

UNIT 4

Database Cursors

(5 Hours) — Database Programming (CACS453)

1. Introduction & Types of Cursors

A cursor is a private work area in memory that Oracle uses to execute a SQL statement and hold the pointer to its result set, allowing the rows returned by a query to be processed one at a time. Every SQL statement executed by the Oracle server uses a cursor internally, but PL/SQL distinguishes between two categories depending on how much control the programmer has over it.

Type	Description	Used For
Implicit Cursor	Automatically created and managed by Oracle for every SQL DML/SELECT INTO statement; the programmer never declares or opens it	Single-row SELECT INTO, INSERT, UPDATE, DELETE
Explicit Cursor	Declared and controlled explicitly by the programmer using DECLARE, OPEN, FETCH, and CLOSE	Queries that return multiple rows, processed one row at a time

1.1 Implicit Cursors

For every SQL statement not associated with an explicitly declared cursor, Oracle opens an implicit cursor automatically named SQL. Its attributes can be checked immediately after the statement executes to determine the outcome.

Attribute	Meaning
SQL%FOUND	TRUE if the most recent statement affected one or more rows
SQL%NOTFOUND	TRUE if the most recent statement affected no rows
SQL%ROWCOUNT	Number of rows affected by the most recent statement
SQL%ISOPEN	Always FALSE for implicit cursors — Oracle opens and closes them automatically

Implicit Cursor Attributes — Example

BEGIN

UPDATE employees SET salary = salary * 1.10

WHERE department_id = 20;

IF SQL%FOUND THEN

DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' row(s) updated.');

ELSE

DBMS_OUTPUT.PUT_LINE('No matching rows found.');

END IF;

END;

/

1.2 Explicit Cursors

An explicit cursor is declared for any **SELECT** statement expected to return zero, one, or many rows, giving the programmer full, row-by-row control over fetching and processing the result set. Working with an explicit cursor always follows four steps:

Step	Statement	Purpose
1	DECLARE	Names the cursor and associates it with a SELECT statement
2	OPEN	Executes the query and positions the cursor before the first row
3	FETCH	Retrieves the current row into variables, then advances the cursor
4	CLOSE	Releases the memory and resources used by the cursor

Explicit Cursor — Basic Example

DECLARE

CURSOR c_emp IS

SELECT emp_name, salary FROM employees WHERE department_id = 20;

v_name employees.emp_name%TYPE;

v_salary employees.salary%TYPE;

BEGIN

OPEN c_emp;

FETCH c_emp INTO v_name, v_salary;

DBMS_OUTPUT.PUT_LINE(v_name || ' earns ' || v_salary);

CLOSE c_emp;

END;

/

1.3 Explicit Cursor Attributes

Attribute	Meaning
cursor_name%FOUND	TRUE if the last FETCH returned a row
cursor_name%NOTFOUND	TRUE if the last FETCH did not return a row (end of result set reached)
cursor_name%ROWCOUNT	Number of rows fetched so far from this cursor
cursor_name%ISOPEN	TRUE if the cursor is currently open

Note: Fetching from a cursor after %NOTFOUND has become TRUE, or fetching from a cursor that is not open, raises the INVALID_CURSOR exception.

2. Cursor Loop

Because a cursor typically returns many rows, it is almost always processed inside a loop that fetches repeatedly until no rows remain. The standard pattern combines a basic LOOP with EXIT WHEN cursor_name%NOTFOUND, so the loop terminates cleanly once the result set is exhausted.

Explicit Cursor Loop — Example

```
DECLARE
  CURSOR c_emp IS
    SELECT emp_name, salary FROM employees WHERE department_id = 20;
  v_name employees.emp_name%TYPE;
  v_salary employees.salary%TYPE;
BEGIN
  OPEN c_emp;
  LOOP
    FETCH c_emp INTO v_name, v_salary;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_name || ' — ' || v_salary);
  END LOOP;
  CLOSE c_emp;
END;
/
```

Note: EXIT WHEN c_emp%NOTFOUND must come immediately after FETCH and before the row is processed — otherwise the last (already-fetched) row's stale values would be processed a second time when the fetch fails.

3. Cursor FOR Loop

The cursor FOR loop is a simplified, less error-prone way to process an explicit cursor. It automatically opens the cursor, fetches each row in turn, implicitly declares a record to hold each row's columns, exits the loop when rows are exhausted, and closes the cursor — all without OPEN, FETCH, EXIT WHEN, or CLOSE being written explicitly.

Cursor FOR Loop — Example

```
DECLARE
  CURSOR c_emp IS
    SELECT emp_name, salary FROM employees WHERE department_id = 20;
BEGIN
  FOR emp_rec IN c_emp LOOP
    DBMS_OUTPUT.PUT_LINE(emp_rec.emp_name || ' — ' || emp_rec.salary);
  END LOOP;
END;
/
```

The loop variable `emp_rec` is an implicitly declared record whose fields match the columns of the cursor's `SELECT` list — here, `emp_rec.emp_name` and `emp_rec.salary` — and it exists only within the scope of the loop.

