Exercises for Chapter 7: Iterative Control: Part II

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 7.1 `CONTINUE` Statement

Answer the following questions:

Using `CONTINUE` Statement

The script for this exercise is based on the script ch06_7a.sql used in the previous chapter. The original script uses the `EXIT` condition to terminate a simple loop, and a special variable, `v_counter`, which keeps count of the loop iterations. With each iteration of the loop, the value of `v_counter` is decremented and displayed on the screen.

The new version of the script used in this exercise adds the `CONTINUE` condition which affects the output produced by the script.

Create the following PL/SQL script:

For Example *ch07_5a.sql*

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

      -- if CONTINUE condition yields TRUE pass control to the first
      -- executable statement of the loop
      IF v_counter > 3
      THEN
         CONTINUE;
      END IF;

      DBMS_OUTPUT.PUT_LINE
```

```
         ('After continue condition, 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) What output was generated by the script?

Answer: The output should look like the following:

```
Before continue condition, v_counter = 4
Before continue condition, v_counter = 3
After continue condition, v_counter = 3
Before continue condition, v_counter = 2
After continue condition, v_counter = 2
Before continue condition, v_counter = 1
After continue condition, v_counter = 1
Before continue condition, v_counter = 0
After continue condition, v_counter = 0
Done…
```

b) Explain the output produced by the script?

Answer: For the first iteration of the loop, the value of the variable `v_counter` changes from 5 to 4, and the `CONTINUE` condition

```
IF v_counter > 3
THEN
   CONTINUE;
END IF;
```

evaluates to `TRUE`. As a result, the control of the execution is passed back to the first executable statement inside the body of the loop. Therefore, only first `DBMS_OUTPUT.PUT_LINE` statement is executed as shown below

```
Before continue condition, v_counter = 4
```

In other words, for the first iteration of the loop, only part of the loop prior to the `CONTINUE` statement is executed.

For the next four iterations of the loop, the `CONTINUE` condition evaluates to `FALSE`, and the entire body of the loop is executed. This is further illustrated by the output below:

```
Before continue condition, v_counter = 3
After continue condition, v_counter = 3
Before continue condition, v_counter = 2
After continue condition, v_counter = 2
Before continue condition, v_counter = 1
After continue condition, v_counter = 1
Before continue condition, v_counter = 0
```

```
After continue condition, v_counter = 0
```

In this case, both `DBMS_OUTPUT.PUT_LINE` statements are executed, and the values of the `v_counter` variable before and after the `CONTINUE` condition are displayed in the Dbms Output window.

Note that for the fifth and last iteration of the loop, the `EXIT` condition evaluates to `TRUE` and the loop terminates, and the last `DBMS_OUTPUT.PUT_LINE` statement is executed as well, so 'Done…' is displayed in the Dbms Output window.

c) How many times was the loop executed?

Answer: The loop was executed five times as the number of the loop iterations is controlled by the `EXIT` condition, and not by the `CONTINUE` condition.

d) Explain how each iteration of the loop will be affected if the `CONTINUE` condition is changed to the

i) `v_counter = 3`?
ii) `v_counter < 3`?

Answer:

i) Changing the `CONTINUE` condition to the

```
IF v_counter = 3
THEN
   CONTINUE;
END IF;
```

affects the third iteration of the loop only.

As long as the value of the variable `v_counter` is not equal to 3, the `CONTINUE` condition will evaluate to `FALSE`. As a result, for the first, third, fourth, and fifth iterations of the loop, all statements inside the body of the loop will be executed. For the second iteration of the loop (highlighted in bold), the `CONTINUE` condition will evaluate to `TRUE` causing partial execution of the loop. Thus, the control of the execution will be passed to the first statement inside the body of the loop as illustrated by the output below:

```
Before continue condition, v_counter = 4
After continue condition, v_counter = 4
Before continue condition, v_counter = 3
Before continue condition, v_counter = 2
After continue condition, v_counter = 2
Before continue condition, v_counter = 1
After continue condition, v_counter = 1
Before continue condition, v_counter = 0
After continue condition, v_counter = 0
Done…
```

ii) Changing the `CONTINUE` condition to the

```
IF v_counter < 3
THEN
   CONTINUE;
END IF;
```

affects all iterations of the loop after the third iteration.

As long as the value of `v_counter` is less than or equal to 3, the `CONTINUE` condition will evaluate to `TRUE`. As a result, for the first two iterations of the loop, all statements inside the body of the loop will be executed. Starting with the third iteration of the loop, the CONTINUE condition will evaluate to TRUE causing partial execution. Note that due to this partial execution, the EXIT condition will never be reached causing this loop to become infinite.

e) How would you modify the script so that the CONTINUE condition v_counter < 3 does not cause infinite loop.

