Chapter
2

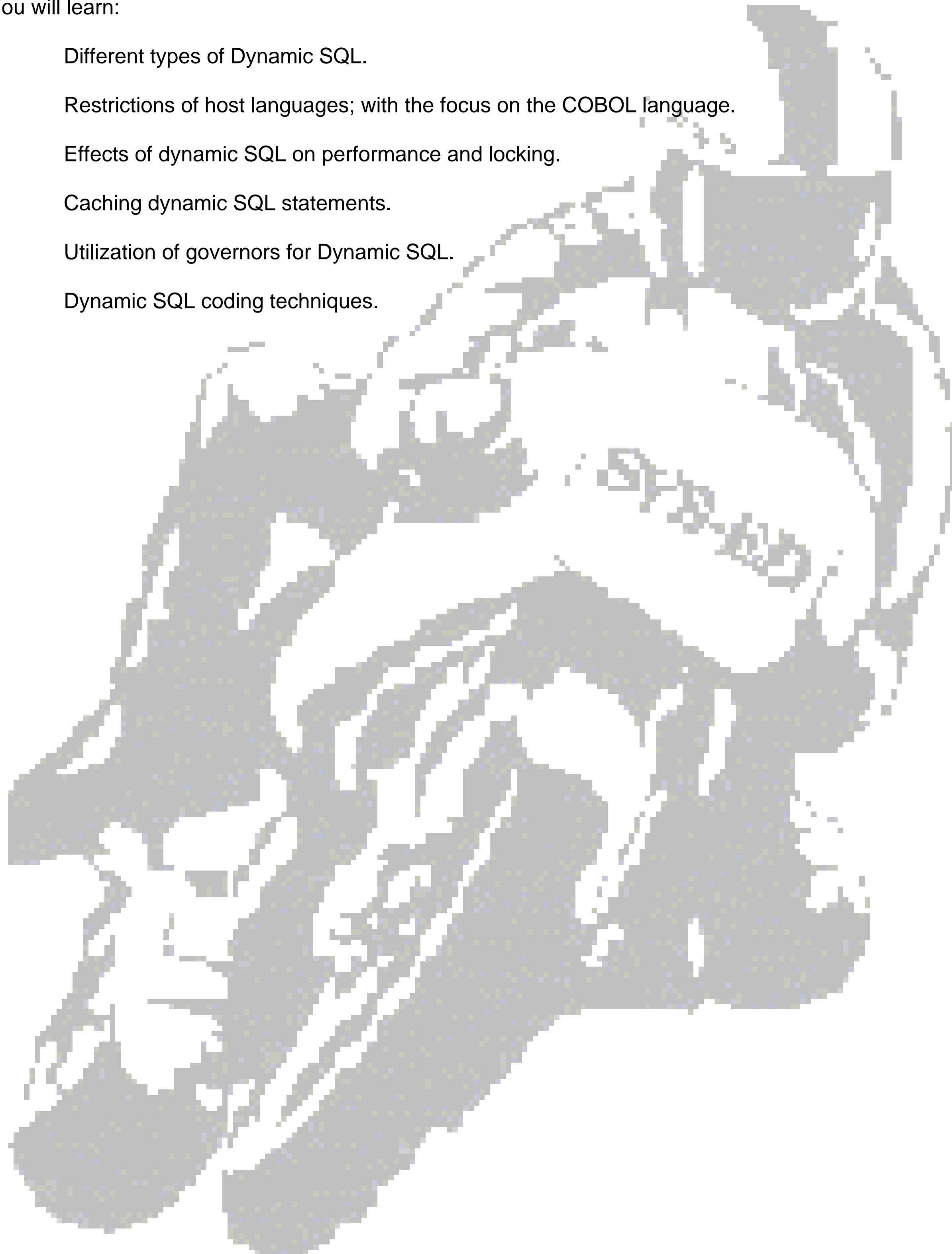
DYNAMIC SQL PROCESSING

SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.

