5

Writing Control Structures

Objectives

After completing this lesson, you should be able to do the following:

- **Identify the uses and types of control structures**
- **Construct an `IF` statement**
- **Use `CASE` statements and `CASE` expressions**
- **Construct and identify different loop statements**
- **Use guidelines when using conditional control structures**

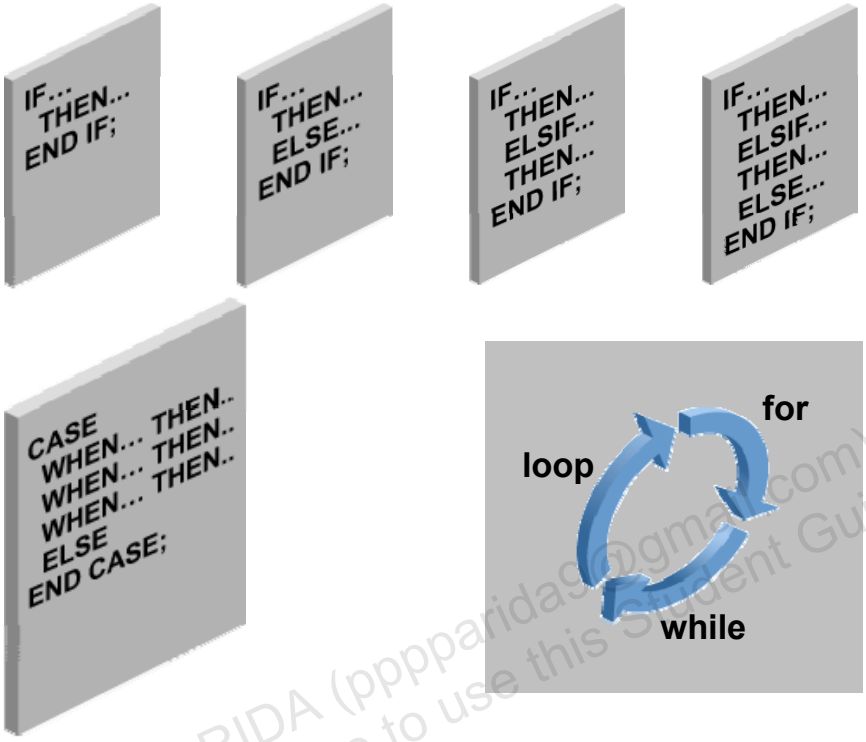
ORACLE

Lesson Aim

You have learned to write PL/SQL blocks containing declarative and executable sections. You have also learned to include expressions and SQL statements in the executable block.

In this lesson, you learn how to use control structures such as `IF` statements, `CASE` expressions, and `LOOP` structures in a PL/SQL block.

Controlling Flow of Execution



Controlling Flow of Execution

You can change the logical flow of statements within the PL/SQL block with a number of control structures. This lesson addresses three types of PL/SQL control structures: conditional constructs with the `IF` statement, `CASE` expressions, and `LOOP` control structures.

IF Statements

Syntax:

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


ORACLE

IF Statements

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

In the syntax:

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
statements	Can be one or more PL/SQL or SQL statements. (They may 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 Statements (continued)

In the syntax:

ELSIF	Is a keyword that introduces a Boolean expression (If the first condition yields FALSE or NULL, the ELSIF keyword introduces additional conditions.)
ELSE	Introduces the default clause that is executed if and only if none of the earlier predicates (introduced by IF and ELSIF) are TRUE. The tests are executed in sequence so that a later predicate that might be true is preempted by an earlier predicate that is true.
END IF	END IF marks the end of an IF statement

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.

Simple IF Statement

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

PL/SQL procedure successfully completed.

ORACLE

Simple IF Statement

The slide shows an example of a simple IF statement with the THEN clause. The variable myage is initialized to 31. The condition for the IF statement returns FALSE because myage is not less than 11. Therefore, the control never reaches the THEN clause. We add code to this example to see the usage of ELSE and ELSIF.

