Chapter 4

Cursors

Limitations of Implicit Cursors

- Example 1
  - Retrieves only one row
  - If multiple rows – ERROR

```
DECLARE
  v_salary employees.salary%TYPE;
BEGIN
  SELECT salary INTO v_salary
    FROM employees;
  DBMS_OUTPUT.PUT_LINE(' Salary is : '|| v_salary);
END;
```

```
ORA-01422: exact fetch returns more than requested number of rows
```

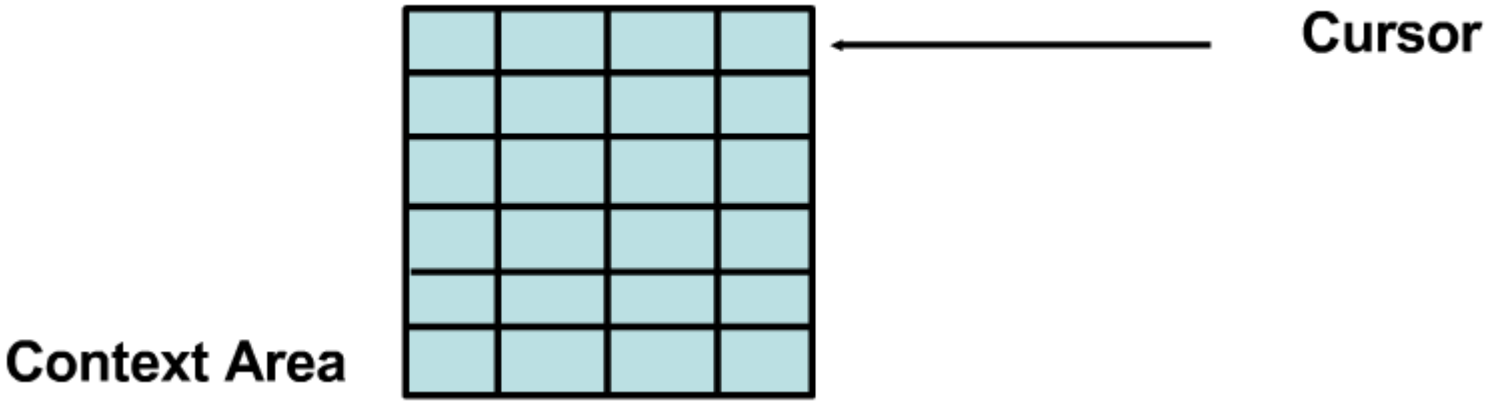
Explicit Cursors

- Declared by the programmer
- For queries that return multiple rows
- A pointer to each row that is retrieved

- Used by PL/SQL to process multiple rows from a database table

Context Areas and Cursors

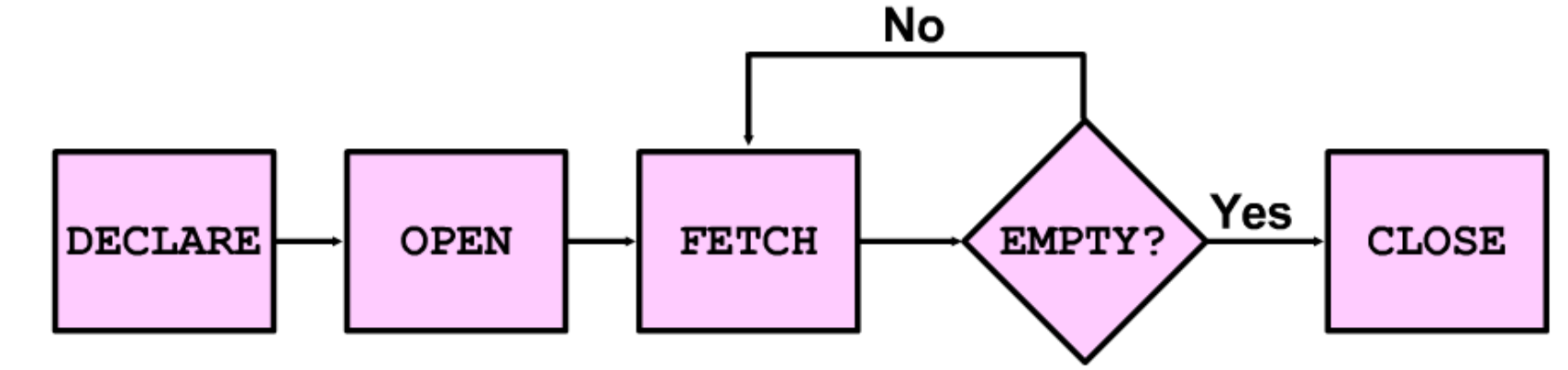
- Context area is a private memory area used to store the result set returned by a SQL statement
- Every context area has a cursor associated with it
  - A reference or pointer to a context area
  - Pointer to each row that is retrieved
  - Access one row at a time



