Writing a Db2 COBOL Cursor Update Program

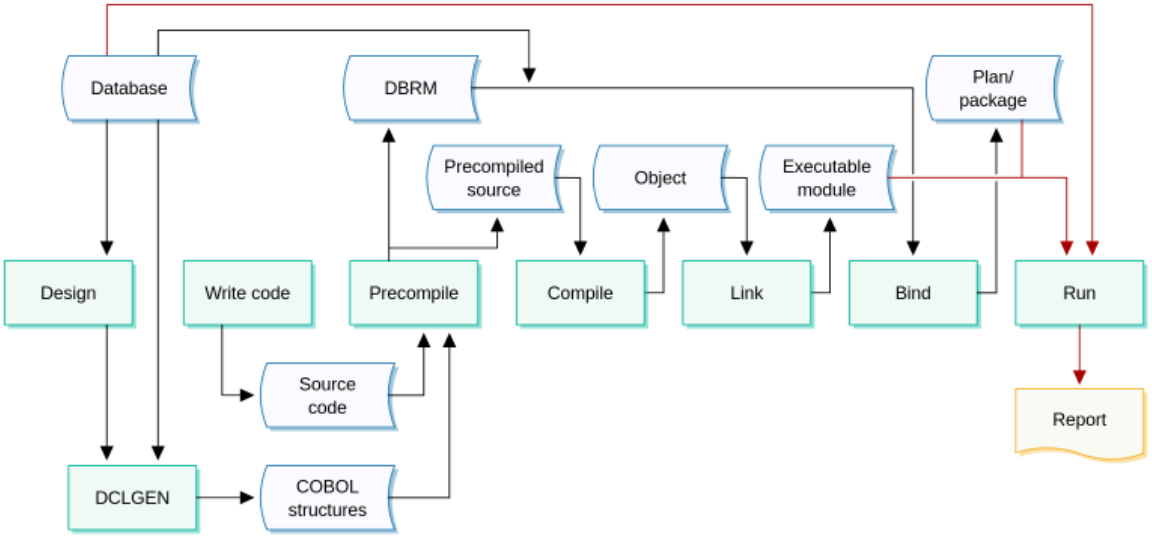
Objectives

Writing a Db2 COBOL Cursor Update Program

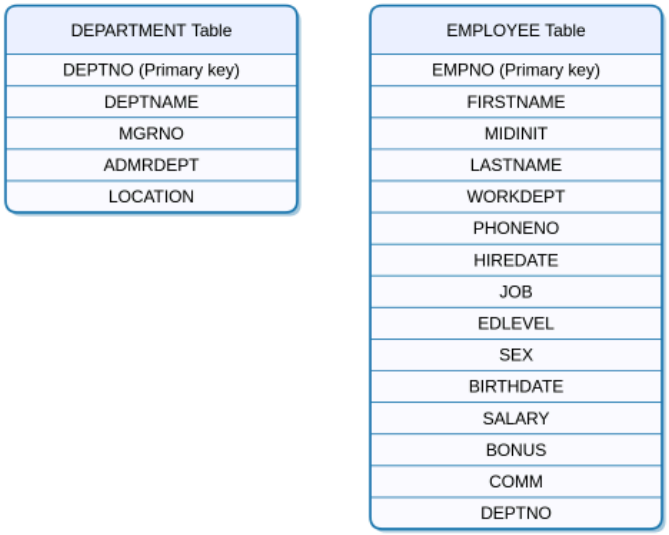
In this module you will be building on your COBOL Db2 program. You are going to change your program to use a cursor and then update the cursor.

After completing this module you will be able to:

- Write a Db2 COBOL Program Using a Cursor
- Write a Db2 COBOL Update Program Using a Cursor



The process of building a cursor program is the same as for any Db2 program. Only the program logic and Db2 structures are different.



This exercise will carry on from the previous course where we used the two tables shown here.

We are going to list the contents of the DEPARTMENT table in preparation for updating the MGRNO field. This update will only need to occur if the WORKDEPT value in the EMPLOYEE table is equal to the DEPTNAME value in the DEPARTMENT table.

