**Dynamic SQL Support Statements**

The dynamic SQL support statements accept a character-string host variable and a statement name as arguments. The host variable contains the SQL statement to be processed dynamically in text form. The statement text is not processed when an application is precompiled. In fact, the statement text does not have to exist at the time the application is precompiled. Instead, the SQL statement is treated as a host variable for pre-compilation purposes and the variable is referenced during application execution. These SQL statements are referred to as *dynamic* SQL.

Dynamic SQL support statements are required to transform the host variable containing SQL text into an executable form and operate on it by referencing the statement name. These statements are:

**EXECUTE IMMEDIATE**

Prepares and executes a statement that does not use any host variables. All EXECUTE IMMEDIATE statements in an application are cached in the same place at run time, so only the last statement is known. Use this statement as an alternative to the PREPARE and EXECUTE statements.

**PREPARE**

Turns the character string form of the SQL statement into an executable form of the statement, assigns a statement name, and optionally places information about the statement in an SQLDA structure.

**EXECUTE**

Executes a previously prepared SQL statement. The statement can be executed repeatedly within a connection.

**DESCRIBE**

Places information about a prepared statement into an SQLDA.

An application can execute most supported SQL statements dynamically.

**Note:** The content of dynamic SQL statements follows the same syntax as static SQL statements, with the following exceptions:

* Comments are not allowed.
* The statement cannot begin with EXEC SQL.
* The statement cannot end with the statement terminator. An exception to this is the CREATE TRIGGER statement which can contain a semicolon (;).

**Dynamic SQL versus Static SQL**

The question of whether to use static or dynamic SQL for performance is usually of great interest to programmers. The answer depends on your situation.

Use the following table when deciding whether to use static or dynamic SQL. Considerations such as security dictate static SQL, while environmental considerations (for example, using DB2 CLI or the CLP) dictate dynamic SQL. When making your decision, consider the following recommendations on whether to choose static or dynamic SQL in a particular situation. In the following table, 'Either' means that there is no advantage to either static or dynamic SQL.

**Note:**

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

| ***Table 1. Comparing Static and Dynamic SQL*** | |
| --- | --- |
| **Consideration** | **Likely Best Choice** |
| Time to run the SQL statement:   * Less than 2 seconds * 2 to 10 seconds * More than 10 seconds | * Static * Either * Dynamic |
| Data Uniformity   * Uniform data distribution * Slight non-uniformity * Highly non-uniform distribution | * Static * Either * Dynamic |
| Range (<,>,BETWEEN,LIKE) Predicates   * Very Infrequent * Occasional * Frequent | * Static * Either * Dynamic |
| Repetitious Execution   * Runs many times (10 or more times) * Runs a few times (less than 10 times) * Runs once | * Either * Either * Static |
| Nature of Query   * Random * Permanent | * Dynamic * Either |
| Run Time Environment (DML/DDL)   * Transaction Processing (DML Only) * Mixed (DML and DDL - DDL affects packages) * Mixed (DML and DDL - DDL does not affect packages) | * Either * Dynamic * Either |
| Frequency of RUNSTATS   * Very infrequently * Regularly * Frequently | * Static * Either * Dynamic |

In general, an application using dynamic SQL has a higher start-up (or initial) cost per SQL statement due to the need to compile the SQL statements before using them. Once compiled, the execution time for dynamic SQL compared to static SQL should be equivalent and, in some cases, faster due to better access plans being chosen by the optimizer. Each time a dynamic statement is executed, the initial compilation cost becomes less of a factor. If multiple users are running the same dynamic application with the same statements, only the first application to issue the statement realizes the cost of statement compilation.

In a mixed DML and DDL environment, the compilation cost for a dynamic SQL statement may vary as the statement may be implicitly recompiled by the system while the application is running. 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 may be more efficient as only those queries executed are recompiled when they are next used. Others are not recompiled. For static SQL, the entire package is rebound once it has been invalidated.

Now suppose your particular application contains a mixture of the above characteristics, and some of these characteristics suggest that you use static while others suggest dynamic. In this case, there is no obvious decision, and you should probably use the method you have the most experience with, and with which you feel most comfortable. Note that the considerations in the above table are listed roughly in order of importance.

**Note:**

Static and dynamic SQL each come in two types that make a difference to the DB2 optimizer. These types are:

1. **Static SQL containing no host variables**

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

* + *Initialization* code
  + Novice training examples

This is actually the best combination from a performance perspective in that there is no run-time performance overhead, and the DB2 optimizer's capabilities can be fully realized.

1. Static SQL containing host variables

This is the traditional *legacy* style of DB2(R) applications. It avoids the run time overhead of a PREPARE and catalog locks acquired during statement compilation. Unfortunately, the full power of the optimizer cannot be utilized because the optimizer does not know the entire SQL statement. A particular problem exists with highly non-uniform data distributions.

1. Dynamic SQL containing no parameter markers

This is the typical style for random query interfaces (such as the CLP), and is the optimizer's preferred *flavor* of SQL. For complex queries, the overhead of the PREPARE statement is usually offset by the improved execution time.

1. Dynamic SQL containing parameter markers

This is the most common type of SQL for CLI applications. The key benefit is that the presence of parameter markers allows the cost of the PREPARE 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. The recommendation is to use *static SQL with host variables* or *dynamic SQL without parameter markers* as the most efficient options.