4

DECLARE AND USE CURSORS

SECTION OBJECTIVES

At the end of this section, you should be able to:

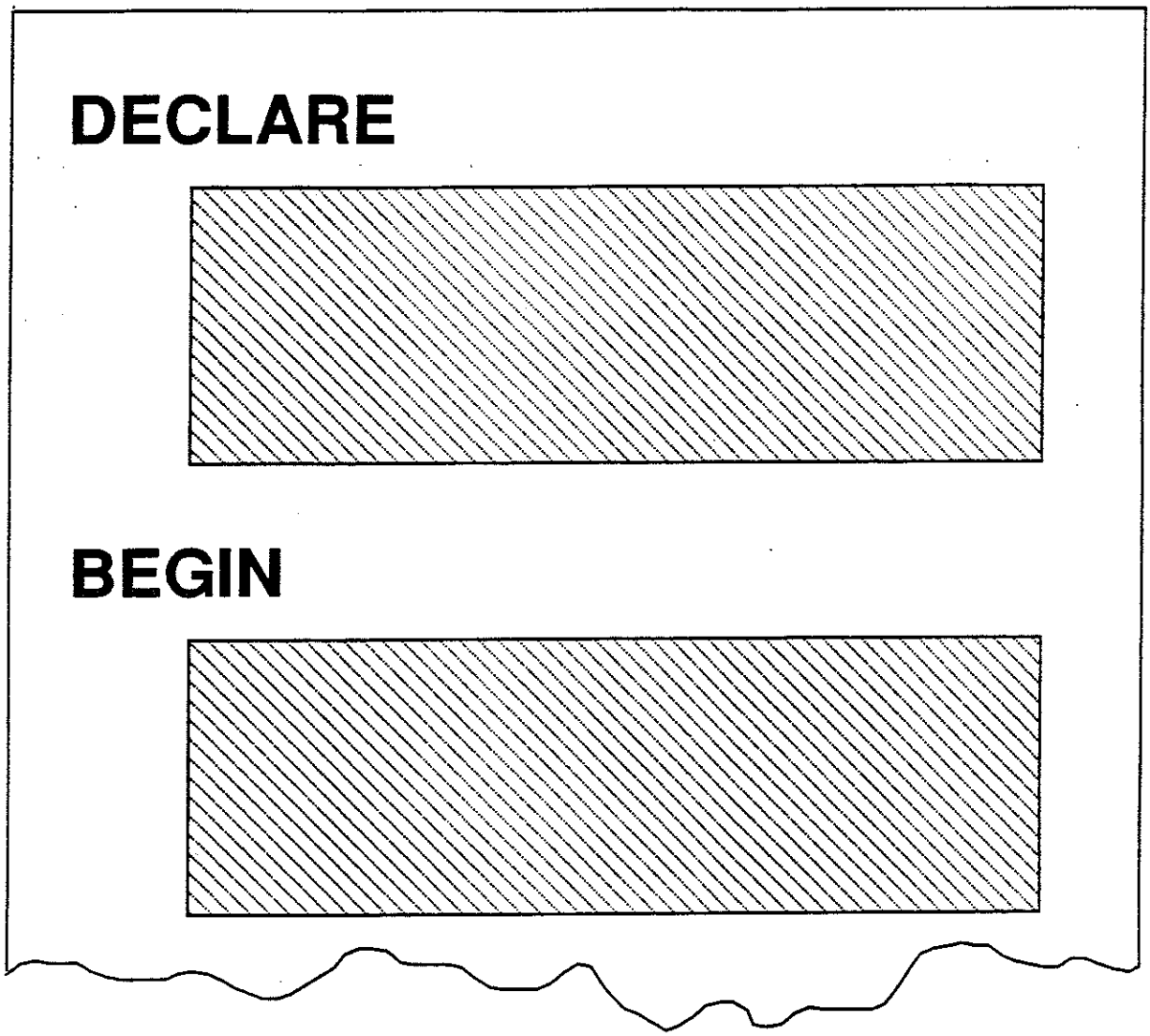
1 Identify the two types of cursors supported by PL/SQL.