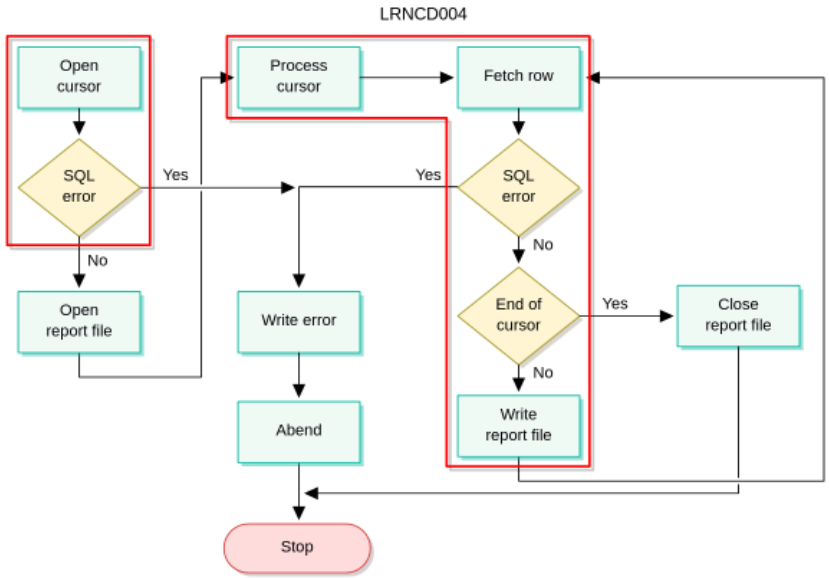
Because our SQL statements will need to process multiple rows, we are going to use a cursor.

```
       WORKING-STORAGE SECTION.

           EXEC SQL
              INCLUDE DEPART
           END-EXEC.

           EXEC SQL
              INCLUDE EMPEE
           END-EXEC.

           EXEC SQL
              INCLUDE SQLCA
           END-EXEC.


       01  PROGRAM-FIELDS.
           05  WS-ZERO                 PIC 9 VALUE 0.
           05  WS-RCODE                PIC  9(04) COMP.
           05  FILEERR-STATUS.
               10  FILEERR-STATUS1     PIC  X(01).
               10  FILEERR-STATUS2     PIC  X(01).
               10  FILEERR-STATUSX2N   PIC  9(03).
           10 WS-NUMBER-OF-DEPTS       PIC S9(4) USAGE COMP.
           10 WS-NUMBER-OF-EMPLOYEES   PIC S9(4) USAGE COMP.
```

This is the simple program you have already used which performs three SQL statements.

You will be using these tables and the Db2 interface is the same, so you do not need additional DCLGENs.

Scroll the window to view all the code.

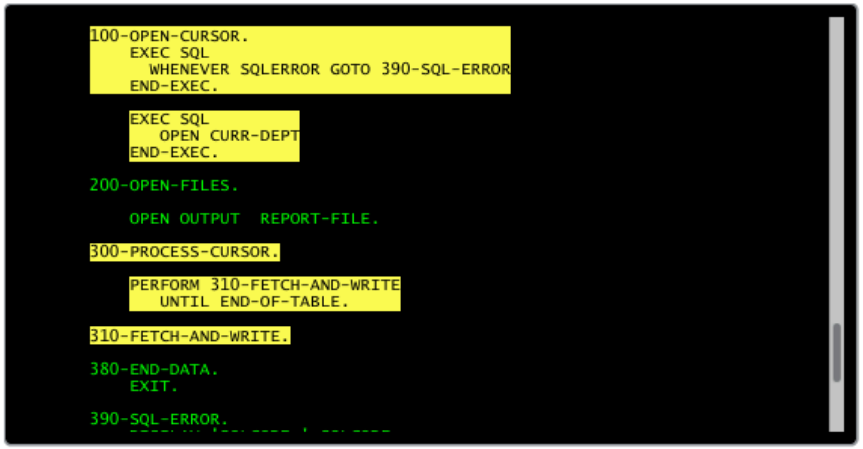


There are some changes in the design of your program.

