
ORACLE
ACADEMY

Database Programming with PL/SQL

4-1
Conditional Control: IF Statements

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Describe a use for conditional control structures
- List the types of conditional control structures
- Construct and use an IF statement
- Construct and use an IF-THEN-ELSE statement
- Create PL/SQL to handle the null condition in IF statements

Purpose

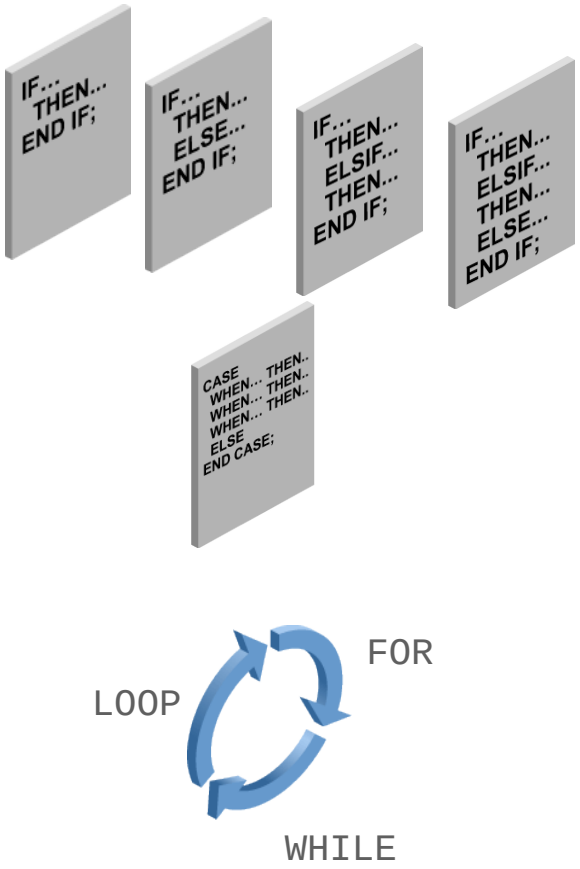
- In this section, you learn how to use the conditional logic in a PL/SQL block. Conditional processing extends the usefulness of programs by allowing the use of simple logical tests to determine which statements are executed.

Purpose

- Think of a logic test as something you do every day. If you get up in the morning and it is cold outside, you will choose to wear cold-weather clothing. If you get up in the morning and it is warm outside, you will choose to wear warm-weather clothing. And if there is a chance of rain, then you will bring a rain coat or an umbrella with you.

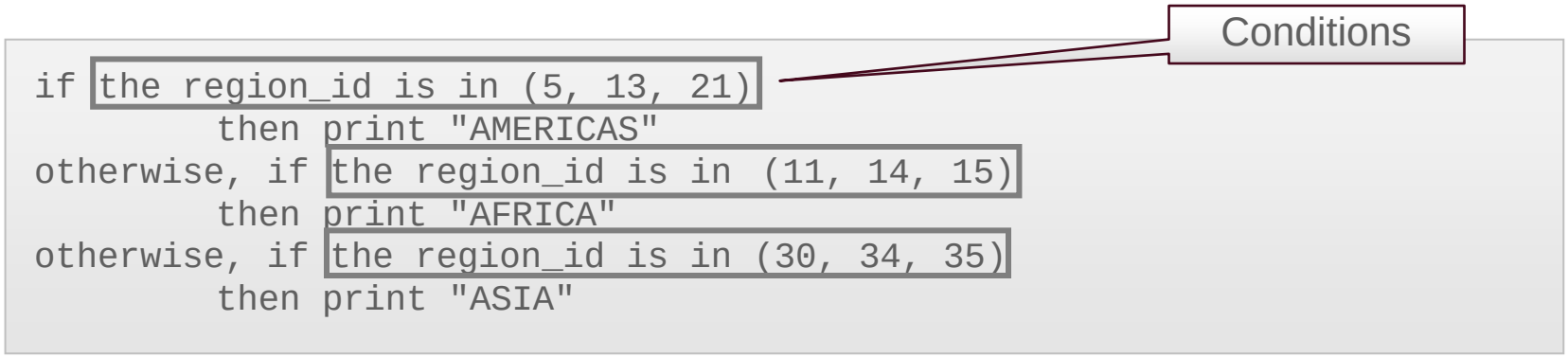
Controlling the Flow of Execution

- You can change the logical flow of statements within the PL/SQL block with a number of control structures.
- This lesson introduces three types of PL/SQL control structures:
  - Conditional constructs with the IF statement
  - CASE expressions
  - LOOP control structures



IF Statement

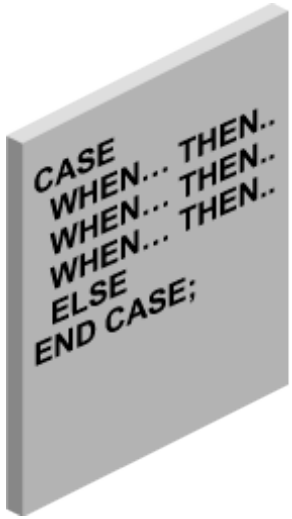
- The IF statement contains alternative courses of action in a block based on conditions. A condition is an expression with a TRUE or FALSE value that is used to make a decision.



