Exercises for Chapter 5: Conditional Control: CASE Statements

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 5.1 CASE Statements

Answer the following questions:

CASE Statements

In this exercise, you will use the CASE statement to display the name of a day based on the number of the day in a week. In other words, if the number of a day of the week is 3, then it is Tuesday.

Create the following PL/SQL script:

For Example *ch05_7a.sql*

```
DECLARE
   v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
   v_day  VARCHAR2(1);
BEGIN
   v_day := TO_CHAR(v_date, 'D');
   CASE v_day
      WHEN '1' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Sunday');
      WHEN '2' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Monday');
      WHEN '3' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Tuesday');
      WHEN '4' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Wednesday');
      WHEN '5' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Thursday');
      WHEN '6' THEN
```

```
      DBMS_OUTPUT.PUT_LINE ('Today is Friday');
   WHEN '7' THEN
      DBMS_OUTPUT.PUT_LINE ('Today is Saturday');
 END CASE;
END;
```

Execute the script, and then answer the following questions:

a) If the value of `v_date` equals '19-SEP-2014', what output is generated by the script?

Answer: The output should look like the following:

```
Today is Friday
```

When the value of 19-SEP-2014 is entered for the variable `v_date`, the number of the day of the week is determined for the variable `v_day` with the help of the `TO_CHAR` function. Next, each expression of the `CASE` statement is compared sequentially to the value of the selector. Because the value of the selector equals '6', the `DBMS_OUTPUT.PUT_LINE` statement associated with the sixth `WHEN` clause is executed. As a result, the message 'Today is Friday' is displayed in the Dbms Output window. The rest of the expressions are not evaluated, and the control of the execution is passed to the first executable statement after `END CASE`.

b) How many times is the `CASE` selector `v_day` evaluated?

Answer: The CASE selector `v_day` is evaluated only once. However, the `WHEN` clauses are checked sequentially. When the value of the expression in the `WHEN` clause equals to the value of the selector, the statements associated with that `WHEN` clause are executed.

c) Rewrite this script using the `ELSE` clause in the `CASE` statement.

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

For Example *ch05_7b.sql*

```
DECLARE
   v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
   v_day  VARCHAR2(1);
BEGIN
   v_day := TO_CHAR(v_date, 'D');
   CASE v_day
       WHEN '1' THEN
          DBMS_OUTPUT.PUT_LINE ('Today is Sunday');
       WHEN '2' THEN
          DBMS_OUTPUT.PUT_LINE ('Today is Monday');
       WHEN '3' THEN
          DBMS_OUTPUT.PUT_LINE ('Today is Tuesday');
       WHEN '4' THEN
          DBMS_OUTPUT.PUT_LINE ('Today is Wednesday');
       WHEN '5' THEN
          DBMS_OUTPUT.PUT_LINE ('Today is Thursday');
       WHEN '6' THEN
          DBMS_OUTPUT.PUT_LINE ('Today is Friday');
       ELSE
          DBMS_OUTPUT.PUT_LINE ('Today is Saturday');
   END CASE;
```

```
END;
```

Notice that the last `WHEN` clause has been replaced by the `ELSE` clause. If '20-SEP-2014' is provided at runtime, the example produces the following output:

```
Today is Saturday
```

None of the expressions listed in the `WHEN` clauses are equal to the value of the selector because the date of '20-SEP-2014' falls on Saturday, which is the seventh day of the week. As a result, the `ELSE` clause is executed, and the message 'Today is Saturday' is displayed in the Dbms Output window.

Searched CASE Statements

In this exercise, you will modify the script **ch05_7a.sql** created in the previous section. In this exercise you will using searched `CASE` statement instead of `CASE` statement.

Create the following PL/SQL script:

For Example *ch05_8a.sql*

```
DECLARE
   v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
BEGIN
   CASE
      WHEN TO_CHAR(v_date, 'D') = '1' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Sunday');
      WHEN TO_CHAR(v_date, 'D') = '2' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Monday');
      WHEN TO_CHAR(v_date, 'D') = '3' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Tuesday');
      WHEN TO_CHAR(v_date, 'D') = '4' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Wednesday');
      WHEN TO_CHAR(v_date, 'D') = '5' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Thursday');
      WHEN TO_CHAR(v_date, 'D') = '6' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Friday');
      WHEN TO_CHAR(v_date, 'D') = '7' THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Saturday');
   END CASE;
END;
```

Try to answer the following questions first, and then execute the script:

a) Explain the differences between the ch05_7a.sql and ch05_8a.sql scripts:

Answer: Because the ch05_8a.sql script employs searched `CASE` statement, there is no need to declare variable `v_day`. This variable was used in the ch05_7a.sql script because `CASE` statement requires a selector. However, for the searched `CASE` statement the `v_date` variable is evaluated by each `WHEN` clause. Otherwise, this version of the script produces the same output as the original script for the same value of the variable `v_date` as demonstrated below:

```
Today is Friday
```

b) What output will be generated by the script if NULL is provided for the variable v_date at the run time?