In your simple program you just issued an SQL statement. Now you must open a cursor and then loop through the cursor, fetching each row. When you have done so, you will close the cursor.

Note: You must also cater for an error when you open the cursor.

Click **Play** to see an example of this concept.

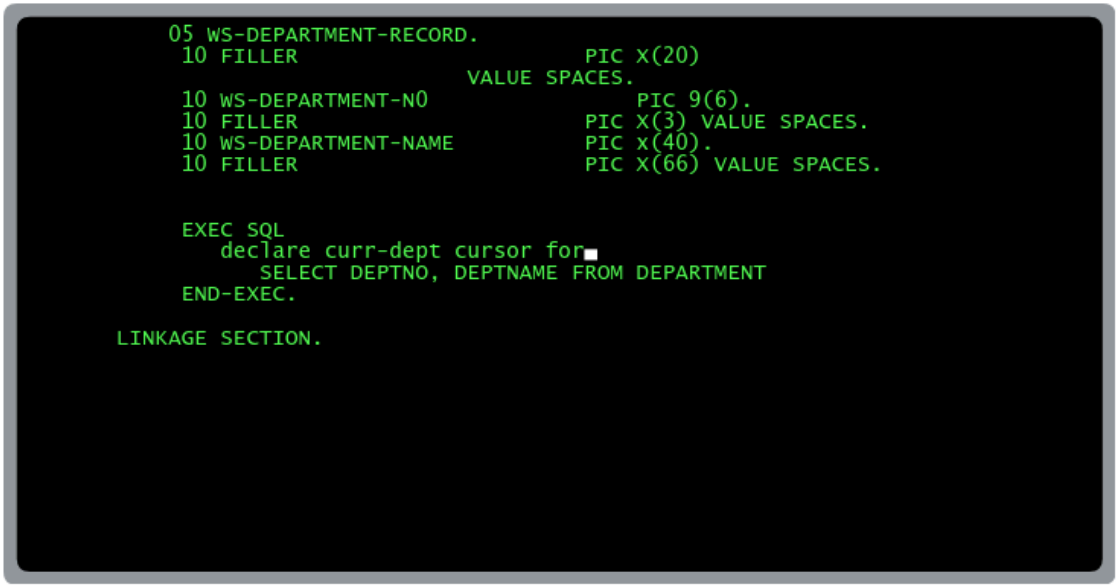


```
100-OPEN-CURSOR.
    EXEC SQL
      WHENEVER SQLERROR GOTO 390-SQL-ERROR
    END-EXEC.

    EXEC SQL
       OPEN CURR-DEPT
    END-EXEC.

200-OPEN-FILES.

    OPEN OUTPUT  REPORT-FILE.

300-PROCESS-CURSOR.

    PERFORM 310-FETCH-AND-WRITE
       UNTIL END-OF-TABLE.

310-FETCH-AND-WRITE.

380-END-DATA.
    EXIT.

390-SQL-ERROR.
```

Here is the structure of your COBOL program. It has been modified by adding extra paragraphs for cursor processing, and some extra flag variables for program control.

The first item you are going to add is the cursor, which must be declared in working storage.

Scroll the window to view all the code.



Step 2 of 3

You insert the SQL into the working storage section using an EXEC SQL block, but you must declare it as a cursor. You will declare it as a cursor named CURR-DEPT.

