

Iterative Control: LOOP Statements

- Loops repeat a statement (or a sequence of statements) multiple times.
- There are three loop types:
 - Basic loop
 - FOR loop
 - WHILE loop

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Iterative Control: LOOP Statements

PL/SQL provides several facilities to structure loops to repeat a statement or sequence of statements multiple times. Loops are mainly used to execute statements repeatedly until an exit condition is reached. It is mandatory to have an exit condition in a loop; otherwise, the loop is infinite.

Looping constructs are the third type of control structures. PL/SQL provides the following types of loops:

- Basic loop that performs repetitive actions without overall conditions

- FOR loops that perform iterative actions based on a count

- WHILE loops that perform iterative actions based on a condition Note:

An EXIT statement can be used to terminate loops. A basic loop must have an EXIT. The cursor FOR loop (which is another type of FOR loop) is discussed in the lesson titled “Using Explicit Cursors.”

Basic Loops

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Basic Loops

The simplest form of a LOOP statement is the basic loop, which encloses a sequence of statements between the LOOP and ENDLOOP keywords. Each time the flow of execution reaches the ENDLOOP statement, control is returned to the corresponding LOOP statement above it. A basic loop allows execution of its statements at least once, even if the EXIT condition is already met upon entering the loop. Without the EXIT statement, the loop would be infinite.

EXIT Statement

You can use the EXIT statement to terminate a loop. Control passes to the next statement after the ENDLOOP statement. You can issue EXIT either as an action within an IF statement or as a stand-alone statement within the loop. The EXIT statement must be placed inside a loop. In the latter case, you can attach a WHEN clause to enable conditional termination of the loop. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop ends and control passes to the next statement after the loop.

A basic loop can contain multiple EXIT statements, but it is recommended that you have only one EXIT point.

Basic Loop: Example

```
DECLARE
  v_countryid  locations.country_id%TYPE := 'CA';
  v_loc_id     locations.location_id%TYPE;
  v_counter    NUMBER(2) := 1;
  v_new_city   locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  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;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Basic Loop: Example

The basic loop example shown in the slide is defined as follows: “Insert three new location IDs for the CA country code and the city of Montreal.”

Note

A basic loop allows execution of its statements until the EXITWHEN condition is met.

If the condition is placed in the loop such that it is not checked until after the loop statements execute, the loop executes at least once.

However, if the exit condition is placed at the top of the loop (before any of the other executable statements) and if that condition is true, the loop exits and the statements never execute.

Results

To view the output, run the code example: code_05_22_s.sql.

WHILE Loops

Syntax:

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

Use the **WHILE** loop to repeat statements while a condition is **TRUE**.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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, no further iterations are performed. Thus, it is possible that none of the statements inside the loop are executed.

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

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

WHILE Loops: 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;
/
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

WHILE Loops: Example

In the example in the slide, three new location IDs for the CA country code and the city of Montreal are added.

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, the code within the loop is executed and a row is inserted into the locations table.

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

Results

To view the output, run the code example: code_05_24_s.sql.

FOR Loops

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

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

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

FOR Loops

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 the PL/SQL performs. In the 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

Do not declare the counter. It is declared implicitly as an integer.

REVERSE Causes the counter to decrement with each iteration from the upper bound to the lower bound

Note: The lower bound is still referenced first.

lower_bound Specifies the lower bound for the range of counter values

upper_bound Specifies the upper bound for the range of counter values
(continued)

Note: The sequence of statements is executed each time the counter is incremented, as determined by the two bounds. The lower bound and upper bound of the loop range can be literals, variables, or expressions, but they must evaluate to integers. The bounds are rounded to integers; that is, 11/3 and 8/5 are valid upper or lower bounds. The lower

bound and upper bound are inclusive in the loop range. If the lower bound of the loop range evaluates to a larger integer than the upper bound, the sequence of statements is not executed.

For example, the following statement is executed only once:

```
FOR i IN 3..3  
LOOP statement1;  
END LOOP;
```

FOR Loops: Example

```
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;
/
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

FOR Loops: Example

You have already learned how to insert three new locations for the CA country code and the city of Montreal by using the basic loop and the WHILE loop. The example in this slide shows how to achieve the same by using the FOR loop.

Results

To view the output, run the code example code_05_27_s.sql.

FOR Loop Rules

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

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

FOR Loop Rules

The slide lists the guidelines to follow when writing a FOR loop.

Note: 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.

Example:

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

Suggested Use of Loops

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

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Suggested Use of Loops

A basic loop allows the execution of its statement at least once, even if the condition is already met upon entering the loop. Without the `EXIT` statement, the loop would be infinite.

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`. If the condition is `FALSE` at the start of the loop, no further iterations are performed.

FOR loops have a control statement before the LOOP keyword to determine the number of iterations that the PL/SQL performs. Use a FOR loop if the number of iterations is predetermined.

Nested Loops and Labels

- You can nest loops to multiple levels.
- Use labels to distinguish between blocks and loops.
- Exit the outer loop with the `EXIT` statement that references the label.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Nested Loops and Labels

You can nest the FOR, WHILE, and basic loops within one another. The termination of a nested loop does not terminate the enclosing loop unless an exception is raised. However, you can label loops and exit the outer loop with the EXIT statement.

