UNIT 4. DYNAMIC AND STATIC SQL

OVERVIEW

- **STATIC SQL**
- **DYNAMIC SQL**

Figure 4.1 Dynamic and Static SQL

STATIC SQL

STATIC SQL

- **can be used if the statements**
  - **TYPE**
  - **TABLES**
  - **COLUMNS, WHERE- Clause**

  Are known when you write your program

- **The complete SQL statement is contained in the program, except for host variable referring to COLUMN VALUES.**

- **BIND creates the access path**

- **Only the "EXECUTE privilege is checked before execution of the access path**

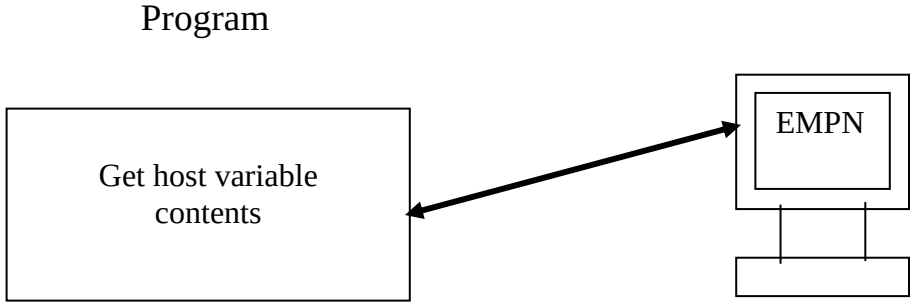

Figure 4.2 Static SQL

DYNAMIC SQL

DYNAMIC SQL

- **MUST be used if ANYTHING ELSE THAN COLUMN VALUES are unknown at program preparation**
- **The statement is contained in a host variable**
- **BIND cannot create an access path but an 'empty' package will be created.**
- **Security is checked at execution time when the access path is created**

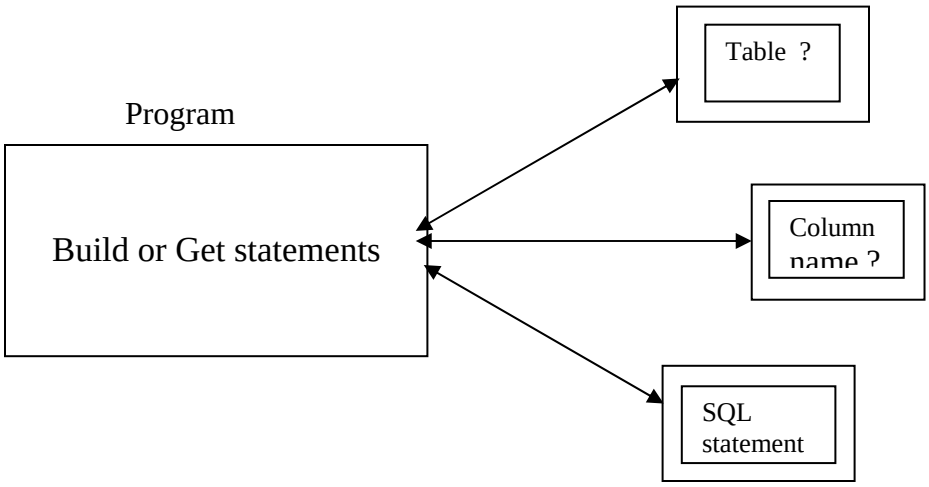

Figure 4.3 Dynamic SQL

Notes:

DYNAMIC SQL (Cont…)

- **NON-SELECT DYNAMIC SQL**
    - **EXECUTE IMMEDIATE**
    - **PREPARE FOR EXECUTE IMMEDIATE**
- **SELECT DYNAMIC SQL**
    - **FIXED-LIST SELECT**
    - **VARYING-LIST SELECT**

Figure: 4.4 Dynamic SQL (Cont.)

Notes:

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE implicitly prepares and executes complete SQL statements coded in host variables. Only a sub-set of SQL statements is available when you use this command, the prominent being SELECT and so this call of dynamic SQL cannot be used for data retrieval. So if you want any data to be retrieved then the SQL portion of your program consists of two steps.

First moving the complete text for the statement to be executed to be executed into the host variable and second issue an EXECUTE IMMEDIATE command giving the host variable as an argument. The statement is prepared and execute automatically.

