Exercises for Chapter 6: Iterative Control: Part I

The Labs below provide you with exercises and suggested answers with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

Lab 6.1 Simple Loops

Answer the following questions:

`EXIT` Statement

In this exercise, you will use the `EXIT` condition to terminate a simple loop. With each iteration of the loop, the value of the loop counter is decremented and then displayed on the screen.

Create the following PL/SQL script:

For Example *ch06_7a.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 5;
BEGIN
   LOOP
      -- decrement loop counter by one
      v_counter := v_counter - 1;
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);

      -- if EXIT condition yields TRUE exit the loop
      IF v_counter = 0 THEN
         EXIT;
      END IF;

   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Execute the script, and then answer the following questions:

a) Execute the script and explain the output produced.

Answer: The output should look like the following:

```
v_counter = 4
v_counter = 3
v_counter = 2
v_counter = 1
v_counter = 0
Done…
```

Every time the loop is run, the statements in the body of the loop are executed. In this script, the value of the loop counter variable `v_counter` is decremented by 1 and displayed in the Dbms Output window. Next, the `EXIT` condition is evaluated. Once the value of the loop counter variable `v_counter` reaches 0, the loop is terminated as the `EXIT` condition evaluates to `TRUE`. After the loop has terminated, 'Done...' is displayed in the Dbms Output window.

b) How many times was the loop executed?

Answer: The loop was executed five times. Once the value of the variable `v_counter` reaches 0, the `IF` statement

```
IF v_counter = 0
THEN
   EXIT;
END IF;
```

evaluates to `TRUE`, and the loop is terminated.

c) What is the `EXIT` condition for this loop?

Answer: The `EXIT` condition for this loop is `v_counter = 0`. The `EXIT` condition is used as a part of an `IF` statement. The `IF` statement evaluates the `EXIT` condition to `TRUE` or `FALSE`, based on the current value of the variable `v_counter`.

d) How many times the value of the variable `v_counter` will be displayed if the `DBMS_OUTPUT.PUT_LINE` statement is moved after the `END IF` statement?

Answer: The value of the variable `v_counter` will be displayed four times. Consider the following code fragment:

```
LOOP
   -- decrement loop counter by one
   v_counter := v_counter - 1;

   -- if EXIT condition yields TRUE exit the loop
   IF v_counter = 0
   THEN
      EXIT;
   END IF;

   DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
END LOOP;
```

Assume that the loop has iterated four times already. Then the value of the variable `v_counter` is decremented by 1, so it becomes 0. Next, the `IF` statement evaluates the `EXIT` condition of the loop. Because the value of the `v_counter` has reached 0, the

EXIT condition yields TRUE, and the loop is terminated. As a result, the DBMS_OUTPUT.PUT_LINE statement is not executed for the last and fifth iteration of the loop because control is passed to the next executable statement after the END LOOP statement. Thus, only four values of the variable v_counter are displayed in the Dbms Output window. This is illustrated further by the output below:

```
v_counter = 4
v_counter = 3
v_counter = 2
v_counter = 1
Done…
```

e) Why does the number of times the loop counter value is displayed differ when the DBMS_OUTPUT.PUT_LINE statement is placed after the END IF statement?

Answer: When the DBMS_OUTPUT.PUT_LINE statement is placed before the IF statement, the value of the variable v_counter is displayed first. Then it is evaluated by the IF statement. As a result, for the fifth iteration of the loop 'v_counter = 0' is displayed first, then the EXIT condition evaluates to TRUE and the loop is terminated. When the DBMS_OUTPUT.PUT_LINE statement is placed after the END IF statement, the EXIT condition is evaluated prior to the execution of the DBMS_OUTPUT.PUT_LINE statement. Thus, for the fifth iteration of the loop, the EXIT condition evaluates to TRUE before the value of the variable v_counter is displayed in the Dbms Output window by the DBMS_OUTPUT.PUT_LINE statement.

