CURSORS

CURSOR MANIPULATION

- To process an SQL statement, ORACLE needs to create an area of memory known as the *context area*; this will have the information needed to process the statement.
- This information includes the number of rows processed by the statement, a pointer to the parsed representation of the statement.
- In a query, the active set refers to the rows that will be returned.

CURSOR MANIPULATION

- A cursor is a handle, or pointer, to the context area.
- Through the cursor, a PL/SQL program can control the context area and what happens to it as the statement is processed.
- Two important features about the cursor are
  1. Cursors allow you to fetch and process rows returned by a SELECT statement, one row at a time.
  2. A cursor is named so that it can be referenced.

Types Of Cursors

- There are two types of cursors:
  1. An *IMPLICIT* cursor is automatically declared by Oracle every time an SQL statement is executed. The user will not be aware of this happening and will not be able to control or process the information in an implicit cursor.
  2. An *EXPLICIT* cursor is defined by the program for any query that returns more than one row of data. That means the programmer has declared the cursor within the PL/SQL code block.

IMPLICIT CURSOR

- Any given PL/SQL block issues an implicit cursor whenever an SQL statement is executed, as long as an explicit cursor does not exist for that SQL statement.
- A cursor is automatically associated with every DML (Data Manipulation) statement (UPDATE, DELETE, INSERT).
- All UPDATE and DELETE statements have cursors that identify the set of rows that will be affected by the operation.
- An INSERT statement needs a place to receive the data that is to be inserted in the database; the implicit cursor fulfills this need.
- The most recently opened cursor is called the “SQL%” Cursor.

The Processing Of An Implicit Cursor

- The implicit cursor is used to process INSERT, UPDATE, DELETE, and SELECT INTO statements.
- During the processing of an implicit cursor, Oracle automatically performs the OPEN, FETCH, and CLOSE operations.
- An implicit cursor cannot tell you how many rows were affected by an update. SQL%ROWCOUNT returns numbers of rows updated. It can be used as follows:

```
BEGIN
UPDATE employees
SET first_name = 'B'
WHERE first_name LIKE 'B%';
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
END;
```

EXPLICIT CURSOR

- The only means of generating an explicit cursor is for the cursor to be named in the DECLARE section of the PL/SQL Block.
- The advantages of declaring an explicit cursor over the indirect implicit cursor are that the explicit cursor gives more programmatic control to the programmer.
- Implicit cursors are less efficient than explicit cursors and thus it is harder to trap data errors.

EXPLICIT CURSOR

- The process of working with an explicit cursor consists of the following steps:
- DECLARING the cursor. This initializes the cursor into memory.
- OPENING the cursor. The previously declared cursor can now be opened; memory is allotted.
- FETCHING the cursor. The previously declared and opened cursor can now retrieve data; this is the process of fetching the cursor.
- CLOSING the cursor. The previously declared, opened, and fetched cursor must now be closed to release memory allocation.

DECLARING A CURSOR

- Declaring a cursor defines the name of the cursor and associates it with a SELECT statement.
- The first step is to Declare the Cursor with the following syntax:

  CURSOR c_cursor_name IS select statement

- Cursor names follow the same rules of scope and visibility that apply to the PL/SQL identifiers.
- Because the name of the cursor is a PL/SQL identifier, it must be declared before it is referenced.
- Any valid select statement can be used to define a cursor, including joins and statements with the UNION or MINUS clause.

RECORD TYPES

- A record is a composite data structure, which means that it is composed of more than one element.
- Records are very much like a row of a database table, but each element of the record does not stand on its own.
- PL/SQL supports three kinds of records:
  1. Table based
  2. Cursor_based,
  3. Programmer-defined.

RECORD TYPES

- A table-based record is one whose structure is drawn from the list of columns in the table.
- A cursor-based record is one whose structure matches the elements of a predefined cursor.
- To create a table-based or cursor_based record use the %ROWTYPE attribute.

  <record_name> <table_name or cursor_name>%ROWTYPE

Example

```
DECLARE
        vr_employe employees%ROWTYPE;
BEGIN
        SELECT *
        INTO vr_employe
        FROM employees
        WHERE  employee_id = 156;
        DBMS_OUTPUT.PUT_LINE (vr_employe.first_name ||vr_employe.last_name||
        ‘ has an ID of 156’);
EXCEPTION
        WHEN no_data_found
        THEN
        RAISE_APPLICATION_ERROR(-2001,‘The Student’ || ‘is not in the
        database’);
END;
```

OPENING A CURSOR

- The next step in controlling an explicit cursor is to open it. When the Open cursor statement is processed, the following four actions will take place automatically:
  1. The variables (including bind variables) in the WHERE clause are examined.
  2. Based on the values of the variables, the active set is determined and the PL/SQL engine executes the query for that cursor. Variables are examined at cursor open time only.
  3. The PL/SQL engine identifies the active set of data—the rows from all involved tables that meet the WHERE clause criteria.
  4. The active set pointer is set to the first row.

