**Simple IF Statement**

**Simple** IF **Example**

The slide shows an example of a simple IF statement with the 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.

**Adding Conditional Expressions**

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

For example:

IF (myfirstname='Christopher' AND v\_myage <11)  
…

The condition uses the AND operator and therefore, evaluates to TRUE only if both conditions are evaluated as TRUE. There is no limitation on the number of conditional expressions. However, these statements must be related with appropriate logical operators.

**IF THEN ELSE Statement**

An ELSE clause is added to the code in the previous slide. The condition has not changed and, therefore, still evaluates to FALSE. Recall that the statements in the THEN clause are executed only if the condition returns TRUE. In this case, the condition returns FALSE and the control moves to the ELSE statement.

The output of the block is shown below the code.

**IF ELSIF ELSE Clause**

The IF clause may contain multiple ELSIF clauses and an ELSE clause. The example illustrates the following characteristics of these clauses:

* + - The ELSIF clauses can have conditions, unlike the ELSE clause.
    - The condition for ELSIF should be followed by the THEN clause, which is executed if the condition for ELSIF returns TRUE.
    - When you have multiple ELSIF clauses, if the first condition is FALSE or NULL, the control shifts to the next ELSIF clause.
    - Conditions are evaluated one by one from the top.
    - If all conditions are FALSE or NULL, the statements in the ELSE clause are executed.
    - The final ELSE clause is optional.

In the example, the output of the block is shown below the code.

**NULL Value in IF Statement**

In the example shown in the slide, the variable v\_myage is declared but not initialized. The condition in the IF statement returns NULL rather than TRUE or FALSE. In such a case, the control goes to the ELSE statement.

**Guidelines**

* + - You can perform actions selectively based on conditions that are being met.
    - When you write code, remember the spelling of the keywords:
      * ELSIF is one word.
      * END IF is two words.
    - If the controlling Boolean condition is TRUE, the associated sequence of statements is executed; if the controlling Boolean condition is FALSE or NULL, the associated sequence of statements is passed over. Any number of ELSIF clauses is permitted.
    - Indent the conditionally executed statements for clarity.

**CASE Expressions**

* A CASE expression returns a result based on one or more alternatives. To return the result, the CASE expression uses a *selector*, which is an expression whose value is used to return one of several alternatives. The selector is followed by one or more WHEN clauses that are checked sequentially. The value of the selector determines which result is returned. If the value of the selector equals the value of a WHEN clause expression, that WHEN clause is executed and that result is returned.
* PL/SQL also provides a searched CASE expression, which has the form:
* CASE  
   WHEN search\_condition1 THEN result1  
   WHEN search\_condition2 THEN result2  
   ...  
   WHEN search\_conditionN THEN resultN  
   [ELSE resultN+1]  
  END;
* A searched CASE expression has no selector. Furthermore, the WHEN clauses in CASE expressions contain search conditions that yield a Boolean value rather than expressions that can yield a value of any type.
* **CASE Expressions: Example**
* In the example in the slide, the CASE expression uses the value in the v\_grade variable as the expression. This value is accepted from the user by using a substitution variable. Based on the value entered by the user, the CASE expression returns the value of the v\_appraisal variable based on the value of the v\_grade value.
* **Result**
* When you enter a or A for v\_grade, as shown in the Substitution Variable window, the output of the example is as follows:
* **Searched CASE Expressions**
* In the previous example, you saw a single test expression, the v\_grade variable.   
  The WHEN clause compared a value against this test expression.
* In searched CASE statements, you do not have a test expression. Instead, the WHEN clause contains an expression that results in a Boolean value. The same example is rewritten in this slide to show searched CASE statements.
* **Result**
* The output of the example is as follows when you enter b or B for v\_grade:

**Handling Nulls**

Consider the following example:

x := 5;  
y := NULL;  
...  
IF x != y THEN -- yields NULL, not TRUE  
 -- **sequence\_of\_statements that are not executed**END IF;

You may expect the sequence of statements to execute because x and y seem unequal. But nulls are indeterminate. Whether or not x is equal to y is unknown. Therefore, the IF condition yields NULL and the sequence of statements is bypassed.

a := NULL;  
b := NULL;  
...  
IF a = b THEN -- yields NULL, not TRUE  
 -- **sequence\_of\_statements that are not executed**END IF;

In the second example, you may 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.

**Logic Tables**

You can build a simple Boolean condition by combining number, character, and date expressions with comparison operators.