Cursor Attributes

Attribute Name	Data type	Description
%ROWCOUNT	Number	Number of rows affected by the SQL statement
%FOUND	Boolean	TRUE if at least one row is affected by the SQL statement, otherwise FALSE
%NOTFOUND	Boolean	TRUE if no rows are affected by the SQL statement, otherwise FALSE

Method 1
Declaring and Controlling an Explicit Cursor



- Name an active set.
- Fill the active set with data.
- Retrieve the current row into variables.
- Test for existing rows.
- Return to FETCH if rows are found.
- Release the active set.

Basic Cursor

```
DECLARE
  v_employee_name     CHARACTER(20);
  v_salary            employees.salary%TYPE;
  C_H1                CHAR(31) := 'Employee name           Salary';
  C_H1_UL             CHAR(31) := '-------------           ------';

  CURSOR v_salaries_cursor IS                                    -- DECLARE cursor
    SELECT first_name || ' ' || last_name as employee_name, salary
    FROM employees
    WHERE department_id = :Enter_department;  -- Try 90

BEGIN
  DBMS_OUTPUT.PUT_LINE( C_H1 );
  DBMS_OUTPUT.PUT_LINE( C_H1_UL );
  DBMS_OUTPUT.NEW_LINE;
  OPEN v_salaries_cursor;                                        -- OPEN cursor
  LOOP
    FETCH v_salaries_cursor                                      -- FETCH current row from cursor
      INTO v_employee_name, v_salary;
    EXIT WHEN v_salaries_cursor%NOTFOUND;                        -- Test for end-of-cursor
    DBMS_OUTPUT.PUT_LINE(v_employee_name || TO_CHAR(v_salary,'$99,999.99') );
  END LOOP;
  CLOSE v_salaries_cursor;                                       -- CLOSE cursor
END;
```

```
Employee name           Salary
-------------           ------

Steven King          $24,000.00
Neena Kochhar        $17,000.00
Lex De Haan          $17,000.00
```

DECLARE Cursor

- DECLARE cursor in the declarative section
- Creates a named cursor (context area) identified by a SELECT statement
- The SELECT statement does not include an INTO clause
- Values in the cursor are moved to PL/SQL variables with the FETCH step

OPEN Cursor

- Allocates memory for a context area
- Executes the SELECT statement in the cursor
- Returns the results into the active set
- Positions the cursor pointer before (at) the first row

FETCH from Cursor

- Retrieves the current row (one at a time) from the cursor into variables
- After each fetch, the cursor advances to the next row in the active set
- Variables are declared to hold the fetched values from the cursor

LOOP/EXIT

- LOOP is used to fetch all the rows

- EXIT statement is used to check for the end of the cursor and exit the loop

CLOSE Cursor

- CLOSE statement:
  - Disables the cursor
  - Clears the active set of rows
  - Frees the memory area used for the cursor
- The cursor can be reopened later if required