Ex:

```
        WORKING-STORAGE SECTION.
                    :
                EXEC SQL
                    INCLUDE SQLCA
                END-EXEC.

    01 WS-HOST-VARIBLE.
            49  WS-HOST-VARBLE-LEN      PIC  S9 (4) COMP.
            49  WS-HOST-VARBLE-TXT      PIC  X(80).

        PROCEDURE DIVISION.

            MOVE +45 TO WS-HOST-VARBLE-LEN.
            MOVE “DELETE FROM EMPLOYEE WHERE DEPT=’CONSUL’”
                    TO WS-HOST-VARBLE-TXT.

            EXEC SQL
                    EXECUTE IMMEDIATE :WS-HOST-VARBLE
            END-EXEC.
```

Figure: 4.5 Execute Immediate

EXECUTE IMMEDIATE (Cont…)

The Execute Immediate supports following statements

ALTER, CREATE, DELETE, DROP, EXPLAIN, COMMIT, GRANT, INSERT, REVOKE and UPDATE.

Disadvantages of EXECUTE IMMEDIATE

- It does not support the SELECT statement.
- It can result in poor performance, if the same SQL statement is performed, the executable form of the SQL is destroyed and the program has to prepare the executable form when it executed again.

Figure: 4.6 Execute Immediate (Cont…)

NON-SELECT DYANMIC SQL

This statement is used to prepare and execute the SQL statements in an application program. This class of SQL statements uses PREPARE and EXECUTE to issue the SQL statements. As the name suggests this class also does not support the SELECT statements and cannot be used for querying tables.

Non-SELECT SQL statements can use the feature of dynamic SQL known as the parameter marker (?), which is a **placeholder** for the host variables in the SQL.

```
WORKING-STORAGE SECTION.

01 TVAL                          PIC   X(10)

01 WS-HOST-VAR.
      49  WS-HOST-VAR-LEN        PIC   S9 (4) COMP.
      49  WS-HOST-VAR-TXT        PIC   X(100).

      EXEC SQL
            INCLUDE SQLCA
      END-EXEC.

PROCEDURE DIVISION.

      MOVE +45 TO WS-HOST-VARBLE-LEN.
      MOVE “DELETE FROM  S  WHERE  S#= ? ” TO WS-HOST-VAR-TXT.

      EXEC SQL
            PREPARE STMT1 FROM  :WS-HOST-VARBLE
      END-EXEC.

      MOVE ‘CONSUL’ TO TVAL.

      EXEC SQL
            EXECUTE STMT1 USING :TVAL
      END-EXEC.
```

Figure: 4.7 Non-Select Dynamic SQL (Cont.)

FIXED-LIST SELECT

A FIXED-LIST SELECT statement can be used to explicitly prepare and execute SQL SELECT statements when the column to be selected are known and unchanging. This is necessary to create the WORKING-STROAGE declaration for the host variable in the program.

Example

```
 WORKING-STORAGE SECTION.

 01 WS-HOST-VAR.
       49  WS-HOST-VAR-LEN          PIC    S9 (4) COMP.
       49  WS-HOST-VAR-TXT          PIC    X(100).
01 TVAL1.
       49  TVAL1-LEN          PIC    S9 (4) COMP.
       49  TVAL1-TXT          PIC    X(3).
01 TVAL2  PIC  X  (10 ).

 PROCEDUER DIVISION.

       MOVE 'SELECT S#, SNAME, STATUS, CITY FROM  S
          WHERE  S# = ? AND CITY= ?'
                TO  FLSQL.

 Move the  'SQL to execute' to WS-HOST-VARBLE.

       EXEC SQL
            DECLARE CRS1 CURSOR FOR FLSQL
       END-EXEC.
```

Figure: 4.8 Fixed-list Select

FIXED-LIST SELECT (Cont…)

```
EXEC SQL
      PREPARE FLSQL FROM :WS-HOST-VAR
END-EXEC.
```

Move the required values to the variables TVAL1 and TVAL2.

```
EXEC SQL
      OPEN CRS1 USING :TVAL1, :TVAL2
END-EXEC.
```

Loop until no more rows to fetch

```
EXEC SQL
    FETCH CRS1 INTO :S#,:SNAME, :STATUS, :CITY
END-EXEC.

EXEC SQL
   CLOSE CRS1
END-EXEC.
```

Figure: 4.9 Fixed-list Select (Cont…)

VARYING-LIST SELECT

Varying-list SELECT SQL statements are used when you do not know the columns that will be retrieved by an application program. This class of SQLs provide more flexibility than any other dynamic SQLs.