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

For Example *ch07_5b.sql*

```
DECLARE
   v_counter BINARY_INTEGER := 5;
BEGIN
   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
         ('Before continue condition, v_counter = '||v_counter);

      -- if CONTINUE condition yields TRUE pass control to the first
      -- executable statement of the loop
      IF v_counter < 3
      THEN
         CONTINUE;
      END IF;

      DBMS_OUTPUT.PUT_LINE
         ('After continue condition, v_counter = '||v_counter);
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done…');
END;
```

In this version of the script, the EXIT condition is moved to the top of the loop, and the placement of the CONTINUE condition remains as is. Note that as long as the EXIT condition is placed prior to the CONTINUE condition, the loop will terminate. In other words, the EXIT condition can be placed anywhere in the loop as long as it is placed before the CONTINUE condition. This version of the script produces the following output.

```
Before continue condition, v_counter = 4
After continue condition, v_counter = 4
Before continue condition, v_counter = 3
After continue condition, v_counter = 3
Before continue condition, v_counter = 2
Before continue condition, v_counter = 1
Done…
```

Using CONTINUE WHEN Statement

In this exercise, you will use the CONTINUE WHEN condition with the numeric FOR loop to calculate to the sum of even integers between 1 and 10.

Create the following PL/SQL script:

For Example *ch07_6a.sql*

```
DECLARE
   v_sum NUMBER := 0;
BEGIN
   FOR v_counter in 1..10
   LOOP
      -- if v_counter is odd, pass control to the top of the loop
      CONTINUE WHEN mod(v_counter, 2) != 0;

      v_sum := v_sum + v_counter;
      DBMS_OUTPUT.PUT_LINE ('Current sum is: '||v_sum);
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Final sum is:   '||v_sum);
END;
```

Execute the script, and then answer the following questions:

a) What output was generated by the script?

Answer: The output should look similar to the following:

```
Current sum is: 2
Current sum is: 6
Current sum is: 12
Current sum is: 20
Current sum is: 30
Final sum is:   30
```

For each iteration of the loop, the value of the variable v_counter is evaluated by the CONTINUE WHEN statement. When the value of the variable v_counter is even, the CONTINUE WHEN condition yields FALSE, and the current value of sum is calculated and displayed in the Dbms Output window. When the value of the variable v_counter is odd, the CONTINUE WHEN condition yields TRUE, and the control of the execution is passed to the top of the loop causing partial execution of the loop. In this case, the statements following the WHEN CONTINUE condition are not executed at all. After the loop has terminated, the final sum is displayed in the Dbms Output window.

b) How many times was the loop executed?

Answer: The loop was executed 10 times because the number of iterations was controlled by the lower and upper limits of the loop which are 1 and 10 respectively.

c) How many iterations of the loop were partial iterations?

Answer: Five iterations of the loop were partial iterations. This is because the `CONTINUE WHEN` condition evaluates to `TRUE` for the odd values of the variable `v_counter` which are 1, 3, 5, 7, and 9. These values of the `v_counter` correspond to the iterations of the loop. In other words, the first, third, fifth, seventh, and ninth iterations of the loop are partial iterations because for these iterations the `CONTINUE WHEN` condition yields TRUE.

d) How would you change the script to calculate the sum of odd integers between 1 and 10?

Answer: The script should look similar to the following script. Changes are highlighted in bold. Note that only `CONTINUE WHEN` condition is modified, the rest of script remains unchanged.

For Example *ch07_6b.sql*

```
DECLARE
   v_sum NUMBER := 0;
BEGIN
   FOR v_counter in 1..10
   LOOP
      -- if v_counter is even, pass control to the top of the loop
      CONTINUE WHEN mod(v_counter, 2) = 0;

      v_sum := v_sum + v_counter;
      DBMS_OUTPUT.PUT_LINE ('Current sum is: '||v_sum);
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Final sum is:   '||v_sum);
END;
```

This version of the script produces the following the output:

```
Current sum is: 1
Current sum is: 4
Current sum is: 9
Current sum is: 16
Current sum is: 25
Final sum is:   25
```

In this version of the script, the `CONTINUE WHEN` condition yields `FALSE` for the odd values of the variable `v_counter` causing the current value of the variable `v_sum` to be calculated and displayed in the Dbms Output window. For the even values of the variable `v_counter`, the `CONTINUE WHEN` condition evaluates to `TRUE` causing the control of the execution to be passed to the top of the loop.

