNIONs and JOINs like the ASP or PHP programmer because the CFQUERY tag functions as a *visual unifier* - a bridge between the query tool and the web script.