You can change tables, columns and predicated during the execution, since every thing about this SQL can change during execution, the number and type of host variables cannot be known beforehand. This is why, this class of SQL is the most complicated among the dynamic SQLs.

Figure: 4.10 Varying-list Select

SELECT STATEMENT WITH VARYING LIST

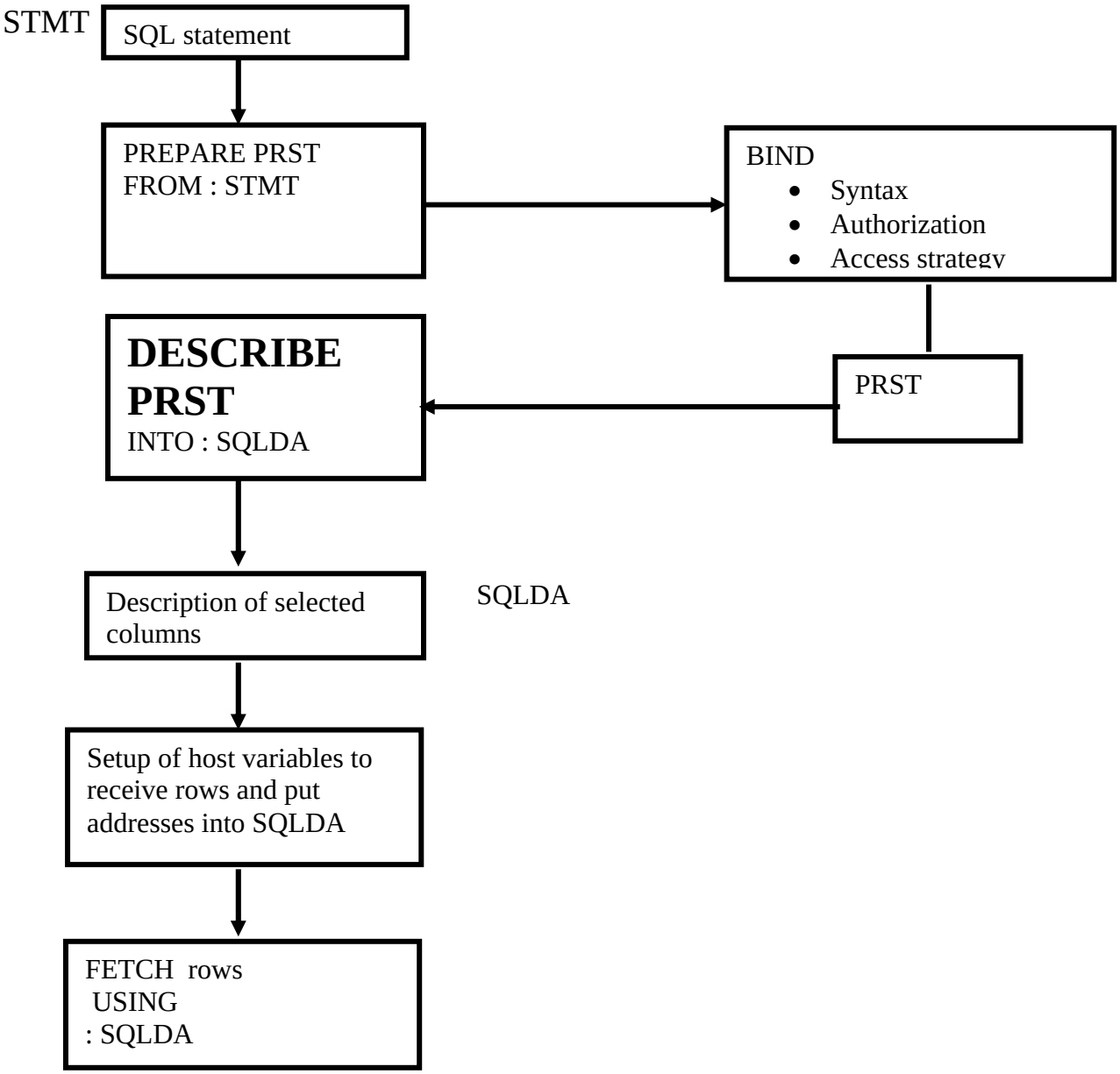

Figure: 4.11 select statement with varying list

SELECT STATEMENT WITH VARYING LIST (Cont...)