Answer: If NULL is provided for the variable v_date at the run time, the script is unable to execute successfully. This is because none of the searched conditions listed account for the value of the variable v_date being NULL as illustrated by the error message below:

```
ORA-06592: CASE not found while executing CASE statement
ORA-06512: at line 4
```

c) Rewrite this script so that it executes successfully when NULL is provided for the variable v_date at the run time.

Answer: Your script should look similar to one of the following scripts. Newly added statements are highlighted bold.

For Example *ch05_8b.sql – Adding WHEN Clause*

```
DECLARE
   v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
BEGIN
   CASE
      WHEN TO_CHAR(v_date, 'D') IS NULL
      THEN
         DBMS_OUTPUT.PUT_LINE ('v_date is NULL');
      WHEN TO_CHAR(v_date, 'D') = '1'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Sunday');
      WHEN TO_CHAR(v_date, 'D') = '2'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Monday');
      WHEN TO_CHAR(v_date, 'D') = '3'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Tuesday');
      WHEN TO_CHAR(v_date, 'D') = '4'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Wednesday');
      WHEN TO_CHAR(v_date, 'D') = '5'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Thursday');
      WHEN TO_CHAR(v_date, 'D') = '6'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Friday');
      WHEN TO_CHAR(v_date, 'D') = '7'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Saturday');
   END CASE;
END;
```

Note that in this version, a new WHEN clause has been added to the searched CASE statement that tests whether v_date variable is NULL. This version of the script produces output as shown below:

```
v_date is NULL
```

Next, consider another version of the script where the searched CASE statement is extended with the ELSE clause. Because the ELSE clause does not check for any specific condition, the message in the newly added DBMS_OUTPUT.PUT_LINE statement has modified to a more generic one.

For Example *ch05_8c.sql – Adding ELSE Clause*

```
DECLARE
   v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
BEGIN
   CASE
      WHEN TO_CHAR(v_date, 'D') = '1'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Sunday');
      WHEN TO_CHAR(v_date, 'D') = '2'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Monday');
      WHEN TO_CHAR(v_date, 'D') = '3'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Tuesday');
      WHEN TO_CHAR(v_date, 'D') = '4'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Wednesday');
      WHEN TO_CHAR(v_date, 'D') = '5'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Thursday');
      WHEN TO_CHAR(v_date, 'D') = '6'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Friday');
      WHEN TO_CHAR(v_date, 'D') = '7'
      THEN
         DBMS_OUTPUT.PUT_LINE ('Today is Saturday');
      ELSE
         DBMS_OUTPUT.PUT_LINE ('Today is Unknown');
   END CASE;
END;
```

This version of the script produces the following output:

```
Today is Unknown
```

Lab 5.2 CASE Expressions

In this exercise, you will modify the script ch04_8c.sql created in the Lab 4.2 of the previous chapter. The original script used ELSIF statement to assign letter grade to the variable v_letter_grade. In this new version of the script, the ELSIF statement is replaced by the CASE expression to display a letter grade for a student registered for a specific section of course number 25.

The original version of the script is provided below for your reference:

For Example *ch04_8c.sql*

```
DECLARE
   v_student_id   NUMBER := 102;
```

```
   v_section_id   NUMBER := 89;
   v_final_grade  NUMBER;
   v_letter_grade CHAR(1);
BEGIN
   SELECT final_grade
     INTO v_final_grade
     FROM enrollment
    WHERE student_id = v_student_id
      AND section_id = v_section_id;

   IF v_final_grade >= 90
   THEN
      v_letter_grade := 'A';
   ELSIF v_final_grade >= 80
   THEN
      v_letter_grade := 'B';
   ELSIF v_final_grade >= 70
   THEN
      v_letter_grade := 'C';
   ELSIF v_final_grade >= 60
   THEN
      v_letter_grade := 'D';
   ELSE
      v_letter_grade := 'F';
   END IF;

   -- control resumes here
   DBMS_OUTPUT.PUT_LINE ('Letter grade is: '||v_letter_grade);

EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      DBMS_OUTPUT.PUT_LINE ('There is no such student or section');
END;
```

Answer the following questions:

a) Modify the script ch04_8c.sql shown above. Replace the `ELSIF` statement with the `CASE` expression so that the value returned by this `CASE` expression is assigned to the variable `v_letter_grade`.