Lab 7.2 Nested Loops

In this exercise, you will use nested numeric `FOR` loops. Create the following PL/SQL script:

For Example *ch07_7a.sql*

```
DECLARE
   v_test NUMBER := 0;
BEGIN
   <<outer_loop>>
   FOR i IN 1..3
   LOOP
      DBMS_OUTPUT.PUT_LINE('Outer Loop');
      DBMS_OUTPUT.PUT_LINE('i = '||i);
      DBMS_OUTPUT.PUT_LINE('v_test = '||v_test);
      v_test := v_test + 1;

      <<inner_loop>>
      FOR j IN 1..2
      LOOP
         DBMS_OUTPUT.PUT_LINE('Inner Loop');
         DBMS_OUTPUT.PUT_LINE('j = '||j);
         DBMS_OUTPUT.PUT_LINE('i = '||i);
         DBMS_OUTPUT.PUT_LINE('v_test = '||v_test);
      END LOOP inner_loop;
   END LOOP outer_loop;
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:

```
Outer Loop
i = 1
v_test = 0
Inner Loop
j = 1
i = 1
v_test = 1
Inner Loop
j = 2
i = 1
v_test = 1
Outer Loop
i = 2
v_test = 1
Inner Loop
j = 1
i = 2
v_test = 2
Inner Loop
j = 2
i = 2
v_test = 2
Outer Loop
i = 3
v_test = 2
Inner Loop
j = 1
i = 3
v_test = 3
Inner Loop
```

```
j = 2
i = 3
v_test = 3
```

Every time the outer loop is run, the value of the loop counter, `i`, is incremented by 1 implicitly and displayed in the Dbms Output window. In addition, the value of the variable `v_test` is displayed and then incremented by 1, as well. Next, the control of the execution is passed to the inner loop.

Every time the inner loop is run, the value of the inner loop counter, `j`, is incremented by 1 and displayed in the Dbms Output window, along with the value of the outer loop counter, `i`, and the variable `v_test`.

b) How many times was the outer loop executed?

Answer: The outer loop was executed three 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 the upper limit is equal to 3.

c) How many times was the inner loop executed?

Answer: The inner loop was executed six times. For each iteration of the outer loop, the inner loop was executed twice. However, the outer loop was executed three times. Overall, the inner loop was executed six times.

d) What are the values of the loop counters, `i` and `j`, after both loops terminate?

Answer: After both loops terminate, both loop counters are undefined and can hold no values. As mentioned earlier, the loop counter ceases to exist once the numeric `FOR` loop is terminated.

e) Rewrite this script using the `REVERSE` option for both loops. How many times will each loop be executed in this case?

Answer: The script should be similar to the script below. Changes are shown in bold.
The outer loop will execute three times, and the inner loop will execute six times.

For Example *ch07_7b.sql*

```
DECLARE
   v_test NUMBER := 0;
BEGIN
   <<outer_loop>>
   FOR i IN REVERSE 1..3
   LOOP
      DBMS_OUTPUT.PUT_LINE('Outer Loop');
      DBMS_OUTPUT.PUT_LINE('i = '||i);
      DBMS_OUTPUT.PUT_LINE('v_test = '||v_test);
      v_test := v_test + 1;

      <<inner_loop>>
      FOR j IN REVERSE 1..2
      LOOP
         DBMS_OUTPUT.PUT_LINE('Inner Loop');
         DBMS_OUTPUT.PUT_LINE('j = '||j);
         DBMS_OUTPUT.PUT_LINE('i = '||i);
         DBMS_OUTPUT.PUT_LINE('v_test = '||v_test);
      END LOOP inner_loop;
   END LOOP outer_loop;
END;
```

This version of the script produces output as follows:

```
Outer Loop
i = 3
v_test = 0
Inner Loop
j = 2
i = 3
v_test = 1
Inner Loop
j = 1
i = 3
v_test = 1
Outer Loop
i = 2
v_test = 1
Inner Loop
j = 2
i = 2
v_test = 2
Inner Loop
j = 1
i = 2
v_test = 2
Outer Loop
i = 1
v_test = 2
Inner Loop
j = 2
i = 1
v_test = 3
Inner Loop
j = 1
i = 1
v_test = 3
```

Notice that the output produced by this version of the script has changed significantly from the output produced by the previous version. The values of the loop counters are decremented because the `REVERSE` option is used. However, the value of the variable `v_test` was not affected by using the `REVERSE` option.

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_10a.sql to calculate factorial of even integers only between 1 and 10. The script should use `CONTINUE` or `CONTINUE WHEN` statement.

Answer: Recall the script ch06_10a.sql:

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 `CONTINUE WHEN` statement. Newly added statements are shown in bold.

For Example *ch07_8a.sql*

