
ORACLE
ACADEMY

Database Programming with PL/SQL

5-3
Cursor FOR Loops

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- List and explain the benefits of using cursor FOR loops
- Create PL/SQL code to declare a cursor and manipulate it in a FOR loop
- Create PL/SQL code containing a cursor FOR loop using a subquery

Purpose

- You have already learned how to declare and use a simple explicit cursor, using DECLARE, OPEN, and FETCH in a loop, testing for %NOTFOUND, and CLOSE statements.
- Wouldn't it be easier if you could do all this with just one statement?
- You can do all of this using a cursor FOR loop.

Cursor FOR Loops

- A cursor FOR loop processes rows in an explicit cursor.
- It is a shortcut because the cursor is opened, a row is fetched once for each iteration in the loop, the loop exits when the last row is processed, and the cursor is closed automatically.
- The loop itself is terminated automatically at the end of the iteration when the last row has been fetched.
- Syntax:

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Cursor FOR Loops

In the syntax:

- *record_name* Is the name of the implicitly declared record (as *cursor_name*%ROWTYPE)
- *cursor_name* Is a PL/SQL identifier for a previously declared cursor

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Cursor FOR Loops

- Note: `v_emp_record` is the record that is implicitly declared.
- You can access the fetched data with this implicit record as shown below.

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

Cursor FOR Loops

- Compare the cursor FOR loop (on the left) with the cursor code you learned in the previous lesson.
- The two forms of the code are logically identical to each other and produce exactly the same results.

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

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

Cursor FOR Loops: A Second Example

- `v_dept_record` has been implicitly declared as `cur_depts%ROWTYPE`.
- How many fields does it contain?

```
DECLARE
  CURSOR cur_depts IS
    SELECT department_id, department_name
      FROM departments
      ORDER BY department_id;
BEGIN
   FOR v_dept_record IN cur_depts LOOP
     DBMS_OUTPUT.PUT_LINE(v_dept_record.department_id || ' '
                          || v_dept_record.department_name);
   END LOOP;
END;
```

Guidelines for Cursor FOR Loops

Guidelines:

- Do not declare the record that controls the loop because it is declared implicitly.
- The scope of the implicit record is restricted to the loop, so you cannot reference the record outside the loop.
- You can access fetched data using *record_name*.*column_name*.

Testing Cursor Attributes

- You can still test cursor attributes, such as %ROWCOUNT.
- This example exits from the loop after five rows have been fetched and processed.
- The cursor is still closed automatically.

```
DECLARE
  CURSOR cur_emps IS
    SELECT employee_id, last_name
      FROM employees;
BEGIN
  FOR v_emp_record IN cur_emps LOOP
    EXIT WHEN cur_emps%ROWCOUNT > 5;
    DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id || ' '
                         || v_emp_record.last_name);
  END LOOP;
END;
```

Cursor FOR Loops Using Subqueries

- You can go one step further. You don’t have to declare the cursor at all!
- Instead, you can specify the SELECT on which the cursor is based directly in the FOR loop.
- The advantage of this is the cursor definition is contained in a single FOR … statement.
- In complex code with lots of cursors, this simplification makes code maintenance easier and quicker.
- The downside is you can't reference cursor attributes.

Cursor FOR Loops Using Subqueries: Example

The SELECT clause in the FOR statement is technically a subquery, so you must enclose it in parentheses.

```
BEGIN
  FOR v_emp_record IN (SELECT employee_id, last_name
    FROM employees WHERE department_id = 50)
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id || ' '
                         || v_emp_record.last_name);
  END LOOP;
END;
```

Cursor FOR Loops Using Subqueries

- Again, compare these two forms of code.
- They are logically identical, but which one would you rather write – especially if you hate typing!

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

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

Terminology

Key terms used in this lesson included:

- Cursor FOR loop

Summary

In this lesson, you should have learned how to:

- List and explain the benefits of using cursor FOR loops
- Create PL/SQL code to declare a cursor and manipulate it in a FOR loop
- Create PL/SQL code containing a cursor FOR loop using a subquery


ORACLE
ACADEMY