Lab 5. Conditional Control

Databases II

Conditional control allows you to control the program's flow of the execution based on a condition. In programming terms, this means that the statements in the program are not executed sequentially. Rather, one group of statements or another is executed, depending on how the condition is evaluated. PL/SQL has three types of conditional control: IF, ELSIF, and CASE statements.

5.1. Conditional Control: IF Statements

An IF statement has two forms: IF-THEN and IF-THEN-ELSE. An IF-THEN statement allows you to specify only one group of actions to take. In other words, this group of actions is taken only when a condition evaluates to TRUE. An IF-THEN-ELSE statement allows you to specify two groups of actions. The second group of actions is taken when a condition evaluates to FALSE or NULL.

IF-THEN STATEMENTS

An IF-THEN statement is the most basic kind of a conditional control; it has the following structure:

```
IF CONDITION THEN
   STATEMENT 1;
   ...
   STATEMENT N;
END IF;
```

The reserved word IF marks the beginning of the IF statement. Statements 1 through N are a sequence of executable statements that consist of one or more standard programming structures. The word *CONDITION* between the keywords IF and THEN determines whether these statements are executed. END IF is a reserved phrase that indicates the end of the IF-THEN construct. When an IF-THEN statement is executed, a condition is evaluated to either TRUE or FALSE. If the condition evaluates to TRUE, control is passed to the first executable statement of the IF-THEN construct. If the condition evaluates to FALSE, control is passed to the first executable statement after the END IF statement.

Example:

Two numeric values are stored in the variables `v_num1` and `v_num2`. You need to arrange their values so that the smaller value is always stored in `v_num1` and the larger value is always stored in `v_num2`.

```
DECLARE
   v_num1 NUMBER := 5;
   v_num2 NUMBER := 3;
   v_temp NUMBER;
BEGIN
   -- if v_num1 is greater than v_num2 rearrange their values
   IF v_num1 > v_num2 THEN
      v_temp := v_num1;
      v_num1 := v_num2;
      v_num2 := v_temp;
   END IF;
   -- display the values of v_num1 and v_num2
   DBMS_OUTPUT.PUT_LINE ('v_num1 = '||v_num1);
   DBMS_OUTPUT.PUT_LINE ('v_num2 = '||v_num2);
END;
```

IF-THEN-ELSE STATEMENT

An IF-THEN statement specifies the sequence of statements to execute only if the condition evaluates to TRUE. When this condition evaluates to FALSE, there is no special action to take, except to proceed with execution of the program.

An IF-THEN-ELSE statement enables you to specify two groups of statements. One group of statements is executed when the condition evaluates to TRUE. Another group of statements is executed when the condition evaluates to FALSE. This is indicated as follows:

```
 IF CONDITION THEN
    STATEMENT 1;
ELSE
    STATEMENT 2;
END IF;
STATEMENT 3;
```

When *CONDITION* evaluates to TRUE, control is passed to *STATEMENT 1*; when *CONDITION* evaluates to FALSE, control is passed to *STATEMENT 2*. After the IF-THEN-ELSE construct has completed, *STATEMENT 3* is executed.

Example:

```
DECLARE
        v_num NUMBER := &sv_user_num;
```

```
BEGIN
   -- test if the number provided by the user is even
   IF MOD(v_num,2) = 0  THEN
      DBMS_OUTPUT.PUT_LINE (v_num||' is even number');
   ELSE
      DBMS_OUTPUT.PUT_LINE (v_num||' is odd number');
   END IF;
   DBMS_OUTPUT.PUT_LINE ('Done');
END;
```

NULL CONDITION

In some cases, a condition used in an IF statement can be evaluated to NULL instead of TRUE or FALSE. For the IF-THEN construct, the statements are not executed if an associated condition evaluates to NULL. Next, control is passed to the first executable statement after END IF. For the IF-THEN-ELSE construct, the statements specified after the keyword ELSE are executed if an associated condition evaluates to NULL.

