ORACLE®
Academy

Database Programming with PL/SQL

5-1
Introduction to Explicit Cursors

Academy

Objectives

This lesson covers the following objectives:

- Distinguish between an implicit and an explicit cursor
- Describe why and when to use an explicit cursor in PL/SQL code
- List two or more guidelines for declaring and controlling explicit cursors
- Create PL/SQL code that successfully opens a cursor and fetches a piece of data into a variable

Objectives

Objectives continued:

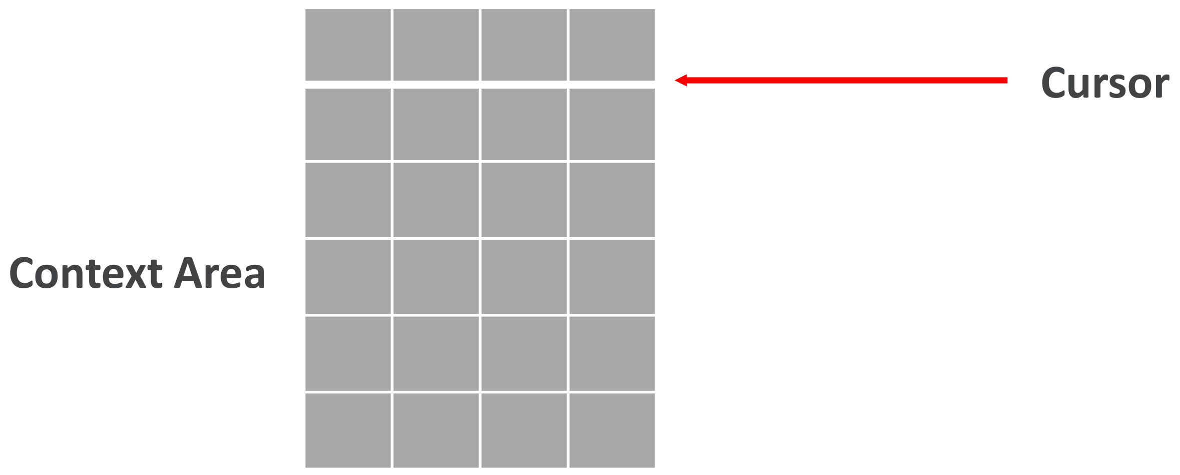
- Use a simple loop to fetch multiple rows from a cursor
- Create PL/SQL code that successfully closes a cursor after fetching data into a variable

Purpose

- You have learned that a SQL SELECT statement in a PL/SQL block is successful only if it returns exactly one row.
- What if you need to write a SELECT statement that returns more than one row?
- For example, you need to produce a report of all employees?
- To return more than one row, you must declare and use an explicit cursor.