Type **DECLARE CURR-DEPT CURSOR FOR** and **press** **Enter**.

```
       100-OPEN-CURSOR.
           EXEC SQL
             WHENEVER SQLERROR GOTO 390-SQL-ERROR
           END-EXEC.

           EXEC SQL
              OPEN CURR-DEPT
           END-EXEC.

       200-OPEN-FILES.

....
....


       910-CLOSE-CURSOR.
           EXEC SQL
              CLOSE CURR-DEPT
           END-EXEC.

       990-ABEND.
           DIVIDE 1 BY WS-ZERO GIVING WS-ZERO.

       9900-EXIT.
           EXIT.
```

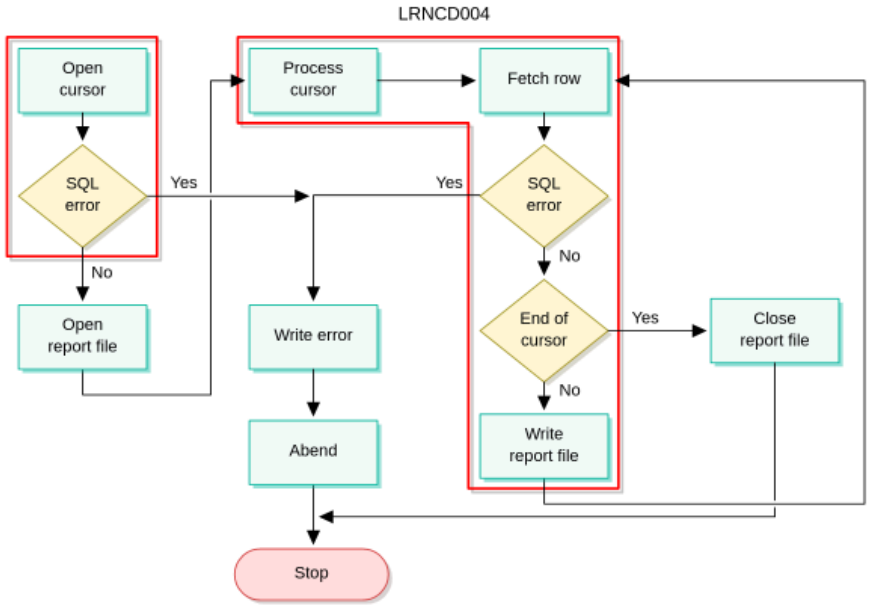
Step 3 of 3

Now you will code the OPEN and CLOSE cursor statements. Cursors are opened and closed by name. Note that Db2 can generate an error on an open cursor, so be sure to check the status or implement a WHENEVER clause, as was done here.

Type **OPEN CURR-DEPT** and **CLOSE CURR-DEPT** and **press** **Enter**.

That is incorrect

Try Again



Now you can code the loop to fetch and process each row, and also to sense when you have processed all rows and exit the loop.

```
300-PROCESS-CURSOR.


    PERFORM 310-FETCH-AND-WRITE
       UNTIL END-OF-TABLE.

310-FETCH-AND-WRITE.
    INITIALIZE DCLDEPARTMENT.
    PERFORM 320-FETCH-NEXT-ROW.
    IF SQLCODE = 100
       MOVE WC-TRUE to WF-END-OF-TABLE-FLAG
    ELSE
       PERFORM 330-WRITE-RECORD
    END-IF.

320-FETCH-NEXT-ROW.
    EXEC SQL
        FETCH CURR-DEPT
        INTO :DEPTNO,
             :DEPTNAME
    END-EXEC.

330-WRITE-RECORD.
    MOVE DEPTNO                 TO WS-DEPARTMENT-NO.
    MOVE DEPTNAME               TO WS-DEPARTMENT-NAME
```

Step 2 of 2

```
FETCH cursor-name
INTO :host-var1
     :host-var2
```

You know the cursor is called CURR-DEPT, the host variable is DEPTNO, and the structure is DEPTNAME. **Type** the FETCH statement using the format here, and **press** **Enter**.