Label names follow the same rules as the other identifiers. A label is placed before a statement, either on the same line or on a separate line. White space is insignificant in all PL/SQL parsing except inside literals. Label basic loops by placing the label before the word LOOP within label delimiters (<<label>>). In FOR and WHILE loops, place the label before FOR or WHILE.

If the loop is labeled, the label name can be included (optionally) after the ENDOLOOP statement for clarity.

Nested Loops and Labels: Example

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

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Nested Loops and Labels: Example

In the example in the slide, 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>>.

The identifiers are placed before the word LOOP within label delimiters (<<label>>). The inner loop is nested within the outer loop. The label names are included after the ENDLOOP statements for clarity.

PL/SQL CONTINUE Statement

– Definition

- Adds the functionality to begin the next loop iteration
- Provides programmers with the ability to transfer control to the next iteration of a loop
- Uses parallel structure and semantics to the EXIT statement

– Benefits

- Eases the programming process
- May provide a small performance improvement over the previous programming workarounds to simulate the CONTINUE statement

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

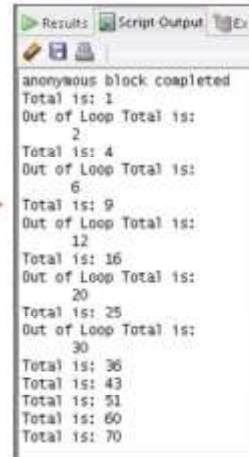
PL/SQL CONTINUE Statement

The CONTINUE statement enables you to transfer control within a loop back to a new iteration or to leave the loop. Many other programming languages have this functionality. With the Oracle Database 11g release, PL/SQL also offers this functionality. Before the Oracle Database 11g release, you could code a workaround by using Boolean variables and conditional statements to simulate the CONTINUE programmatic functionality. In some cases, the workarounds are less efficient. The CONTINUE statement offers you a simplified means to control loop iterations. It may be more efficient than the previous coding workarounds.

The CONTINUE statement is commonly used to filter data within a loop body before the main processing begins.

PL/SQL CONTINUE Statement: Example 1

```
DECLARE
  v_total SIMPLE_INTEGER := 0;
BEGIN
  FOR i IN 1..10 LOOP
    ① v_total := v_total + i;
      dbms_output.put_line
        ('Total is: ' || v_total);
      CONTINUE WHEN i > 5;
    ② v_total := v_total + i;
      dbms_output.put_line
        ('Out of Loop Total is:
        ' || v_total);
      END LOOP;
  END;
```



```
anonymous block completed
Total is: 1
Out of Loop Total is:
2
Total is: 4
Out of Loop Total is:
6
Total is: 9
Out of Loop Total is:
12
Total is: 16
Out of Loop Total is:
20
Total is: 25
Out of Loop Total is:
30
Total is: 36
Total is: 43
Total is: 51
Total is: 60
Total is: 70
```

PL/SQL CONTINUE Statement: Example 1

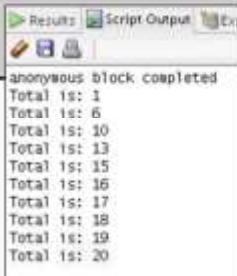
This is a 11g only code. Continue statement was introduced only in 11g. In the example, there are two assignments using the v_total variable:

1. The first assignment is executed for each of the 10 iterations of the loop.
 2. The second assignment is executed for the first five iterations of the loop.
- The CONTINUE statement transfers control within a loop back to a new iteration, so for the last five iterations of the loop, the second TOTAL assignment is not executed.

The end result of the TOTAL variable is 70.

PL/SQL CONTINUE Statement: Example 2

```
DECLARE
  v_total NUMBER := 0;
BEGIN
  <<BeforeTopLoop>>
  FOR i IN 1..10 LOOP
    v_total := v_total + 1;
    dbms_output.put_line
      ('Total is: ' || v_total);
    FOR j IN 1..10 LOOP
      CONTINUE BeforeTopLoop WHEN i + j > 5;
      v_total := v_total + 1;
    END LOOP;
  END LOOP;
END two_loop;
```



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

PL/SQL CONTINUE Statement: Example 2

You can use the CONTINUE statement to jump to the next iteration of an outer loop. This is a 11g only code.

To do this, provide the outer loop a label to identify where the CONTINUE statement should go.

The CONTINUE statement in the innermost loop terminates that loop whenever the WHEN condition is true (just like the EXIT keyword). After the innermost loop is terminated by the CONTINUE statement, control transfers to the next iteration of the outermost loop labeled BeforeTopLoop in this example.

When this pair of loops completes, the value of the TOTAL variable is 20.

You can also use the CONTINUE statement within an inner block of code, which does not contain a loop as long as the block is nested inside an appropriate outer loop.

Restrictions

The CONTINUE statement cannot appear outside a loop at all—this generates a compiler error.

You cannot use the CONTINUE statement to pass through a procedure, function, or method boundary—this generates a compiler error.