PL/SQL has four types of loops: simple loops, WHILE loops, numeric FOR loops, and cursor FOR loops.

6.1. Simple Loops

A simple loop, as you can tell from its name, is the most basic kind of loop. It has the following structure:

```
LOOP
  STATEMENT 1;
  STATEMENT 2;
  ...
  STATEMENT N;
END LOOP;
```

The reserved word LOOP marks the beginning of the simple loop. Statements 1 through N are a sequence of statements that is executed repeatedly. These statements consist of one or more standard programming structures. END LOOP is a reserved phrase that indicates the end of the loop construct.

Every time the loop iterates, a sequence of statements is executed, and then control is passed back to the top of the loop. The sequence of statements is executed an infinite number of times, because no statement specifies when the loop must terminate. Hence, a simple loop is called an infinite loop because there is no means to exit the loop. A properly constructed loop needs an exit condition that determines when the loop is complete. This exit condition has two forms: EXIT and EXIT WHEN.

EXIT STATEMENT

The EXIT statement causes a loop to terminate when the EXIT condition evaluates to TRUE. The EXIT condition is evaluated with the help of an IF statement. When the EXIT condition is evaluated to TRUE, control is passed to the first executable statement after the END LOOP statement. This is indicated by the following:

```
LOOP
     STATEMENT 1;
     STATEMENT 2;
     IF CONDITION THEN
        EXIT;
     END IF;
  END LOOP;
  STATEMENT 3;
```

In this example, you can see that after the EXIT condition evaluates to TRUE, control is passed to *STATEMENT 3*, which is the first executable statement after the END LOOP statement.

Example:

```
SET SERVEROUTPUT ON
     DECLARE
        v_counter BINARY_INTEGER := 0;
     BEGIN
```

```
   LOOP
      -- increment 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 = 5 THEN
         EXIT;
      END IF;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done...');
END;
```

EXIT WHEN STATEMENT

The EXIT WHEN statement causes a loop to terminate only if the EXIT WHEN condition eval- uates to TRUE. Control is then passed to the first executable statement after the END LOOP statement. The structure of a loop using an EXIT WHEN clause is as follows:

```
LOOP
      STATEMENT 1;
      STATEMENT 2;
      EXIT WHEN CONDITION;
   END LOOP;
   STATEMENT 3;
```

Example:

```
SET SERVEROUTPUT ON
  DECLARE
     v_course        course.course_no%type := 430;
     v_instructor_id instructor.instructor_id%type := 102;
     v_sec_num       section.section_no%type := 0;
     BEGIN
        LOOP
        -- increment section number by one
        v_sec_num := v_sec_num + 1;
        INSERT INTO section
           (section_id, course_no, section_no, instructor_id,
            created_date, created_by, modified_date,
            modified_by)
        VALUES
           (section_id_seq.nextval, v_course, v_sec_num,
            v_instructor_id, SYSDATE, USER, SYSDATE, USER);
        -- if number of sections added is four exit the loop
        EXIT WHEN v_sec_num = 4;
        END LOOP;
        -- control resumes here
        COMMIT;
     END;
```

6.2. WHILE Loops

A WHILE loop has the following structure:

```
WHILE CONDITION LOOP
      STATEMENT 1;
      STATEMENT 2;
      ...
      STATEMENT N;
```

```
           END LOOP;
```

The reserved word WHILE marks the beginning of a loop construct. The word CONDITION is the test condition of the loop that evaluates to TRUE or FALSE. The result of this evaluation determines whether the loop is executed. Statements 1 through N are a sequence of statements that is executed repeatedly. The END LOOP is a reserved phrase that indicates the end of the loop construct. If the test condition evaluates to TRUE, the sequence of statements is executed, and control is passed to the top of the loop for the next evaluation of the test condition. If the test condition evaluates to FALSE, the loop is terminated, and control is passed to the next executable statement following the loop.

