# Why You should worry about your execution plan

Posted At : June 28, 2005 11:15 AM | Posted By : Mark Kruger
Related Categories: SQL tips, MS SQL Server, Coldfusion & Databases

No I'm not talking about dead man walking. I'm talking about your *database execution plan*. I want to give fair warning to all of you Microsoft haters out there (and you know who you are) that I'm going to use lingo from Microsoft SQL Server 2000. It's a ubiquitous and full-featured database with good documentation regarding this subject. Much of what is said here applies to other databases as well (no doubt using different lingo). So please, feel free to post comments of how Oracle or MySQL or PostgreSQL or Interbase or your flat file - all have a great way of doing this. But please *don't* post about how your favorite DB is so great and Microsoft is the spawn of Satan. That's not helpful and it's makes me want to poke out my eye with an ice pick! Whew! Now that that's out of the way.

The database execution plan is the series of steps that a database takes to deliver a particular query or task. These steps are cached on the DB server. When you run a query it looks in the cache for a plan that matches. If it finds one, it uses it. If not, it creates a new one. Why is this important? Because the more often your DB Server finds a matching plan in the cache, the better it performs. In fact, it can run *significantly faster* when it is not tasked with constantly building execution plans from scratch. Here's the rub, much of the query code written in Coldfusion requires the RDBMS to compile a new execution plan. Here's why.

### The Execution Context

When you run a query the database determines the *execution context*. The execution context is used to hold the data necessary to run the query. Think of it as an array of parameters. Consider this example.

```
<cfquery name="get" datasource="#datadsn#">
   SELECT * FROM users
   WHERE    username = 'BOB'
   AND    password = 'BOBSPASSWORD'
   AND      active = 1
</cfquery>
```

The execution context here is:

```
paramater 1 = Bob
parameter 2 = BobsPassword
parameter 3 = 1
```

You can see there are really 2 parts to the query that the DB needs to run the query. There is the context - the parameters to use in the query, and there is the execution code. This is comprised of the keywords and syntax. The DB server parses the execution code, inserting the context variables in the right spot, and then builds a query plan.

### The query plan

The query plan is the steps necessary to run the query. In it's simplest form:

```
<cfquery name="get" datasource="#datadsn#">
   SELECT * FROM users
   WHERE    username = [context param 1]
   AND    password = [context param 2]
   AND      active     = [context param 3]
</cfquery>
```

To use the plan, the server only needs the context - the rest is already pre-compiled. So, if could pass in an array of three parameters with a query that matches this one I could get the plan to match and hit the cache. Cool!

### Coldfusion Gotcha

You probably knew there was a catch didn't you. You see there is one more piece of data that the DB server needs to make a hit on the cache. It needs to know the "type" of the data in the context. Why? For 2 reasons. 1) Databases support conversion. For example, if you pass in 2.43 to a field that holds an INT you will end up with 2. The DB will know what to do with it. If you search "username = 1" (no single quotes) - the db may know enough to treat it as a string. 2) Databases have to "sort out" the list of key words from the list of parameters. If you do not specify a "type" for a parameter you force the DB to parse through the whole string and sort out the execution code from the context.

Let me illustrate it with some code. If you write a query like the one above, here's what the SQL server will see.

```
<cfquery name="get" datasource="#datadsn#">
   SELECT * FROM users
   WHERE    username = [some unknown type of variable]
   AND    password = [some unknown type of variable]
   AND      active     = [some unknown type of variable]
</cfquery>
```

The DB server will then have to take what it does know (the column names) and "look up" the type of the column. So it would first look and see that username and password are both strings, and treat parameters 1 and 2 as strings. Then it would look up the column "active" and examine parameter 4 to "see" if it tested correctly as an INT. Then it would compile an execution plan based on what it had discovered. You are not "Telling" the DB enough to be efficient.

### The Solution - CFQUERYPARAM

The solution is CFQUERYPARAM. With it you provide a "typed execution context". You tell the DB enough to look up the plan straightway. It doesn't even need to look for the column name types and match them with the context. Why? Because if it finds a cached execution plan it means that this query has been run successfully before, and the bindings ensure that the context data is typed correctly. It has a measure of guarantee that the plan will succeed. So, using the same query:

```
<cfquery name="get" datasource="#datadsn#">
   SELECT * FROM users
   WHERE    username = <cfqueryparam cfsqltype="CF_SQL_CHAR" value="BOB">
   AND    password = <cfqueryparam cfsqltype="CF_SQL_CHAR" value="BOBSPASSWORD">
   AND      active = <cfqueryparam cfsqltype="CF_SQL_INTEGER" value="1">
</cfquery>
```

we "tell" the DB server enough that it can skip a few steps and hit the cached plan. Here's what the server sees.

```
<cfquery name="get" datasource="#datadsn#">
   SELECT * FROM users
   WHERE    username = [a String variable]
   AND    password = [a String variable]
   AND      active     = [an INT variable]
</cfquery>
```

There are other reasons to use data binding (SQL Injection attack prevention is chief among them), but the performance benefit can be quite dramatic. I have seen as much as a 40% decrease in query execution time - just by adding cfqueryparam. Here's another tip, add the shell for CF_SQL_CHAR and CF_SQL_INTEGER to your IDE as a short cut (snippet). It will save you a lot of typing. These are by far the 2 most common types.

One more tip. On MS SQL server, fully qualifying the objects in the query can further increase your chances of hitting the cache. So "SELECT * FROM dbo.users" has a better chance than just "SELECT * FROM users". SQL 2000 is far superior to SQL 7 in this regard - but even it can benefit from fully qualified objects.