2 Outline the procedural steps for using explicit cursors in PL/SQL blocks.

3 Demonstrate an understanding of PL/SQL cursors by writing a simple PL/SQL procedure using explicit cursors

4 Utilize implicit cursor attributes.

DECLARE AND USE CURSORS: OVERVIEW



Declare and Use Cursors: Overview—cont'd

Every SQL DML statement processed by PL/SQL has an associated cursor.

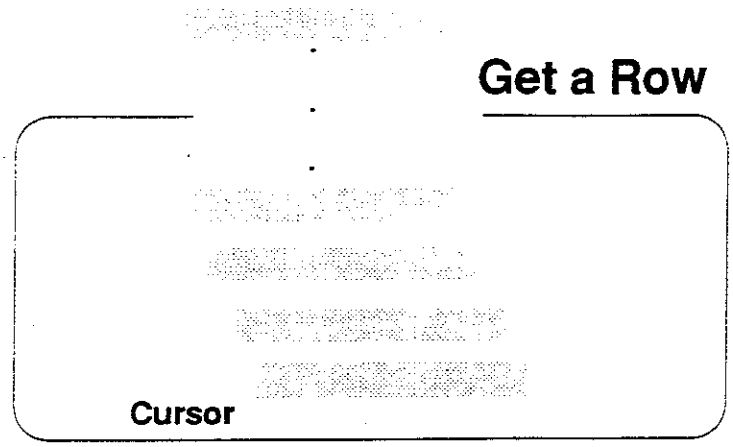
SQL Statements with Associated Cursor

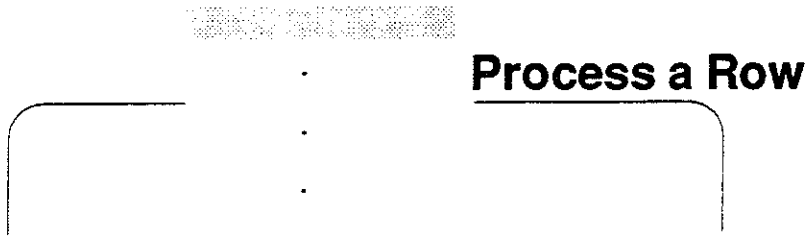
- INSERT
- UPDATE
- DELETE
- SELECT...INTO
- COMMIT
- ROLLBACK

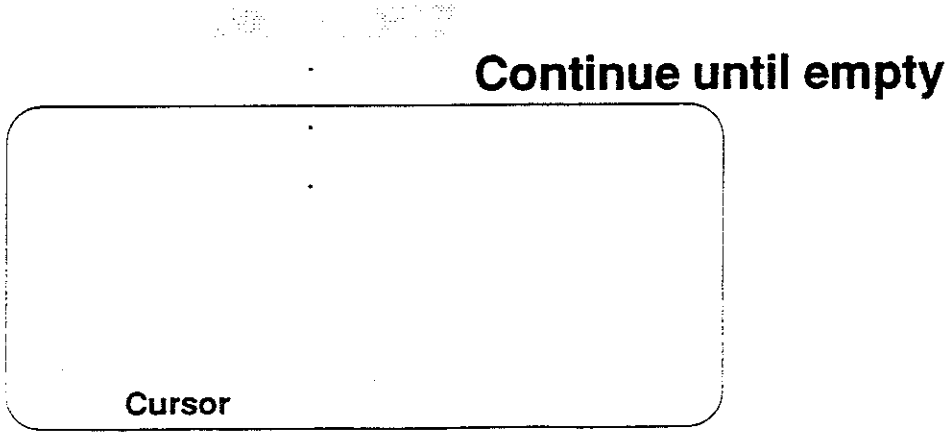
Types of Cursors

Type	Description
EXPLICIT	Multiple row SELECT statements
IMPLICIT	All INSERT statements All UPDATE statements All DELETE statements Single row SELECT...INTO statements

DECLARE AND USE EXPLICIT CURSORS







Declare and Use Explicit Cursors—cont'd

The set of rows returned by a query can consist of zero, one, or many rows, depending upon the number of rows that meet the query's search condition.

When a query returns multiple rows, a cursor can be explicitly defined to:

- process beyond the first row returned by the query.
- keep track of which row is currently being processed.

Steps to Declare and Use a Cursor

1 Declare the cursor

2 Open the cursor

3 Fetch data from the cursor

4 Close the cursor

Declare and Use Explicit Cursors—cont'd

Declare the cursor to associate its name with a SELECT statement.

