Getting DB Meta Data From Access

Posted At : August 26, 2005 11:27 AM | Posted By : Mark Kruger
Related Categories: Coldfusion & Databases

Sometimes it's useful to be able to query a database for information *about itself*. For example, I have a survey application that allows a user to send a survey to a "population" of users. The user has multiple SQL database containing multiple possible recipients. He also receives ad-hoc leads from marketing that are temporarily "dumped" into his database. In other words, he knows the data is in there but he doesn't know the table name and he doesn't know the columns inside the table from which to draw the name and email. In MS SQL you can select from the "sys" tables or us the stored procedures "sp_tables" and "sp_columns" to get a list of all the tables, columns and data types. Wouldn't it be nice to do the same in access? After all, access is what often used as portable transport for transferring leads around - right?

It turns out that Access has system tables too. The only problem is that Access system tables are not accessible by default. You have to have *read rights* on the tables to make them available to your CFQUERY. Before you can even do that they have to be made *visible*. That means you will need to open the database *in access* before you place it on the server. Here are the steps:

- Open the database
- Go to tools-->Options
- Check the box under *show* that says *system Objects*. You will notice that several new tables appear in table view with the prefix MSys - MSysAccessObjects, MSysAces, MSysObjects, MSysQueries, MSysRelationships, and there may be a few others depending on your version of Access.
- Now Go to Tools-->Security-->User and Group Permissions
- Select the MSysObjects table and check the box that says "Read Data"

You are now ready to copy the file back to the server and access it using CFQUERY.

Querying MSysObjects

Ok, so I can access the system table. What's in it? Well frankly there's a lot of stuff that won't matter to you. There's an ID, a creation date and an update date, a foreign Name, some columns with binary OLE object data and some other unintelligible items. The two we will use in our example are *name* and *type*. Try this query using the "cfdocexamples" data source that ships with a default installation of MX 7.

```
<Cfquery name="getSysData" datasource="cfdocexamples">
       SELECT name, type FROM MSysObjects
    </CFQUERY>

   <cfdump var="#getSysData#">
```

Here are the **results** of that query. You will notice that everything that has a type of "1" seems to be a table. So let's alter our query to be:

```
<Cfquery name="getSysData" datasource="cfdocexamples">
   SELECT name, type FROM MSysObjects
   WHERE type = 1
</CFQUERY>
```

So far so good. Now we just need to filter out the system tables:

```
<Cfquery name="getSysData" datasource="cfdocexamples">
   SELECT    name, type
   FROM    MSysObjects
   WHERE    type = 1
   AND      name NOT LIKE 'MSys%'
</CFQUERY>
```

Ok, now we have a list of 27 tables - that's exactly the number we expected.

Getting Column Types

Having a list of tables is great, but remember my original plan was to allow my user to build his own populations of leads. That means I have to allow him to select from the tables as well - and *that* means I have to know about the columns in the tables *and* their types. In MX 7 this is easy with the *getMetaData( )* function. Try the following code against your cfdocexamples database.

```
<Cfquery name="getSysData" datasource="cfdocexamples">
   SELECT    id,name, type
   FROM    MSysObjects
   WHERE    type = 1
   AND      name NOT LIKE 'MSys%'
</CFQUERY>

<cfloop query="getSysData">
    <Cfquery name="getTableStuff" datasource="cfdocexamples">
      SELECT   TOP 1 *
      FROM   #NAME#
   </CFQUERY>

   <cfset md = getMetaData(getTableStuff)>

   <Cfdump var="#md#" label="#Name#">
   <br>
</cfloop>
```

Each of the metadata variables ("md" in the example) contains an array of structures. The structure contains 3 keys - "IsCaseSensitive","Name" (the name of the column) and "type" (varchar, INT etc.). Here's the table called *tblCartItems* from the cfdocexamples db.

tblCartItems - array

1

tblCartItems - struct

IsCaseSensitive	NO
Name	ItemIDPK
TypeName	VARCHAR

2

tblCartItems - struct

IsCaseSensitive	NO
Name	CartIDPK
TypeName	VARCHAR

3

tblCartItems - struct

IsCaseSensitive	NO
Name	Quantity
TypeName	INT

Using these arrays you could "figure out" how to select against a particular column. It's certainly not as easy or accessible as MS SQL or other production quality DB servers, but something like the task described above it works pretty well.