OPENING A CURSOR

- The syntax for opening a cursor is:

  OPEN cursor_name;

FETCHING ROWS IN A CURSOR

- After the cursor has been declared and opened, you can then retrieve data from the cursor.
- The process of getting the data from the cursor is referred to as fetching the cursor.
- There are two methods of fetching a cursor, done with the following command:

  FETCH cursor_name INTO PL/SQL variables;

  or

  FETCH cursor_name INTO PL/SQL record;

FETCHING ROWS IN A CURSOR

- When the cursor is fetched the following occurs:

1. The fetch command is used to retrieve one row at a time from the active set. This is generally done inside a loop. The values of each row in the active set can then be stored into the corresponding variables or PL/SQL record one at a time, performing operations on each one successively.
2. After each FETCH, the active set pointer is moved forward to the next row. Thus, each fetch will return successive rows of the active set, until the entire set is returned. The last FETCH will not assign values to the output variables; they will still contain their prior values.

Example

```
DECLARE
        CURSOR c_zip IS
        SELECT *
        FROM employees;
        vr_zip c_zip%ROWTYPE;
BEGIN
        OPEN c_zip;
        LOOP
        FETCH c_zip INTO vr_zip;
        EXIT WHEN c_zip%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(vr_zip.employee_id);
        END LOOP;
END;
```

CLOSING A CURSOR

- Once all of the rows in the cursor have been processed (retrieved), the cursor should be closed.
- This tells the PL/SQL engine that the program is finished with the cursor, and the resources associated with it can be freed.
- The syntax for closing the cursor is:

```
CLOSE cursor_name;
```

- Once a cursor is closed, it is no longer valid to fetch from it.
- Likewise, it is not possible to close an already closed cursor (either one will result in an Oracle error).

Table 8.1 ■ Explicit Cursor Attributes

Cursor Attribute	Syntax	Explanation
%NOTFOUND	cursor_name%NOTFOUND	A Boolean attribute that returns TRUE if the previous FETCH did not return a row, and FALSE if it did.
%FOUND	cursor_name%FOUND	A Boolean attribute that returns TRUE if the previous FETCH returned a row, and FALSE if it did not.
%ROWCOUNT	cursor_name%ROWCOUNT	# of records fetched from a cursor at that point in time.
%ISOPEN	cursor_name%ISOPEN	A Boolean attribute that returns TRUE if cursor is open, FALSE if it is not.

USING CURSOR FOR LOOPS AND NESTING CURSORS

- When using the cursor FOR LOOP, the process of opening, fetching, and closing are implicitly handled.
- This makes the blocks much simpler to code and easier to maintain.
- The cursor FOR LOOP specifies a sequence of statements to be repeated once for each row returned by the cursor.
- Use the cursor FOR LOOP if you need to FETCH and PROCESS each and every record from a cursor.

Example

```
DECLARE
        CURSOR c_student IS
        SELECT student_id, last_name, first_name
        FROM student
        WHERE student_id < 110;
BEGIN
        FOR r_student IN c_student
        LOOP
        INSERT INTO table_log
        VALUES(r_student.last_name);
        END LOOP;
END;
```

PROCESSING NESTED CURSORS

- Cursors can be nested inside each other.
- It is just a loop inside a loop, much like nested loops.
- If you had one parent cursor and two child cursors, then each time the parent cursor makes a single loop, it will loop through each child cursor once and then begin a second round.
- In the following example, you will encounter a nested cursor with a single child cursor.

Example

```
1 DECLARE
2 v_zip zipcode.zip%TYPE;
3 CURSOR c_zip IS
4 SELECT zip, city, state
5 FROM zipcode
6 WHERE state = 'CT';
7 CURSOR c_student IS
8 SELECT first_name,
        last_name
9 FROM student
10 WHERE zip = v_zip;
11 BEGIN
12 FOR r_zip IN c_zip
13 LOOP
```

- There are two cursors in this example.
- The first is a cursor of the zipcodes and the second cursor is a list of students.
- The variable v_zip is initialized in line 14 to be the zipcode of the current record of the c_zip cursor.
- The c_student cursor ties in c_zip cursor by means of this variable.