Since we don't know which columns will be accessed, we cannot pre-allocate any storage for host variables until we know that information. This will be the thing to do.

If we have an idea how many columns we will have to access, it will already simplify our case.

In Order to now the data types and names of the columns, we will have to ask DB2 to "DESCRIBE" the table we access. DB2 needs storage to be able to do this describe. If you know how many columns you will need, you can at least allocate enough storage for DB2 to be able to do the DESCRIBE. Once this is done, you can allocate storage for the host variables and execute the statements.

Figure: 4.12 select statement with varying list (Cont…)

SQL DESCRIPTOR AREA (SQLDA) - FUNCTION

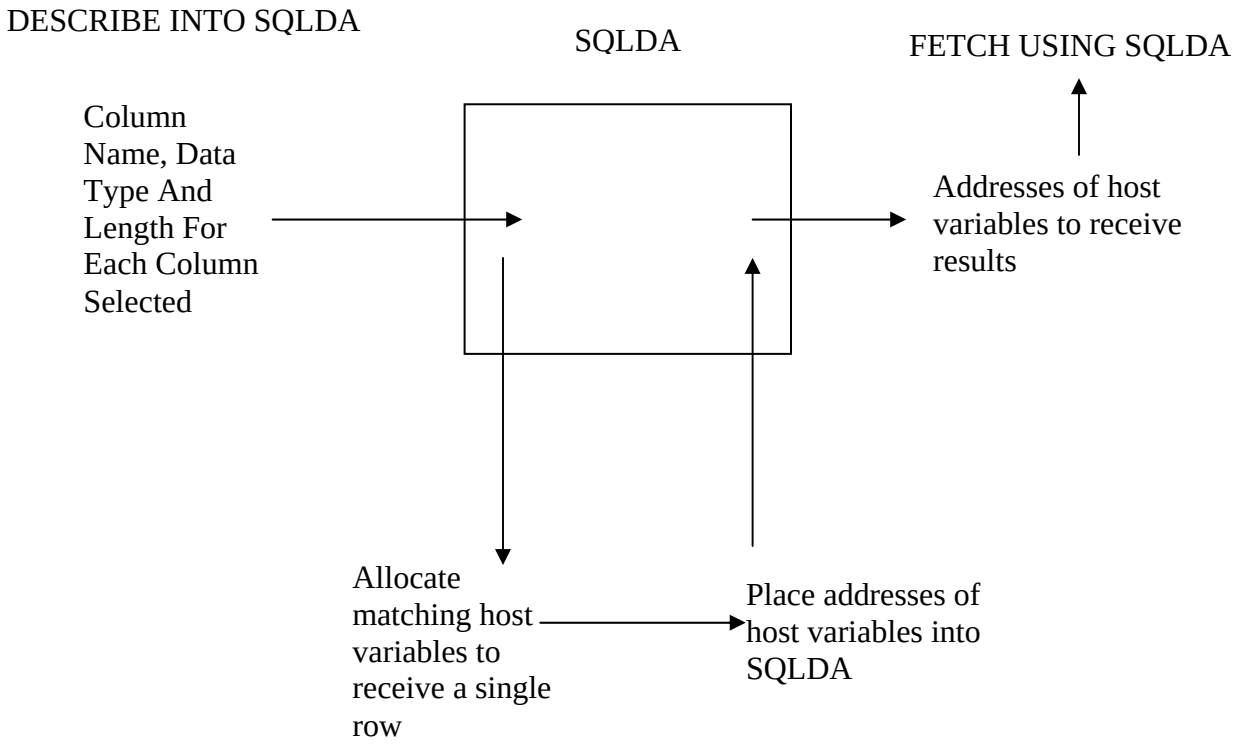

Figure: 4.13 SQL Descriptor Area (SQLDA)- Function

Notes:

This is the host structure used by DB2 “describe” an SQL statement (number of columns and data type of each column). This will be the input for the allocation of the host variable storage.

SQLDA - FORMAT

SQLDAID CHAR(8)	**SQLD ABC INTEGER**
SQLN SMALLINT	**SQLD SMALLINT**
SQLTYPE SMALLINT	**SQLLEN SMALLINT**
SQLDATA POINTER	**SQLIND POINTER**
SQLNAME VARCHAR (30)	
OTHER SQLVARS	

One Structure For Each Column Selected (SQLVAR)

Figure: 4.14 SQLDA- Format

Notes:

The SQLDA has a “header” containing some very useful information (number of columns, etc), and then for each column a descriptive are called SALVAR.