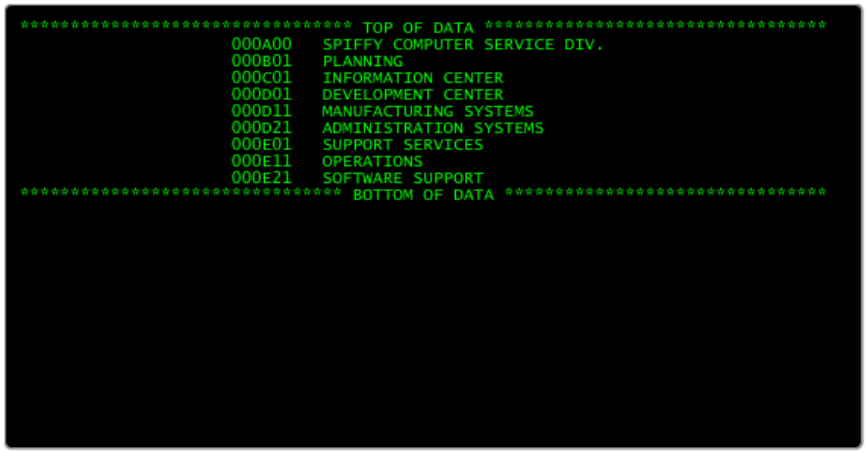
That is incorrect

Try Again

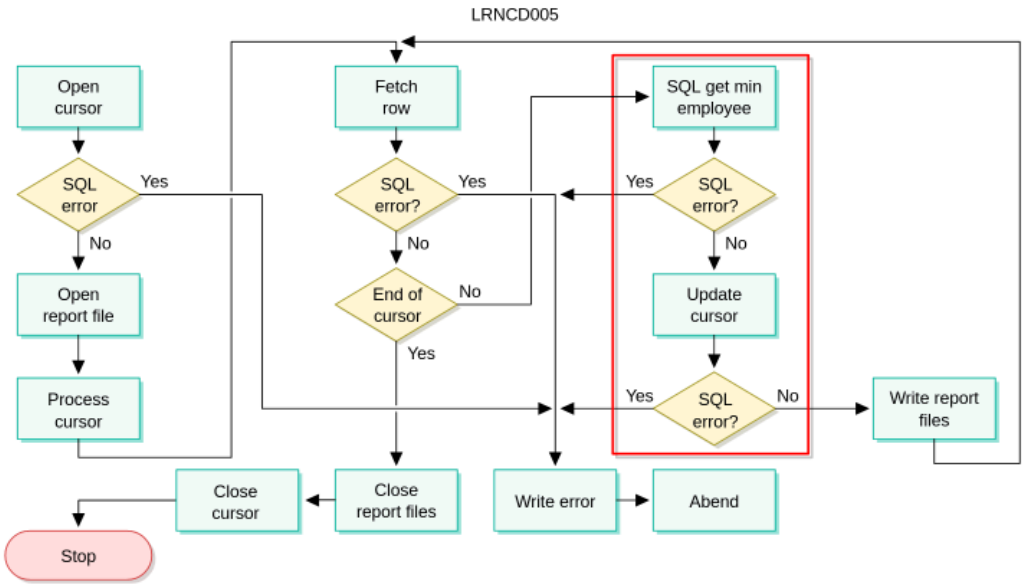
```
310-FETCH-AND-WRITE.
    INITIALIZE DCLDEPARTMENT.
    PERFORM 320-FETCH-NEXT-ROW.
    IF SQLCODE = 100
       MOVE WC-TRUE to WF-END-OF-TABLE-FLAG
    ELSE
       PERFORM 330-WRITE-RECORD
    END-IF.

320-FETCH-NEXT-ROW.
    EXEC SQL
        FETCH CURR-DEPT
        INTO  :DEPTNO,
              :DEPTNAME
    END-EXEC.

330-WRITE-RECORD.
    MOVE DEPTNO                 TO WS-DEPARTMENT-NO.
    MOVE DEPTNAME-TEXT          TO WS-DEPARTMENT-NAME

    MOVE    WS-DEPARTMENT-RECORD          TO  PRINT-AREA.
    WRITE   REPORT-RECORD  AFTER 1.

380-END-DATA.
```

Here is your completed program. Now you will run it and see the output.

Scroll the window to view all the code.