Contd.

```
14 v_zip := r_zip.zip;
15 DBMS_OUTPUT.PUT_LINE
       (CHR(10));
16 DBMS_OUTPUT.PUT_LINE
       ('Students living in '||
17 r_zip.city);
18 FOR r_student in c_student
  19 LOOP
  20 DBMS_OUTPUT.PUT_LINE
       (r_student.first_name||
  21 ' '||r_student.last_name);
  22 END LOOP;
23 END LOOP;
24* END;
```

Thus, when the cursor is processed in lines 18–22, it is retrieving students that have the zipcode of the current record for the parent cursor.

The parent cursor is processed from lines 12–23. Each iteration of the parent cursor will only execute the DBMS_OUTPUT in lines 16 and 17 once.

The DBMS_OUTPUT in line 20 will be executed once for each iteration of the child loop, producing a line of output for each student.

CURSORS WITH PARAMETERS

- A cursor can be declared with parameters.
- This enables a cursor to generate a specific result set, which is, on the one hand, more narrow, but on the other hand, reusable.
- A cursor of all the data from the zipcode table may be very useful, but it would be more useful for certain data processing if it held information for only one state.

```
CURSOR c_zip (p_state IN zipcode.state%TYPE) IS
    SELECT zip, city, state
    FROM zipcode
    WHERE state = p_state;
```

CURSORS WITH PARAMETERS

- Cursor parameters make the cursor more reusable.
- Cursor parameters can be assigned default values.
- The scope of the cursor parameters is local to the cursor.
- The mode of the parameters can only be IN.
- When a cursor has been declared as taking a parameter, it must be called with a value for that parameter.
- The c_zip cursor that was just declared is called as follows:

OPEN c_zip (parameter_value)

USING A FOR UPDATE CURSOR

- The CURSOR FOR UPDATE clause is only used with a cursor when you want to update tables in the database.
- Generally, when you execute a SELECT statement, you are not locking any rows.
- The purpose of using the FOR UPDATE clause is to lock the rows of the tables that you want to update, so that another user cannot perform an update until you perform your update and release the lock.
- The next COMMIT or ROLLBACK statement releases the lock.

USING A FOR UPDATE CURSOR

- The syntax is simply to add FOR UPDATE to the end of the cursor definition.
- If there are multiple items being selected, but you only want to lock one of them, then end the cursor definition with the following syntax:

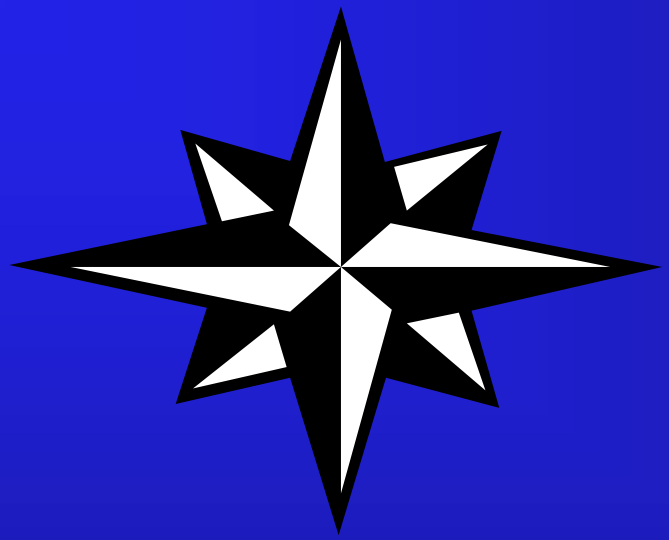
FOR UPDATE OF <item_name>

WHERE CURRENT OF CLAUSE

- Use WHERE CURRENT OF when you want to update the most recently fetched row.
- WHERE CURRENT OF can only be used with a FOR UPDATE OF cursor.
- The advantage of the WHERE CURRENT OF clause is that it enables you to eliminate the WHERE clause in the UPDATE statement.

Example

```
DECLARE
        CURSOR c_stud_zip IS
        SELECT s.studid_id, z.city
        FROM student s, zipcode z
        WHERE z.city = 'Brooklyn'
        AND s.szip = z.zip
        FOR UPDATE OF sphone;
BEGIN
        FOR r_stud_zip IN c_stud_zip
        LOOP
        DBMS_OUTPUT.PUT_LINE(r_stud_zip.studid);
        UPDATE student
        SET sphone = '718'||SUBSTR(sphone,4)
        WHERE CURRENT OF c_stud_zip;
        END LOOP;
END;
```



END