SQLDA – COLUMN DESCRIPTION

SQLTYPE	DATA TYPE	LENGTH
384	DATE, NOT NULL	Inst. Dep
388	TIME, NOT NULL	Inst. Dep.
392	TIMESTAMP, NOT NULL	26
448	VARCHAR, NOT NULL	Length
452	CHAR, NOT NULL	Length
480	FLOAT, NOT NULL	8
480	REAL, NOT NULL	4
484	DECIMAL, NOT NULL	2(prec/scale)
496	INTEGER, NOT NULL	4
500	SMALLINT, NOT NULL	2

SQLTYPE + 1 = same as SQLTYPE, WITH nulls allowed

Figure: 4.15 SQLDA – Column Description

Notes:

The SQLDA column description uses “SQL TYPES” to identify the different DB2 data types for columns. The Null attribute is also implicitly included.

SQLDA – Technique 1

Allocate SQLDA1 for pre-defined number of columns (e.g. sqln = 50)

SQLDAID	SQLDABC	SQLN	SQLD	
SQLTYPE	SQLLEN	SQLDATA	SQLIND	SQLNAME
: : : ……				

SQLN #50

DESCRIBE statement INTO SQLDA1

If less than 50 columns returned

Allocate host variable and indicator variable storage

- OPEN CURSOR and FETCH

If more than 50 columns returned

- **Read SQLD (# cols)**
- **Reallocate SQLDA2**
- **Describe statement INTO SQLDA2**

Figure: 4.16 SQLDA – Technique 1

SQLDA – Technique 1(Cont…)

The problem is that you don't always know in advance how many columns will be returned, nor do you know how many SQLVARs you should allocate for the DESCRIBE.

The first technique "guesses" that 50 will be about right.
If it was optimistic and more than 50 columns are returned, two DESCRIBES will be necessary, a first one to find out how many rows will be returned (SQLD contains this info) and a second one in a freshly allocated, correctly sized SQLDA.

Figure: 4.17 SQLDA - Technique 1

SQLDA – Technique 2

- Allocate “minimal” SQLDA1 (SQLN = 0)

SQLDAID	SQLDABC	SQLN	SQLD

- DESCRIBE statement INTO SQLDA1
- Read SQLD (#of columns returned)
- Allocate large enough SQLDA2 (SQLN = SQLD)

SQLDAID	SQLDABC	SQLN	SQLD	
SQLTYPE	SQLLEN	SQLDATA	SQLIND	SQLNAME
: : :				
…… ……				

SQLN = SQLD

- **DESCRIBE statement INTO SQLDA2**
- **Allocate host + Indicator variable storage**
- **OPEN CURSOR**
- **FETCH**

Figure: 4.18 SQLDA – Technique 2

Notes:

This is the case where you have no idea at all about the number of returned columns or where you want to use the same processing logic whatever that number may be.

