

# WEEK 7 CONTROL STRUCTURE - LOOPING

**IM101 - Advance Database System** 

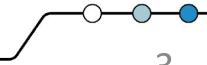
#### LEARNING OUTCOMES:

At the end of the session, the students should be able to:

- 1. Recognize different types of PL/SQL loops.
- 2. Create PL/SQL blocks containing basic loop, for loop, and while loop, nested loop.

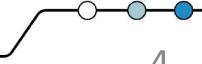
## PL/SQL Looping Statement

- Looping constructs are the second type of control structure.
- Loops are used to execute statements repeatedly until an EXIT condition is reached.
- PL/SQL provides three ways to structure loops: basic loops, FOR loops, and WHILE loops.



# PL/SQL – Types of Loops

- Basic loops that perform repetitive actions without overall conditions
- FOR loops that perform iterative actions based on a counter
- WHILE loops that perform repetitive actions based on a condition



### **Basic Loops**

 Use the basic loop when the statements inside the loop must execute at least once.

Encloses a sequence of statements between the keywords LOOP

and END LOOP.

```
BEGIN
  LOOP
    statements;
  EXIT [WHEN condition];
  END LOOP;
END;
```

#### **Basic Loop –** Simple Example

We simply display the loop counter each time we repeat the loop.

```
DECLARE
  v_counter    NUMBER(2) := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('Loop execution #' || v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 5;
END LOOP;
END;
```

#### **Basic Loop** – More Complex Example

In this example, three new location IDs for Montreal, Canada, are inserted in the LOCATIONS table.

```
DECLARE
v loc id locations.location id%TYPE;
v counter NUMBER(2) := 1;
BEGIN
 SELECT MAX(location id) INTO v loc id FROM locations
 WHERE country id = 2;
 LOOP
  INSERT INTO locations (location id, city, country id)
 VALUES((v loc id + v counter), 'Montreal', 2);
  v counter := v counter + 1;
 EXIT WHEN v counter > 3;
 END LOOP;
END:
```

#### **Basic Loop** – EXIT Statement

 You can use the EXIT statement to terminate a loop and pass control to the next statement after the END LOOP statement.

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
   DBMS_OUTPUT.PUT_LINE('Counter is ' || v_counter);
   v_counter := v_counter + 1;
   IF v_counter > 10 THEN EXIT;
   END IF;
  END LOOP;
END;
```

#### **Basic Loop** – EXIT WHEN Statement

If the WHEN clause evaluates to TRUE, the loop ends and control
passes to the next statement following END LOOP.

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('Counter is ' || v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
END LOOP;
END;
```

#### WHILE Loop

- The WHILE loop is a looping construct which requires the controlling condition be evaluated at the start of each iteration.
- You can use the WHILE loop to repeat a sequence of statements until the controlling condition is no longer TRUE.

```
Syntax:
```

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

```
DECLARE
   a number(2) := 10;
BEGIN
   WHILE a < 20 LOOP
      dbms_output.put_line('value of a: ' || a);
      a := a + 1;
   END LOOP;
END;</pre>
```

### WHILE Loop - Example

In this example, three new location IDs for Montreal, Canada, are inserted in the LOCATIONS table.

```
DECLARE
v_loc_id locations.location_id%TYPE;
v counter NUMBER := 1;
BEGIN
 SELECT MAX (location id) INTO v loc id FROM locations
 WHERE country id = 2;
 WHILE v counter <= 3 LOOP
  INSERT INTO locations (location id, city, country id)
 VALUES((v loc id + v counter), 'Montreal', 2);
 v counter := v counter + 1;
END LOOP;
END;
```

#### WEEK 7 – Control Structure – Looping

#### **FOR Loop**

- 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
- Do not declare the counter; it is declared implicitly. 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.
- *lower\_bound .. upper\_bound* specifies the range of the counter value. It is required.

```
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. •

This slide shows you how to achieve the same by using the FOR loop.

```
DECLARE
v_loc_id locations.location_id%TYPE;
BEGIN

SELECT MAX(location_id) INTO v_loc_id FROM locations
   WHERE country_id = 2;
FOR i IN 1..3 LOOP
   INSERT INTO locations(location_id, city, country_id)
   VALUES((v_loc_id + i), 'Montreal', 2);
END LOOP;
END;
```

#### **FOR Loop Expression**

- 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;
```

#### **Nested Loop**

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

#### **WEEK 7 – Control Structure – Looping**

#### Reference

PL/SQL User's Guide and Reference, Release 9.0.1 Part No. A89856-01

Copyright © 1996, 2001, Oracle Corporation. All rights reserved.

Primary Authors: Tom Portfolio, John Russell

Oracle Academy

Database Programming with PL/SQL 4-3 Iterative Control: Basic Loops

Database Programming with PL/SQL 4-4 Iterative Control: While and For Loops

Database Programming with PL/SQL 4-5 Iterative Control: Nested Loops

# END OF PRESENTATION. THANK YOU!