Determining when to execute SQL statements statically or dynamically in embedded SQL applications

Last Updated: 2024-02-09

There are several factors that you must consider before determining whether to issue a static or dynamic SQL statement in an embedded SQL application.
The following table lists the considerations associated with use of static and dynamic SQL statements.

Note: These are general suggestions only. Your application requirement, its intended usage, and working environment dictate the actual choice. When in doubt, prototyping your statements as static SQL, then as dynamic SQL, and comparing the differences is the best approach.

Table 1. Comparing Static and Dynamic SQL

Consideration	Likely Best Choice
Uniformity of data being queried or operated upon by the SQL statement	
– Uniform data distribution	– Static
– Slight non-uniformity	– Either
– Highly non-uniform distribution	– Dynamic
Quantity of range predicates within the query	
– Few	– Static
– Some	– Either
– Many	– Dynamic
Likelihood of repeated SQL statement execution	
– Runs many times (10 or more times)	– Either
– Runs a few times (less than 10 times)	– Either
– Runs once	– Static
Nature of Query	
– Random	– Dynamic
– Permanent	– Either
Types of SQL statements (DML/DDL/DCL)	

Consideration	Likely Best Choice
– Transaction Processing (DML Only)	– Either
– Mixed (DML and DDL - DDL affects packages)	– Dynamic
– Mixed (DML and DDL - DDL does not affect packages)	– Either
Frequency with which the RUNSTATS command is issued	
– Infrequently	– Static
– Regularly	– Either
– Frequently	– Dynamic

SQL statements are always compiled before they are run.

The difference is that dynamic SQL statements are compiled at runtime, so the application might be slower due to the increased resource use associated with compiling each of the dynamic statements at application runtime versus during a single initial compilation stage as is the case with static SQL.

In a mixed environment, the choice between static and dynamic SQL must also factor in the frequency in which packages are invalidated. If the DDL does invalidate packages, dynamic SQL is more efficient as only those queries issued are recompiled when they are next used. Others are not recompiled. For static SQL, the entire package is rebound once it has been invalidated.

There are times when it does not matter whether you use static SQL or dynamic SQL. For example it might be the case within an application that contains mostly references to SQL statements to be issued dynamically that there might be one statement that might more suitably be issued as static SQL. In such a case, to be consistent in your coding, it might make sense to issue that one statement dynamically too. Note that the considerations in the previous table are listed roughly in order of importance.

Do not assume that a static version of an SQL statement is always faster than the equivalent dynamic statement. In some cases, static SQL is faster because of the resource use required to prepare the dynamic statement. In other cases, the same statement prepared dynamically issues faster, because the optimizer can make use of current database statistics, rather than the database statistics available at an earlier bind time. Note that if your transaction takes less than a couple of seconds to complete, static SQL will generally be faster. To choose which method to use, you should prototype both forms of binding.

Note: Static and dynamic SQL each come in two types, statements which make use of host variables and ones which don't. These types are:

1. Static SQL statements containing no host variables

   This is an unlikely situation which you may see only for:

   - Initialization code
   - Simple SQL statements

Simple SQL statements without host variables perform well from a performance perspective in that there is no runtime performance increase, and the Db2® optimizer capabilities can be fully realized.

2. Static SQL containing host variables
   Static SQL statements which make use of host variables are considered as the traditional style of Db2 applications. The static SQL statement avoids the runtime resource usage associated with the PREPARE and catalog locks acquired during statement compilation. Unfortunately, the full power of the optimizer cannot be used because the optimizer does not know the entire SQL statement. A particular problem exists with highly non-uniform data distributions.

3. Dynamic SQL containing no parameter markers
   This is typical of interfaces such as the CLP, which is often used for executing on-demand queries. From the CLP, SQL statements can only be issued dynamically.

4. Dynamic SQL containing parameter markers
   The key benefit of dynamic SQL statements is that the presence of parameter markers allows the cost of the statement preparation to be amortized over the repeated executions of the statement, typically a select, or insert. This amortization is true for all repetitive dynamic SQL applications. Unfortunately, just like static SQL with host variables, parts of the Db2 optimizer will not work because complete information is unavailable.

Parent topic:

→ Static and dynamic SQL statement execution in embedded SQL applications