Objectives

You will learn:

- Different types of Dynamic SQL.
- Restrictions of host languages; with the focus on the COBOL language.
- Effects of dynamic SQL on performance and locking.
- Caching dynamic SQL statements.
- Utilization of governors for Dynamic SQL.
- Dynamic SQL coding techniques.



1 Monitoring the Database

The database performance and space utilization is measured with several utilities:

- The RUNSTATS utility obtains information about the efficiency of space utilization for table spaces and indexes.

- The STOSPACE utility obtains information about the amount of space used by storage groups and related table spaces and indexes.

- The DB2 trace facility reports on a variety of internal system events, under many different conditions.

- The DB2 EXPLAIN PLAN table, the EXPLAIN statement, and the EXPLAIN options of BIND and REBIND, tell how an SQL statement will be executed.

1.1 RUNSTATS Utility

The RUNSTATS utility updates the DB2 catalog with data about space utilization and row clustering, so that DB2 has current information with which to select access paths.

To ensure that the information is available and current, the user should run RUNSTATS:

- After loading a table space and before binding application plans that will access it.

- After reorganizing a table space. The user should rebind the application plans where performance is critical.

- After extensive inserts and deletes. Again, the user should rebind the application plans where performance is critical.

  Use the updated information to determine when to reorganize (REORG TABLESPACE statement) the table spaces and indexes.

- Extract this information using SQL statements that access the DB2 catalog.

2 Dynamic SQL in Application Programs

Before deciding whether to use dynamic SQL, evaluate whether static SQL or dynamic SQL is the best technique for an application.

For most DB2 users, static SQL--embedded in a host language program and bound before the program runs--provides a straightforward, efficient path to DB2 data. Static SQL can be used when it is known before run time what SQL statements an application needs to execute.

Dynamic SQL prepares and executes the SQL statements within a program, while the program is running.

There are four types of dynamic SQL:

- Embedded dynamic SQL.
- Interactive SQL.
- Deferred embedded SQL.
- Dynamic SQL executed through ODBC functions.

3 Host Variables in Static SQL

When using static SQL, the form of SQL statements can not be changed changes are made to the program.

However, host variables can be used for increasing the flexibility of the statements by using host variables.

```
01  IOAREA.
        02  EMPID                   PIC X(06).
        02  NEW-SALARY              PIC S9(7)V9(2) COMP-3.
.
.
.
(Other declarations)
READ CARDIN RECORD INTO IOAREA
AT END MOVE 'N' TO INPUT-SWITCH.
.
.
.
(Other COBOL statements)
EXEC SQL
        UPDATE DSN8610.EMP
        SET SALARY = :NEW-SALARY
         WHERE EMPNO = :EMPID
END-EXEC.
```

The statement (UPDATE) does not change, nor does its basic structure, but the input can change the results of the UPDATE statement.

Dynamic SQL is completely flexible.

- Query Management Facility (QMF), is a program which provides an alternative interface to DB2 that accepts almost any SQL statement.
- SPUFI is another example; it accepts SQL statements from an input data set, and then processes and executes them dynamically.

A program that provides for dynamic SQL accepts as input or generates an SQL statement in the form of a character string. The programming can be simplified by planning that the program not use SELECT statements, or to use only those that return a known number of values of known types.

In the most general case, in which it is not known in advance which SQL statements will be executed, the program takes these steps:

1. Translates the input data, including any parameter markers, into an SQL statement.
2. Prepares the SQL statement to execute and acquires a description of the result table.
3. Obtains, for SELECT statements, enough main storage to contain retrieved data.
4. Executes the statement or fetches the rows of data.
5. Processes the information returned.
6. Handles SQL return codes.

Performance

To access DB2 data, an SQL statement requires an access path.

Two big factors in the performance of an SQL statement are the amount of time that DB2 uses to determine the access path at run time and whether the access path is efficient.

DB2 determines the access path for a statement at either of these times:

- When binding the plan or package that contains the SQL statement.
- When the SQL statement executes.