- → Declare Cursor
- Open Cursor
- Fetch Data
- Close Cursor

Declare the Cursor

```
DECLARE
      CURSOR <cursor_name>
            IS <regular_select_statement>;
```

The <regular SELECT statement> must not include the INTO clause.

Declared cursors are scoped just as variables are.

Names a cursor and associates a query with it.

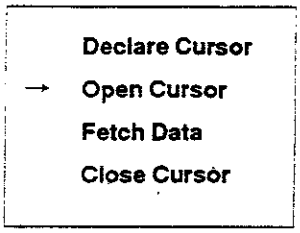
Example - CURSOR declaration

```
DECLARE
      v_ename                 VARCHAR2(20)
      x                       NUMBER(7,2);
      total                   NUMBER(5);
      lower_sal_limit         CONSTANT
                              NUMBER(4) := 1200;
      CURSOR c1 IS SELECT ename FROM emp
            WHERE sal > lower_sal_limit;
BEGIN...
```

```
BEGIN
    OPEN c1;
    LOOP
      fetch c1 into v_ename;
      -- processing
      exit when c1%notfound;
    END LOOP;
    CLOSE c1;
END;
```

Declare and Use Explicit Cursors—cont'd

Open the cursor to process the SELECT statement and store the returned rows in the cursor.



Open the Cursor

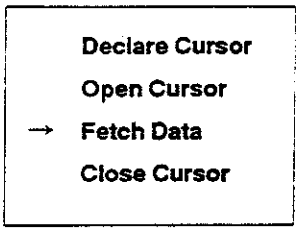
```
OPEN <cursor name>;
```

Evaluates the SELECT statement associated with the cursor.

Allocates resources used by ORACLE to process the query.

Identifies the active set.

Fetch data from the cursor and store it in specified variables.



Fetch Data

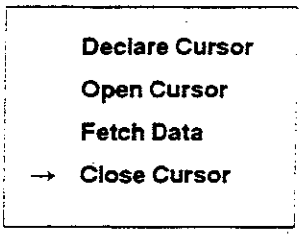
```
FETCH <cursor name> INTO <var1, var2...>;
```

There must be exactly one INTO variable for each column selected by the SELECT statement.

The first column gets assigned to *var1*, the second assigned *var2*, and so on.

Declare and Use Explicit Cursors—cont'd

Close the cursor to free up resources.



Close the Cursor

```
CLOSE <cursor name>;
```

Marks resources held by opened cursor as reusable.

No more rows can be fetched from a closed cursor.

REFERENCE EXPLICIT CURSOR ATTRIBUTES

%NOTFOUND evaluates to TRUE if prior FETCH returned no rows.

Example - %NOTFOUND

```
LOOP
      FETCH my_cursor INTO my_ename, my_sal;
      EXIT WHEN my_cursor%NOTFOUND;
         -- process data here
END LOOP;
```

%FOUND evaluates to TRUE if prior FETCH returned a row

Example - %FOUND

```
FETCH my_cursor INTO my_ename, my_sal;
WHILE my_cursor%FOUND LOOP
      -- process data here
      FETCH my_cursor INTO my_ename, my_sal;
END LOOP;
```

Reference Explicit Cursor Attributes—cont'd

%ROWCOUNT evaluates to the total number of rows FETCHed thus far.

Example - %ROWCOUNT

```
LOOP
     FETCH my_cursor INTO my_ename, my_sal;
     EXIT WHEN (my_cursor%NOTFOUND)
               OR  (my_cursor%ROWCOUNT > 10);
        -- process data here
END LOOP;
```

%ISOPEN evaluates to TRUE if the associated cursor is open.