A cursor `FOR` loop can also use a query directly, without a separately declared cursor:

Cursor FOR Loop with Inline Query — Example

```
BEGIN
  FOR emp_rec IN (SELECT emp_name, salary FROM employees
    WHERE department_id = 20) LOOP
    DBMS_OUTPUT.PUT_LINE(emp_rec.emp_name || ' — ' || emp_rec.salary);
  END LOOP;
END;
/
```

Aspect	Explicit Cursor Loop	Cursor FOR Loop
OPEN / FETCH / CLOSE	Written explicitly	Handled automatically
Exit condition	Must code <code>EXIT WHEN %NOTFOUND</code>	Handled automatically
Row variable	Declared separately (or <code>%ROWTYPE</code>)	Implicitly declared per iteration
Code Length	Longer, more explicit control	Shorter, less error-prone

4. Parameterized Cursors

A cursor can be declared with formal parameters, similar to a procedure, allowing the same cursor definition to be reused with different filter values each time it is opened — instead of hard-coding a value into the WHERE clause or declaring several nearly identical cursors.

Syntax

```
CURSOR cursor_name (param1 datatype, param2 datatype, ...) IS  
SELECT ...;
```

Parameterized Cursor — Example

```
DECLARE  
  
CURSOR c_emp_by_dept (p_dept_id NUMBER) IS  
    SELECT emp_name, salary FROM employees  
    WHERE department_id = p_dept_id;  
  
BEGIN  
    FOR emp_rec IN c_emp_by_dept(20) LOOP  
        DBMS_OUTPUT.PUT_LINE('Dept 20: ' || emp_rec.emp_name);  
    END LOOP;  
  
    FOR emp_rec IN c_emp_by_dept(30) LOOP  
        DBMS_OUTPUT.PUT_LINE('Dept 30: ' || emp_rec.emp_name);  
    END LOOP;  
  
END;  
/
```

The same cursor `c_emp_by_dept` is opened twice with two different argument values (20 and 30), returning a different result set each time, without redeclaring the cursor or its query.

Note: Cursor parameters, like procedure parameters, can also be given default values (e.g. `p_dept_id NUMBER := 10`), making the argument optional when the cursor is opened.

5. Nested Cursors

A nested cursor arrangement occurs when one cursor's FOR loop (the outer cursor) contains, inside its loop body, another cursor's declaration and FOR loop (the inner cursor) — typically used to process a master/detail (parent/child) relationship, such as departments and the employees within each department.

The inner cursor is commonly parameterized using a column value from the current row of the outer cursor, so that each pass through the outer loop drives a fresh, filtered result set from the

inner cursor.

Nested Cursors — Departments and Their Employees

```
DECLARE
  CURSOR c_dept IS
    SELECT department_id, department_name FROM departments;
  CURSOR c_emp (p_dept_id NUMBER) IS
    SELECT emp_name, salary FROM employees
    WHERE department_id = p_dept_id;
BEGIN
  FOR dept_rec IN c_dept LOOP
    DBMS_OUTPUT.PUT_LINE('Department: ' || dept_rec.department_name);
    FOR emp_rec IN c_emp(dept_rec.department_id) LOOP
      DBMS_OUTPUT.PUT_LINE('  - ' ||
        emp_rec.emp_name || ' (' || emp_rec.salary || ')');
    END LOOP;
  END LOOP;
END;
/
```

Sample Output

Department: Sales

- Anita Rai (55000)
- Ramesh Shrestha (48000)

Department: IT

- Sita Gurung (62000)

For each row fetched by the outer cursor `c_dept`, the inner cursor `c_emp` is opened afresh with that row's `department_id` as its parameter, fully fetched to completion, and then closed automatically before the outer loop advances to the next department.

Note: Because the inner cursor's FOR loop runs to completion for every single iteration of the outer loop, nested cursors can be expensive for large tables — indexing the join column (e.g. `employees.department_id`) is important for performance.

Key Takeaways

- ✓ Implicit cursors (the automatic SQL cursor) handle single-row SELECT INTO and DML statements; explicit cursors give full row-by-row control over multi-row queries.
- ✓ An explicit cursor follows DECLARE, OPEN, FETCH, CLOSE, and its progress is tracked with %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.
- ✓ A cursor loop combines LOOP / FETCH / EXIT WHEN %NOTFOUND / CLOSE to process every row of an explicit cursor's result set.
- ✓ A cursor FOR loop automates OPEN, FETCH, EXIT, and CLOSE, and implicitly declares a record for each row — shorter and less error-prone than a manual cursor loop.
- ✓ Parameterized cursors accept arguments, letting the same cursor definition be reopened with different filter values.
- ✓ Nested cursors place one cursor's FOR loop inside another's, typically to process parent/child data such as departments and their employees.

— End of Unit 4 —