CASE Expressions

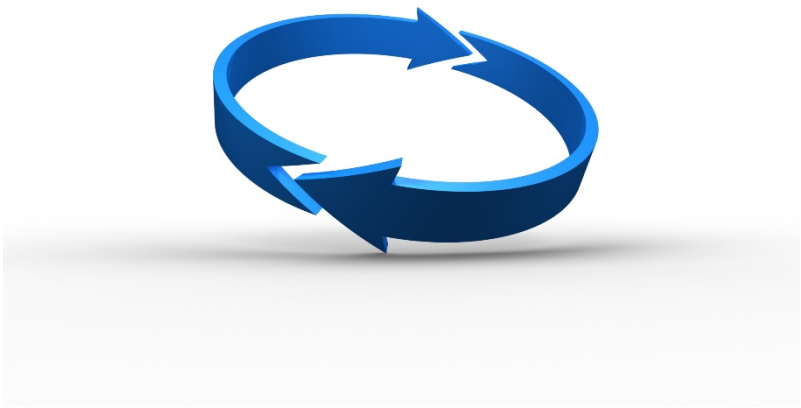
- CASE expressions are similar to IF statements in that they also determine a course of action based on conditions. They are different in that they can be used outside of a PLSQL block in an SQL statement. Consider the following example:

```
If the region_id is
        5 then print  "AMERICAS"
        13 then print "AMERICAS"
        21 then print "AMERICAS"
        11 then print "AFRICA"
        14 then print "AFRICA"
        15 then print "AFRICA"
        30 then print "ASIA" …
```



LOOP Control Structures

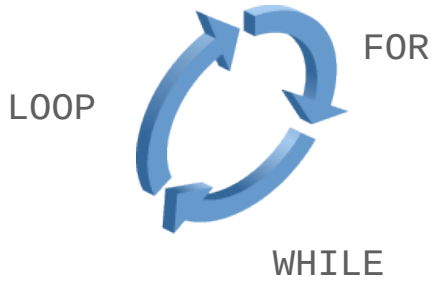
- Loop control structures are repetition statements that enable you to execute statements in a PLSQL block repeatedly.
- Three types of loop control structures are supported by PL/SQL: BASIC, FOR, and WHILE.



LOOP Control Structures

- Consider the following example:
- Print the numbers 1–5 by using a loop and a counter.

```
Loop Counter equals: 1
Loop Counter equals: 2
Loop Counter equals: 3
Loop Counter equals: 4
Loop Counter equals: 5
Statement processed.
```



IF Statements Structure

- The structure of the PL/SQL IF statement is similar to the structure of IF statements in other procedural languages. It enables PL/SQL to perform actions selectively based on conditions.

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

IF Statements

- Condition is a Boolean variable or expression that returns TRUE, FALSE, or NULL. THEN introduces a clause that associates the Boolean expression with the sequence of statements that follows it.

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

IF Statements

- Statements can be one or more PL/SQL or SQL statements.
- They can include further IF statements containing several nested IF, ELSE, and ELSIF statements.
- The statements in the THEN clause are executed only if the condition in the associated IF clause evaluates to TRUE.

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

IF Statements

- ELSIF is a keyword that introduces a Boolean expression. (If the first condition yields FALSE or NULL, then the ELSIF keyword introduces additional conditions. ELSIF is the correct spelling, not ELSEIF.)

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

IF Statements

- ELSE introduces the default clause that is executed if and only if none of the earlier conditions (introduced by IF and ELSIF) are TRUE. The tests are executed in sequence so that a later condition that might be true is pre-empted by an earlier condition that is true.
- END IF; marks the end of an IF statement.

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

IF Statements Note

- ELSIF and ELSE are optional in an IF statement. You can have any number of ELSIF keywords but only one ELSE keyword in your IF statement. END IF marks the end of an IF statement and must be terminated by a semicolon.

