February / March 2011

zData Perspectives

Dynamic SQL Versus Static SQL

by Craig S. Mullins

Many DB2 professionals are not comfortable determining when to choose dynamic SQL instead of static. This month's column examines criteria that can be used to help make that decision.

Performance Sensitivity: For very frequent, high performance SQL statements weigh the overhead required for the PREPARE against the potential for better access paths when choosing between dynamic and static. When performance is crucial, favor static SQL. Using dynamic SQL incurs a higher initial cost per SQL statement due to the PREPARE. Once prepared, the difference in execution time for dynamic versus static diminishes. If multiple users are running the same dynamic application with the same statements, and dynamic statement caching is implemented, only the first application to issue the statement realizes the cost of preparing.

Range Predicates: Range predicates refer to greater-than (>), less-than (<), BETWEEN, and LIKE. The more frequently you need to use range predicates the more you should consider dynamic SQL. This is so because the optimizer can take advantage of distribution statistics and histogram statistics with dynamic SQL to formulate better access paths because the actual range will be known.

Repetitious Execution: If the statement runs once a day, for example, it should not really matter whether dynamic or static SQL is used. Because the statement does not run frequently the impact of dynamic preparation is likely moot. As the frequency of execution increases, you should favor static SQL or dynamic SQL with KEEPDYNAMIC YES. The cost of the PREPARE becomes a smaller percentage of the overall run time of the statement the more frequently it runs (if the cached prepare is reused).

Nature of the Query: If you need all or part of the SQL statement to be generated during application execution, favor

In some cases, the access paths chosen by the optimizer for dynamic SQL can be better but authorization checking costs and prepare statement lookup can often mean up to 30% more CPU time for dynamic SQL the first time the statement is executed.

Language: For COBOL, it still makes sense to favor static over dynamic as a default. Static is generally easier to code, monitor, and tune. This does not mean that dynamic SQL has no place within a COBOL shop, just that static is the standard and you should deviate only when absolutely necessary. For Java, C and web developers, dynamic SQL is easier to use because of the IDEs and CLIs that simplify coding. Of course, the same performance issues that impact COBOL programs exist for Java, too.

Data Uniformity: When data is very non-uniformly distributed (e.g. beer drinkers skew male) it makes sense to favor dynamic SQL over static. Dynamic SQL can result in more efficient access paths than static for non-uniform distributions. This is so because DB2 can potentially derive different access paths for different host variable values. Columns that have a limited number of values can result in significantly different access paths for the same SQL statement when the DB2 optimizer has access to the values.

Data can also be correlated, which is a similar yet separate concept. For example, CITY, STATE, and ZIP_CODE data will be correlated. The ZIP_CODE determines the CITY and STATE. And there will be many more occurrences of HOUSTON, TX than there would be of HOUSTON, CA (if, indeed, that is even valid data).

statement to be generated during application execution, favor dynamic over static SQL. A common scenario involves a transaction where the user is presented with a screen and can specify various criteria for querying. The SQL statement is based on what the user enters and there can be many variations. Coding such an application with static SQL is quite difficult because of the numerous variations that need to be supported. But it is a snap with dynamic SQL because the SQL can be built on the fly based on what is entered.

Runtime Environment: Favor dynamic SQL when you need to build an application where the database objects may not exist at precompile time. Binding a static SQL program that attempts to access a non-existent table will not succeed unless you also specify VALIDATE(RUN); but then every time you run the program, DB2 must validate that the required database objects exist, thereby degrading performance.

RUNSTATS Frequency: Consider dynamic SQL when your application needs to access data that changes frequently and dramatically. Using dynamic SQL increases the probability of using the most optimal access path, based on current statistics. With static SQL you would have to either use REOPT, or make sure that you REBIND the program as frequently as you run RUNSTATS -- and even then if the data is skewed you may not be getting the best access path for every SQL statement in your program.

Summary: Of course, the issues we've discussed here are conditional, and you will need to apply knowledge of your business, application, and specific performance requirements in order to make informed decisions on dynamic versus static SQL. But you can use the scenarios here to drive your decision-making process.

From zJournal, Feb / March 2011.

DB2PORTAL.com