Basic Cursor

```
DECLARE
  v_employee_name     CHARACTER(20);
  v_salary            employees.salary%TYPE;
  C_H1                CHAR(31) := 'Employee name           Salary';
  C_H1_UL             CHAR(31) := '-------------           ------';

  CURSOR v_salaries_cursor IS                                    -- DECLARE cursor
    SELECT first_name || ' ' || last_name as employee_name, salary
    FROM employees
    WHERE department_id = :Enter_department;  -- Try 90

BEGIN
  DBMS_OUTPUT.PUT_LINE( C_H1 );
  DBMS_OUTPUT.PUT_LINE( C_H1_UL );
  DBMS_OUTPUT.NEW_LINE;
  OPEN v_salaries_cursor;                                        -- OPEN cursor
  LOOP
    FETCH v_salaries_cursor                                      -- FETCH current row from cursor
      INTO v_employee_name, v_salary;
    EXIT WHEN v_salaries_cursor%NOTFOUND;                        -- Test for end-of-cursor
    DBMS_OUTPUT.PUT_LINE(v_employee_name || TO_CHAR(v_salary,'$99,999.99') );
  END LOOP;
  CLOSE v_salaries_cursor;                                       -- CLOSE cursor
END;
```

```
Employee name           Salary
-------------           ------

Steven King          $24,000.00
Neena Kochhar        $17,000.00
Lex De Haan          $17,000.00
```

Method 2
%ROWTYPE Attribute

- The %ROWTYPE attribute:
  - Used to declare a record structure based on a cursor
  - The record structure contains the same variables as the cursor on which it is based
  - Each variable is referenced by dot-prefixing the record-name with the field-name

  - Example next slide

%ROWTYPE Attribute

```
DECLARE
  CURSOR emp_cursor IS
  SELECT employee_id, last_name
    FROM employees;
  v_emp_record      emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
   FETCH emp_cursor INTO v_emp_record;
   EXIT WHEN emp_cursor%ROWCOUNT > 10 OR emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id
     ||' '|| v_emp_record.last_name);
  END LOOP;
  CLOSE emp_cursor;
END;
```

%ROWCOUNT and %NOTFOUND

- %ROWCOUNT and %NOTFOUND attributes for exit conditions in a loop

```
DECLARE
  CURSOR emp_cursor IS
  SELECT employee_id, last_name
    FROM employees;
  v_emp_record     emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
   FETCH emp_cursor INTO v_emp_record;
   EXIT WHEN emp_cursor%ROWCOUNT > 10 OR emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id
     ||' '|| v_emp_record.last_name);
  END LOOP;
  CLOSE emp_cursor;
END;
```

Explicit Cursor Attributes in SQL Statements

- Cannot use an explicit cursor attribute directly in an SQL statement
- The following code returns an error:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, salary FROM employees
    ORDER BY SALARY DESC;
  v_emp_record  emp_cursor%ROWTYPE;
  v_count       NUMBER;
BEGIN
  OPEN emp_cursor;
  LOOP
   FETCH emp_cursor INTO v_emp_record;
   EXIT WHEN emp_cursor%NOTFOUND;
   INSERT INTO top_paid_emps
    (employee_id, rank, salary)
   VALUES
    (v_emp_record.employee_id, emp_cursor%ROWCOUNT,
     v_emp_record.salary);
```

Method 3
Cursor FOR loop

- Previous example – Basic LOOP used to process cursor
- Cursor FOR loop performs the following automatically:
  - %ROWTYPE (v_emp_record) is declared
  - The cursor is opened
  - A row is fetched once for each iteration
  - No variables are declared to hold the fetched data by using the INTO clause
  - The loop exits and terminates when the last row is processed
  - The cursor is closed

Example Comparison

- The two examples produce exactly the same results

```
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name
   FROM employees
   WHERE department_id = 50;
BEGIN
  FOR v_emp_record IN emp_cursor
    LOOP
      DBMS_OUTPUT.PUT_LINE(…);
    END LOOP;
END;
```

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id = 50;
  v_emp_record  emp_cursor%ROWTYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
     FETCH emp_cursor INTO
       v_emp_record;
     EXIT WHEN emp_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(…);
   END LOOP;
   CLOSE emp_cursor;
END;
```

Method 4 - SKIP
Cursor FOR Loops Using Subqueries

- The SELECT statement on which the cursor is based is specified directly in the FOR loop
- The SELECT statement in the FOR statement is technically a subquery, so you must enclose it in parentheses