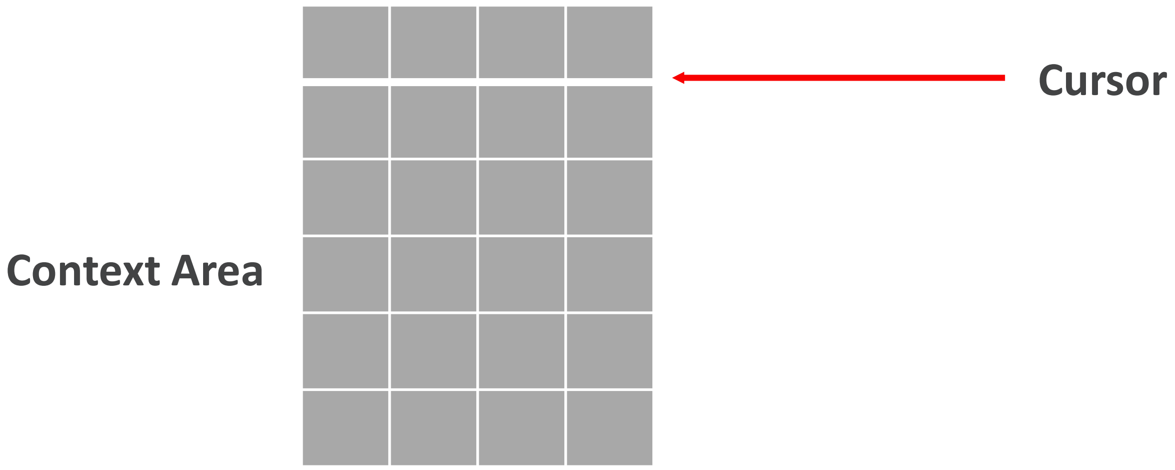
Context Areas and Cursors

- The Oracle server allocates a private memory area called a context area to store the data processed by a SQL statement.
- Every context area (and therefore every SQL statement) has a cursor associated with it.



Context Areas and Cursors

- You can think of a cursor either as a label for the context area, or as a pointer to the context area.
- In fact, a cursor is both of these items.



Implicit and Explicit Cursors

There are two types of cursors:

- Implicit cursors: Defined automatically by Oracle for all SQL DML statements (INSERT, UPDATE, DELETE, and MERGE), and for SELECT statements that return only one row.
- Explicit cursors: Declared by the programmer for queries that return more than one row.
  - You can use explicit cursors to name a context area and access its stored data.

Limitations of Implicit Cursors

- Programmers must think about the data that is possible as well as the data that actually exists now.
- If there ever is more than one row in the EMPLOYEES table, the SELECT statement below (without a WHERE clause) will cause an 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
```

Limitations of Implicit Cursors

To limit the SELECT statement so it returns only one row, the WHERE clause must be matching on either the primary key or other unique column. For example:

- SELECT …
- FROM employees
- WHERE employee_id = 101;
- Group functions without a GROUP BY clause also return exactly one row. For example:
- SELECT COUNT(*) INTO v_number_of_employees
- FROM employees;

Explicit Cursors

- With an explicit cursor, you can retrieve multiple rows from a database table, have a pointer to each row that is retrieved, and work on the rows one at a time.
- Reasons to use an explicit cursor:
  – It is the only way in PL/SQL to retrieve more than one row from a table.
  – Each row is fetched by a separate program statement, giving the programmer more control over the processing of the rows.

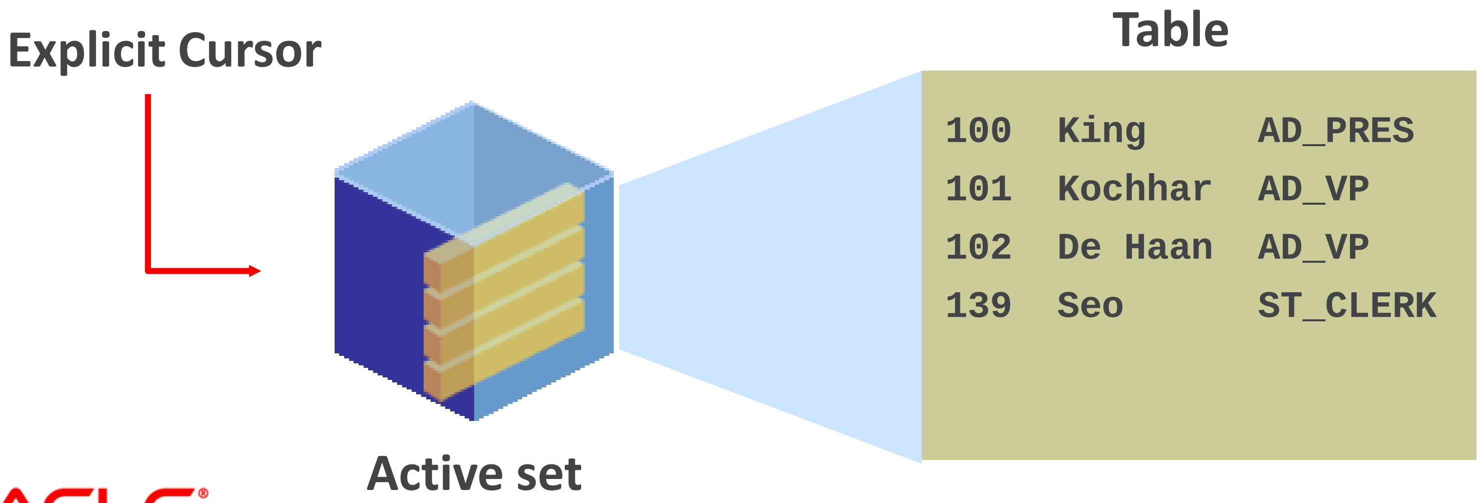
Example of an Explicit Cursor

- The following example uses an explicit cursor to display each row from the departments table.