Example - %ISOPEN

```
IF my_cursor%ISOPEN THEN
     FETCH my_cursor INTO my_ename, my_sal;
ELSE
     OPEN my_cursor;
END IF;
```

Reference Explicit Cursor Attributes—cont'd

Example

```
DECLARE
     sal_limit       NUMBER(4) := 0;
     my_ename        emp.ename%TYPE;
     my_sal          emp.sal%TYPE;
     CURSOR my_cursor IS SELECT ename, sal FROM emp
          WHERE sal > sal_limit;
BEGIN
     sal_limit := 1200;
     OPEN my_cursor; -- uses 1200 as sal_limit
     LOOP
        FETCH my_cursor INTO my_ename, my_sal;
        EXIT WHEN my_cursor%NOTFOUND; -- nothing returned
        INSERT INTO new_table VALUES (my_ename, my_sal);
     END LOOP;
     CLOSE my_cursor;
     COMMIT;
END;
```

Reference Explicit Cursor Attributes—cont'd

Cursors must be closed before they can be reopened.

Note: Due to PL/SQL's cursor handling mechanism, closing a cursor and then immediately reopening it will not cause the SELECT statement to be re-parsed.

Example - Reopen a cursor

```
DECLARE
      total_bonus     NUMBER(8,2) := 0;
      min_sal         emp.sal%TYPE := 1000; -- starting guess
      bonus_amt       emp.sal%TYPE;
      CURSOR bonus IS
         SELECT sal * .10 FROM emp WHERE sal > min_sal;
BEGIN
      OPEN bonus; -- uses min_sal value of 1000
      LOOP
         FETCH bonus INTO bonus_amt;
         EXIT WHEN bonus%NOTFOUND;
         total_bonus := total_bonus + bonus_amt;
         IF total_bonus > 2000 THEN
            /* up the minimum and try again */
               min_sal := min_sal + 500;
               total_bonus := 0  -- reset the total
               CLOSE bonus;
               OPEN bonus;  -- re-open with new min_sal
         END IF;
      END LOOP;
         -- you may want to store min_sal somewhere
END;
```

REFERENCE THE CURRENT CURSOR ROW

Reference the current row with the WHERE CURRENT OF statement.

Syntax

```
WHERE CURRENT OF <cursor_name>
```

The cursor must be declared with a FOR UPDATE OF clause.

Example

```
DECLARE
CURSOR my_cur IS SELECT ename, sal FROM emp
     FOR UPDATE of sal;
emp_name    emp.ename%TYPE;
salary      emp.sal%TYPE;
BEGIN
OPEN my_cur;
LOOP
     FETCH my_cur INTO emp_name, salary;
     EXIT WHEN my_cur%NOTFOUND;
     IF salary > 3000 THEN
        DELETE FROM emp WHERE CURRENT OF my_cur;
     END IF;
     ...
END LOOP;
```

REFERENCE CURSOR FOR LOOPS

Specify a sequence of statements to be repeated once for each row that is returned by the cursor with the cursor FOR loop.

Syntax—Cursor FOR loop

```
FOR <record_name> IN <cursor_name> LOOP
        -- statements to be repeated go here
END LOOP;
```

Quick Notes

- Cursor FOR loops are similar to numeric FOR loops.
- Cursor FOR loops specify a set of rows from a table using the cursor's name. Numeric FOR loops specify an integer range.
- A cursor FOR loop record takes on the values of each row. A numeric FOR loop index takes on each value in the range.
- *record_name* is implicitly declared as:

```
record_name    cursor_name%ROWTYPE;
```

- To reference an element of the record, use the *record_name.column_name* notation.

Declare
cursor c_emp is
select empno, ename, sal from emp where sal>2000;

empno	ename	sal
17369	SMITH	2500

X emp c_emp%rowtype;

Begin
X open c_emp
for r_emp in c_emp loop
X fetch c_emp into c_emp;
X exit when c_emp%notfound;
--possessary
end loop
X close c_emp

X - unbelievable

Reference Cursor FOR Loops—cont'd

Conceptual Cursor Loop Model

- When a cursor loop is initiated, an implicit OPEN *cursor_name* is executed.
- For each row that satisfies the query associated with the cursor, an implicit FETCH is executed into the components of *record_name*.
- When there are no more rows left to FETCH, an implicit CLOSE *cursor_name* is executed and the loop is exited.