You can build a complex Boolean condition by combining simple Boolean conditions with the logical operators AND, OR, and NOT. The logical operators are used to check the Boolean variable values and return TRUE, FALSE, or NULL. In the logic tables shown in the slide:

* + - FALSE takes precedence in an AND condition, and TRUE takes precedence in an OR condition
    - AND returns TRUE only if both of its operands are TRUE
    - OR returns FALSE only if both of its operands are FALSE
    - NULL AND TRUE always evaluates to NULL because it is not known whether the second operand evaluates to TRUE

**Note:** The negation of NULL (NOT NULL) results in a null value because null values are indeterminate.

**Boolean Expressions or Logical Expression?**

The AND logic table can help you to evaluate the possibilities for the Boolean condition in the slide.

**Answers**

1. TRUE

2. FALSE

3. NULL

4. FALSE

**Iterative Control: LOOP Statements**

PL/SQL provides several facilities to structure loops to repeat a statement or sequence of statements multiple times. Loops are mainly used to execute statements repeatedly until an exit condition is reached. It is mandatory to have an exit condition in a loop; otherwise, the loop is infinite.

Looping constructs are the third type of control structures. PL/SQL provides the following types of loops:

* + - Basic loop that performs repetitive actions without overall conditions
    - FOR loops that perform iterative actions based on a count
    - WHILE loops that perform iterative actions based on a condition

**Note:** An EXIT statement can be used to terminate loops. A basic loop must have an EXIT. The cursor FOR loop (which is another type of FOR loop) is discussed in the lesson titled “Using Explicit Cursors.”

**Basic Loops**

The simplest form of a LOOP statement is the basic loop, which encloses a sequence of statements between the LOOP and END LOOP keywords. Each time the flow of execution reaches the END LOOP statement, control is returned to the corresponding LOOP statement above it. A basic loop allows execution of its statements at least once, even if the EXIT condition is already met upon entering the loop. Without the EXIT statement, the loop would be infinite.

**EXIT Statement**

You can use the EXIT statement to terminate a loop. Control passes to the next statement after the END LOOP statement. You can issue EXIT either as an action within an IF statement or as a stand-alone statement within the loop. The EXIT statement must be placed inside a loop. In the latter case, you can attach a WHEN clause to enable conditional termination of the loop. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop ends and control passes to the next statement after the loop.   
A basic loop can contain multiple EXIT statements, but it is recommended that you have only one EXIT point.

**Basic Loop: Example**

The basic loop example shown in the slide is defined as follows: “Insert three new location IDs for the CA country code and the city of Montreal.”

**Note**

* + - A basic loop allows execution of its statements until the EXIT WHEN condition is met.
    - If the condition is placed in the loop such that it is not checked until after the loop statements execute, the loop executes at least once.
    - However, if the exit condition is placed at the top of the loop (before any of the other executable statements) and if that condition is true, the loop exits and the statements never execute.

**Results**

To view the output, run the code example: code\_05\_22\_s.sql.

**WHILE Loops**

You can use the WHILE loop to repeat a sequence of statements until the controlling condition is no longer TRUE. The condition is evaluated at the start of each iteration. The loop terminates when the condition is FALSE or NULL. If the condition is FALSE or NULL at the start of the loop, no further iterations are performed. Thus, it is possible that none of the statements inside the loop are executed.

In the syntax:

*condition* Is a Boolean variable or expression (TRUE, FALSE, or NULL)

*statement* Can be one or more PL/SQL or SQL statements

If the variables involved in the conditions do not change during the body of the loop, the condition remains TRUE and the loop does not terminate.

**Note:** If the condition yields NULL, the loop is bypassed and control passes to the next statement.

**WHILE Loops: Example**

In the example in the slide, three new location IDs for the CA country code and the city of Montreal are added.

* + - With each iteration through the WHILE loop, a counter (v\_counter) is incremented.
    - If the number of iterations is less than or equal to the number 3, the code within the loop is executed and a row is inserted into the locations table.
    - After v\_counter exceeds the number of new locations for this city and country, the condition that controls the loop evaluates to FALSE and the loop terminates.

**Results**

To view the output, run the code example: code\_05\_24\_s.sql.

**FOR Loops**

FOR loops have the same general structure as the basic loop. In addition, they have a control statement before the LOOP keyword to set the number of iterations that the PL/SQL performs.

In the syntax:

Do not declare the counter. It is declared implicitly as an integer.