EXIT WHEN Statement

In this exercise, you will use the EXIT WHEN statement to terminate the loop after ten iterations. For each iteration of the loop, you will display a text showing the value of the loop counter and indicating whether it is even or odd.

Create the following PL/SQL script:

For Example *ch06_8a.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 0;
BEGIN
   LOOP
      -- increment loop counter by one
      v_counter := v_counter + 1;

      -- determine whether v_counter is even or odd
      CASE
         WHEN MOD(v_counter, 2) = 0
         THEN
            DBMS_OUTPUT.PUT_LINE (v_counter||' is even');
         ELSE
            DBMS_OUTPUT.PUT_LINE (v_counter||' is odd');
         END CASE;

      -- if EXIT WHEN condition yields TRUE exit the loop
      EXIT WHEN v_counter = 10;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Execute the script, and then answer the following questions:

a) Execute the script and explain the output produced.

Answer: The output should look like the following:

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
Done…
```

The loop in the script above executes 10 times. For each iteration of the loop, the loop counter variable `v_counter` is incremented by 1. Then, its value is checked via the `CASE` statement. If the value happens to be an even number, then the message '… is even' is displayed in the Dbms Output window. If the value happens to be an odd number, then '… is odd' is displayed in the Dbms Output window. Finally, the `EXIT WHEN` condition is evaluated. Once the value of the loop counter variable `v_counter` reaches 10, the loop is terminated as the `EXIT WHEN` condition evaluates to `TRUE`. After the loop is terminated, 'Done...' is displayed in the Dbms Output window.

b) How many times will the loop be executed if the variable `v_counter` is incremented by 2?

Answer: The loop will be executed 5 times. Consider modified version of the script (changes are highlighted in bold).

For Example *ch06_8b.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 0;
BEGIN
   LOOP
      -- increment loop counter by two
      v_counter := v_counter + 2;

      -- determine whether v_counter is even or odd
      CASE
         WHEN MOD(v_counter, 2) = 0
         THEN
            DBMS_OUTPUT.PUT_LINE (v_counter||' is even');
         ELSE
            DBMS_OUTPUT.PUT_LINE (v_counter||' is odd');
         END CASE;
      -- if EXIT WHEN condition yields TRUE exit the loop
      EXIT WHEN v_counter = 10;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Note that in this version, for each iteration of the loop, the variable `v_counter` is incremented by 2. This means that for iteration 1, its value becomes 2, for iteration 2, it becomes 4, and so forth. As a result, the `EXIT` condition of the loop is reached on the fifth iteration of the loop, and the value of the variable `v_counter` is never odd. This is further illustrated by the output below:

```
2 is even
4 is even
6 is even
8 is even
10 is even
Done…
```

c) How many times the loop will be executed if the statement that increments the loop counter variable is moved to after the `EXIT WHEN` statement as indicated below?

For Example *ch06_8c.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 0;
BEGIN
   LOOP
      -- determine whether v_counter is even or odd
      CASE
         WHEN MOD(v_counter, 2) = 0
         THEN
            DBMS_OUTPUT.PUT_LINE (v_counter||' is even');
         ELSE
            DBMS_OUTPUT.PUT_LINE (v_counter||' is odd');
         END CASE;

      -- if EXIT WHEN condition yields TRUE exit the loop
      EXIT WHEN v_counter = 10;

      -- increment loop counter by 1
      v_counter := v_counter + 1;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Answer: The loop will be executed 11 times. In the original version of the script, the loop counter is incremented by 1 before the `EXIT` condition is evaluated. As a result, for the

first iteration of the loop the value of the variable `v_counter` is 1 when the `EXIT` condition is evaluated for the first time. Similarly, for the second iteration of the loop, the value of the variable `v_counter` is 2 when the `EXIT` condition is evaluated for the second time. This behavior continues for all latter iterations of the loop.
In the new version of the script, the loop counter remains 0 when the `EXIT` condition is evaluated for the first time, and it is incremented by 1 after the `EXIT WHEN` statement. Then, for the second iteration of the loop, the value of `v_counter` is 1 when the `EXIT` condition is evaluated for the second time, and so on. For the last (eleventh) iteration of the loop, the value of `v_counter` is 10 and the `EXIT` condition evaluates to `TRUE` causing the loop to terminate. This is illustrated further by the output below:

```
0 is even
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
Done…
```

Lab 6.2 `WHILE` Loops

Answer the following questions:

Using `WHILE` Loops

In this exercise, you will use a `WHILE` loop to calculate the sum of the integers between 1 and 10.

Create the following PL/SQL script:

For Example *ch06_9a.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 1;
   v_sum     BINARY_INTEGER := 0;
BEGIN
   WHILE v_counter <= 10
   LOOP
      v_sum := v_sum + v_counter;
      DBMS_OUTPUT.PUT_LINE ('Current sum is: '||v_sum);

      -- increment loop counter by one
      v_counter := v_counter + 1;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('The sum of integers between 1 and 10 is: '||v_sum);
END;
```

Execute the script, and then answer the following questions:

a) Execute the script and explain the output produced.

Answer: The output should look like the following:

```
Current sum is: 1
Current sum is: 3
Current sum is: 6
Current sum is: 10
Current sum is: 15
Current sum is: 21
Current sum is: 28
Current sum is: 36
Current sum is: 45
Current sum is: 55
The sum of integers between 1 and 10 is: 55
```

Every time the loop is run, the value of the variable `v_counter` is checked in the test condition. While the value of the variable `v_counter` is less than or equal to 10, the statements inside the body of the loop are executed. In this script, the value of the variable `v_sum` is calculated and displayed in the Dbms Output window. Next, the value of the variable `v_counter` is incremented, and the control of the execution is passed back to the top of the loop. Once the value of `v_counter` reaches 11, the loop is terminated. For the first iteration of the loop, the value of the variable `v_sum` equals 1 based on the statement

```
v_sum := v_sum + v_counter
```

Then, for the second iteration of the loop, the value of the variable `v_sum` equals 3, because 2 (value of `v_counter`) is added to the old value of `v_sum` (which is 1), and so on. After the loop terminates, the message 'The sum of integers...' is displayed in the Dbms Output window.

b) What is the test condition for this loop?

Answer: The test condition for this loop is `v_counter <= 10`.

c) How many times was the loop executed?

Answer: The loop was executed 10 times. Once the value of the variable `v_counter` reaches 11, the test condition

```
v_counter <= 10
```

evaluates to `FALSE`, and the loop terminates. As mentioned earlier, the loop counter tracks the number of times the loop is executed. You will notice that in this exercise, the maximum value of `v_counter` equals to the number of times the loop iterates.

d) How many times will the loop be executed

i) if the variable `v_counter` is not initialized?

ii) if the variable `v_counter` is initialized to 0?

iii) if the varaible `v_counter` is initialized to 10?

Answer:

i) If the value of the variable `v_counter` is not initialized to some value, the loop will not execute at all. In order for the loop to execute at least once, the test condition must evaluate to TRUE at least once. If the value of the variable v_counter is only declared and not initialized, it is NULL. It is important to remember that null variables cannot be compared to other variables or values. Therefore, the test condition

```
v_counter <= 10
```

never evaluates to TRUE, and the loop is not executed at all.

ii) If the variable v_counter is initialized to 0, the loop will execute 11 times since the minimum value of v_counter has decreased by 1. When the variable v_counter is initialized to 0, the range of integers for which the test condition of the loop evaluates to TRUE becomes 0 to 10. The given range of the integers has eleven numbers in it. As a result, the loop will iterate eleven times.

iii) If the variable v_counter is initialized to 10, the loop will execute only once. When initial value of v_counter equals to 10, the test condition evaluates to TRUE for the first iteration of the loop. Inside the body of the loop, the value of the variable v_counter is incremented by one. As a result, for the second iteration of the loop, the test condition evaluates to FALSE, since 11 is not less than or equal to 10, and control of the execution is passed to the first statement after the loop.

e) How will the value of the variable v_sum change

i) if the variable v_counter is not initialized?
ii) if the variable v_counter is initialized to 0?
iii) if the varaible v_counter is initialized to 10?