Example

```
DECLARE
     sal_limit       NUMBER(4) := 0;
     total_sal       NUMBER(9,2) := 0;
     CURSOR my_cursor IS SELECT ename, sal FROM emp
           WHERE sal > sal_limit;

BEGIN
     sal_limit := 1200;
     -- implicit OPEN done next
     FOR cursor_row IN my_cursor LOOP
           -- an implicit fetch done here
              INSERT INTO new_table
                 VALUES (cursor_row.ename, cursor_row.sal);
              total_sal := total_sal + cursor_row.sal;
     END LOOP;  -- an implicit close done here
     -- store total_sal in temp table
     INSERT INTO temp (COL1,MESSAGE)
        VALUES (total_sal, 'Total Salary');
     COMMIT;
END;
```

Reference Cursor FOR Loops—cont'd

Load data into a PL/SQL table with a cursor FOR loop.

Example

```
DECLARE
      TYPE ename_tab_type IS TABLE OF emp.ename%TYPE
         INDEX BY BINARY_INTEGER;
      TYPE sal_tab_type IS TABLE OF emp.sal%TYPE
         INDEX BY BINARY_INTEGER;
      ename_tab   ename_tab_type;
      sal_tab     sal_tab_type;
      i           BINARY_INTEGER := 0;
      CURSOR empcur IS SELECT ename, sal FROM emp;
BEGIN
-- load employee names and salaries into PL/SQL tables
FOR emprec IN empcur LOOP
      i := i + 1;
      ename_tab(i) := emprec.ename;
      sal_tab(i) := emprec.sal;
END  LOOP;
-- process the PL/SQL tables
. . .
END;
```

DECLARE EXPLICIT CURSORS PARAMETERS

Declare cursors to use parameters.

Syntax

```
DECLARE
     CURSOR <cursor_name> [(param_name param_type)]
          IS <regular select statement>;
```

Example

```
DECLARE
     CURSOR emp_cur (sal_value NUMBER) IS
          SELECT ename FROM emp WHERE sal > sal_value;
BEGIN
     OPEN emp_cur (1200);
     ...
END;
```

Same as

```
DECLARE
     sal_value      NUMBER;
     CURSOR emp_cur IS
          SELECT ename FROM emp WHERE sal > sal_value;
BEGIN
     sal_value := 1200;
     OPEN emp_cur;
     ...
END;
```

IMPLICIT CURSOR OVERVIEW

An implicit cursor is automatically associated with any SQL DML statement that does not have an explicit cursor associated with it.

Statements Associated with Implicit Cursors

- All INSERT statements
- All UPDATE statements
- All DELETE statements
- All SELECT...INTO statements

Quick Notes

- Implicit cursor is called the SQL cursor—it stores information concerning the processing of the last SQL statement not associated with an explicit cursor.
- OPEN, FETCH, and CLOSE do not apply.
- All cursor attributes apply.

REFERENCE IMPLICIT CURSOR ATTRIBUTES

SQL%NOTFOUND evaluates to TRUE if the most recently executed SQL statement affects no rows.

Example - SQL%NOTFOUND

```
UPDATE emp SET sal = sal*10.0 WHERE ename = 'WARD';
IF SQL%NOTFOUND THEN  -- WARD wasn't found
     INSERT INTO emp(empno, ename, sal) VALUES
          (1234, 'WARD', 99999);
END IF;
```

SQL%FOUND evaluates to TRUE if the most recently executed SQL statement affects one or more rows.