```
DECLARE
 CURSOR cur_depts IS
  SELECT department_id, department_name FROM departments;
 v_department_id   departments.department_id%TYPE;
 v_department_name  departments.department_name%TYPE;
BEGIN
 OPEN cur_depts;
 LOOP
  FETCH cur_depts INTO v_department_id, v_department_name;
  EXIT WHEN cur_depts%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE(v_department_id||' '||v_department_name);
 END LOOP;
 CLOSE cur_depts;
END;
```

Explicit Cursor Operations

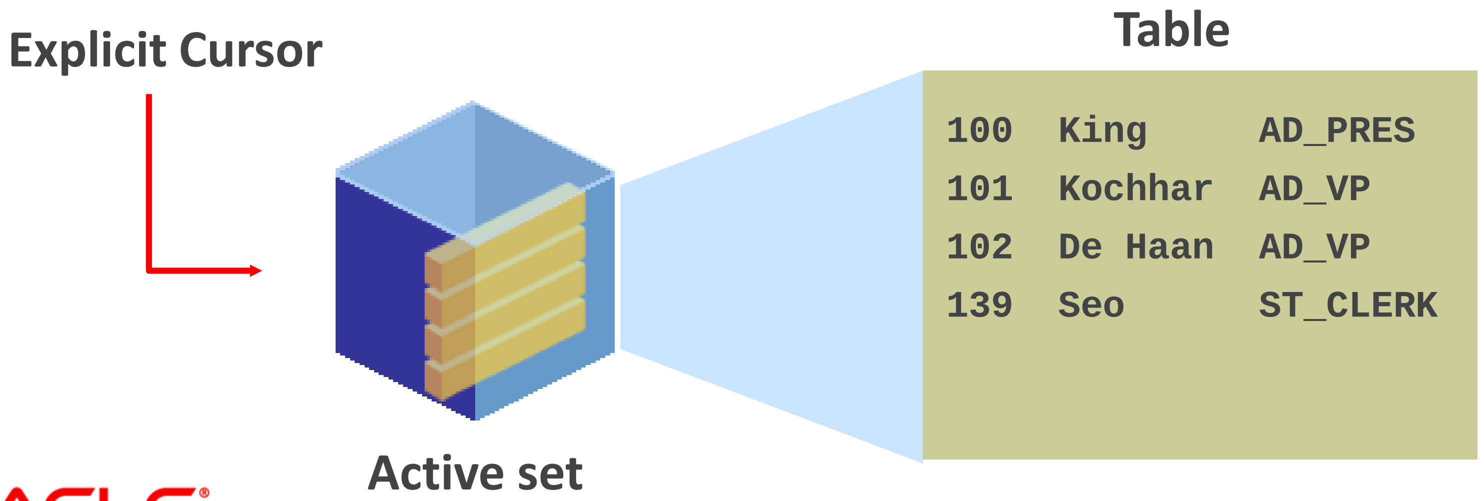
- The set of rows returned by a multiple-row query is called the active set, and is stored in the context area.
- Its size is the number of rows that meet your query criteria.