Example:

```
DECLARE
   v_num1 NUMBER := 0;
   v_num2 NUMBER;
BEGIN
   IF v_num1 = v_num2 THEN
      DBMS_OUTPUT.PUT_LINE ('v_num1 = v_num2');
   ELSE
      DBMS_OUTPUT.PUT_LINE ('v_num1 != v_num2');
  END IF;
END;
```

ELSIF STATEMENTS

An ELSIF statement has the following structure:

```
IF CONDITION 1 THEN
   STATEMENT 1;
ELSIF CONDITION 2 THEN
   STATEMENT 2;
ELSIF CONDITION 3 THEN
   STATEMENT 3;
...
ELSE
   STATEMENT N;
END IF;
```

The reserved word IF marks the beginning of an ELSIF construct. *CONDITION 1* through *CONDITION N* are a sequence of the conditions that evaluate to TRUE or FALSE. These conditions are mutually exclusive. In other words, if *CONDITION 1* evaluates to TRUE, *STATEMENT 1* is executed, and control is passed to the first executable statement after the reserved phrase END IF. The rest of the ELSIF construct is ignored. When *CONDITION 1* evaluates to FALSE, control is passed to the ELSIF part and *CONDITION 2* is evaluated, and so forth. If none of the specified conditions yields TRUE, control is passed to the ELSE part of the ELSIF construct. An ELSIF statement can contain any number of ELSIF clauses.

Example:

```
DECLARE
   v_num NUMBER := &sv_num;
BEGIN
   IF v_num < 0 THEN
      DBMS_OUTPUT.PUT_LINE (v_num||' is a negative number');
   ELSIF v_num = 0 THEN
      DBMS_OUTPUT.PUT_LINE (v_num||' is equal to zero');
   ELSE
      DBMS_OUTPUT.PUT_LINE (v_num||' is a positive number');
   END IF;

END;
```

When using an ELSIF construct, it is not necessary to specify what action should be taken if none of the conditions evaluates to TRUE. In other words, an ELSE clause is not required in the ELSIF construct. Consider the following example:

Example:

```
DECLARE
   v_num NUMBER := &sv_num;
BEGIN
   IF v_num < 0 THEN
      DBMS_OUTPUT.PUT_LINE (v_num||' is a negative number');
   ELSIF v_num > 0 THEN
      DBMS_OUTPUT.PUT_LINE (v_num||' is a positive number');
   END IF;
   DBMS_OUTPUT.PUT_LINE ('Done...');
END;
```

NESTED IF STATEMENTS

You have encountered different types of conditional controls: the IF-THEN statement, the IF- THEN-ELSE statement, and the ELSIF statement. These types of conditional controls can be nested inside one another. For example, an IF statement can be nested inside an ELSIF, and vice versa.

Example

```
DECLARE
   v_num1  NUMBER := &sv_num1;
   v_num2  NUMBER := &sv_num2;
   v_total NUMBER;
BEGIN
   IF v_num1 > v_num2 THEN
      DBMS_OUTPUT.PUT_LINE ('IF part of the outer IF');
      v_total := v_num1 - v_num2;
   ELSE
      DBMS_OUTPUT.PUT_LINE ('ELSE part of the outer IF');
      v_total := v_num1 + v_num2;
      IF v_total < 0 THEN
         DBMS_OUTPUT.PUT_LINE ('Inner IF');
         v_total := v_total * (-1);
      END IF;
   END IF;
   DBMS_OUTPUT.PUT_LINE ('v_total = '||v_total);
END;
```

The IF-THEN-ELSE statement is called an outer IF statement because it encompasses the IF-THEN statement (shown in bold). The IF-THEN statement is called an inner IF statement because it is enclosed by the body of the IF-THEN-ELSE statement.

5.2 CASE Statements

A CASE statement has two forms: CASE and searched CASE. A CASE statement allows you to specify a selector that determines which group of actions to take. A searched CASE statement does not have a selector; it has search conditions that are evaluated in order to determine which group of actions to take.

CASE STATEMENTS

A CASE statement has the following structure:

```
CASE SELECTOR
   WHEN EXPRESSION 1 THEN STATEMENT 1;
   WHEN EXPRESSION 2 THEN STATEMENT 2;
   ...
   WHEN EXPRESSION N THEN STATEMENT N;
   ELSE STATEMENT N+1;
```

```
END CASE;
```

The reserved word CASE marks the beginning of the CASE statement. A selector is a value that determines which WHEN clause should be executed. Each WHEN clause contains an *EXPRESSION* and one or more executable statements associated with it. The ELSE clause is optional. It works much like the ELSE clause used in the IF-THEN-ELSE statement. END CASE is a reserved phrase that indicates the end of the CASE statement.
The selector is evaluated only once, and the WHEN clauses are evaluated sequentially. The value of an expression is compared to the value of the selector. If they are equal, the statement associated with a particular WHEN clause is executed, and subsequent WHEN clauses are not evaluated. If no expression matches the value of the selector, the ELSE clause is executed.

