Data Migration and Coldfusion

Posted At : November 18, 2005 12:02 PM | Posted By : Mark Kruger
Related Categories: Coldfusion & Databases, Coldfusion Tips and Techniques

If you've been working with Coldfusion very long, chances are you've written a data import script. There are many tools that allow you to migrate data from one database platform or schema to another, and I'm well aware that "guru dogma" states that Coldfusion is *not the best tool* for things like long running tasks that *can* be performed by the database. I'm also a big advocate for letting the database do its job. So it may surprise you to learn that I believe Coldfusion *is actually a pretty good choice* in many cases - especially if you have to do anything tricky with the data. Take looping for example:

In T-SQL, if you want to loop through a record set and apply logic to the data you will need a cursor. Cursor code isn't hard to write, but it will definitely take you longer and require a bit more skill - especially debugging. Take this example from books on line....

```
DECLARE authors_cursor CURSOR FOR
SELECT au_lname, au_fname FROM authors
WHERE au_lname LIKE "B%"
ORDER BY au_lname, au_fname

OPEN authors_cursor

-- Perform the first fetch and store the values in variables.
-- Note: The variables are in the same order as the columns
-- in the SELECT statement.

FETCH NEXT FROM authors_cursor
INTO @au_lname, @au_fname

-- Check @@FETCH_STATUS to see if there are any more rows to fetch.
WHILE @@FETCH_STATUS = 0
BEGIN

-- Concatenate and display the current values in the variables.
PRINT "Author: " + @au_fname + " " + @au_lname

-- This is executed as long as the previous fetch succeeds.
FETCH NEXT FROM authors_cursor
INTO @au_lname, @au_fname
END

CLOSE authors_cursor
DEALLOCATE authors_cursor
```

This code works by declaring a query as a cursor, looping through each row and loading local variables @au_lname and @au_fname with values from the row with each iteration. It takes a good bit of cryptic code to do what Coldfusion can do like this:

```
<cfquery name="authors" datasource="pubs">
   SELECT au_lname, au_fname FROM authors
   WHERE au_lname LIKE "B%"
   ORDER BY au_lname, au_fname
```

```
</cfquery>

<cfoutput query="authors">
   Author: #au_fname# #au_lname# <br/>
</cfoutput>
```

Even a novice could see what is happening here.

Sometimes data import tasks are very challenging and require a good deal of data manipulation. Yes, you can write very complex SQL to do such things, but you can do it faster in Coldfusion. If you read my blog you know I'm an advocate for letting the Database do what it does best. But I'm also an advocate for being cost effective. If you are doing a data migration as part of a deployment, then there will be times when Coldfusion will allow you to quickly import records that may have taken a much greater effort in SQL. For example, you may be tasked with merging 2 databases and eliminating the duplicate data between them. You might write code like this.

```
<!--- get current data --->
<Cfquery name="qryAll" datasource="#myDsn#">
   SELECT username
   FROM    oldTable
</CFQUERY>
<!--- loop through the data --->
<cfloop query="qryAll">
   <!--- check to see if it exists --->
   <Cfquery name="check" datasource="#myDsn#">
      SELECT   username
      FROM   newTable
      WHERE   username = '#username#'
   </CFQUERY>
   <!--- if notput it in the db --->
   <Cfif NOT check.recordcount>

   <Cfquery name="put" datasource="#myDsn#">
      insert into newtable
         (username)
      VALUES
         ('#username#')
   </CFQUERY>
   </CFIF>

</cfloop>
```

In this simple example we are checking to see if a username from "oldtable" exists in "newtable" and if it does not, we are inserting it. Obviously, typical migration code is vastly more complicated. In this case we could eliminate the "check" query using SQL like this (which I prefer):

```
<Cfquery name="put" datasource="#myDsn#">
      IF NOT EXISTS
         (select username
          from newtable
          where username = '#username#')
      BEGIN
         insert into newtable
            (username)
         VALUES
            ('#username#')
```

```
      END
   </CFQUERY>
```

That would save us a step and a hit on the db.

When you do this sort of thing you have an issue you may not have thought about. When using Coldfusion Each *connection* to the db is an implicit commit. If you were doing it in T-SQL then each *block* would be an implicit commit. If, during SQL cursor code, an error is thrown, the transaction is rolled back - as if the process had never begun. This is not the case in your Coldfusion code however. That leaves you with a possibility that, in the case of an error, your data would be half imported to "newtable".

Using Cftransaction

This is where cftransaction comes to the rescue. You can treat the loop code as if it were a block of SQL - even though it may contain things *other than* cfquery. An exception will cause a roll-back of all the DB calls that have been made so far. Using cftransaction is simple.

```
<Cftransaction action="BEGIN">

<cfloop query="qryAll">
   <Cfquery name="put" datasource="#myDsn#">
      IF NOT EXISTS
         (select username
          from newtable
          where username = '#username#')
      BEGIN
         insert into newtable
            (username)
         VALUES
            ('#username#')
      END
   </CFQUERY>
</cfloop>
</Cftransaction>
```

Note that I'm not using a isolation level and I'm not explicitly telling the db to commit. The isolation level is probably not an issue if you are doing a pre-deployment data migration script. The "commit" is implied by the end transaction tag. You can get quite granular in your control of specific commits and roll backs if you like, but in this example we are saying simply, "if the process errors out before it completes then kill the whole thing". It's all or nothing.

Now you may be wondering about the accuracy of your duplicate record check. If I insert a username "BOB" inside of my cftransaction, then 10 rows later I try and insert "BOB" again, will the database "see" the first bob? After all, it's not committed yet - right? The answer is yes. The database will see bob number 1 and not insert bob number 2. Event though the transaction is not yet committed, the db can read from the new pages because it is "inside" the transaction. Your checks will work as you expect.