Example:

```
DECLARE
   v_counter NUMBER := 5;
BEGIN
   WHILE v_counter < 5 LOOP
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      -- decrement the value of v_counter by one
      v_counter := v_counter - 1;
   END LOOP;
END;
```

Boolean expressions can also be used to determine when the loop should terminate.

```
DECLARE
   v_test BOOLEAN := TRUE;
BEGIN
   WHILE v_test LOOP
      STATEMENTS;
      IF TEST_CONDITION THEN
         v_test := FALSE;
      END IF;
   END LOOP;
END;
```

When using a Boolean expression as a test condition of a loop, you must make sure that a different value is eventually assigned to the Boolean variable to exit the loop. Otherwise, the loop becomes infinite.

PREMATURE TERMINATION OF THE LOOP

The EXIT and EXIT WHEN statements can be used inside the body of a WHILE loop. If the EXIT condition evaluates to TRUE before the test condition evaluates to FALSE, the loop is terminated prematurely. If the test condition yields FALSE before the EXIT condition yields TRUE, there is no premature termination of the loop. This is indicated as follows:

```
WHILE TEST_CONDITION LOOP
      STATEMENT 1;
      STATEMENT 2;
      IF EXIT_CONDITION THEN
         EXIT;
      END IF;
  END LOOP;
  STATEMENT 3;
```

```
WHILE TEST_CONDITION LOOP
   STATEMENT 1;
   STATEMENT 2;
   EXIT WHEN EXIT_CONDITION;
END LOOP;
STATEMENT 3;
```

Example

```
DECLARE
   v_counter NUMBER := 1;
BEGIN
   WHILE v_counter <= 5 LOOP
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      IF v_counter = 2 THEN
         EXIT;
      END IF;
      v_counter := v_counter + 1;
   END LOOP;
END;
```

Notice that according to the test condition, the loop should execute five times. However, the loop is executed only twice, because the EXIT condition is present inside the body of the loop. Therefore, the loop terminates prematurely.

Example

```
DECLARE
   v_counter NUMBER := 1;
BEGIN
   WHILE v_counter <= 2 LOOP
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      v_counter := v_counter + 1;
      IF v_counter = 5 THEN
         EXIT;
      END IF;
   END LOOP;
END;
```

These examples demonstrate not only the use of the EXIT statement inside the body of the WHILE loop, but also a bad programming practice. In the first example, the test condition can be changed so that there is no need to use an EXIT condition, because essentially they both are used to terminate the loop. In the second example, the EXIT condition is useless, because its terminal value is never reached. You should never use unnecessary code in your program.

6.3. Numeric FOR Loops

A numeric FOR loop is called numeric because it requires an integer as its terminating value. Its structure is as follows:

```
FOR loop_counter IN [REVERSE] lower_limit..upper_limit LOOP
   STATEMENT 1;
   STATEMENT 2;
   ...
   STATEMENT N;
END LOOP;
```

The reserved word FOR marks the beginning of a FOR loop construct. The variable loop_counter is an implicitly defined index variable. There is no need to define the loop counter in the declaration section of the PL/SQL block. This variable is defined by the loop construct. lower_limit and upper_limit are two integer numbers or expressions that evaluate to integer values at runtime, and the double dot (..) serves as the range operator. lower_limit and upper_limit define the number of iterations for the loop, and their values are evaluated once, for the first iteration of the loop. At this point, it is determined how many times the loop will iterate. Statements 1 through N are a sequence of statements that is executed repeatedly. END LOOP is a reserved phrase that marks the end of the loop construct. The reserved word IN or IN REVERSE must be present when the loop is defined. If the REVERSE keyword is used, the loop counter iterates from the upper limit to the lower limit. However, the syntax for the limit specification does not change. The lower limit is always referenced first.