Cursor FOR Loops Using Subqueries : A Comparison - SKIP

- No cursor defined, but it is important to understand that a cursor is being used under the hood

```
BEGIN
  FOR v_dept_rec IN (SELECT *
     FROM departments)
  LOOP
    DBMS_OUTPUT.PUT_LINE(…);
  END LOOP;
END;
```

```
DECLARE
  CURSOR dept_cursor IS
    SELECT * FROM departments;
  v_dept_rec dept_cursor%ROWTYPE;
BEGIN
   OPEN dept_cursor;
   LOOP
     FETCH dept_cursor INTO
       v_dept_rec;
     EXIT WHEN
dept_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(…);
   END LOOP;
   CLOSE dept_cursor;
END;
```

Cursors with Parameters

Cursors with Parameters

- Each parameter named in the cursor declaration must have a corresponding value in the OPEN statement
- Parameter data types are the same as those for scalar variables, but you do not give them sizes
- The parameter names are used in the WHERE clause of the cursor SELECT statement

- Example next slide

Cursors with Parameters

```
DECLARE
 C_H1                CHAR(50) := ' ID Employee name        Salary';
 C_H1_UL             CHAR(31) := ' -- -------------        ------';
 CURSOR emp_cursor (p_deptid NUMBER) IS   -- p_deptid is a parameter ; size is not required
   SELECT employee_id, last_name, salary
     FROM employees
     WHERE department_id = p_deptid
     ORDER BY employee_id;

 v_employee_rec emp_cursor%ROWTYPE;

BEGIN
  DBMS_OUTPUT.PUT_LINE( C_H1 );
  DBMS_OUTPUT.PUT_LINE( C_H1_UL );
  DBMS_OUTPUT.NEW_LINE;
 OPEN emp_cursor(:Enter_department_id);  -- Enter department_id (90)
 LOOP
   FETCH emp_cursor INTO v_employee_rec;
   EXIT WHEN emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(v_employee_rec.employee_id || ' '
         || RPAD( v_employee_rec.last_name, 15 ) || ' ' || TO_CHAR(v_employee_rec.salary,'$99,999.99') );
 END LOOP;
 CLOSE emp_cursor;
END;
```

```
 ID Employee name         Salary
 -- -------------         ------

100 King              $24,000.00
101 Kochhar           $17,000.00
102 De Haan           $17,000.00
```

Using Cursors for UPDATE

Declaring a Cursor with FOR UPDATE

- The FOR UPDATE clause is specified in the cursor declaration
- When a cursor is declared FOR UPDATE
  - Each row is locked with open cursor
  - Prevents other users from modifying the rows while the cursor is open
  - Does not prevent other users from reading the rows

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, salary
      FROM copy_employees
      WHERE salary <= 20000
      FOR UPDATE NOWAIT;  -- FOR UPDATE clause locks the rows first

  v_emp_rec emp_cursor%ROWTYPE;
```

WHERE CURRENT OF clause in UPDATE Statement

- Use the WHERE CURRENT OF clause in the UPDATE statement to reference the current row in the explicit cursor

```
BEGIN
 OPEN emp_cursor;
 LOOP
   FETCH emp_cursor INTO v_emp_rec;
   EXIT WHEN emp_cursor%NOTFOUND;
   UPDATE copy_employees
     SET salary = v_emp_rec.salary * 1.1
     WHERE CURRENT OF emp_cursor;  -- WHERE CURRENT OF clause references the current row in the cursor
 END LOOP;
 CLOSE emp_cursor;
 COMMIT;
END;
```

NOWAIT Keyword

- Tells the Oracle server **not to wait** if any of the requested rows have already been locked by another user
- Returns an Oracle server error immediately
- Control is immediately returned to the program so that it can do other work before trying again to acquire the lock

```
DECLARE
   CURSOR emp_cursor IS
      SELECT employee_id, salary
        FROM copy_employees
        WHERE salary <= 20000
        FOR UPDATE NOWAIT;
```

WAIT n Keyword

- If the rows have already been locked by another session:
  - WAIT n waits for n seconds, and returns an Oracle server error if the other session is still locking the rows at the end of that time
  - If the NOWAIT / WAIT keywords are omitted, the Oracle server waits indefinitely until the rows are available

FOR UPDATE OF column-name

- If the cursor is based on a join of two tables, we may want to lock the rows of one table but not the other
- To do this, we specify any column of the table we want to lock