```
100   King      AD_PRES
101   Kochhar   AD_VP
102   De Haan   AD_VP
139   Seo       ST_CLERK
```

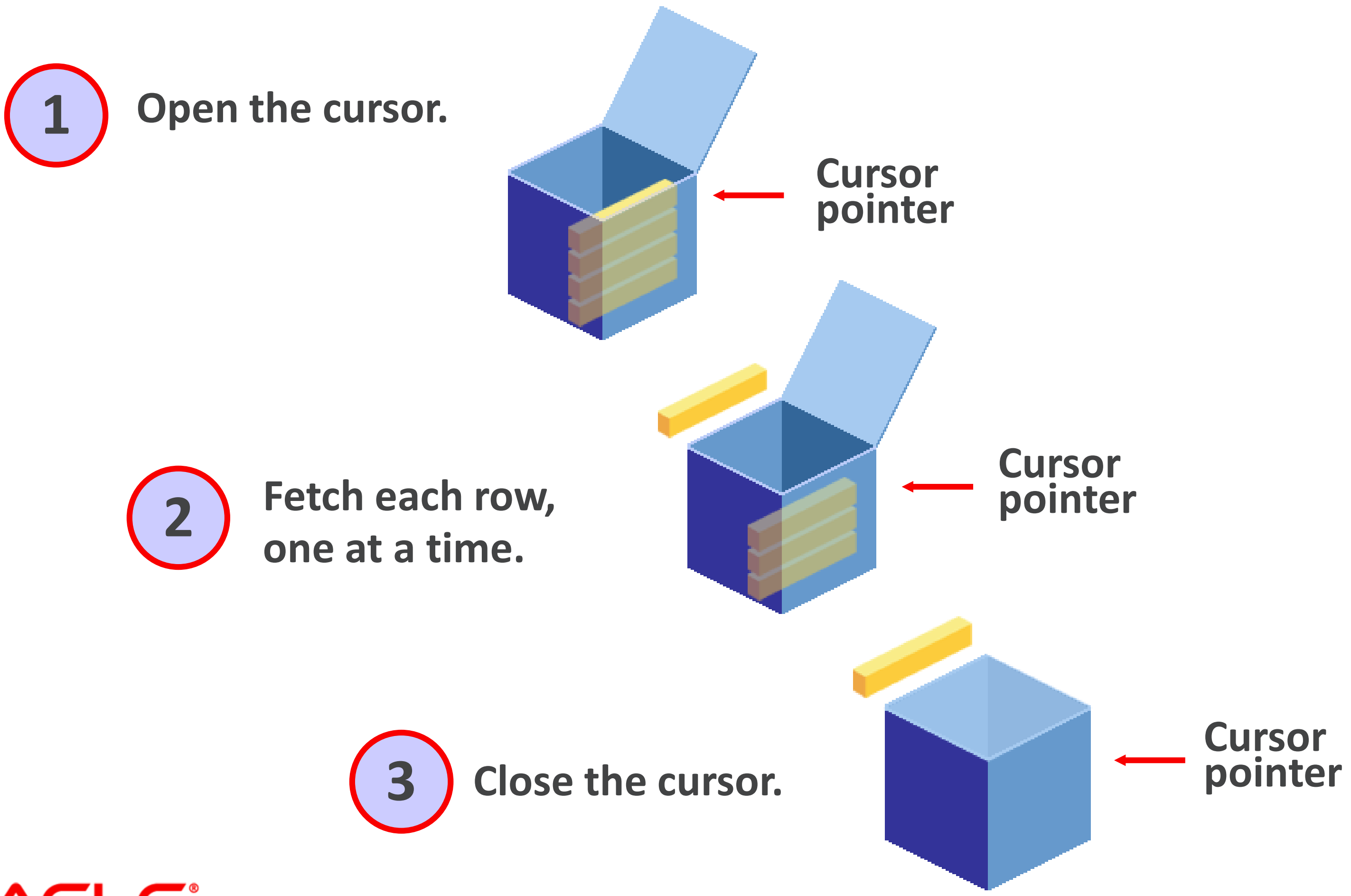
Explicit Cursor Operations

- Think of the context area (named by the cursor) as a box, and the active set as the contents of the box.
- To get at the data, you must OPEN the box and FETCH each row from the box one at a time.
- When finished, you must CLOSE the box.



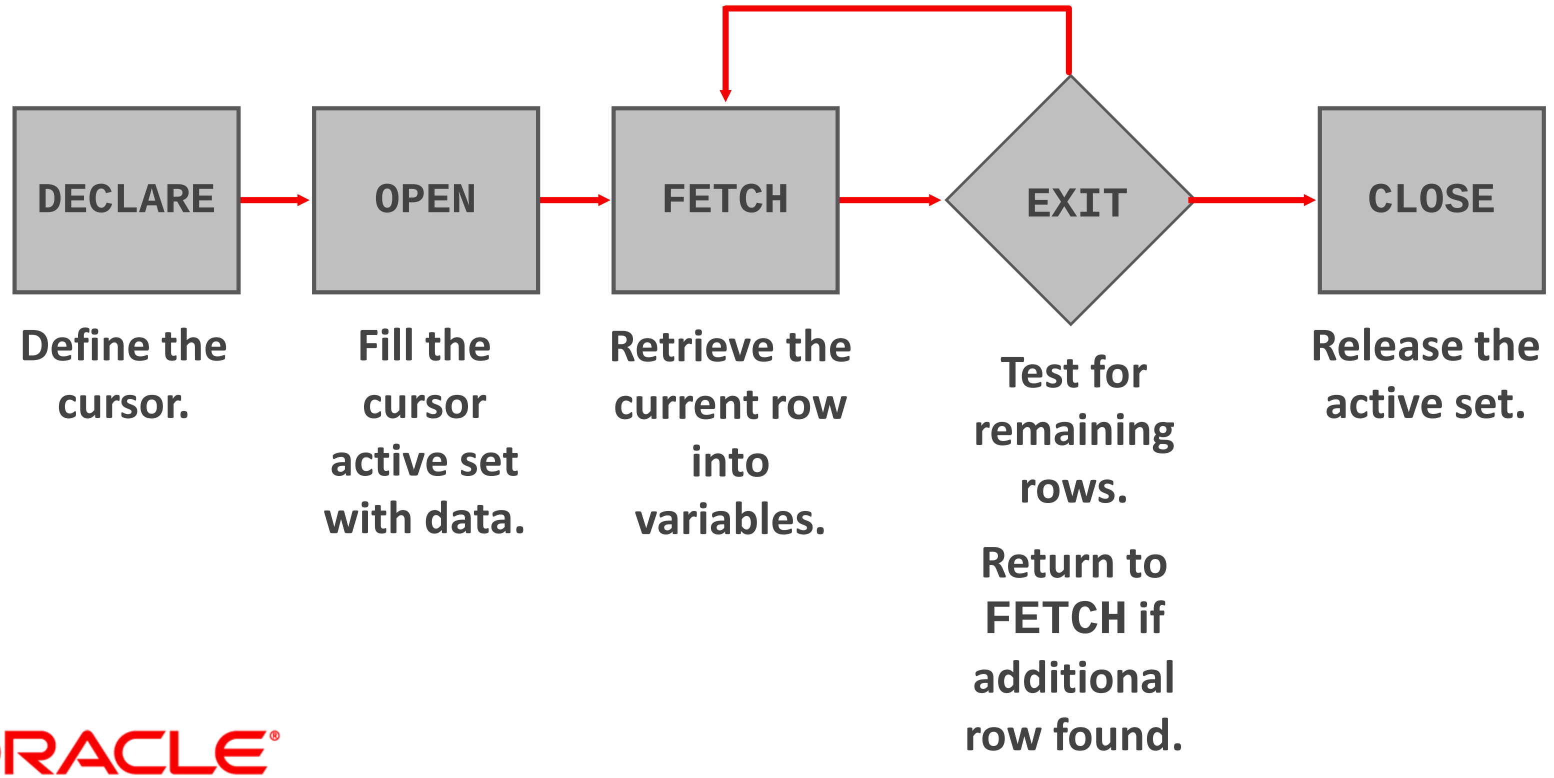
100	King	AD_PRES
101	Kochhar	AD_VP
102	De Haan	AD_VP
139	Seo	ST_CLERK

Controlling Explicit Cursors



Steps for Using Explicit Cursors

- You first DECLARE a cursor, and then you use the OPEN, FETCH, and CLOSE statements to control a cursor.


DECLARE
OPEN
FETCH
EXIT
CLOSE
Define the cursor.
Fill the cursor active set with data.
Retrieve the current row into variables.
Test for remaining rows.
Return to FETCH if additional row found.
Release the active set.

Steps for Using Explicit Cursors

- Now that you have a conceptual understanding of cursors, review the steps to use them:
- DECLARE the cursor in the declarative section by naming it and defining the SQL SELECT statement to be associated with it.
- OPEN the cursor.
  - This will populate the cursor's active set with the results of the SELECT statement in the cursor's definition.
  - The OPEN statement also positions the cursor pointer at the first row.

Steps for Using Explicit Cursors

- Steps to use cursors continued:
- 3. FETCH each row from the active set and load the data into variables.
  - After each FETCH, the EXIT WHEN checks to see if the FETCH reached the end of the active set. If it did, the NOTFOUND attribute is set to TRUE.
  - If the end of the active set was reached, the LOOP is exited.
- 4. CLOSE the cursor.
  - The CLOSE statement releases the active set of rows.
  - It is now possible to reopen the cursor to establish a fresh active set using a new OPEN statement.

Declaring the Cursor

When declaring the cursor:

- Do not include the INTO clause in the cursor declaration because it appears later in the FETCH statement.
- If processing rows in a specific sequence is required, then use the ORDER BY clause in the query.
- The cursor can be any valid SELECT statement, including joins, subqueries, and so on.
- If a cursor declaration references any PL/SQL variables, these variables must be declared before declaring the cursor.

Syntax for Declaring the Cursor

- The active set of a cursor is determined by the SELECT statement in the cursor declaration.
- Syntax:

```
CURSOR cursor_name IS
     select_statement;
```

- In the syntax:

– cursor_name	Is a PL/SQL identifier
– select_statement	Is a SELECT statement without an INTO clause

Declaring the Cursor Example 1

- The cur_emps cursor is declared to retrieve the employee_id and last_name columns of the employees working in the department with a department_id of 30.

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
...
```

- Note that even if there is currently only one employee in department #30, we should still declare a cursor, because in the future there may be more than one employee.

Declaring the Cursor Example 2

- The cur_depts cursor is declared to retrieve all the details for the departments with the location_id 1700.
- You want to fetch and process these rows in ascending sequence by department_name.

```
DECLARE
  CURSOR cur_depts IS
    SELECT * FROM departments
      WHERE location_id = 1700
      ORDER BY department_name;
...
```

Declaring the Cursor Example 3

- A SELECT statement in a cursor declaration can include joins, group functions, and subqueries.
- This example retrieves each department that has at least two employees, giving the department name and number of employees.

```
DECLARE
  CURSOR cur_depts_emps IS
    SELECT department_name, COUNT(*) AS how_many
      FROM departments d, employees e
         WHERE d.department_id = e.department_id
      GROUP BY d.department_name
      HAVING COUNT(*) > 1;
...
```

Opening the Cursor

- The OPEN statement executes the query associated with the cursor, identifies the active set, and positions the cursor pointer to the first row.
- The OPEN statement is included in the executable section of the PL/SQL block.

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name FROM employees
      WHERE department_id = 30;
...
BEGIN
  OPEN cur_emps;
...
```

Opening the Cursor

- The OPEN statement performs the following operations:
  - Allocates memory for a context area (creates the box to hold the data)
  - Executes the SELECT statement in the cursor declaration, returning the results into the active set (fills the box with the data)
  - Positions the pointer to the first row in the active set

Fetching Data from the Cursor

- The FETCH statement retrieves the rows from the cursor one at a time.
- After each successful fetch, the cursor advances to the next row in the active set.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
      WHERE department_id =10;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  ...
END;
```

Fetching Data from the Cursor

- Two variables, v_empno and v_lname, were declared to hold the values fetched from the cursor.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
      WHERE department_id =10;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  ...
END;
```

Fetching Data from the Cursor

- The previous code successfully fetched the values from the first row in the cursor into the variables.
- If there are other employees in department 50, you have to use a loop as shown below to access and process each row.

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name FROM employees
       WHERE  department_id =50;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN cur_emps;
  LOOP
    FETCH cur_emps INTO v_empno, v_lname;
    EXIT WHEN cur_emps%NOTFOUND;     // exit loop when NOTFOUND is TRUE
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP; …
END;
```

ORACLE Academy

Guidelines for Fetching Data From the Cursor

- Follow these guidelines when fetching data from the cursor:
- Include the same number of variables in the INTO clause of the FETCH statement as columns in the SELECT statement, and be sure that the data types are compatible.
- Match each variable to correspond to the columns position in the cursor definition.
- Use %TYPE to insure data types are compatible between variable and table.

