Database Programming with PL/SQL

Iterative Control: Nested Loops

ORACLE® ACADEMY

Objectives

This lesson covers the following objectives:

- Construct and execute PL/SQL using nested loops
- Label loops and use the labels ın EXIT statements
- Evaluate a nested loop construct and identify the exit point

Purpose

You’ve learned about looping constructs in PL/SQL. This lesson discusses how you can nest loops to multiple levels. You can nest `FOR`, `WHILE`, and basic loops within one another.

Nested Loop Example

In PL/SQL, you can nest loops to multiple levels. You can nest `FOR`, `WHILE`, and basic loops within one another.

```
BEGIN
  FOR v_outerloop IN 1..3 LOOP
    FOR v_innerloop IN REVERSE 1..5 LOOP
     DBMS_OUTPUT.PUT_LINE('Outer loop is:'||v_outerloop||
                        ' and inner loop is: '||v_innerloop);
    END LOOP;
  END LOOP;
END;
```

Nested Loops

This example contains EXIT conditions in nested basic loops. What if you want to exit from the outer loop at step A?

```
DECLARE
  v_outer_done   CHAR(3) := 'NO';
  v_inner_done   CHAR(3) := 'NO';
BEGIN
  LOOP                  -- outer loop
    ...
    LOOP                -- inner loop
      ...
      ...               -- step A
      EXIT WHEN v_inner_done = 'YES';
      ...

    END LOOP;
    ...
    EXIT WHEN v_outer_done = 'YES';
    ...
  END LOOP;
END;
```

Loop Labels

```
DECLARE
  ...
BEGIN
 <<outer_loop>>
  LOOP                -- outer loop
    ...
    <<inner_loop>>
    LOOP              -- inner loop
      EXIT outer_loop WHEN ...  -- Exits both loops
      EXIT WHEN v_inner_done = 'YES';
      ...
    END LOOP;
    ...
    EXIT WHEN v_outer_done = 'YES';
    ...
  END LOOP;
END;
```

Loop Labels (cont.)

Loop label names follow the same rules as other identifiers. A label is placed before a statement, either on the same line or on a separate line. In `FOR` or `WHILE` loops, place the label before `FOR` or `WHILE` within label delimiters (<<*label*>>).

If the loop is labeled, the label name can optionally be included after the `END LOOP` statement for clarity.

Loop Labels (cont.)

Label basic loops by placing the label before the word LOOP within label delimiters (<<*label*>>).

```
DECLARE
  v_outerloop PLS_INTEGER :=0;
  v_innerloop PLS_INTEGER :=5;
BEGIN
 <<Outer_loop>>
  LOOP
    v_outerloop := v_outerloop + 1;
    v_innerloop := 5;
    EXIT WHEN v_outerloop > 3;
    <<Inner_loop>>
    LOOP
      DBMS_OUTPUT.PUT_LINE('Outer loop is:'||v_outerloop||
                     ' and inner loop is: '||v_innerloop);
      v_innerloop := v_innerloop - 1;
      EXIT WHEN v_innerloop =0;
    END LOOP Inner_loop;
  END LOOP Outer_loop;
END;
```

Nested Loops and Labels

In this example, there are two loops. The outer loop is identified by the label `<<Outer_Loop>>`, and the inner loop is identified by the label `<<Inner_Loop>>`.

```
…BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN v_total_done = 'YES';
      -- Leave both loops
      EXIT WHEN v_inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
```

Summary

In this lesson, you should have learned how to:

- Construct and execute PL/SQL using nested loops
- Label loops and use the labels ın EXIT statements
- Evaluate a nested loop construct and identify the exit point