Example:

```
DECLARE
   v_num NUMBER := &sv_user_num;
BEGIN
   -- test if the number provided by
      the user is even
   IF MOD(v_num,2) = 0  THEN
      DBMS_OUTPUT.PUT_LINE (v_num||'
          is even number');
   ELSE
      DBMS_OUTPUT.PUT_LINE (v_num||'
          is odd number');
   END IF;
   DBMS_OUTPUT.PUT_LINE ('Done');
END;
```

```
DECLARE
   v_num NUMBER := &sv_user_num;
   v_num_flag NUMBER;
BEGIN
   v_num_flag := MOD(v_num,2);
   -- test if the number provided by
      the user is even
   CASE v_num_flag
      WHEN 0 THEN
        DBMS_OUTPUT.PUT_LINE
         (v_num||’ is even number’);
      ELSE
        DBMS_OUTPUT.PUT_LINE
         (v_num||’ is odd number’);
   END CASE;
   DBMS_OUTPUT.PUT_LINE ('Done’);
END;
```

SEARCHED CASE STATEMENTS

A searched CASE statement has search conditions that yield Boolean values: TRUE, FALSE, or NULL. When a particular search condition evaluates to TRUE, the group of statements associated with this condition is executed. This is indicated as follows:

```
CASE
   WHEN SEARCH CONDITION 1 THEN STATEMENT 1;
   WHEN SEARCH CONDITION 2 THEN STATEMENT 2;
   ...
   WHEN SEARCH CONDITION N THEN STATEMENT N;
   ELSE STATEMENT N+1;
END CASE;
```

When a search condition evaluates to TRUE, control is passed to the statement associated with it. If no search condition yields TRUE, statements associated with the ELSE clause are executed. Note that the ELSE clause is optional.

Example:

```
DECLARE
   v_num NUMBER := &sv_user_num;
BEGIN
   -- test if the number provided by the user is even
   CASE
      WHEN MOD(v_num,2) = 0 THEN
         DBMS_OUTPUT.PUT_LINE (v_num||' is even number');
      ELSE
         DBMS_OUTPUT.PUT_LINE (v_num||' is odd number');
   END CASE;
   DBMS_OUTPUT.PUT_LINE ('Done');
END;
```

DIFFERENCES BETWEEN CASE AND SEARCHED CASE STATEMENTS

The differences between CASE and searched CASE statements are:

- the searched CASE statement does not have a selector.
- the searched CASE statement's WHEN clauses contain search conditions that yield a Boolean value similar to the IF statement, not expressions that can yield a value of any type except a PL/SQL record, an index-by-table, a nested table, a vararray, BLOB, BFILE, or an object type.

Example:

```
DECLARE
   v_num      NUMBER := &sv_user_num;
   v_num_flag NUMBER;
BEGIN
   v_num_flag := MOD(v_num,2);
   -- test if the number provided by the user is even
   CASE v_num_flag
      WHEN 0 THEN
         DBMS_OUTPUT.PUT_LINE (v_num||' is even number');
...
and
DECLARE
   v_num NUMBER := &sv_user_num;
BEGIN
```

```
         -- test if the number provided by the user is even
         CASE
            WHEN MOD(v_num,2) = 0 THEN
      ...
```

In the first code fragment, `v_num_flag` is the selector. It is a PL/SQL variable that has been defined as NUMBER. Because the value of the expression is compared to the value of the selector, the expression must return a similar datatype. The expression `0` contains a number, so its datatype is also numeric. In the second code fragment, each searched expression evaluates to TRUE or FALSE, just like conditions of an IF statement.

NULLIF and COALESCE FUNCTIONS

Both functions can be used as a variation on the CASE expression.

THE NULLIF FUNCTION

The NULLIF function compares two expressions. If they are equal, the function returns NULL; otherwise, it returns the value of the first expression. NULLIF has the following structure:

```
      NULLIF (expression1, expression2)
```

If *expression1* is equal to *expression2*, NULLIF returns NULL. If *expression1* does not equal *expression2*, NULLIF returns *expression1*. The NULLIF function is equivalent to the following CASE expression:

```
      CASE
         WHEN expression1 = expression2 THEN NULL
         ELSE expression1
      END
```

Consider the following example of NULLIF:

Example:

```
      DECLARE
         v_num       NUMBER := &sv_user_num;
         v_remainder NUMBER;
      BEGIN
         -- calculate the remainder and if it is zero return NULL
         v_remainder := NULLIF(MOD(v_num,2),0);
         DBMS_OUTPUT.PUT_LINE ('v_remainder: '||v_remainder);
      END;
```

THE COALESCE FUNCTION

The COALESCE function compares each expression to NULL from the list of expressions and returns the value of the first non-null expression. The COALESCE function has the following structure:

```
    COALESCE (expression1, expression2, ..., expressionN)
```

If *expression1* evaluates to NULL, *expression2* is evaluated. If *expression2* does not evaluate to NULL, the function returns *expression2*. If *expression2* also evaluates to NULL, the next expression is evaluated. If all expressions evaluate to NULL, the function returns NULL.