Example(IN)

```
BEGIN
   FOR v_counter IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
   END LOOP;
END;
```

Example(IN REVERSE)

```
BEGIN
   FOR v_counter IN REVERSE 1..5 LOOP
     DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
   END LOOP;
END;
```

PREMATURE TERMINATION OF THE LOOP

The EXIT and EXIT WHEN statements covered in the previous labs can be used inside the body of a numeric FOR loop as well. If the EXIT condition evaluates to TRUE before the loop counter reaches its terminal value, the FOR loop is terminated prematurely. If the loop counter reaches its terminal value before the EXIT condition yields TRUE, the FOR loop doesn't terminate prematurely. Consider the following:

```
FOR LOOP_COUNTER IN LOWER_LIMIT..UPPER_LIMIT LOOP
      STATEMENT 1;
      STATEMENT 2;
      IF EXIT_CONDITION THEN
         EXIT;
      END IF;
   END LOOP;
   STATEMENT 3;
```

or

```
FOR LOOP_COUNTER IN LOWER_LIMIT..UPPER_LIMIT LOOP
      STATEMENT 1;
      STATEMENT 2;
      EXIT WHEN EXIT_CONDITION;
  END LOOP;
  STATEMENT 3;
```

Example:

```
BEGIN
   FOR v_counter IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE ('v_counter = '||v_counter);
      EXIT WHEN v_counter = 3;
   END LOOP;
END;
```

6.4. The CONTINUE Statement

The CONTINUE condition has two forms: CONTINUE and CONTINUE WHEN.

THE CONTINUE STATEMENT

The CONTINUE statement causes a loop to terminate its current iteration and pass control to the next iteration of the loop when the CONTINUE condition evaluates to TRUE. The CONTINUE condition is evaluated with the help of an IF statement. When the CONTINUE condition evaluates to TRUE, control is passed to the first executable statement in the body of the loop. This is indicated by the following:

```
LOOP
   STATEMENT 1;
   STATEMENT 2;
   IF CONTINUE_CONDITION THEN
      CONTINUE;
   END IF;
   EXIT WHEN EXIT_CONDITION;
END LOOP;
STATEMENT 3;
```

In this example, you can see that after the CONTINUE condition evaluates to TRUE, control is passed to *STATEMENT 1*, which is the first executable statement inside the body of the loop. In this case, it causes partial execution of the loop as the statements following the CONTINUE condition inside the body of the loop are not executed.

Example:

```
SET SERVEROUTPUT ON
DECLARE
   v_counter BINARY_INTEGER := 0;
BEGIN
   LOOP
      -- increment 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 = 5 THEN
         EXIT;
      END IF;
   END LOOP;
   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Done...');
END;
```

THE CONTINUE WHEN STATEMENT

The CONTINUE WHEN statement causes a loop to terminate its current iteration and pass control to the next iteration of the loop only if the CONTINUE WHEN condition evaluates to TRUE. Control is then passed to the first executable statement inside the body of the loop. The structure of a loop using a CONTINUE WHEN clause is as follows:

```
LOOP
   STATEMENT 1;
   STATEMENT 2;
   CONTINUE WHEN CONTINUE_CONDITION;
   EXIT WHEN EXIT_CONDITION;
END LOOP;
STATEMENT 3;
```

Example:

```
SET SERVEROUTPUT ON
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;
```

6.5. Nested Loops

You have explored three types of loops: simple loops, WHILE loops, and numeric FOR loops. Any of these three types of loops can be nested inside one another. For example, a simple loop can be nested inside a WHILE loop, and vice versa. Consider the following example:

Example:

```
DECLARE
   v_counter1 INTEGER := 0;
   v_counter2 INTEGER;
BEGIN
   WHILE v_counter1 < 3 LOOP
       DBMS_OUTPUT.PUT_LINE ('v_counter1: '||v_counter1);
       v_counter2 := 0;
       LOOP
         DBMS_OUTPUT.PUT_LINE ('v_counter2: '||v_counter2);
         v_counter2 := v_counter2 + 1;
         EXIT WHEN v_counter2 >= 2;
       END LOOP;
       v_counter1 := v_counter1 + 1;
   END LOOP;

END;
```

LOOP LABELS

Loops can be labeled in a similar manner as blocks, as follows:

```
<<label_name>>
FOR LOOP_COUNTER IN LOWER_LIMIT..UPPER_LIMIT LOOP
   STATEMENT 1;
   ...
   STATEMENT N;
END LOOP label_name;
```