```
DROP TABLE COPY_DEPARTMENTS
CREATE TABLE copy_departments AS SELECT * FROM departments;

DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, salary, department_name
      FROM copy_employees
           JOIN copy_departments USING (department_id)
      FOR UPDATE OF salary NOWAIT;    -- lock copy_employees table only
BEGIN
  FOR v_emp_record IN emp_cursor LOOP
    UPDATE copy_employees
      SET salary = v_emp_record.salary * 1.1
        WHERE CURRENT OF emp_cursor;
  END LOOP;
  COMMIT;
END;
```

Multiple Cursors

Multiple Cursors

A Sample Problem Statement

- Produce a report that lists each department as a sub-heading, immediately followed by a listing of the employees in that department, followed by the next department, and so on
- Two cursors are required
  - One for each of the two tables
  - The cursor based on EMPLOYEES is opened several times, once for each department

Multiple Cursors Problem Solution: Step 1

- Declare two cursors, one for each table, plus associated record structures

```
DECLARE
  CURSOR department_cursor IS
    SELECT department_id, department_name
    FROM departments
    ORDER BY department_name;

  CURSOR employee_cursor (p_department_id NUMBER) IS
        SELECT first_name, last_name
      FROM employees
      WHERE department_id = p_department_id
      ORDER BY last_name;

  v_department_record   department_cursor%ROWTYPE;
  v_employee_record     employee_cursor%ROWTYPE;

BEGIN
  OPEN department_cursor;  LOOP
    FETCH department_cursor INTO v_department_record;
    EXIT WHEN department_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_department_record.department_name);
  END LOOP;
  CLOSE department_cursor;
END;
```

Why is cursor employee_cursor declared with a parameter?

```
DECLARE
  CURSOR department_cursor IS
    SELECT department_id, department_name
    FROM departments
    ORDER BY department_name;

  CURSOR employee_cursor (p_department_id NUMBER) IS
        SELECT first_name, last_name
      FROM employees
      WHERE department_id = p_department_id
      ORDER BY last_name;

  v_department_record   department_cursor%ROWTYPE;
  v_employee_record     employee_cursor%ROWTYPE;

BEGIN
  OPEN department_cursor;  LOOP
    FETCH department_cursor INTO v_department_record;
    EXIT WHEN department_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_department_record.department_name);
  END LOOP;
  CLOSE department_cursor;
END;
```

Multiple Cursors
Problem Solution: Step 2

- Open the department cursor and fetch and display the DEPARTMENTS rows in the usual way

```
DECLARE
  CURSOR department_cursor IS
    SELECT department_id, department_name
    FROM departments
    ORDER BY department_name;

v_department_record   department_cursor%ROWTYPE;
```

```
BEGIN
  OPEN department_cursor;  LOOP
    FETCH department_cursor INTO v_department_record;
    EXIT WHEN department_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_department_record.department_name);
  END LOOP;
  CLOSE department_cursor;
END;
```

```
Accounting
Administration
Contracting
Executive
IT
Marketing
Sales - Americas
Sales - Europe
Shipping
```

Multiple Cursors
Problem Solution: Step 3

- As each DEPARTMENT row is fetched and displayed, fetch and display the EMPLOYEES in that department

```
BEGIN
  OPEN department_cursor;  LOOP
    FETCH department_cursor INTO v_department_record;
    EXIT WHEN department_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(' ');
    DBMS_OUTPUT.PUT_LINE(v_department_record.department_name);
    OPEN employee_cursor (v_department_record.department_id);
    LOOP
      FETCH employee_cursor INTO v_employee_record;
      EXIT WHEN employee_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('   ' || v_employee_record.last_name || ' ' ||
           v_employee_record.first_name);
    END LOOP;
    CLOSE employee_cursor;
  END LOOP;
  CLOSE department_cursor;
END;
```

```
Accounting
  Duric Jelena
  Gietz William
  Higgins Shelley
  Loermans Jennifer
  Reinhard Matthias

Administration
  Hernandez Katia
  Ricci Guido
  Saikawa Mizuto
  Whalen Jennifer
```


Q & A
Questions & Answers