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

For Example *ch05_9a.sql*

```
DECLARE
   v_student_id   NUMBER := 102;
   v_section_id   NUMBER := 89;
   v_final_grade  NUMBER;
   v_letter_grade CHAR(1);
BEGIN
   SELECT final_grade
     INTO v_final_grade
     FROM enrollment
```

```
    WHERE student_id = v_student_id
      AND section_id = v_section_id;

   v_letter_grade := CASE
                        WHEN v_final_grade >= 90 THEN 'A'
                        WHEN v_final_grade >= 80 THEN 'B'
                        WHEN v_final_grade >= 70 THEN 'C'
                        WHEN v_final_grade >= 60 THEN 'D'
                        ELSE 'F'
                     END;

   DBMS_OUTPUT.PUT_LINE ('Letter grade is: '||v_letter_grade);

EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      DBMS_OUTPUT.PUT_LINE ('There is no such student or section');
END;
```

b) Run the script created in part (a) and explain the output produced.

Answer: The output should look similar to the following:

```
Letter grade is: A
```

The `SELECT INTO` statement returns a value of 92 that is assigned to the variable `v_final_grade`. As a result, the first searched condition of the `CASE` expression evaluates to `TRUE` and returns a value of 'A'. This value is then assigned to the variable `v_letter_grade` and displayed in the Dbms Output window via the `DBMS_OUTPUT.PUT_LINE` statement.

c) Rewrite the script created in part (a), ch05_9a.sql so that the result of the `CASE` expression is assigned to the variable `v_letter_grade` via the `SELECT INTO` statement.

Answer: The script should look similar to the following. Newly modified `SELECT INTO` statement is shown on bold.

For Example *ch05_9b.sql*

```
DECLARE
   v_student_id   NUMBER := 102;
   v_section_id   NUMBER := 89;
   v_letter_grade CHAR(1);
BEGIN
   SELECT CASE
             WHEN final_grade >= 90 THEN 'A'
             WHEN final_grade >= 80 THEN 'B'
             WHEN final_grade >= 70 THEN 'C'
             WHEN final_grade >= 60 THEN 'D'
             ELSE 'F'
          END
     INTO v_letter_grade
     FROM enrollment
    WHERE student_id = v_student_id
```

```
      AND section_id = v_section_id;
   DBMS_OUTPUT.PUT_LINE ('Letter grade is: '||v_letter_grade);

EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      DBMS_OUTPUT.PUT_LINE ('There is no such student or section');
END;
```

In the previous version of the script, the variable `v_final_grade` was used to hold the value of the numeric grade as follows:

```
SELECT final_grade
  INTO v_final_grade
  FROM enrollment
 WHERE student_id = v_student_id
   AND section_id = v_section_id;
```

This value of the numeric grade was then used by the `CASE` expression to assign proper letter grade to the variable `v_letter_grade` as follows:

```
CASE
   WHEN v_final_grade >= 90 THEN 'A'
   WHEN v_final_grade >= 80 THEN 'B'
   WHEN v_final_grade >= 70 THEN 'C'
   WHEN v_final_grade >= 60 THEN 'D'
   ELSE 'F'
END;
```

In the current version of the script, the `CASE` expression is used as part of the `SELECT INTO` statement. As a result, the column `FINAL_GRADE` is used by the `CASE` expression as indicated below

```
CASE
   WHEN final_grade >= 90 THEN 'A'
   WHEN final_grade >= 80 THEN 'B'
   WHEN final_grade >= 70 THEN 'C'
   WHEN final_grade >= 60 THEN 'D'
   ELSE 'F'
END
```

This eliminates the need for the variable `v_final_grade` as the letter grade is assigned to the variable `v_letter_grade` via the `SELECT INTO` statement.

Lab 5.3 `NULLIF` and `COALESCE` Functions

Answer the following questions:

NULLIF Function

In this exercise, you will modify the following script. Instead of using the searched `CASE` expression, you will use the `NULLIF` function.

