Legacy Code Part 16: How You Get Data

Posted At : September 18, 2013 6:03 PM | Posted By : Cutter
Related Categories: Legacy Code, Development, ColdFusion, Application Setup, Adobe

One of the greatest culprits in poorly performing ColdFusion applications, Legacy Code or modern, is badly written SQL. ColdFusion was originally written as a middle tier between the web server and the database, giving us the ability to output queried data in html format. In fact CFML was originally DBML. One of it's greatest strengths has always been the ease with which one could get data back from a multitude of datasources, from a variety of platforms, and display it to a user. One of it's greatest weaknesses is that it allows you, the developer, to write those queries just as poorly as you want to as well.

How many queries in your system start with *SELECT **? Of the 35 columns in that table, are you really using more than five or ten? And, do you have multiple <cfquery> calls, one after another, with each using data from the query before? Can this be translated into a single query using some well thought out joins? By the way, when's the last time you analyzed your query performances? How about rebuilt your table indexes?

In most applications that I've worked on, Legacy and modern, major bottlenecks occurred due to the poor performance of queries. Applying &ltcfqueryparam>s helped a little, but truly reviewing each query, running it through a query analyzer, rebuilding and creating new indexes.

Some people, especially writing quick small apps and prototypes, have used ORM frameworks for their database interaction, such as **Reactor**, **Transfer**, and now ColdFusion's own built-in implementation of **Hibernate via ColdFusion ORM**. These are very popular, and great for quickly standing up new product, but they are also very object intensive, and don't necessarily give you deep introspection into what those data transactions are truly doing under the covers. Yes, they can allow you to quickly build new applications, but that doesn't necessarily mean that those applications will scale well, or continue to perform three years later the same way they did on day one.

There was a **really good article by Chris Travers** on DZone, a few weeks back, defending the choice to continue to hand code your SQL. It's not a very long article, but one paragraph really stood out for me, and is something that I already do.

> I find that my overall development time is not slowed down by hand-writing SQL. This remains true even as the software matures. The time-savings of automatic query tools is traded for the fact that one doesn't get to spend time thinking about how to best utilize queries in the application. The fact

> is that, as application developers, we tend to do a lot in application code that could be done better as part of a query. Sitting down and *thinking* about how the queries fit into the application is one of the single most productive exercises one can do.

This is something that is very easy for me to identify with, and also a good argument for the (occasional and well thought out) use of stored procedures. Many developers stay away from stored procedures because a) they don't know how to write or use them, and/or b) those procedures aren't stored in code, so it's not as easy to introspect or search for things when you're making changes. While both of these may be valid arguments, in some way, there is the performance trade off. Stored procedures due, typically, perform better, having compiled cached execution plans on the SQL server. If you can overcome your other obstacles (and you can), you can gain from placing complex SQL logic inside of stored procs.

Again, this type of change, in a large Legacy Code system, can be long and arduous. Set a goal, with each bit of maintenance that you do, to review the queries you are using in the requests you are adjusting. Tools like **FusionReactor** can help you identify slow running queries, for you to target your efforts. It may pay well to hire an outside SQL resource to review your databases, monitor query performance, and provide detailed analysis and suggestions on improvement. A good DBA, even a part timer, can save a Legacy Code application from extinction.

In our next post we'll dive into some of the things that you can do at the Application Server level, to help you get the most out of your Legacy Code.

This article is the sixteenth in a series of articles on bringing life back to your legacy ColdFusion applications. Follow along in the ***Legacy Code*** *category.*