Adding Cfqueryparams to a Legacy Site Without Losing Your Hair

Posted At : July 26, 2008 2:28 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Security, Coldfusion Tips and Techniques

So you got hit with the latest SQLi attack eh? SQLi is the hip acronym for "sql injection" that fancy pants security people use. You've put in some stop gap measures and now you are slogging through 3000 queries trying to add cfqueryparam to everything. It's a laborious task to be sure. Here are some special tips from the muse that might help shorten it.

Identifying Vulnerable Queries

First, there are a couple of excellent code scanners released that can identify vulnerable queries in your CF Code. They can serve as an excellent starting point. I won't detail them here. Rather, I will point you to Brad Wood's oustanding blog, **Coders Revolution** where he has some reviews and tips on using them. One of these tools created by David Banttari (a **Webapper** guru) actually takes a stab at fixing queries for you. While I wouldn't use it on live code, it might save some time and effort. I suspect that you will need to spend time scanning, fixing, testing and rescanning no matter what you use. Is there anything that can help with the task of just slogging through the queries, editing and testing? I have a couple of thoughts.

Snippets

First, and I hate to say this dear muse readers because I have a high regard for most of you and I appreciate what you do, but one of the reasons developers don't use Cfqueryparam is that they are just a bit lazy. It's a lengthy tag with some long attributes and even using autofill it seems like a bother. My trick (a simply one to be sure) is to keep 3 snippets around. You may know this, but snippets are handy little bits of code you can use in CF Eclipse or Homesite. I especially like homesite for a task like this. Here's a link to my **snippets** for Cfqueryparam if you care to try them. Just put them in your %homesite%/userData/snippets directory and then assign a short cut key to each of them. I have Ctrl+1,2,5 assign to CF_SQL_INTEGER, CF_SQL_CHAR and CF_SQL_DECIMAL. My snippets are actually kind of unique. Each of them includes the tag, the type, and value=" with no closing tag. Like so:

```
<cfqueryparam cfsqltype="CF_SQL_INTEGER" value="
<cfqueryparam cfsqltype="CF_SQL_CHAR" value="
<cfqueryparam cfsqltype="CF_SQL_DECIMAL" scale="2" value="
```

Why? I simply find it easier to use when typing. I can add them inline as I write queries or position my cursor after the fact to the left of a variable and hit control 1, 2 or 5 to insert the code - then polish it off. You might also note that I'm using a decimal="2" in the decimal snippet. I find this to be the most common input for decimal types.

Figuring Out Your Data Types

This might be another spot where developers drop the ball when it comes to Cfqueryparam. It's frustrating to have to "look up" the data type all the time to figure out what cfsqltype to use in the tag. I have 3 tips that might help.

First, you probably already know that about 80 to 90 percent of the variables in *most* applications are either some sort of character type (Char, Varchar, Text, nText, nVarchar etc.) or an Int type (int, bigint, bit). For each of these types the CF_SQL_CHAR type and the CF_SQL_INTEGER type is going to work fine. So that takes care of a good deal of decision making.

In addition, the CF_SQL_DECIMAL type can cover for most floating point or decimal related types like decimal, money, float and real. The database and driver implicitly convert the type to it's proper form for storage (and this process is *usually* more about how many bytes to use for storage than it is about accuracy unless you are developing a scientific or statistical analysis application).

Finally, the CF_SQL_CHAR type is useful in another way. You can pass a properly formatted date string using CF_SQL_CHAR to a smalldatetime column. For example:

```
<Cfquery ....>
    UPDATE    birthdays
    SET        myBirthday = <cfqueryparam cfsqltype="CF_SQL_CHAR" value="3/31/1965"/>
    WHERE    myName = <cfqueryparam cfsqltype="CF_SQL_CHAR" value="Mark"/>
</cfquery>
```

Other strings work as well including yyyy-mm-dd. If you want to include time you can do that as well as in "yyyy-mm-dd 12:25:00". MSSQL will convert each of these formats into datetime and store them properly.

What About Retro-Fitting and Finding Data Types

My final tip has to do with trying to figure out which queries need which data types. Take this query for example.

```
<Cfquery ....>
    UPDATE    Products
    SET        cost = #prodcost#
    AND        inStock = #instock#
    WHERE    prodID = #prodid#
</cfquery>
```

If I'm not careful I might end up converting this to something like this:

```
<Cfquery ....>
    UPDATE    Products
    SET        cost = <cfqueryparam cfsqltype="CF_SQL_INTEGER" value="#prodcost#"/>
    AND        qtyOnhand = <cfqueryparam cfsqltype="CF_SQL_INTEGER" value="#instock#"/>
    WHERE    prodID = <cfqueryparam cfsqltype="CF_SQL_INTEGER" value="#prodid#"/>
</cfquery>
```

Now the use of the integer type is *probably* right for prodid and it *may* be right for qtyOnHand (although this could be a percentage or something). But if I *accidentally* alter the query in this way then I'm going to have a ticklish bug on my hand. Why? Because of implicit conversion this query will *succeed!*. The JDBC driver will take your numeric value, and create an INT from it (meaning your cost of 9.95 just became "9"). It will pass a value of INT to the DB. The DB will look up the column and decide it's a decimal (18,2). It will add 2 decimal places (.00) and you will end up inserting 9.00 into the DB. This can result in some very hard to find bugs - especially when you are bleary eyed after a weekend of typing in C F Q U E R Y P.....

Never fear - I have a special code snippet that can help. This code will lookup every tablename/column that is of a type decimal, float, money, numeric, real, and small

money in your data base. Run this code in query analyzer. Even a very complex DB will probably have a fairly small set of results - 20 to 50 columns. Then use that information to further scan your code for anomalies like the one described above.

```
select a.name as tbl,b.name as col, 'decimal' AS dType

from sysobjects a,syscolumns b

where a.id=b.id and a.xtype='u'

AND    b.xtype = 106

UNION

select a.name as tbl,b.name as col, 'float' AS dType

from sysobjects a,syscolumns b

where a.id=b.id and a.xtype='u'

AND    b.xtype = 62

UNION

select a.name as tbl,b.name as col, 'money' AS dType

from sysobjects a,syscolumns b

where a.id=b.id and a.xtype='u'

AND    b.xtype = 60

UNION

select a.name as tbl,b.name as col, 'numeric' AS dType

from sysobjects a,syscolumns b

where a.id=b.id and a.xtype='u'

AND    b.xtype = 108

UNION

select a.name as tbl,b.name as col, 'Real' AS dType

from sysobjects a,syscolumns b

where a.id=b.id and a.xtype='u'
```

```
AND    b.xtype = 59

UNION

select a.name as tbl,b.name as col, 'small money' AS dType

from sysobjects a,syscolumns b

where a.id=b.id and a.xtype='u'

AND    b.xtype = 122
```

Final Cavaets

Finally, muse readers, keep in mind these are shortcuts and tips. They are not designed to keep you from coding or testing. You will need to find a way to test every query you change to make sure it is still doing what you believe it should be doing.