Create the following PL/SQL script:

For Example *ch05_10a.sql*

```
DECLARE
   v_final_grade NUMBER;
BEGIN
   SELECT CASE
             WHEN e.final_grade = g.numeric_grade THEN NULL
             ELSE g.numeric_grade
          END
     INTO v_final_grade
     FROM enrollment e
     JOIN grade g
       ON (e.student_id = g.student_id
      AND  e.section_id = g.section_id)
    WHERE e.student_id = 102
      AND e.section_id = 86
      AND g.grade_type_code = 'FI';

   DBMS_OUTPUT.PUT_LINE ('Final grade: '||v_final_grade);
END;
```

In the preceding script, the value of the final grade is compared to the value of the numeric grade. If these values are equal, the `CASE` expression returns `NULL`. In the opposite case, the `CASE` expression returns the numeric grade. The result of the `CASE` expression is assigned to the variable `v_final_grade` which is then displayed in the Dbms Output window via the `DBMS_OUTPUT.PUT_LINE` statement.

Answer the following questions:

a) Modify script ch05_10a.sql. Substitute the `CASE` expression with the `NULLIF` function.

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

For Example *ch05_10b.sql*

```
DECLARE
   v_final_grade NUMBER;
BEGIN
   SELECT NULLIF(g.numeric_grade, e.final_grade)
     INTO v_final_grade
     FROM enrollment e
     JOIN grade g
       ON (e.student_id = g.student_id
      AND  e.section_id = g.section_id)
    WHERE e.student_id = 102
      AND e.section_id = 86
      AND g.grade_type_code = 'FI';

   DBMS_OUTPUT.PUT_LINE ('Final grade: '||v_final_grade);
END;
```

In the original version of the script, the `CASE` expression is used in order to assign a value to the variable `v_final_grade` as follows:

```
CASE
```

```
    WHEN e.final_grade = g.numeric_grade THEN NULL
    ELSE g.numeric_grade
END
```

In this case, the value stored in the column FINAL_GRADE is compared to the value stored in the column NUMERIC_GRADE. If these values are equal, then NULL is assigned to the variable v_final_grade; otherwise, the value stored in the column NUMERIC_GRADE is assigned to the variable v_final_grade.

In the new version of the script, the CASE expression is replaced by the NULLIF function as follows:

```
NULLIF(g.numeric_grade, e.final_grade)
```

By the Way

Did you notice that the NUMERIC_GRADE column is referenced first in the NULLIF function? This is because the NULLIF function compares expression1 to expression2. If expression1 equals expression2, the NULLIF functions returns NULL. If expression1 does not equal expression2, the NULLIF function returns expression1. In order to return the value stored in the column NUMERIC_GRADE, it must be referenced first in the NULLIF function.

b) Run the modified version of the script and explain the output produced.

Answer: The output should look similar to the following:

```
Final grade: 85
```

The NULLIF function compares values stored in the columns NUMERIC_GRADE and FINAL_GRADE. Because the column FINAL_GRADE is not populated, the NULLIF function returns the value stored in the column NUMERIC_GRADE. This value is assigned to the variable v_final_grade and displayed in the Dbms Output window with the help of the DBMS_OUTPUT.PUT_LINE statement.

c) Change the order of columns in the NULLIF function. Run the modified version of the script and explain the output produced.

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

For Example *ch05_10c.sql*

```
DECLARE
   v_final_grade NUMBER;
BEGIN
   SELECT NULLIF(e.final_grade, g.numeric_grade)
     INTO v_final_grade
     FROM enrollment e
     JOIN grade g
       ON (e.student_id = g.student_id
      AND  e.section_id = g.section_id)
    WHERE e.student_id = 102
      AND e.section_id = 86
      AND g.grade_type_code = 'FI';

   DBMS_OUTPUT.PUT_LINE ('Final grade: '||v_final_grade);
END;
```

The version of the script produces output as indicated below:

```
Final grade:
```

In this version of the script, the columns NUMERIC_GRADE and FINAL_GRADE are listed in the opposite order as follows:

```
NULLIF(e.final_grade, g.numeric_grade)
```

The value stored in the column FINAL_GRADE is compared to the value stored in the column NUMERIC_GRADE. Because these values are not equal, the NULLIF function returns the value of the column FINAL_GRADE. Since this column is not populated, NULL is assigned to the variable v_final_grade.

COALESCE Function

In this exercise, you will modify the following script. Instead of using the nested CASE expressions, you will use the COALESCE function.

