Craig S. Mullins

Return to Home Page

June / July 2008

zData Perspectives

by Craig S. Mullins

The Rise of Dynamic SQL

Have you noticed that dynamic SQL is more popular today than ever before?

There are a number of factors contributing to the success of dynamic SQL. Commercial off the shelf applications, like SAP, Siebel and Peoplesoft, utilize dynamic SQL when DB2 for z/OS is used as the database server. In many cases, too, dynamic SQL is the default choice for in-house application development. Development of Java applications using an IDE that interacts with JDBC can be simpler for programmers than building static applications from scratch. Applications that use JDBC and ODBC will result in dynamic SQL. Furthermore, dynamic SQL is prevalent in most web-based development projects.

This trend to embrace dynamic SQL is in stark contrast to the mindset of a decade or so ago when the prevailing rule was to avoid dynamic SQL at all costs. Of course, back then, dynamic SQL in a DB2 environment meant COBOL programs with embedded dynamic SQL, which was not as easy to code.

DBA groups created those rules about avoiding dynamic SQL, and usually for good reason. Dynamic SQL was unpredictable because it was built "on the fly" in the program, and it was more difficult to monitor and tune because dynamic SQL is optimized at runtime, instead of beforehand during the bind process.

Even today, the performance of dynamic SQL still is a widely debated DB2 topic. Some shops still try to avoid it, while many more place controls on its use. But most of the past concerns can be relegated to the dustbin of history. In this day and age, a hard and fast rule of "no dynamic SQL" is unwarranted. IBM has made significant performance improvements and provided new options to help. As such, the differences between static and dynamic SQL have lessened over time. Today, there are various options at your disposal to make static act more like dynamic, as well as to make dynamic act more like static.

Turning things upside down, dynamic SQL can even offer advantages over static SQL when it is used appropriately. With dynamic SQL, access paths are not pre-determined at BIND-time so DB2 can use the most up-to-date statistics to build more optimal query execution plans. And for queries having predicates written on columns with non-uniformly distributed data, DB2 can factor the host variable values into the access path criterion. This can produce performance improvements over static SQL which typically has no knowledge of the host variable values (but you can use the REOPT parameter to change that, too).

Dynamic statement caching (DSC) removes yet another impediment to dynamic SQL performance. With DSC, DB2 saves prepared dynamic statements in a cache. After a dynamic SQL statement has been prepared and saved in the cache, subsequent prepare requests for that same SQL statement can avoid the costly preparation process by reusing the statement from the cache.

Of course, there are disadvantages to dynamic SQL, too. Whenever dynamic SQL statements are prepared and not in the DSC, total statement execution time increases. This is so because the time to prepare the dynamic statement must be added to the overall execution time. And, keep in mind, in order for the dynamic prepare to be reused, the dynamic SQL statement has to be **exactly** the same as the one that caused the prepared statement to be cached -- even down to the same number of spaces in the text of the statement.

From the perspective of performance monitoring and tuning, dynamic SQL still can be more difficult to manage. For static SQL, the statement text is available in the DB2 catalog and the access path information is available in the associated plan tables after binding with the EXPLAIN(YES). There is no PLAN_TABLE that contains the access paths for dynamic SQL, nor is the SQL available in the DB2 catalog. For this reason some DBAs view dynamic SQL as a performance black hole. Of course, you can EXPLAIN the statements in the DSC, and there ***are*** tools that allow you to shine a light into this darkness by capturing statements from the DSC and explaining them.

Error detection can also be problematic because dynamic SQL is only compiled at runtime, so errors in the SQL statement may not be detected until it is run. And, of course, authorization and security can be more burdensome for dynamic SQL applications. For example, using dynamic SQL puts more responsibility on programmers to avoid security exposures, such as SQL injection attacks.

But the bottom line is that dynamic SQL is a viable option for application development in the 21st century. It is wise to avoid stringent rules that prohibit its use in your shop.

From zJournal, June / July 2008
.

Home.