Answer:

i) If the value of the variable v_counter is not initialized to some value, the loop will not execute at all. Therefore, the value of v_sum does not change from its initial value; it stays 0.

ii) If the variable v_counter is initialized to 0, the loop will execute 11 times since the minimum value of v_counter has decreased by 1. Thus, the value of the variable v_sum is calculated 11 times, as well. However, after the loop completes, the value of v_sum remains 55, because 0 is added to the variable v_sum during first iteration of the loop.

iii) If the variable v_counter is initialized to 10, the loop will execute only once. As a result, the value of the variable v_sum is incremented only once by 10 and it remains 10 after the loop completes.

f) What will be the value of the variable v_sum if it is not initialized at the beginning of the script?

Answer: The value of the variable v_sum will remain NULL if it is not initialized to some value. The value of the variable v_sum is calculated in the statement

```
v_sum := v_sum + 1
```

Because NULL + 1 always evaluates to NULL, the value of the variable v_sum will remain NULL regardless of the number of times the loop iterates.

Lab 6.3 Numeric `FOR` Loops

Answer the following questions:

Using the `IN` Option in the Loop

In this exercise, you will use a numeric FOR loop to calculate a factorial of 10 (10! = 1*2*3…*10).

Create the following PL/SQL script:

For Example *ch06_10a.sql*

```
DECLARE
   v_factorial NUMBER := 1;
BEGIN
   FOR v_counter IN 1..10
   LOOP
      v_factorial := v_factorial * v_counter;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Factorial of ten is: '||v_factorial);
END;
```

Execute the script, and then answer the following questions:

a) What output was generated by the script?

Answer: The output should look like the following:

```
Factorial of ten is: 3628800
```

Every time the loop is run, the value of the loop counter `v_counter` is incremented by 1 implicitly, and the current value of the factorial is calculated. Once the value of the loop counter `v_counter` increases to 10, the loop is run for the last time. At this point, the final value of the factorial is calculated, and the loop is terminated. After the loop has terminated, control is passed to the first statement outside of the loop, in this case, `DBMS_OUTPUT.PUT_LINE` statement.

b) How many times was the loop executed?

Answer: The loop was executed ten times according to the range specified by the lower limit and the upper limit of the loop. In this example, the lower limit is equal to 1, and upper limit is equal to 10.

c) What is the value of the loop counter before the loop?

Answer: The loop counter is defined implicitly by the loop. Therefore, before the loop, the loop counter is undefined and has no value.

d) What is the value of the loop counter after the loop?

Answer: Similarly to the above, after the loop has completed, the loop counter is undefined again and can hold no value.

e) How many times will the loop be executed if the value of the loop counter `v_counter` is incremented by 5 inside the body of the loop?

   Answer: If the value of the loop counter `v_counter` is incremented by 5 inside the body of the loop, the PL/SQL block will not compile successfully. The statement

   ```
   v_counter := v_counter + 5;
   ```

   will cause the following error message:

   ```
   ORA-06550: line 7, column 7:
   PLS-00363: expression 'V_COUNTER' cannot be used as an assignment target
   ORA-06550: line 7, column 7:
   PL/SQL: Statement ignored
   ```

Using the REVERSE Option in the Loop

In this exercise, you will use the `REVERSE` option to specify the range of numbers used by the loop to iterate. You will display a list of even numbers starting from 10 going down to 0. Try to answer the questions before you run the script. Once you have answered the questions, run the script and check your results.