For Example *ch05_11a.sql*

```
DECLARE
   v_num1   NUMBER := &sv_num1;
   v_num2   NUMBER := &sv_num2;
   v_num3   NUMBER := &sv_num3;
   v_result NUMBER;
BEGIN
   v_result := CASE
                  WHEN v_num1 IS NOT NULL
                  THEN
                     v_num1
                  ELSE
                     CASE
                        WHEN v_num2 IS NOT NULL
                        THEN
                           v_num2
                        ELSE
                           v_num3
                     END
                  END;
   DBMS_OUTPUT.PUT_LINE ('Result: '||v_result);
END;
```

In the preceding script, the list consisting of three numbers is evaluated by the nested CASE expressions as follows:

- If the value of the first number is not NULL, then the outer CASE expression returns the value of the first number.
- Otherwise, the control of the execution is passed to the inner CASE expression, which evaluates the second number.
  - If the value of the second number is not NULL, then the inner CASE expression returns the value of the second number; in the opposite case, it returns the value of the third number.

The preceding CASE expression is equivalent to the following two CASE expressions:

```
CASE
   WHEN v_num1 IS NOT NULL
   THEN
      v_num1
   WHEN v_num2 IS NOT NULL
   THEN
      v_num2
   ELSE
      v_num3
END
```

and

```
CASE
   WHEN v_num1 IS NOT NULL
   THEN
      v_num1
   ELSE
      COALESCE(v_num2, v_num3)
END
```

Answer the following questions:

a) Modify script ch05_11a.sql. Substitute the nested `CASE` expressions with the `COALESCE` function.

Answer: The script should look similar to the following script. Newly modified statement is shown in bold.

For Example *ch05_11b.sql*

```
DECLARE
   v_num1   NUMBER := &sv_num1;
   v_num2   NUMBER := &sv_num2;
   v_num3   NUMBER := &sv_num3;
   v_result NUMBER;
BEGIN
   v_result := COALESCE(v_num1, v_num2, v_num3);
   DBMS_OUTPUT.PUT_LINE ('Result: '||v_result);
END;
```

The original version of the script uses nested `CASE` expressions in order to assign a value to the variable `v_result` as follows:

```
CASE
   WHEN v_num1 IS NOT NULL
   THEN
      v_num1
   ELSE
      CASE
         WHEN v_num2 IS NOT NULL
         THEN
            v_num2
         ELSE
            v_num3
      END
```

```
END;
```

In the new version of the script, the nested `CASE` expression is replaced with the `COALESCE` function as shown below:

```
COALESCE(v_num1, v_num2, v_num3)
```

Based on the values stored in the variables `v_num1`, `v_num2`, and `v_num3`, the `COALESCE` function returns value of the first non-null variable.

b) Run the modified version of the script and explain the output produced. Use the following values for the list of numbers: `NULL`, 1, 2.

Answer: The output should look similar to the following:

```
Result: 1
```

The `COALESCE` function evaluates its expressions in the sequential order. The variable `v_num1` is evaluated first. Because the variable `v_num1` is `NULL`, the `COALESCE` function evaluates the variable `v_num2` next. Because the variable `v_num2` is not `NULL`, the `COALSECE` function returns the value of the variable `v_num2`. This value is assigned to the variable `v_result` and is displayed in the Dbms Output window via `DBMS_OUTPUT.PUT_LINE` statement.

c) What output will be produced by the modified version of the script if `NULL` is provided for all three numbers? Try to explain your answer before you run the script.

Answer: The variables `v_num1`, `v_num2`, and `v_num3` are evaluated by the `COALESCE` function in the sequential order. When `NULL` is assigned to all three variables, none of the evaluations produce a non-null result. So the `COALESCE` function returns `NULL` when all of its expressions evaluate to `NULL`. This is illustrated further by the output below:

```
Result:
```

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) Create the following script. Modify the script created in Chapter 4, Question 1 of the Try It Yourself section, ch04_10a.sql. You can use either `CASE` statement or searched `CASE` statement. The output should look similar to the output produced by the example created in Chapter 4.