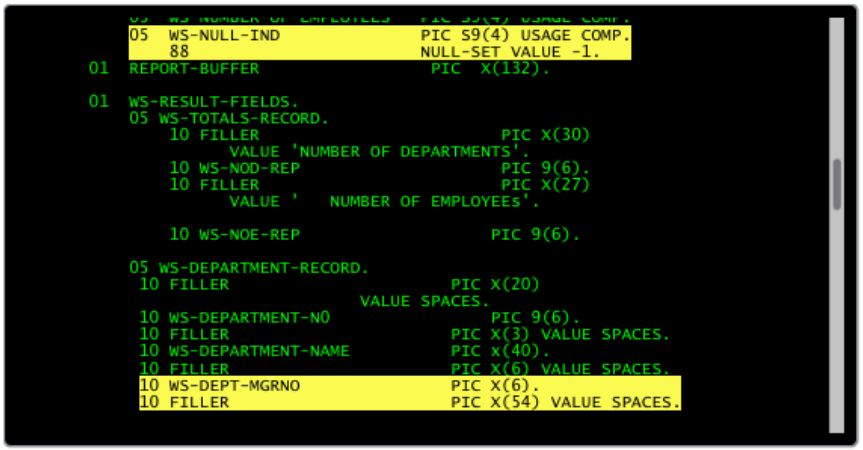
You have produced a list of all the departments held in the department table.



The next step in our exercise is to update the MGRNO column in each row of the DEPARTMENT table, but only if the WORKDEPT value in the EMPLOYEE table is equal to the DEPTNAME value in the DEPARTMENT table.

The value used in the update process will be obtained from the minimum EMPNO value relating to each department.

If for some reason, departments do not have any employees (could be a new structure being set-up, or a redundant department), you will need to set the MGRNO value to '000000'.



```
05  WS-NULL-IND                 PIC S9(4) USAGE COMP.
    88                          NULL-SET VALUE -1.
01  REPORT-BUFFER                PIC  X(132).

01  WS-RESULT-FIELDS.
    05 WS-TOTALS-RECORD.
       10 FILLER                           PIC X(30)
             VALUE 'NUMBER OF DEPARTMENTS'.
       10 WS-NOD-REP                       PIC 9(6).
       10 FILLER                           PIC X(27)
             VALUE '   NUMBER OF EMPLOYEES'.

       10 WS-NOE-REP                    PIC 9(6).

    05 WS-DEPARTMENT-RECORD.
     10 FILLER                      PIC X(20)
                              VALUE SPACES.
     10 WS-DEPARTMENT-NO                 PIC 9(6).
     10 FILLER                     PIC X(3) VALUE SPACES.
     10 WS-DEPARTMENT-NAME         PIC x(40).
     10 FILLER                     PIC X(6) VALUE SPACES.
     10 WS-DEPT-MGRNO              PIC X(6).
     10 FILLER                     PIC X(54) VALUE SPACES.
```

Your program is shown here with a few changes. Fields have been added: one to hold a null indicator and others to display the updated MGRNO.

You must also change the way you declare the cursor and code the update logic.

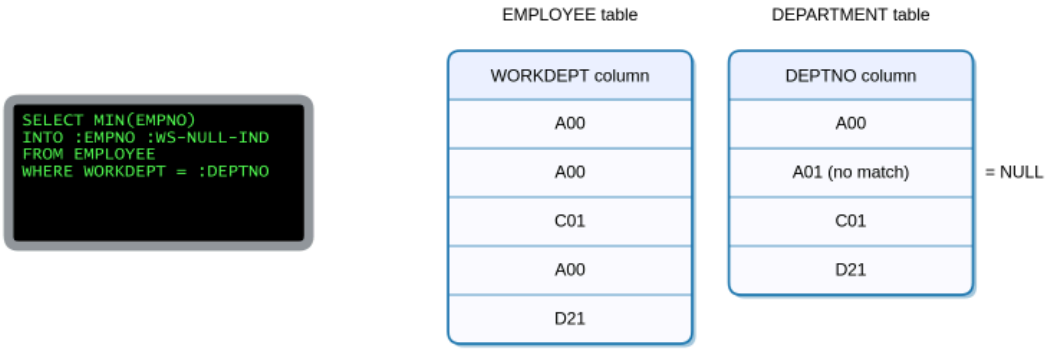
Scroll the window to view all the code.

```
SELECT MIN(EMPNO)
INTO :EMPNO
FROM EMPLOYEE
WHERE WORKDEPT = :DEPTNO
```