Create the following PL/SQL script:

For Example *ch06_11a.sql*

```
BEGIN
   FOR v_counter IN REVERSE 0..10
   LOOP
      -- if v_counter is even, display its value on the screen
      IF MOD(v_counter, 2) = 0
      THEN
         DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      END IF;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Answer the following questions first, and then execute the script:

a) What output do you think will be generated by this script?

   Answer: The output should look like the following:

   ```
   v_counter = 10
   v_counter = 8
   v_counter = 6
   v_counter = 4
   v_counter = 2
   v_counter = 0
   Done…
   ```

   Notice that the values of the loop counter `v_counter` are displayed in the decreasing order from 10 to 0 because the `REVERSE` option is used. However, remember that regardless of the option used, the lower limit is referenced first in the `FOR…LOOP` clause.

b) How many times will the body of the loop be executed?

Answer: The body of the loop will be executed eleven times, since the range of the integer numbers specified varies from 0 to 10.

c) How many times will the value of the loop counter `v_counter` be displayed in the Dbms Output window?

Answer: The value of the loop counter `v_counter` will be displayed six times, since the `IF` statement will evaluate to `TRUE` only for even integers.

d) How would you change this script to start the list from 0 and go up to 10?

Answer: The script should look similar to the script shown below. Changes are shown in bold. Essentially, to start the list of integers from 0 and go up to 10, the `IN` option needs to be used in the loop.

For Example *ch06_11b.sql*

```
BEGIN
   FOR v_counter IN 0..10
   LOOP
      -- if v_counter is even, display its value on the screen
      IF MOD(v_counter, 2) = 0
      THEN
         DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      END IF;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

This version of the script produces the following output:

```
v_counter = 0
v_counter = 2
v_counter = 4
v_counter = 6
v_counter = 8
v_counter = 10
Done…
```

Notice that when the `IN` option is used, the value of the loop counter `v_counter` is initialized to 0, and, with each iteration of the loop, it is incremented by 1. When the `REVERSE` option is used, the loop counter `v_counter` is initialized to 10, and its value is decremented by 1 with each iteration of the loop.

e) How would you change the script to display only odd numbers in the Dbms Output window?

Answer: The script should look similar to the following script. Changes are highlighted bold.

For Example *ch06_11c.sql*

```
BEGIN
   FOR v_counter IN REVERSE 0..10
   LOOP
      -- if v_counter is odd, display its value on the screen
      IF MOD(v_counter, 2) != 0
      THEN
         DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      END IF;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Notice that only the test condition of the `IF` statement is changed in order to display the list of odd integers, and the following output is produced:

```
v_counter = 9
v_counter = 7
v_counter = 5
v_counter = 3
v_counter = 1
Done…
```

f) How many times will the loop be executed in this case?

Answer: In this case the loop will be executed eleven times. Based on the test condition used in the `IF` statement, even or odd integers are displayed in the Dbms Output window. Depending on the test condition, the number of times the loop counter `v_counter` is displayed varies. However, the loop is executed eleven times as long as the number range specified for the lower and upper limits is 0 to 10.

Try It Yourself

The projects in this section are meant to have you use all of the skills that you have acquired throughout this chapter. Here are some exercises that will help you test the depth of your understanding.

1. Rewrite script ch06_7a.sql using a `WHILE` loop instead of a simple loop. Make sure that the output produced by this script does not differ from the output produced by the original example.

Answer: Consider the original script:

For Example *ch06_7a.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 5;
BEGIN
   LOOP
      -- decrement loop counter by one
      v_counter := v_counter - 1;
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);

      -- if EXIT condition yields TRUE exit the loop
      IF v_counter = 0 THEN
         EXIT;
      END IF;

   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

Next, consider a new version of the script that uses a `WHILE` loop. All changes are shown in bold.

For Example *ch06_7b.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 5;
BEGIN
   WHILE v_counter > 0
   LOOP
      -- decrement loop counter by one
      v_counter := v_counter - 1;
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE('Done…');
END;
```