Note that the COALESCE function is like a nested NVL function:

```
    NVL(expression1, NVL(expression2, NVL(expression3,...)))
```

The COALESCE function can also be used as an alternative to a CASE expression. For example,

```
    COALESCE (expression1, expression2)
```
is equivalent to

```
    CASE
       WHEN expression1 IS NOT NULL THEN expression1
       ELSE expression2
    END
```

If more than two expressions need to be evaluated,

```
    COALESCE (expression1, expression2, ..., expressionN)
```

is equivalent to

```
    CASE
       WHEN expression1 IS NOT NULL THEN expression1
       ELSE COALESCE (expression2, ..., expressionN)
    END
```

which in turn is equivalent to

```
    CASE
       WHEN expression1 IS NOT NULL THEN expression1
       WHEN expression2 IS NOT NULL THEN expression2
       ...
       ELSE expressionN
```

```
END
```

Example:

```
SELECT e.student_id, e.section_id, e.final_grade,
       g.numeric_grade,
       COALESCE(e.final_grade, g.numeric_grade, 0) 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';
```

The COALESCE function shown in the preceding example is equivalent to the following NVL statement and CASE expressions:

```
NVL(e.final_grade, NVL(g.numeric_grade, 0))
CASE
  WHEN e.final_grade IS NOT NULL THEN e.final_grade
  ELSE COALESCE(g.numeric_grade, 0)
END
```

and

```
CASE
   WHEN e.final_grade   IS NOT NULL THEN e.final_grade
   WHEN g.numeric_grade IS NOT NULL THEN g.numeric_grade
   ELSE 0
END
```

Exercisses:

1. Rewrite the following script.

```
SET SERVEROUTPUT ON
DECLARE
  v_date DATE := TO_DATE('&sv_user_date', 'DD-MON-YYYY');
  v_day  VARCHAR2(15);
BEGIN
  v_day := RTRIM(TO_CHAR(v_date, 'DAY'));
  IF v_day IN ('SATURDAY', 'SUNDAY') THEN
     DBMS_OUTPUT.PUT_LINE (v_date||' falls on weekend');
  END IF;
        --- control resumes here
  DBMS_OUTPUT.PUT_LINE ('Done...');
END;
```

Instead of getting information from the user for the variable `v_date`, define its value with the help of the function SYSDATE. After it has been determined that a certain day falls on the weekend, check to see if the time is before or after noon. Display the time of day together with the day.

2. Create a new script. For a given instructor, determine how many sections he or she is teaching. If the number is greater than or equal to 3, display a message saying that the instructor needs a vacation. Otherwise, display a message saying how many sections this instructor is teaching.

3. Create the following script. Modify the script you created at Exercise 1. You can use either the CASE statement or the searched CASE statement. The output should look similar to the output produced by the example you created at Exercise 1.

4. Create the following script. Modify the script you created at Exercise 2. You can use either the CASE statement or the searched CASE statement. The output should look similar to the output produced by the example you created at Exercise 2.

5. Execute the following two PL/SQL blocks, and explain why they produce different output for the same value of the variable `v_num`.

```
-- Block 1
 DECLARE
   v_num NUMBER := NULL;
 BEGIN
    IF v_num > 0 THEN
       DBMS_OUTPUT.PUT_LINE ('v_num is greater than 0');
    ELSE
       DBMS_OUTPUT.PUT_LINE ('v_num is not greater than 0');
    END IF;
 END;
-- Block 2
 DECLARE
    v_num NUMBER := NULL;
 BEGIN
    IF v_num > 0 THEN
       DBMS_OUTPUT.PUT_LINE ('v_num is greater than 0');
    END IF;
    IF NOT (v_num > 0) THEN
       DBMS_OUTPUT.PUT_LINE ('v_num is not greater than 0');
    END IF;
 END;
```

6. Execute the following two SELECT statements, and explain why they produce different output:

```
SELECT e.student_id, e.section_id, e.final_grade, g.numeric_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, e.final_grade, g.numeric_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';
```