UNIT 3. CURSOR HANDLING

OVERVIEW

- **DEFINING CURSORS**
- **CURSOR DECLARATION**
- **UPDATING/DELETING WITH CURSORS**
- **READ ONLY CURSORS**

Figure 3.1 Cursor Handling

DEFINING CURSORS

Two types of embedded SQL SELECT STATEMENTS.

- **Singleton SELECT**
- **Cursor SELECT**

SQL statements operate on a set of data and return a set of data. Host language programs, on the other hand, operate on a row at a time.

A singleton SELECT is simply on SQL SELECT statement that returns a single row. The singleton SELECT differs from the ordinary SELECT statement in that it contains an INTO clause. The INTO clause is where you code your host variables that accept the data returned by DB2. But if such a SELECT statement returns more than one row, the values of the first row is placed in the host variable and DB2 issues an SQLCODE of –811. So, in your application program, if the SELECT will return more than one row then one must use Cursors.

DB2 uses cursors to navigate through a set of rows returned by an embedded SQL SELECT statement. A cursor can be compared to a pointer. The programmer declares a cursor and defines the SQL statement for the cursor. After that you can use the cursor like a sequential file. The cursor is opened, rows are fetched from the cursor, and one row at a time and at the end of processing the cursor is closed.

Figure 3.2 Defining Cursors

DEFINING CURSORS (Cont...)

The four operations that must be performed for the successful working of cursors.

- **DECLARE** - This statement defines the cursors, gives a name to it and assigns an SQL statement to it. The DECLARE statement does not execute the SQL statement but merely defines it.

- **OPEN** – This readies the cursor for row retrieval. OPEN is an executable statement. It reads the SQL search fields, executes the SQL statements and sometimes builds the result table.

- **FETCH** - This statement returns data from the result table one row at a time to the host variables. If the result table is not built at the OPEN time, it is built during FETCH.

- **CLOSE** – Releases all resources used by the cursor.

Figure 3.3 Defining Cursors (Cont…)

Notes:

SAMPLE PROGRAM FOR CURSOR

```
WORKING-STORAGE SECTION.

        EXEC SQL
             INCLUDE dclgenmem
        END-EXEC.
        EXEC SQL
             INCLUDE SQLCA
        END-EXEC.

PROCEDURE DIVISION.

        EXEC SQL
             DECLARE CUR CURSOR FOR SELECT * FROM S
        END-EXEC.
        EXEC SQL
             OPEN CUR
        END-EXEC.
        PERFORM FETCH-PARA UNTIL SQLCODE = 100.

FETCH-PARA.

        EXEC SQL
             FETCH CUR INTO :DCLS
        END-EXEC.

        IF SQLCODE = 100
             EXEC SQL
                  CLOSE CUR
             END-EXEC
             STOP RUN
        END-IF.

        DISPLAY  DCLS.
```

CURSOR DECLARATION

```
EXEC SQL DECLARE CURSOR-NAME
CURSOR (WITH HOLD) FOR select-statement
```

- **A Cursor :**
  - **Is required for select of multiple rows**
  - **Is never used for INSERT**
  - **May be reused (CLOSE + new OPEN)**
  - **Will be closed at COMMIT**
  - **Unless declared with "WITH HOLD"**

- **Multiple Cursors :**
  - **May be defined in a program**
  - **May work with the same table**
  - **May be open simultaneously**

Figure: 3.4 Cursor Declarations

Notes:
The DECLARE CURSOR statement is used to associate a cursor with the SQL statement. No data access is performed at this stage.

The WITH HOLD option is important if you want to issue COMMIT inside the fetch loop. If you omit this keyword, you will have to re-open (and DB2 will have to re-execute the access path) the cursor after each COMMIT, because its position will be lost due to the COMMIT.

UPDATING /DELETING WITH THE CURSORS

Row-at a time UPDATE OR DELETE (‘Positioned’ Updates)

- Declare
  FOR UPDATE OF…….option

```
EXEC SQL DECLARE CUR1 CURSOR FOR
        SELECT EMPNO, LASTNAME
            FROM EMP
            WHERE WORKDEPT =: DPT
            FOR UPDATE OF LASTNAME
```

- and use
  WHERE CURRENT OF…option

```
EXEC SQL OPEN CUR1
EXEC SQL FETCH CUR1 INTO: EMPNO, NAME
IF…
    EXEC SQL UPDATE EMP
                SET LASTNAME =: NEWNAME
                WHERE CURRENT OF CUR1.

    EXEC SQL FETCH CUR1 INTO: EMPNO, NAME
    IF …THEN
        EXEC SQL DELETE FROM EMP
                        WHERE CURRENT OF CUR1.
```

Figure: 3.5 Updating /Deleting With The Cursors

Notes:

Once the cursor is positioned on a row, you can do a ‘Positioned’ update or delete. It means that the update/delete does’nt require that DB2 re-access the data and it would have to do if you used “delete where key =”.

READ-ONLY CURSORS

EXEC SQL DECLARE cursor-name CURSOR FOR select-statement FOR FETCH/READ ONLY

- **The SELECT Statement contains**
  - **FOR FETCH ONLY/FOR READ ONLY**
  - **ORDER BY**
  - **UNION or UNION ALL**

- **The first SELECT contains**
  - **DISTINCT**
  - **A FUNCTION OR EXPRESSION**

- **The outer subselect contains**
  - **GROUP BY / HAVING**

- **The same table is used in SELECT statement and SUBQUERY**

- **Select from multiple tables (join)**

- **A nested table expression is used in the first from**

- **Isolation UR is used without FOR UPDATE OF**

Figure: 3.6 Read-Only Cursors

READ-ONLY CURSORS (Cont...)

A Read-only cursor cannot be the target of a positioned update/delete

However, some cursors are "read/only". The reason behind this is that for some SQL statements, it will not be possible for DB2 to 'position' itself before the first result row without accessing the entire result set. Take the example of a SELECT…….ORDER BY. DB2 will have to get all qualifying rows and sort them before it can position its cursor. In addition, the cursor will be positioned on an intermediate result table and not on the actual data itself.

Figure: 3.7 Read-Only Cursors (Cont..)