Reference Implicit Cursor Attributes—cont'd

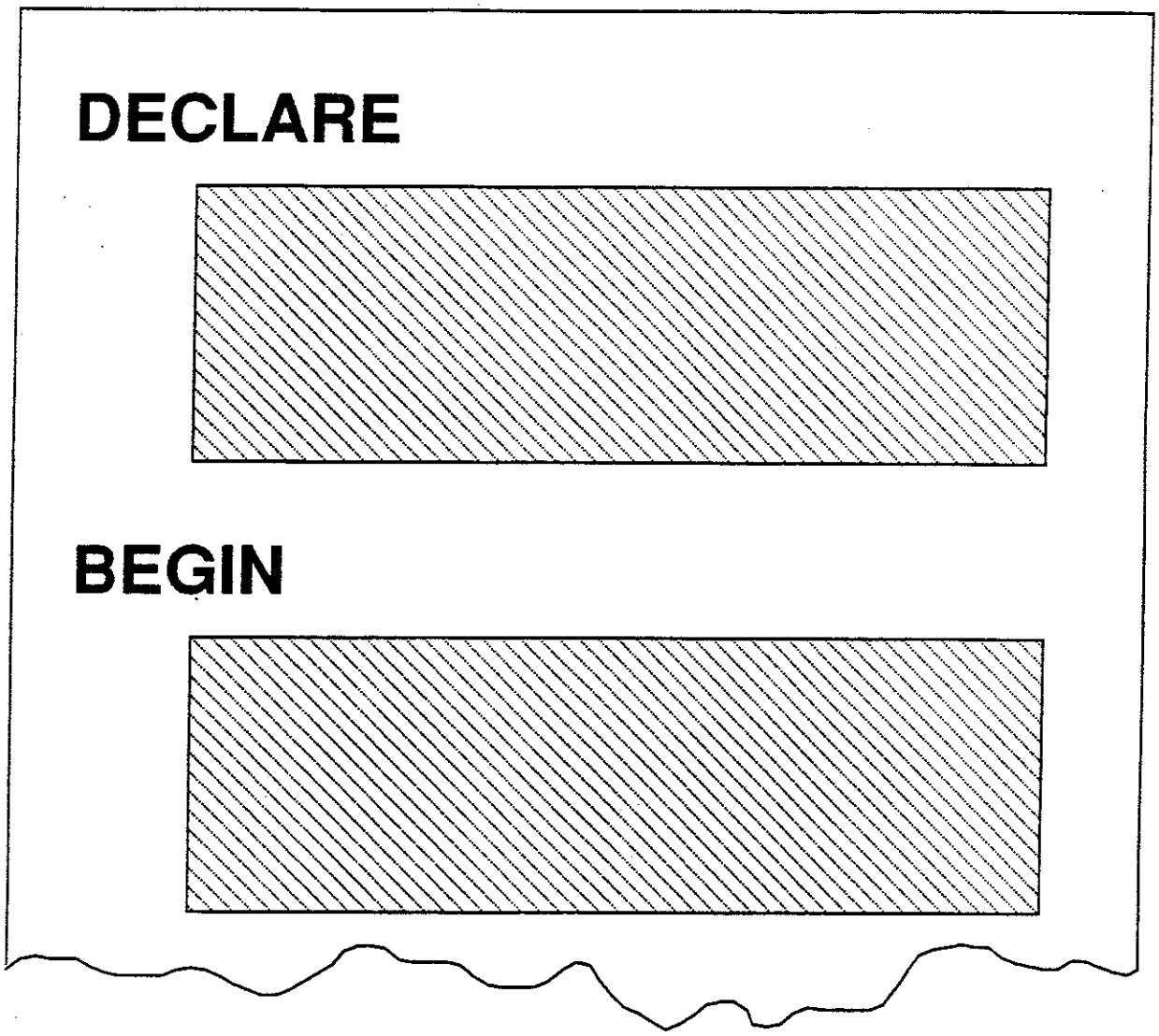
SQL%ROWCOUNT evaluates to the number of rows affected by a DELETE or an UPDATE, the number of rows INSERTed, or the number of rows returned by a SELECT...INTO.

Example - SQL%ROWCOUNT

```
DELETE FROM baseball_team WHERE batting_avg < .100;
IF SQL%ROWCOUNT > 5 THEN
     INSERT INTO temp (message)
          VALUES ('Your team needs help.');
END IF;
```

SQL%ISOPEN always evaluates to FALSE.

DECLARE AND USE CURSORS: SUMMARY



Declare and Use Cursors: Summary—cont'd

Every SQL DML statement processed by PL/SQL has an associated cursor.

Two Types of Cursors

- Explicit
- Implicit

Use Explicit Cursors

- Declare
- Open
- Fetch
- Close

Reference Cursor Attributes

- %FOUND
- %NOTFOUND
- %ROWCOUNT
- %ISOPEN

LAB 4-1

Write a PL / SQL block to satisfy the scenario below. Test your block by executing it within SQL*Plus.

1 Get a number *n* from the operator, using an & variable.

2 Using a simple loop, get the name and salary of the people with the top *n* salaries in the company. All names and salaries may be found in the EMP table (see below).

EMP

[CHAR(10)] ENAME	[NUMBER(7,2)] SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

3 Store them in a table called TOP_SALS, with columns NAME CHAR(20) and SALARY NUMBER(7,2).

4 Repeat the procedure using a cursor FOR loop.