Database Programming with PL/SQL

Iterative Control: WHILE and FOR Loops

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Construct and use the `WHILE` looping construct in PL/SQL
- Construct and use the `FOR` looping construct in PL/SQL
- Describe when a `WHILE` loop is used in PL/SQL
- Describe when a `FOR` loop is used in PL/SQL

Purpose

The previous lesson discussed the basic loop, which allows the statements inside the loop to execute at least once.

This lesson introduces the `WHILE` loop and `FOR` loop. The `WHILE` loop is a looping construct which requires that the `EXIT` condition be evaluated at the start of each iteration. The `FOR` loop should be used if the number of iterations is known.

`WHILE` Loops

You can use the `WHILE` loop to repeat a sequence of statements until the controlling condition is no longer `TRUE`. The condition is evaluated at the start of each iteration.

The loop terminates when the condition is `FALSE` or `NULL`. If the condition is `FALSE` or `NULL` at the start of the loop, then no further iterations are performed.

```
WHILE condition LOOP
   statement1;
   statement2;
   . . .
END LOOP;
```

WHILE Loops (cont.)

In the syntax:

- Condition is a Boolean variable or expression (TRUE, FALSE, or NULL)
- Statement can be one or more PL/SQL or SQL statements

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

`WHILE` Loops (cont.)

In the syntax:

- If the variables involved in the conditions do not change during the body of the loop, then the condition remains `TRUE` and the loop does not terminate.
  - Note: If the condition yields `NULL`, then the loop is bypassed and control passes to the statement that follows the loop.

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

`WHILE` Loops (cont.)

In this example, three new location IDs for the country code CA and the city Montreal are being added. The counter is explicitly declared in this example.

```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
  v_counter     NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
```

WHILE Loops (cont.)

With each iteration through the WHILE loop, a counter (v_counter) is incremented. If the number of iterations is less than or equal to the number 3, then the code within the loop is executed and a row is inserted into the locations table.

```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
  v_counter     NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
```

WHILE Loops (cont.)

After the counter exceeds the number of new locations for this city and country, the condition that controls the loop evaluates to FALSE and the loop is terminated.

```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
  v_counter     NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
  END LOOP;
END;
```

FOR Loops Described

FOR loops have the same general structure as the basic loop. In addition, they have a control statement before the LOOP keyword to set the number of iterations that PL/SQL performs.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

FOR Loop Rules

FOR loop rules:

- Use a FOR loop to shortcut the test for the number of iterations.
- Do not declare the counter; it is declared implicitly.
- *lower_bound .. upper_bound* is the required syntax.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

`FOR` Loops Syntax

- *`Counter`* is an implicitly declared integer whose value automatically increases or decreases (decreases if the `REVERSE` keyword is used) by 1 on each iteration of the loop until the upper or lower bound is reached.
- `REVERSE` causes the counter to decrement with each iteration from the upper bound to the lower bound. (Note that the lower bound is referenced first.)

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

`FOR` Loops Syntax (cont.)

- *`lower_bound`* specifies the lower bound for the range of counter values.
- *`upper_bound`* specifies the upper bound for the range of counter values.
- Do not declare the counter. It is declared implicitly as an integer.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

FOR Loop Example

You have already learned how to insert three new locations for the country code CA and the city Montreal by using the simple LOOP and the WHILE loop. The slide shows you how to achieve the same by using the FOR loop.

```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id
    FROM locations
    WHERE country_id = v_countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + i), v_new_city, v_countryid );
  END LOOP;
END;
```

`FOR` Loop Guidelines

- Reference the counter within the loop only; it is undefined outside the loop.
- Do not reference the counter as the target of an assignment.
- Neither loop bound should be `NULL`.

FOR Loop Expression Example

While writing a FOR loop, the lower and upper bounds of a LOOP statement do not need to be numeric literals. They can be expressions that convert to numeric values.

```
DECLARE
  v_lower  NUMBER := 1;
  v_upper  NUMBER := 100;
BEGIN
  FOR i IN v_lower..v_upper LOOP
  ...
  END LOOP;
END;
```

Guidelines For When to Use Loops

- Use the basic loop when the statements inside the loop must execute at least once.
- Use the `WHILE` loop if the condition has to be evaluated at the start of each iteration.
- Use a `FOR` loop if the number of iterations is known.

Terminology

Key terms used in this lesson included:

- `FOR` loops
- `WHILE` loops

Summary

In this lesson, you should have learned how to:

- Construct and use the `WHILE` looping construct in PL/SQL
- Construct and use the `FOR` looping construct in PL/SQL
- Describe when a `WHILE` loop is used in PL/SQL
- Describe when a `FOR` loop is used in PL/SQL