Good Developers Practice Safe Query Caching

Posted At : September 19, 2010 12:00 AM | Posted By : Mark Kruger
Related Categories: Coldfusion & Databases, Coldfusion Tips and Techniques

First let me say that query caching on a CF server is *not a panacea*. There are many ways to improve performance and there are many techniques for caching. All techniques have trade-offs. Still, there are instances where caching will save you time and money - and those are both things in short supply. More to the point, query caching is *so easy to implement* that it can be done for an entire site or application in a relatively short time - as long as you follow some simple rules and take some precautions (please people - use "safe caching").

How do I Cache?

First, you must make sure that caching is enabled in the Coldfusion Administrator. Go into CF Administrator and click on the Caching link in the side menu. Make sure there is a number entered for "maximum cached queries" in the box provided. How many you wish to cache depends on your server resources, site activity, database etc. See comments below under the section explaining the FIFO buffer.

Now it's time to fiddle with the CFQUERY tag. There are actually 2 attributes to cfquery that deal with caching. They are ***cachedAfter*** and ***cachedWithin***. CachedAfter takes a *date* as a parameter. It will cache the query AFTER the date that is specified. So, if I added the cachedAfter tag with tomorrows date in it, the query would run "live" *until tomorrow* and run from the cache from then on. To be honest I've never had the occasion to use cachedafter. It would be more useful if it took a time value as well as a date.

The second caching attribute is the one that I find useful. It takes the results of "createTimespan()" as its argument. Create timespan allows you to create "length-of-time" variable down to the second. It takes a list of 4 integers for Days, hours, minutes and seconds. So, for example, a time span of 2 hours and 13 minutes would be #createTimespan(0,2,13,0)#. If I wanted to cache a query for 2 hours and 13 minutes I would doe something like this:

```
<cfquery name="NightOnTown" datasource="dsn" cachedwithin="#createTimespan(0,2,13,0)#">
   SELECT   Hose, Slip, Garters, shoes, fashion_sense
   FROM      myCloset
   WHERE   username = 'bubbaLouie'
</cfquery>
```

How does it work?

This is where some folks who are new to query caching get nervous. If I'm caching the query "NightOnTown" and username "samIam" shows up, will he see BubbaLouie's shoes? The answer is *no*. Bubbalouie's pumps will remain in the closet (along with his garters hopefully). To pull from the cache, more than just the name of the query must match. Here's the list:

- Same query "Name"
- Exact same SQL statement - "where username='bubbaLouie'" and "where username = 'samIam'" are 2 different statements, ergo 2 different queries in the cache - even if they are both "named" NightOnTown.

- Same Datasource - for those of you who fail to assume and stumbled onto that thought.
- Same Username and password - This is interesting to note. If you have a site with a shared datasource but multiple db usernames you may not get the benefit from caching that you think you should.
- Same DBTYPE

How does it Help?

Recently I was working on a "sick server" and ran across a query that was being run with every page request. The purpose of the query was to build a drop down list of choices. This busy site was receiving over 200 views per minute. That means (for those of you taking notes) that the database is being tapped for what is largely the same information 200 times per minute or 12000 times an hour. I suggested caching the query. The client was concerned because the choices in the drop down list changed frequently - several times an hour. So I suggested caching the query for just *three minutes*.

Three minutes? How can caching for so short a time make any difference? Well in this case the number of db hits went from 12,000 per hour to 20 per hour for that single query. So you see, even a small amount of caching can make a difference. Minimizing the number of calls to the database has an exponential effect on your server. Remember, *database activity is virtually always the single most costly process on your web server*. Getting to know your data, database schema, database platform and advanced SQL techniques will *probably* do more for your site than advanced CF programming (although you should strive for both). When the number of calls to the DB goes down, there are more threads in the pool and the JDBC manager doesn't have to work so hard. Resources are freed up for other things. Requests are less likely to be queued and you diminish the possibility of the snowball effect on your resources. In other words, it's a good thing.

What are the rules?

Now don't go off and add caching willy nilly. Here are some things to keep in mind.

CFQUERYPARM and Caching are Mutually exclusive on ColdFusion 7 and below - you can't have both on the same server. Why? Because data binding is based on "typing" the data for the DB server. The SQL Statement becomes "where username = [variable of type varchar]" instead of "where username = 'bubbaLouie'". Since the "statement" part of the query is generic, it would cause bubbaLouie and samIam to *seem* like they were the same query. One more important note, if you can't use CFQUERYPARAM and caching together, then you *must remember to validate* user entered variables. Otherwise you will be exposed to SQL injection attacks. **NOTE: Starting with ColdFusion 8 Queries using cfqueryparam can now be cached**. Apparently Adobe figured it out so now you need not worry about that issue.

Know your Data Patterns - To use caching you should be aware of how often the data you are caching changes as the result of updates or inserts. You should also be aware of *how significant* the lag time might be to a user. If the information changes quite frequently, but a lag of 5 or 10 minutes doesn't make any difference to those who need the data, then you can cache safely for 5 or 10 minutes. If the users cannot tolerate *any lag time*, then you may have to abandon caching even if the data doesn't change

often.

Beware the Delete - One special category of changes is the "delete". If data that you intend on caching is frequently deleted then you may need to implement a "cache refresh" technique (I'll blog about that sometime). Consider what might happen if data is being displayed on your site that has already been deleted. If a user has to interact with that data you will throw the site open to errors in your database code where an expected record is not found, or (worse) you will end up creating orphan records or badly synchronized data that will not be tenable.

FIFO is Not Always Your Friend - 2 things control the cache. One is the time span of hours, minutes or even days that you choose to cache the query, but the other is the "max cached queries" setting that we referenced above. Consider the scenario where a developer has chosen to cache the users login information. Let's suppose the max cache limit is 500 queries. When the 501rst user logs in, his query is cached and the "oldest" query is expired from the cache. That means someone's query who is already logged in will be re-run "live" (not pulled from the cache), and at that point his or her query will be cached forcing out another query - and so on... (I can hear Elton John singing "..it's the circle of life....").

If the query you are caching is run thousands of times with different parameters it may not be useful to cache it. You could turn your FIFO (first in first out) cache buffer into a wildly spinning revolving door. Queries that may be larger and more useful to cache might get pushed out so often as to make caching them equally useless. You could set the number into the thousands, but that might just overload the resources of your web server. As you can see, it's important to know that level and activity on your server.

Happy Safe Caching!