```
DECLARE
   v_factorial NUMBER := 1;
BEGIN
   FOR v_counter IN 1..10
   LOOP
      CONTINUE WHEN MOD(v_counter, 2) != 0;
      v_factorial := v_factorial * v_counter;
   END LOOP;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE
      ('Factorial of even numbers between 1 and 10 is: '||v_factorial);
END;
```

In this version of the script, a `CONTINUE WHEN` statement passes control to the top of the loop if the current value of the loop counter `v_counter` is not an even number. The rest of the script remains unchanged. Note that you could specify the `CONTINUE` condition using an `IF-THEN` statement as well:

```
IF MOD(v_counter, 2) != 0
THEN
   CONTINUE;
END IF;
```

When run, this example shows the following output:

```
Factorial of even numbers between 1 and 10 is: 3840
```

2) Rewrite script ch07_7a.sql using a simple loop instead of the outer `FOR` loop, and a `WHILE` loop for the inner `FOR` loop. Make sure that the output produced by this script does not differ from the output produced by the original script.

Answer: Consider the original version of the script:

For Example *ch07_7a.sql*

```
DECLARE
   v_test NUMBER := 0;
BEGIN
   <<outer_loop>>
   FOR i IN 1..3
   LOOP
      DBMS_OUTPUT.PUT_LINE('Outer Loop');
      DBMS_OUTPUT.PUT_LINE('i = '||i);
      DBMS_OUTPUT.PUT_LINE('v_test = '||v_test);
      v_test := v_test + 1;

      <<inner_loop>>
      FOR j IN 1..2
      LOOP
         DBMS_OUTPUT.PUT_LINE('Inner Loop');
         DBMS_OUTPUT.PUT_LINE('j = '||j);
         DBMS_OUTPUT.PUT_LINE('i = '||i);
         DBMS_OUTPUT.PUT_LINE('v_test = '||v_test);
      END LOOP inner_loop;
   END LOOP outer_loop;
END;
```

Next, consider a modified version of the script that uses simple and `WHILE` loops. All changes are highlighted in bold.

For Example *ch07_9a.sql*

```
DECLARE
   i      BINARY_INTEGER := 1;
   j      BINARY_INTEGER := 1;
   v_test NUMBER := 0;
BEGIN
   <<outer_loop>>
   LOOP
      DBMS_OUTPUT.PUT_LINE ('Outer Loop');
      DBMS_OUTPUT.PUT_LINE ('i = '||i);
      DBMS_OUTPUT.PUT_LINE ('v_test = '||v_test);
      v_test := v_test + 1;

      -- reset inner loop counter
      j := 1;

      <<inner_loop>>
      WHILE j <= 2
      LOOP
         DBMS_OUTPUT.PUT_LINE ('Inner Loop');
         DBMS_OUTPUT.PUT_LINE ('j = '||j);
         DBMS_OUTPUT.PUT_LINE ('i = '||i);
         DBMS_OUTPUT.PUT_LINE ('v_test = '||v_test);
         j := j + 1;
      END LOOP inner_loop;
      i := i + 1;
      -- EXIT condition of the outer loop
      EXIT WHEN i > 3;
   END LOOP outer_loop;
END;
```

Note that this version of the script contains changes that are important due to the nature of the loops that are used:

- First, both counters, for outer and inner loops, must be declared and initialized. Moreover, the counter for the inner loop must be initialized to 1 prior to each execution of the inner loop, and not just in the declaration section of this script. In other words, the inner loop executes three times. It is important not to confuse the term *execution of the loop* with the term *iteration*. *Each execution of the `WHILE` loop causes the statements inside this loop to iterate twice.* Before each execution, the loop counter `j` must reset to 1 again. This step is necessary because the `WHILE` loop does not initialize its counter implicitly like numeric `FOR` loop. As a result, after the first execution of the `WHILE` loop is complete, the value of counter `j` remains 3. If this value is not reset to 1 again, the `WHILE` loop will not execute second time.
- Second, both loop counters must be incremented so that simple and `WHILE` loops do not become infinite.
- Third, the `EXIT` condition must be specified for the outer loop, and the test condition must be specified for the inner loop.

When run, the version of the script produces output as indicated below:

```
Outer Loop
i = 1
v_test = 0
Inner Loop
j = 1
i = 1
v_test = 1
Inner Loop
j = 2
i = 1
v_test = 1
Outer Loop
i = 2
v_test = 1
Inner Loop
j = 1
i = 2
v_test = 2
Inner Loop
j = 2
i = 2
v_test = 2
Outer Loop
i = 3
v_test = 2
Inner Loop
j = 1
i = 3
v_test = 3
```

```
Inner Loop
j = 2
i = 3
v_test = 3
```