Guidelines for Fetching Data From the Cursor

- Follow these guidelines when fetching data from the cursor:
- Test to see whether the cursor contains rows.
- If a fetch acquires no values, then there are no rows to process (or left to process) in the active set and no error is recorded.
- You can use the %NOTFOUND cursor attribute to test for the exit condition.

Fetching Data From the Cursor Example 1

- What is wrong with this example?

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name, salary FROM employees
      WHERE  department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
  v_sal   employees.salary%TYPE;
BEGIN
  OPEN cur_emps;
  LOOP
    FETCH cur_emps INTO v_empno, v_lname;
    EXIT WHEN cur_emps%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP;
  …
END;
```

Fetching Data From the Cursor Example 2

- There is only one employee in department 10.
- What happens when this example is executed?

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name FROM employees
      WHERE  department_id =10;
  v_empno    employees.employee_id%TYPE;
  v_lname    employees.last_name%TYPE;
BEGIN
  OPEN cur_emps;
  LOOP
    FETCH cur_emps INTO v_empno, v_lname;
    DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || v_lname);
  END LOOP;
  …
END;
```

Closing the Cursor

- The CLOSE statement disables the cursor, releases the context area, and undefines the active set.
- You should close the cursor after completing the processing of the FETCH statement.

```
...
  LOOP
    FETCH cur_emps INTO v_empno, v_lname;
    EXIT WHEN cur_emps%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_empno || ' ' ||
v_lname);
  END LOOP;
  CLOSE cur_emps;
END;
```

Closing the Cursor

- You can reopen the cursor later if required.
- Think of CLOSE as closing and emptying the box, so you can no longer FETCH its contents.

```
...
  LOOP
    FETCH cur_emps INTO v_empno, v_lname;
    EXIT WHEN cur_emps%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_empno || ' ' ||
v_lname);
  END LOOP;
  CLOSE cur_emps;
END;
```

Guidelines for Closing the Cursor

- Follow these guidelines when closing the cursor:
- A cursor can be reopened only if it is closed.
- If you attempt to fetch data from a cursor after it has been closed, then an INVALID_CURSOR exception is raised.
- If you later reopen the cursor, the associated SELECT statement is re-executed to re-populate the context area with the most recent data from the database.

Putting It All Together

- Now, when looking at an explicit cursor, you should be able to identify the cursor-related keywords and explain what each statement is doing.

```
DECLARE
  CURSOR cur_depts IS
    SELECT department_id, department_name FROM departments
  v_department_id     departments.department_id%TYPE;
  v_department_name   departments.department_name%TYPE;
BEGIN
  OPEN cur_depts;
  LOOP
    FETCH cur_depts INTO v_department_id, v_department_name;
    EXIT WHEN cur_depts%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_department_id||' '||v_department_name);
  END LOOP;
  CLOSE cur_depts;
END;
```

Terminology

Key terms used in this lesson included:

- Active set
- CLOSE
- Context area
- Cursor
- Explicit cursor
- FETCH
- Implicit cursor
- OPEN

Summary

In this lesson, you should have learned how to:

- Distinguish between an implicit and an explicit cursor
- Describe why and when to use an explicit cursor in PL/SQL code
- List two or more guidelines for declaring and controlling explicit cursors
- Create PL/SQL code that successfully opens a cursor and fetches a piece of data into a variable

Summary

In this lesson, you should have learned how to:

- Use a simple loop to fetch multiple rows from a cursor
- Create PL/SQL code that successfully closes a cursor after fetching data into a variable

ORACLE®

Academy