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Simple IF Statement

- This is an example of a simple IF statement with a THEN clause. The v_myage variable is initialized to 31. The condition for the IF statement returns FALSE because v_myage is not less than 11. Therefore, the control never reaches the THEN clause and nothing is printed to the screen.

```
DECLARE
  v_myage NUMBER:=31;
BEGIN
  IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
```

IF THEN ELSE Statement

- The ELSE clause has been added to this example. The condition has not changed, thus it still evaluates to FALSE. Remember that the statements in the THEN clause are only executed if the condition returns TRUE. In this case, the condition returns FALSE, so control passes to the ELSE statement.

```
DECLARE
  v_myage NUMBER:=31;
BEGIN
  IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
```

IF ELSIF ELSE Clause

- The IF statement now contains multiple ELSIF clauses as well as an ELSE clause.
- Notice that the ELSIF clauses add additional conditions. As with the IF statement, each ELSIF condition is followed by a THEN clause, which is executed only if the ELSIF condition returns TRUE.

```
DECLARE
  v_myage NUMBER:=31;
BEGIN
  IF v_myage < 11
    THEN
      DBMS_OUTPUT.PUT_LINE('I am a child');
  ELSIF v_myage < 20
    THEN
      DBMS_OUTPUT.PUT_LINE('I am young');
  ELSIF v_myage < 30
    THEN
      DBMS_OUTPUT.PUT_LINE('I am in my
twenties');
  ELSIF v_myage < 40
    THEN
      DBMS_OUTPUT.PUT_LINE('I am in my
thirties');
  ELSE
    DBMS_OUTPUT.PUT_LINE('I am always young ');
  END IF;
END;
```

IF ELSIF ELSE Clause

- When you have multiple clauses in the IF statement and a condition is FALSE or NULL, control then passes to the next clause. Conditions are evaluated one by one. If all conditions are FALSE or NULL, then the statements in the ELSE clause are executed. The final ELSE clause is optional.

```
…IF    v_myage < 11 THEN
     DBMS_OUTPUT.PUT_LINE(' I am a child ');
 ELSIF v_myage < 20 THEN
     DBMS_OUTPUT.PUT_LINE(' I am young ');
 ELSIF v_myage < 30 THEN
     DBMS_OUTPUT.PUT_LINE(' I am in my twenties ');
 ELSIF v_myage < 40 THEN
     DBMS_OUTPUT.PUT_LINE(' I am in my thirties ');
 ELSE
     DBMS_OUTPUT.PUT_LINE(' I am always young ');
END IF;…
```

IF Statement with Multiple Expressions

- An IF statement can have multiple conditional expressions related with logical operators, such as AND, OR, and NOT.

IF Statement with Multiple Expressions

- The example uses the AND operator. Therefore, it evaluates to TRUE only if both the first name and age conditions are evaluated as TRUE. There is no limitation on the number of conditional expressions that can be used; however, these statements must be connected with the appropriate logical operators.

```
DECLARE
  v_myage       NUMBER       := 31;
  v_myfirstname VARCHAR2(11) := 'Christopher';
BEGIN
  IF v_myfirstname ='Christopher' AND v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child named Christopher');
  END IF;
END;
```

NULL Values in IF Statements

- In this example, the v_myage variable is declared but is not initialized. The condition in the IF statement returns NULL, which is neither TRUE nor FALSE. In such a case, the control goes to the ELSE statement because, just like FALSE, NULL is not TRUE.

```
DECLARE
  v_myage NUMBER;
BEGIN
  IF v_myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
```

Handling Nulls

- When working with nulls, you can avoid some common mistakes by keeping in mind the following rules:
- Simple comparisons involving nulls always yield NULL.
- Applying the logical operator NOT to a null yields NULL.
- In conditional control statements, if a condition yields NULL, it behaves just like a FALSE, and the associated sequence of statements is not executed.

Handling Nulls Example

- In this example, you might expect the sequence of statements to execute because a and b seem equal. But, again, equality is unknown. So the IF condition yields NULL and the sequence of statements is bypassed.

```
a := NULL;
b := NULL;
...
IF a = b THEN … -- yields NULL, not TRUE and the
         sequence of statements is not executed
END IF;
```

Guidelines for Using IF Statements

Follow these guidelines when using IF statements:

- You can perform actions selectively when a specific condition is being met.
- When writing code, remember the spelling of the keywords:
  - ELSIF is one word
  - END IF is two words

Guidelines for Using IF Statements

- If the controlling Boolean condition is TRUE, then the associated sequence of statements is executed; if the controlling Boolean condition is FALSE or NULL, then the associated sequence of statements is passed over. Any number of ELSIF clauses is permitted.
- Indent the conditionally executed statements for clarity.

Terminology

Key terms used in this lesson included:

- CASE
- Condition
- IF
- LOOP

Summary

In this lesson, you should have learned how to:

- Describe a use for conditional control structures
- List the types of conditional control structures
- Construct and use an IF statement
- Construct and use an IF-THEN-ELSE statement
- Create PL/SQL to handle the null condition in IF statements


ORACLE
ACADEMY