The label must appear right before the beginning of the loop. This syntax example shows that the label can optionally be used at the end of the loop statement. It is very helpful to label nested loops, because labels improve readability. Consider the following example:

Example:

```
BEGIN
   <<outer_loop>>
   FOR i IN 1..3 LOOP
      DBMS_OUTPUT.PUT_LINE ('i = '||i);
      <<inner_loop>>
      FOR j IN 1..2 LOOP
         DBMS_OUTPUT.PUT_LINE ('j = '||j);
      END LOOP inner_loop;
   END LOOP outer_loop;
END;
```

For both outer and inner loops, the statement END LOOP must be used. If the loop label is added to each END LOOP statement, it is easier to understand which loop is being terminated.

6.6. Built-in Exceptions

As mentioned earlier, a PL/SQL block has the following structure:

```
DECLARE ...

     BEGIN
        EXECUTABLE STATEMENTS;
     EXCEPTION
        WHEN EXCEPTION_NAME THEN
           ERROR-PROCESSING STATEMENTS;
END;
```

When an error occurs that raises a built-in exception, the exception is said to be raised implic- itly. In other words, if a program breaks an Oracle rule, control is passed to the exception- handling section of the block. At this point, the error-processing statements are executed. It is important to realize that after the exception-handling section of the block has executed, the block terminates. Control does not return to the executable section of the block. The following example illustrates this point:

Example:

```
DECLARE
   v_student_name VARCHAR2(50);
BEGIN
```

```
   SELECT first_name||' '||last_name
     INTO v_student_name
     FROM student
    WHERE student_id = 101;
    DBMS_OUTPUT.PUT_LINE ('Student name is '||v_student_name);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('There is no such student');
END;
```

The following list describes some commonly used predefined exceptions and how they are raised:

- NO_DATA_FOUND: This exception is raised when a SELECT INTO statement that makes no calls to group functions, such as SUM or COUNT, does not return any rows.
- TOO_MANY_ROWS: This exception is raised when a SELECT INTO statement returns more than one row. By definition, a SELECT INTO can return only a single row. If a SELECT INTO statement returns more than one row, the definition of the SELECT INTO statement is violated. This causes the TOO_MANY_ROWS exception to be raised.
- ZERO_DIVIDE: This exception is raised when a division operation is performed in the program and a divisor is equal to 0.
- LOGIN_DENIED: This exception is raised when a user tries to log in to Oracle with an invalid username or password.
- PROGRAM_ERROR: This exception is raised when a PL/SQL program has an internal problem.
- VALUE_ERROR: This exception is raised when a conversion or size mismatch error occurs. For example, suppose you select a student's last name into a variable that has been defined as VARCHAR2(5). If the student's last name contains more than five characters, the VALUE_ERROR exception is raised.
- DUP_VALUE_ON_INDEX: This exception is raised when a program tries to store a duplicate value in the column or columns that have a unique index defined on them. For example, suppose you are trying to insert a record into the SECTION table for course number 25, section 1. If a record for the given course and section number already exists in the SECTION table, the DUP_VAL_ON_INDEX exception is raised, because these columns have a unique index defined on them.

Example (different exceptions in the PL/SQL block)

```
DECLARE
   v_student_id NUMBER := &sv_student_id;
   v_enrolled   VARCHAR2(3) := 'NO';
BEGIN
   DBMS_OUTPUT.PUT_LINE ('Check if the student is enrolled');
   SELECT 'YES'
     INTO v_enrolled
     FROM enrollment
    WHERE student_id = v_student_id;
    DBMS_OUTPUT.PUT_LINE ('The student is enrolled into one course');
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('The student is not enrolled');
   WHEN TOO_MANY_ROWS THEN
       DBMS_OUTPUT.PUT_LINE
          ('The student is enrolled in too many courses');
END;
```

Notice that this example contains two exceptions in a single exception-handling section. The first exception, NO_DATA_FOUND, is raised if there are no records in the ENROLLMENT table for a particular student.