In the next part of our exercise, you need to obtain the minimum EMPNO value relating to each department. The following is the additional criteria required for this statement:

- It needs to be retrieved from the EMPLOYEE table and stored in the :EMPNO host variable.
- A WHERE statement needs to be added so that the data is only stored if the WORKDEPT value in the EMPLOYEE table is equal to the :DEPTNO host variable that was set in an earlier SELECT statement.

An example of what is required is shown here.



We also need to cater for a NULL condition if there are no employee rows whose WORKDEPT value matches :DEPTNO. In this situation we can provide a null indicator variable as shown above.

```
       325-UPDATE-ROW.
            EXEC SQL
               SELECT MIN(EMPNO)
               INTO :EMPNO :WS-NULL-IND
               FROM EMPLOYEE
               WHERE WORKDEPT = :DEPTNO
            END-EXEC.
......
       330-WRITE-RECORD.
            MOVE DEPTNO                 TO WS-DEPARTMENT-NO.
            MOVE DEPTNAME-TEXT          TO WS-DEPARTMENT-NAME.
            MOVE MGRNO                  TO WS-DEPT-MGRNO.

            MOVE    WS-DEPARTMENT-RECORD          TO  PRINT-AREA.
            WRITE   REPORT-RECORD  AFTER 1.

       380-END-DATA.
            EXIT.
.....
```

Step 1 of 4

```
INTO :EMPNO :WS-NULL-IND
FROM EMPLOYEE
WHERE WORKDEPT = :DEPTNO
```

Type the above SELECT statement into the program and **press** **Enter**.

That is incorrect

Try Again

```
       325-UPDATE-ROW.
            EXEC SQL
               SELECT MIN(EMPNO)
               INTO  :EMPNO :WS-NULL-IND
               FROM EMPLOYEE
               WHERE WORKDEPT = :DEPTNO
            END-EXEC.
            IF NOT NULL-SET
               MOVE EMPNO TO MGRNO
            ELSE
               MOVE '000000' TO MGRNO
            END-IF.

            EXEC SQL
              UPDATE DEPARTMENT
                 SET MGRNO = :MGRNO
              where current of curr-dept
            END-EXEC.

      330-WRITE-RECORD.
            MOVE DEPTNO                 TO WS-DEPARTMENT-NO.
            MOVE DEPTNAME-TEXT          TO WS-DEPARTMENT-NAME.
            MOVE MGRNO                  TO WS-DEPT-MGRNO.
.....
```

Step 4 of 4

You only want to update the current cursor row. To do this, you must restrict the update using a WHERE CURRENT OF cursor-name clause.

Your cursor name is CURR-DEPT.

Type the WHERE statement above and **press** **Enter**.

Try Again

```
           WRITE REPORT-RECORD AFTER 1.

       380-END-DATA.
           EXIT.

       390-SQL-ERROR.
           DISPLAY 'SQLCODE ' SQLCODE.
           DISPLAY 'SQLSTATE' SQLSTATE.
           PERFORM  990-ABEND.


       900-CLOSE-FILES.

           CLOSE   REPORT-FILE.

       910-CLOSE-CURSOR.
           EXEC SQL
              CLOSE CURR-DEPT
           END-EXEC.

       990-ABEND.
           DIVIDE 1 BY WS-ZERO GIVING WS-ZERO.

       9900-EXIT.
           EXIT.
```

You have now completed your cursor update program. You can browse the complete program above.

If you precompile, compile, link, and bind the program, it is now ready to run.

Scroll the window to view all the code.