Answer: Consider the original script created in Chapter 4:

For Example *ch04_10a.sql*

```
DECLARE
   v_instructor_id NUMBER := &sv_instructor_id;
   v_total NUMBER;
BEGIN
   SELECT COUNT(*)
     INTO v_total
     FROM section
```

```
    WHERE instructor_id = v_instructor_id;

   -- check if instructor teaches 3 or more sections
   IF v_total >= 3
   THEN
      DBMS_OUTPUT.PUT_LINE ('This instructor needs a vacation');
   ELSE
      DBMS_OUTPUT.PUT_LINE ('This instructor teaches '||v_total||' sections');
   END IF;
   -- control resumes here
END;
```

Next, consider modified version script:

For Example *ch05_12a.sql*

```
DECLARE
   v_instructor_id NUMBER := &sv_instructor_id;
   v_total NUMBER;
BEGIN
   SELECT COUNT(*)
     INTO v_total
     FROM section
    WHERE instructor_id = v_instructor_id;

   -- check if instructor teaches 3 or more sections
   CASE
      WHEN v_total >= 3
      THEN
         DBMS_OUTPUT.PUT_LINE ('This instructor needs a vacation');
      ELSE
         DBMS_OUTPUT.PUT_LINE ('This instructor teaches '||v_total||' sections');
    END CASE;
   -- control resumes here
END;
```

Note that this version of the script employs searched `CASE` statement and has minimal changes. Next, consider yet another version of the script that uses `CASE` statement. The changes in this version are more significant.

For Example *ch05_12b.sql*

```
DECLARE
   v_instructor_id NUMBER := &sv_instructor_id;
   v_total NUMBER;
BEGIN
   SELECT COUNT(*)
     INTO v_total
     FROM section
    WHERE instructor_id = v_instructor_id;

   -- check if instructor teaches 3 or more sections
   CASE SIGN(v_total - 3)
```

```
      WHEN -1
      THEN
         DBMS_OUTPUT.PUT_LINE ('This instructor teaches '||v_total||' sections');
      ELSE
         DBMS_OUTPUT.PUT_LINE ('This instructor needs a vacation');
    END CASE;
   -- control resumes here
END;
```

This version of the script employs the SIGN function as the CASE statement requires a selector value that determines which WHEN clause should be executed. The result of the SIGN function is such a selector as it returns –1 if v_total is less than 3, 0 if v_total equals to 3, and 1 if v_total is greater than 3. In this case, as long as the SIGN function returns –1, the message 'This instructor teaches…' is displayed in the Dbms Output window. In all other cases, the message 'This instructor needs a vacation' is displayed in the Dbms Output window.

2) Execute the following two SELECT statements and explain why they produce different output:

```
SELECT e.student_id, e.section_id, g.numeric_grade, e.final_grade,
       COALESCE(g.numeric_grade, e.final_grade) grade
  FROM enrollment e, grade g
 WHERE e.student_id = g.student_id
   AND e.section_id = g.section_id
   AND e.student_id = 102
   AND g.grade_type_code = 'FI';

SELECT e.student_id, e.section_id, g.numeric_grade, e.final_grade,
       NULLIF(g.numeric_grade, e.final_grade) grade
  FROM enrollment e, grade g
 WHERE e.student_id = g.student_id
   AND e.section_id = g.section_id
   AND e.student_id = 102
   AND g.grade_type_code = 'FI';
```

Answer: Consider outputs produced by the following SELECT statements:

```
STUDENT_ID SECTION_ID NUMERIC_GRADE FINAL_GRADE      GRADE
---------- ---------- ------------- -----------  ---------
       102         86            85                     85
       102         89            92           92        92


STUDENT_ID SECTION_ID NUMERIC_GRADE FINAL_GRADE      GRADE
---------- ---------- ------------- ----------- ----------
       102         86            85                     85
       102         89            92          92
```

Take a closer look at the output returned by the first SELECT statement. This statement uses the COALESCE function to derive the value of the GRADE column. Recall that the COALESCE function compares each expression to NULL from the list of expressions and returns the value of the first non-null expression. This list of expressions consists of two columns, NUMERIC_GRADE and FINAL_GRADE, and in both rows, the NUMERIC_GRADE column contains non-null values, so the value of GRADE column always equals to the value of the NUMERIC_GRADE column.

Next, take a closer look at the second `SELECT` statement. This statement uses the `NULLIF` function to derive the value of the `GRADE` column. Recall that the `NULLIF` function compares two expressions. If they are equal, then the function returns `NULL`; otherwise, it returns the value of the first expression. In this case, the `NULLIF` function is comparing `NUMERIC_GRADE` and `FINAL_GRADE` columns. Since in the first row, the `FINAL_GRADE` column is `NULL`, the `NULLIF` function returns the value of the `NUMERIC_GRADE` column. In the second row, both `NUMERIC_GRADE` and `FINAL_GRADE` columns contain the same value, thus, the `NULLIF` function return value of `NULL` for the derived `GRADE` column.