SQLDA USAGE – SUMMARY

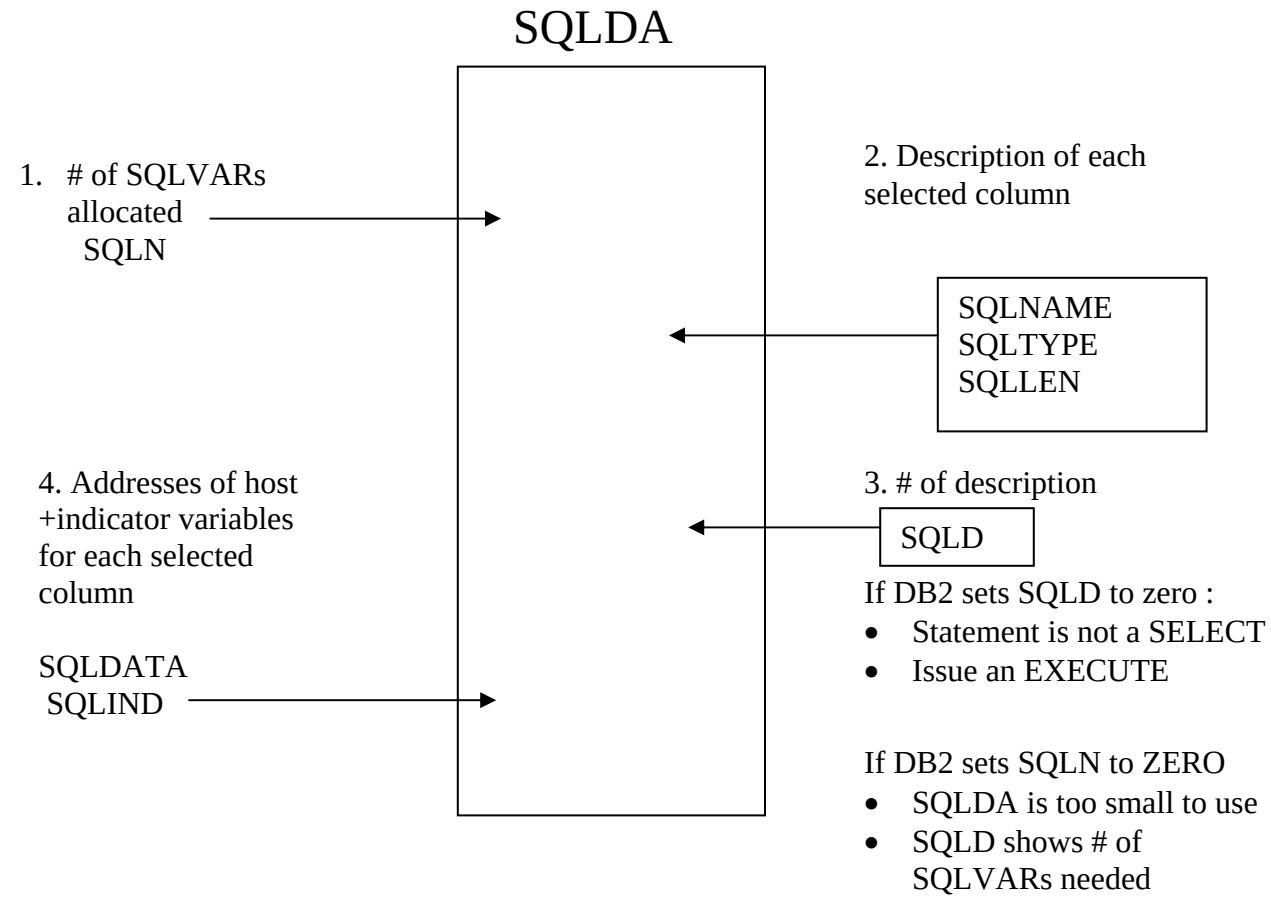

Figure: 4.19 SQLDA Usage – Summary

COMMIT AND ROLLBACK

COMMIT and ROLL BACK are not really database operations at all, in the sense that SELECT, UPDATE are database operations.
The COMMIT and ROLLBACK statements are not instructions to the database management system. Instead, they are instructions to the transaction manager and the transaction manager is certainly not a part of the DBMS
On the contrary, the DBMS is subordinate to the transaction manager, in the sense that the DBMS is just one of possibly several "resource managers" that provide services to the actions running under that transaction manager.

- **A transaction running under CICS can also use the services of three resources managers- IMS/DB, CICS and DB2. Here CICS acts as the transaction manager.**

- **In the TSO and 'pure batch' environments, where DB2 itself serves as the transaction manager, they are requested via the explicit SQL operators COMMIT and ROLLBACK**

Before getting details of COMMIT and ROLLBACK statements as such, we first need to know "SYNCHRONIZATION POINT' abbreviated as SYNCPOINT.

A SYNCPOINT represents a boundary point between two consecutive transactions, loosely speaking, it corresponds to the end of a logical unit of work and thus to a point at which the database is in a state of consistency. Program initiation, COMMIT and ROLLBACK each establish a synchpoint and no other operation does.

Figure: 4.20 Commit and Rollback

COMMIT AND ROLLBACK (Cont...)

COMMIT :

The SQL COMMIT statement takes the form

COMMIT (WORK)

A successful end-of-transaction is signaled and a synchpoint is established. All updates made by the program since the previous synchpoint are committed. All open cursors are closed, except for cursors whose declaration includes the optional specification WITH HOLD , all rowlocks are released, except for locks needed to hold position for cursors not closed.

ROLLBACK

The SQL ROLLBACK statement takes the form

ROLLBACK [WORK]

An unsuccessful end-of-transaction is signed and a synchpoint is established. All updates made by the program since the previous synchpoint are undone.
All open cursors are closed.

Figure: 4.21 Commit and Rollback (Cont…)