The second exception, TOO_MANY_ROWS, is raised if a particular student is enrolled in more than one course.

6.7. User-Defined Exceptions

Often in your programs you may need to handle problems that are specific to the program you write. For example, your program asks a user to enter a value for student ID. This value is then assigned to the variable v_student_id that is used later in the program. Generally, you want a positive number for an ID. By mistake, the user enters a negative number. However, no error occurs, because the variable v_student_id has been defined as a number, and the user has supplied a legitimate numeric value. Therefore, you may want to implement your own excep- tion to handle this situation.

This type of exception is called a user-defined exception because the programmer defines it. As a result, before the exception can be used, it must be declared. A user-defined exception is declared in the declaration section of a PL/SQL block:

```
DECLARE
   exception_name EXCEPTION;
```

Example:

```
DECLARE
   e_invalid_id EXCEPTION;
BEGIN ...
EXCEPTION
   WHEN e_invalid_id THEN
      DBMS_OUTPUT.PUT_LINE ('An id cannot be negative');
END;
```

A user-defined exception must be raised explicitly. In other words, you need to specify in your program under what circumstances an exception must be raised:

```
DECLARE
        exception_name EXCEPTION;
BEGIN ...
        IF CONDITION THEN
           RAISE exception_name;
ELSE ...
        END IF;
     EXCEPTION
        WHEN exception_name THEN
           ERROR-PROCESSING STATEMENTS;
END;
```

Example:

```
DECLARE
   v_student_id    student.student_id%type := &sv_student_id;
   v_total_courses NUMBER;
   e_invalid_id    EXCEPTION;
BEGIN
   IF v_student_id < 0 THEN
      RAISE e_invalid_id;
   ELSE
      SELECT COUNT(*)
        INTO v_total_courses
        FROM enrollment
       WHERE student_id = v_student_id;
      DBMS_OUTPUT.PUT_LINE ('The student is registered for '||
         v_total_courses||' courses');
   END IF;
   DBMS_OUTPUT.PUT_LINE ('No exception has been raised');
EXCEPTION
   WHEN e_invalid_id THEN
      DBMS_OUTPUT.PUT_LINE ('An id cannot be negative');
END;
```

Exercisses:

1. Rewrite the following script using a WHILE loop instead of a simple loop.

```
SET SERVEROUTPUT ON
     DECLARE
        v_counter BINARY_INTEGER := 0;
     BEGIN
        LOOP
           -- increment 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 = 5 THEN
              EXIT;
           END IF;
        END LOOP;
        -- control resumes here
        DBMS_OUTPUT.PUT_LINE ('Done...');
    END;
```

2. Rewrite the following script using a numeric FOR loop instead of a WHILE loop.

```
SET SERVEROUTPUT ON
     DECLARE
        v_counter BINARY_INTEGER := 1;
            v_sum NUMBER := 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;
```

3. Rewrite the following script using a simple loop instead of a numeric FOR loop.

```
SET SERVEROUTPUT ON
     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;
```

4. Rewrite the script to calculate the factorial of even integers only between 1 and 10.The script should use a CONTINUE or CONTINUE WHEN statement.

```
SET SERVEROUTPUT ON
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;
```

5. Rewrite the following script 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.

```
SET SERVEROUTPUT ON
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;
```

6. Create the following script: Check to see whether there is a record in the STUDENT table for a given student ID. If there is not, insert a record into the STUDENT table for the given student ID.
7. Create the following script: For a given instructor ID, check to see whether it is assigned to a valid instructor. Then check to see how many sections this instructor teaches, and display this information on the screen.
8. Create the following script: For a course section provided at runtime, determine the number of students registered. If this number is equal to or greater than 10, raise the user-defined exception e_too_many_students and display an error message. Otherwise, display how many students are in a section. Raise a user-defined exception.Otherwise, display how many students are in a section. Make sure your program can process all sections.