PL/SQL - FOR LOOP STATEMENT

A **FOR LOOP** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

```
FOR counter IN initial_value .. final_value LOOP
   sequence_of_statements;
END LOOP;
```

Here is the flow of control in a for loop:

- The initial step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the condition i.e. *initial_value* .. *final_value* is evaluated. If it is TRUE, the body of the loop is executed. If it is FALSE, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the value of the *counter* variable in increased or decreased.
- The condition is now evaluated again. If it is TRUE, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes FALSE, the FOR-LOOP terminates.

Following are some special characteristics of PL/SQL for loop:

- The *initial_value* and *final_value* of the loop variable or *counter* can be literals, variables, or expressions but must evaluate to numbers. Otherwise, PL/SQL raises the predefined exception VALUE_ERROR.
- The *initial_value* need not to be 1; however, the **loop counter increment (or decrement) must be 1**.
- PL/SQL allows determine the loop range dynamically at run time.

Example:

```
DECLARE
   a number(2);
BEGIN
   FOR a in 10 .. 20 LOOP
       dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20

PL/SQL procedure successfully completed.
```

Reverse FOR LOOP Statement

By default, iteration proceeds from the initial value to the final value, generally upward from the lower bound to the higher bound. You can reverse this order by using the **REVERSE** keyword. In such case, iteration proceeds the other way. After each iteration, the loop counter is decremented.

However, you must write the range bounds in ascending (not descending) order. The following program illustrates this:

```
DECLARE
   a number(2) ;
BEGIN
   FOR a IN REVERSE 10 .. 20 LOOP
      dbms_output.put_line('value of a: ' || a);
   END LOOP;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
value of a: 20
value of a: 19
value of a: 18
value of a: 17
value of a: 16
value of a: 15
value of a: 14
value of a: 13
value of a: 12
value of a: 11
value of a: 10

PL/SQL procedure successfully completed.
```