For static SQL statements that do not contain input host variables, DB2 determines the access path when the plan or package is being bound. This combination yields the best performance because the access path is already determined when the program executes.

4 Static SQL Statements with Input Host Variables

The time at which DB2 determines the access path depends on whether the bind option NOREOPT(VARS) or REOPT(VARS) is specified. NOREOPT(VARS) is the default.

- If NOREOPT(VARS) is specified, DB2 determines the access path at bind time, just as it does when there are no input variables.

- If REOPT(VARS) is specified, DB2 determines the access path at bind time and again at run time, using the values in these types of input variables:

Host variables	Parameter markers	Special registers

This means that DB2 must spend extra time determining the access path for statements at run time, but if DB2 determines a significantly better access path using the variable values, then there might be an overall performance improvement.

In general, using REOPT(VARS) can make static SQL statements with input variables perform like dynamic SQL statements with constants.

5 Dynamic SQL Statements

For dynamic SQL statements, DB2 determines the access path at run time when the statement is prepared. This can make the performance inferior to that of static SQL statements.

However, if the same SQL statement is executed often, then the dynamic statement cache can be used for decreasing the number of times that the dynamic statements must be prepared.

Dynamic SQL Statements with Input Host Variables

The REOPT(VARS) option should be used when binding applications that contain dynamic SQL statements with input host variables.

PREPARE statements should be coded to minimize overhead. With REOPT(VARS), DB2 prepares an SQL statement at the same time as it processes OPEN or EXECUTE for the statement.
DB2 will processes the statement as if DEFER(PREPARE) had been specified.

However, if the DESCRIBE statement is executed before the PREPARE statement in a program, or if the PREPARE statement is used with the INTO parameter, DB2 prepares the statement twice.

The first time DB2 will determine the access path without using input variable values; the second time DB2 will uses the input variable values. The extra prepare can result in a performance degradation .

6 Caching Dynamic SQL Statements

As DB2's ability to optimize SQL has improved, the cost of preparing a dynamic SQL statement has grown. Applications that use dynamic SQL might be forced to pay this cost more than once.

- When an application performs a commit operation, it must issue another PREPARE statement if that SQL statement is to be executed again.
- For a SELECT statement, the ability to declare a cursor WITH HOLD provides some relief, but requires that the cursor be open at the commit point.
- WITH HOLD also causes some locks to be held for any objects that the prepared statement is dependent on. Also, WITH HOLD offers no relief for SQL statements that are not SELECT statements.

DB2 can save prepared dynamic statements in a cache. The cache is a DB2-wide cache in the EDM pool that all application processes can use to store and retrieve prepared dynamic statements.

After a SQL statement has been prepared, it is automatically stored in the cache. Subsequent prepare requests for that same SQL statement can then avoid the costly preparation process by using the statement in the cache.

Cached statements can be shared among different threads, plans, or packages.

7 Conditions for Statement Sharing

Consider a scenario, where S1 and S2 are source statements, and P1 is the prepared version of S1.

P1 is in the prepared statement cache.

The following conditions must be met before DB2 can use statement P1 instead of preparing statement S2:

- S1 and S2 must be identical.

  The statements must pass a character by character comparison and must be the same length.

  If either of these conditions are not true, DB2 cannot use the statement in the cache.

- The authorization ID that was used to prepare S1 must be used to prepare S2:

8 Keeping Prepared Statements After Commit Points

The bind option KEEPDYNAMIC(YES) allows for dynamic statements to be held past a commit point for an application process. An application can issue a PREPARE for a statement once and omit subsequent PREPAREs for that statement.

In order to understand how the KEEPDYNAMIC bind option works, it will be necessary to differentiate between:

- the executable form of a dynamic SQL statement, the prepared statement.
- the character string form of the statement, the statement string.

9 Limiting Dynamic SQL with the Resource Limit Facility

The resource limit facility or governor limits the amount of CPU time an SQL statement can consume. This serves to prevent SQL statements from making excessive requests.