In this version of the script, a simple loop is replaced by a `WHILE` loop. It is important to remember that a simple loop executes at least once because the `EXIT` condition is placed in the body of the loop. On the other hand, a `WHILE` loop may not execute at all because a condition is tested outside the body of the loop. So, in order to achieve the same results using the `WHILE` loop, the `EXIT` condition

```
v_counter = 0
```

used in the original version of the script is replaced by the test condition

```
v_counter > 0
```

As a result, when run, this version of the script produces the same output as the original example:

```
v_counter = 4
v_counter = 3
v_counter = 2
v_counter = 1
v_counter = 0
Done…
```

2. Rewrite script ch06_9a.sql using a numeric `FOR` loop instead of a `WHILE` loop. Make sure that the output produced by this script does not differ from the output produced by the original example.

Answer: Consider the original script:

For Example *ch06_9a.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 1;
   v_sum     BINARY_INTEGER := 0;
BEGIN
   WHILE v_counter <= 10
   LOOP
      v_sum := v_sum + v_counter;
      DBMS_OUTPUT.PUT_LINE ('Current sum is: '||v_sum);

      -- increment loop counter by one
      v_counter := v_counter + 1;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('The sum of integers between 1 and 10 is: '||v_sum);
END;
```

Next, consider a new version of the script that uses a numeric `FOR` loop. All changes are shown in bold letters.

For Example *ch06_9b.sql*

```
DECLARE
   v_sum BINARY_INTEGER := 0;
BEGIN
   FOR v_counter in 1..10
   LOOP
      v_sum := v_sum + v_counter;
      DBMS_OUTPUT.PUT_LINE ('Current sum is: '||v_sum);
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('The sum of integers between 1 and 10 is: '||v_sum);
END;
```

In this version of the script, the `WHILE` loop by the numeric `FOR` loop. As a result, there is no need to declare variable `v_counter` and increment it by 1 as these steps are implicitly handled by the loop itself. When run, this version of the script produces output identical to the output produced by the original version:

```
Current sum is: 1
Current sum is: 3
Current sum is: 6
Current sum is: 10
Current sum is: 15
Current sum is: 21
Current sum is: 28
Current sum is: 36
Current sum is: 45
Current sum is: 55
```

```
The sum of integers between 1 and 10 is: 55
```

3. Rewrite script ch06_10a.sql using a simple loop instead of a numeric `FOR` loop. Make sure that the output produced by this script does not differ from the output produced by the original example.

   Answer: Consider the original script:

   For Example *ch06_10a.sql*

```
DECLARE
   v_factorial NUMBER := 1;
BEGIN
   FOR v_counter IN 1..10
   LOOP
      v_factorial := v_factorial * v_counter;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Factorial of ten is: '||v_factorial);
END;
```

   Next, consider a new version of the script that uses a simple loop. All changes are highlighted in bold.

   For Example *ch06_10b.sql*

```
DECLARE
   v_counter   NUMBER := 1;
   v_factorial NUMBER := 1;
BEGIN
   LOOP
      v_factorial := v_factorial * v_counter;

      v_counter := v_counter + 1;
      EXIT WHEN v_counter = 10;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Factorial of ten is: '||v_factorial);
END;
```

   In this version of the script, numeric `FOR` loop is replaced by the simple loop. As a result, there are three important changes that are made to the script. First, a loop counter variable `v_counter` is declared and initialized to 1. Note that in the original version of the script, this loop counter is implicitly defined and initialized by the `FOR` loop itself. Second, the variable v_counter must be incremented by 1 explicitly. This is very important because if the statement

```
v_counter := v_counter + 1;
```

   is not included in the body of the simple loop, it becomes infinite. The step is not necessary when using numeric `FOR` loop because it is done by the loop itself.
   Third, the `EXIT` condition of the simple loop must be specified as follows:

```
EXIT WHEN v_counter = 10;
```

Note that the `EXIT` condition could also be specified via `IF-THEN` statement as well:

```
IF v_counter = 10 THEN
   EXIT;
END IF;
```

When run, this version of the script produces the same output as the original example:

```
Factorial of ten is: 362880
```