**FOR Loops (continued)**

**Note:** The sequence of statements is executed each time the counter is incremented, as determined by the two bounds. The lower bound and upper bound of the loop range can be literals, variables, or expressions, but they must evaluate to integers. The bounds are rounded to integers; that is, 11/3 and 8/5 are valid upper or lower bounds. The lower bound and upper bound are inclusive in the loop range. If the lower bound of the loop range evaluates to a larger integer than the upper bound, the sequence of statements is not executed.   
For example, the following statement is executed only once:

FOR i IN 3..3   
LOOP   
 statement1*;*END LOOP;

**FOR Loop Rules**

The slide lists the guidelines to follow when writing a FOR loop.

**Note:** The lower and upper bounds of a LOOP statement do not need to be numeric literals. They can be expressions that convert to numeric values.

**Example:**

DECLARE  
 v\_lower NUMBER := 1;  
 v\_upper NUMBER := 100;  
BEGIN  
 FOR i IN v\_lower..v\_upper LOOP  
 ...  
 END LOOP;  
END;

/

**Suggested Use of Loops**

A basic loop allows the execution of its statement at least once, even if the condition is already met upon entering the loop. Without the EXIT statement, the loop would be infinite.

You can use the WHILE loop to repeat a sequence of statements until the controlling condition is no longer TRUE. The condition is evaluated at the start of each iteration. The loop terminates when the condition is FALSE. If the condition is FALSE at the start of the loop, no further iterations are performed.

FOR loops have a control statement before the LOOP keyword to determine the number of iterations that the PL/SQL performs. Use a FOR loop if the number of iterations is predetermined.

**Nested Loops and Labels**

You can nest the FOR, WHILE, and basic loops within one another. The termination of a nested loop does not terminate the enclosing loop unless an exception is raised. However, you can label loops and exit the outer loop with the EXIT statement.

Label names follow the same rules as the other identifiers. A label is placed before a statement, either on the same line or on a separate line. White space is insignificant in all PL/SQL parsing except inside literals. Label basic loops by placing the label before the word LOOP within label delimiters (<<*label*>>). In FOR and WHILE loops, place the label before FOR or WHILE.

If the loop is labeled, the label name can be included (optionally) after the END LOOP statement for clarity.

**Nested Loops and Labels: Example**

In the example in the slide, there are two loops. The outer loop is identified by the label <<Outer\_Loop>> and the inner loop is identified by the label <<Inner\_Loop>>.   
The identifiers are placed before the word LOOP within label delimiters (<<*label*>>). The inner loop is nested within the outer loop. The label names are included after the END LOOP statements for clarity.

**PL/SQL CONTINUE Statement**

The CONTINUE statement enables you to transfer control within a loop back to a new iteration or to leave the loop. Many other programming languages have this functionality. With the Oracle Database 11*g* release, PL/SQL also offers this functionality. Before the Oracle Database 11*g* release, you could code a workaround by using Boolean variables and conditional statements to simulate the CONTINUE programmatic functionality. In some cases, the workarounds are less efficient.

The CONTINUE statement offers you a simplified means to control loop iterations. It may be more efficient than the previous coding workarounds.

The CONTINUE statement is commonly used to filter data within a loop body before the main processing begins.

**PL/SQL CONTINUE Statement: Example 1**

In the example, there are two assignments using the v\_total variable:

1. The first assignment is executed for each of the 10 iterations of the loop.

2 The second assignment is executed for the first five iterations of the loop. The CONTINUE statement transfers control within a loop back to a new iteration, so for the last five iterations of the loop, the second TOTAL assignment is not executed.

The end result of the TOTAL variable is 70.

**PL/SQL CONTINUE Statement: Example 2**

You can use the CONTINUE statement to jump to the next iteration of an outer loop. To do this, provide the outer loop a label to identify where the CONTINUE statement should go.

The CONTINUE statement in the innermost loop terminates that loop whenever the WHEN condition is true (just like the EXIT keyword). After the innermost loop is terminated by the CONTINUE statement, control transfers to the next iteration of the outermost loop labeled BeforeTopLoop in this example.

When this pair of loops completes, the value of the TOTAL variable is 20.

You can also use the CONTINUE statement within an inner block of code, which does not contain a loop as long as the block is nested inside an appropriate outer loop.

**Restrictions**

* + - The CONTINUE statement cannot appear outside a loop at all—this generates a compiler error.
    - You cannot use the CONTINUE statement to pass through a procedure, function, or method boundary—this generates a compiler error.

**Answer: 1**

**Loop Types**

PL/SQL provides the following types of loops:

* + - Basic loops that perform repetitive actions without overall conditions
    - FOR loops that perform iterative actions based on a count
    - WHILE loops that perform iterative actions based on a condition