The predictive governing function of the resource limit facility provides an estimate of the processing cost of SQL statements before they run. In order to predict the cost of an SQL statement, the EXPLAIN statement needs to be executed in order to put information about the statement cost in DSN_STATEMNT_TABLE.

The governor controls only the dynamic SQL manipulative statements SELECT, UPDATE, DELETE, and INSERT.

Each dynamic SQL statement used in a program is subject to the same limits. The limit can be a reactive governing limit or a predictive governing limit.

- If the statement exceeds a reactive governing limit, the statement receives an error SQL code.
- If the statement exceeds a predictive governing limit, it receives a warning or error SQL code.

The system administrator can establish the limits for:

- individual plans or packages
- individual users.
- all users who do not have personal limits.

10 Choosing a Host Language for Dynamic SQL Applications

Programs that use dynamic SQL are usually written in Assembler, C, PL/I, and versions of COBOL other than OS/VS COBOL. It is possible to write non-SELECT and fixed-list SELECT statements in any of the DB2 supported languages.

A program containing a varying-list SELECT statement is more difficult to write in FORTRAN, because the program cannot run without the help of a subroutine to manage address variables (pointers) and storage allocation.

Dynamic SQL for non-SELECT Statements

The easiest way to use dynamic SQL is not to use SELECT statements dynamically.

Because it is not required to dynamically allocate any main storage, a program can be written in any host language, including OS/VS COBOL and FORTRAN.

The program must take the following steps:

1. Include an SQLCA.

   The requirements for an SQL communications area (SQLCA) are the same as for static SQL statements.

2. Load the input SQL statement into a data area.

   Execute the statement; either of these methods can be used:

   Dynamic execution using EXECUTE IMMEDIATE".
   Dynamic execution using PREPARE and EXECUTE".

3. Handle any errors that might result.

   The requirements are the same as those for static SQL statements.

   The return code from the most recently executed SQL statement appears in the host variables SQLCODE and SQLSTATE or corresponding fields of the SQLCA.

4. Dynamic execution using EXECUTE IMMEDIATE.

Consider a program which has been designed to read SQL DELETE statements from a terminal:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = '000190'
DELETE FROM DSN8610.EMP WHERE EMPNO = '000220'
```

After reading a statement, the program is to execute it immediately.

The SQL statements must be precompiled and bound using static SQL statements before they can be used. Dynamic SQL statements can not be prepared in advance.

The SQL statement EXECUTE IMMEDIATE causes an SQL statement to prepare and execute, dynamically, at run time.

To execute the statements:

Read a DELETE statement into the host variable DSTRING.

```
EXEC SQL
        EXECUTE IMMEDIATE :DSTRING;
```

- DSTRING is a character-string host variable.
- EXECUTE IMMEDIATE causes the DELETE statement to be prepared and executed immediately.

11 Dynamic Execution Using PREPARE and EXECUTE

Consider a scenario where it will be necessary to execute DELETE statements repeatedly using a list of employee numbers.

This can be performed by writing the DELETE statement as a static SQL statement:

```
 < Read a value for EMP from the list. >
DO UNTIL (EMP = 0);
        EXEC SQL
        DELETE FROM DSN8610.EMP WHERE EMPNO = :EMP ;
< Read a value for EMP from the list. >
 END;
```

The loop repeats until it reads an EMP value of 0.

Parameter Markers

Dynamic SQL statements cannot use host variables. Therefore, it will not be possible to dynamically execute an SQL statement which contains host variables.

Instead, substitute a parameter marker, indicated by a question mark (?), for each host variable in the statement. DB2 can use a parameter marker which represents a host variable of a certain data type by specifying the parameter marker as the argument of a CAST function.

When the statement executes, DB2 converts the host variable to the data type in the CAST function.

- A parameter marker included in a CAST function is called a typed parameter marker.
- A parameter marker without a CAST function is called an untyped parameter marker.

Since DB2 can evaluate an SQL statement with typed parameter markers more efficiently than a statement with untyped parameter markers, it is recommend that typed parameter markers be used whenever possible.

To prepare this statement:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = :EMP;
```

use the following string:

```
DELETE FROM DSN8610.EMP WHERE EMPNO = CAST(? AS CHAR(6))
```

Host variable :EMP can be associated with the parameter marker when the prepared statement is executed.

Example:

Assume S1 is the prepared statement.

Then the EXECUTE statement would be:

```
EXECUTE S1 USING :EMP;
```

11.1 The PREPARE Statement

PREPARE and EXECUTE is essentially an EXECUTE IMMEDIATE done in two steps.

The PREPARE step turns a character string into an SQL statement, and then assigns it a designated name.

Example:

Consider a scenario where the variable :DSTRING assigns the value "DELETE FROM DSN8610.EMP WHERE EMPNO = ?".

To prepare an SQL statement from that string and assign it the name S1, use the following code:

```
EXEC SQL PREPARE S1 FROM :DSTRING;
```

The prepared statement will contain a parameter marker, for which a value must be supplied when the statement executes. After the statement is prepared, the table name is fixed, but the parameter marker allows the same statement to be executed many times with different values of the employee number.

11.2 The EXECUTE Statement

EXECUTE executes a prepared SQL statement, naming a list of one or more host variables, or a host structure, that supplies values for all of the parameter markers.

After a statement is prepared, it can be executed many times within the same unit of work. In most cases, COMMIT or ROLLBACK destroys statements prepared in a unit of work. The statement would then have to be prepared again before it can be re-executed.

However, if a cursor is declared for a dynamic statement and the option WITH HOLD is used, a commit operation will not destroy the prepared statement if the cursor is still open. The statement can be executed in the next unit of work without being prepared again.

To execute the prepared statement S1 once, use a parameter value contained in the host variable :EMP

```
EXEC SQL EXECUTE S1 USING :EMP;
```

12 Dynamic SQL for Fixed-list SELECT Statements

A fixed-list SELECT statement returns rows containing a known number of values of a known type. When a fixed-list statement is used, it will be necessary to know in advance what kinds of host variables are needed to declare in order to store the results.

The term "fixed-list" does not imply that it is necessary to know in advance the number rows of data that will return; however, it is required that the number of columns and the data types of those columns be known.

A fixed-list SELECT statement returns a result table that can contain any number of rows; the program will then use the FETCH statement to look at the rows one at a time.

Each successive fetch returns the same number of values and the values have the same data types each time. Host variables can be specified in the same way as static SQL.

13 Declare a Cursor for the Statement Name

Dynamic SELECT statements cannot use INTO; a cursor must be used to put the results into host variables. When declaring the cursor, use the statement name (call it STMT), and give the cursor itself a name.

```
EXEC SQL DECLARE C1 CURSOR FOR STMT;
```

Prepare a statement, STMT, from DSTRING.

Example:

```
EXEC SQL PREPARE STMT FROM :DSTRING;
```

As with non-SELECT statements, the fixed-list SELECT can contain parameter markers.

In order to execute STMT, a program open the cursors, fetches rows from the result table, and closes the cursor.

13.1 Open the Cursor

The OPEN statement evaluates the SELECT statement named STMT.

Example:

Without parameter markers:

```
EXEC SQL OPEN C1;
```

If STMT contains parameter markers, then the USING clause of OPEN needs to provide values for all of the parameter markers in STMT.

If there are four parameter markers in STMT, the following will be required:

```
EXEC SQL OPEN C1
USING :PARM1, :PARM2, :PARM3, :PARM4
```

13.2 Fetch Rows from the Result Table

A program can repeatedly execute a statement.

Example:

```
EXEC SQL FETCH C1 INTO :NAME, :PHONE;
```

The statement uses a list of host variables to receive the values returned by FETCH.

The list has a known number of items (two--:NAME and :PHONE) of known data types; both are character strings, of lengths 15 and 4, respectively.

13.3 Close the Cursor

This step is the same as for static SQL.

Example:

A WHENEVER NOT FOUND statement in a program can name a routine that contains this statement:

```
EXEC SQL CLOSE C1;
```