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

```
IF (myfirstname='Christopher' AND 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

```
SET SERVEROUTPUT ON
DECLARE
myage number:=31;
BEGIN
IF myage < 11
 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
 ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```

```
I am not a child
PL/SQL procedure successfully completed.
```

ORACLE

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 in the slide.

IF ELSIF ELSE Clause

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

```
I am in my thirties
PL/SQL procedure successfully completed.
```

ORACLE

IF ELSIF ELSE Clause

The IF clause now contains multiple ELSIF clauses and an ELSE. Notice that 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 of the 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.

NULL Values in IF Statements

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

```
I am not a child
PL/SQL procedure successfully completed.
```


ORACLE

NULL Values in IF Statements

In the example shown in the slide, the variable 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 writing 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 are permitted.
- Indent the conditionally executed statements for clarity.

CASE Expressions

- A CASE expression selects a result and returns it.
- To select the result, the CASE expression uses expressions. The value returned by these expressions is used to select one of several alternatives.

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```


ORACLE

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, its WHEN clauses contain search conditions that yield a Boolean value rather than expressions that can yield a value of any type.

CASE Expressions: Example

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DECLARE
   grade CHAR(1) := UPPER('&grade');
   appraisal VARCHAR2(20);
BEGIN
   appraisal :=
      CASE grade
         WHEN 'A' THEN 'Excellent'
         WHEN 'B' THEN 'Very Good'
         WHEN 'C' THEN 'Good'
         ELSE 'No such grade'
      END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| grade || '
                      Appraisal ' || appraisal);
END;
/
```


CASE Expressions: Example

In the example in the slide, the `CASE` expression uses the value in the `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 `appraisal` variable based on the value of the `grade` value. The output of the example is as follows when you enter `a` or `A` for the grade:

```
Grade: A Appraisal Excellent
PL/SQL procedure successfully completed.
```

Searched CASE Expressions

```
DECLARE
   grade CHAR(1) := UPPER('&grade');
   appraisal VARCHAR2(20);
BEGIN
    appraisal :=
     CASE
         WHEN grade = 'A' THEN 'Excellent'
         WHEN grade IN ('B','C') THEN 'Good'
         ELSE 'No such grade'
     END;
   DBMS_OUTPUT.PUT_LINE ('Grade: '|| grade || '
                  Appraisal ' || appraisal);
END;
/
```


Searched CASE Expressions

In the previous example, you saw a single test expression that was the `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.

CASE Statement

```
DECLARE
   deptid NUMBER;
   deptname VARCHAR2(20);
   emps NUMBER;
   mngid NUMBER:= 108;
BEGIN
  CASE  mngid
   WHEN  108 THEN
    SELECT department_id, department_name
     INTO deptid, deptname FROM departments
     WHERE manager_id=108;
    SELECT count(*) INTO emps FROM employees
     WHERE department_id=deptid;
   WHEN  200 THEN
    ...
 END CASE;
DBMS_OUTPUT.PUT_LINE ('You are working in the '|| deptname||
' department. There are '||emps ||' employees in this
department');
END;
/
```


CASE Statement

Recall the use of the IF statement. You may include n number of PL/SQL statements in the THEN clause and also in the ELSE clause. Similarly, you can include statements in the CASE statement. The CASE statement is more readable compared to multiple IF and ELSIF statements.

How Is a CASE Expression Different from a CASE Statement?

A CASE expression evaluates the condition and returns a value. On the other hand, a CASE statement evaluates the condition and performs an action. A CASE statement can be a complete PL/SQL block. CASE statements end with END CASE; but CASE expressions end with 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`.**
- **If the condition yields `NULL` in conditional control statements, its associated sequence of statements is not executed.**

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

Build a simple Boolean condition with a comparison operator.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

ORACLE

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 or not.

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

Boolean Conditions

What is the value of `flag` in each case?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG
TRUE	TRUE	? (1)
TRUE	FALSE	? (2)
NULL	TRUE	? (3)
NULL	FALSE	? (4)

ORACLE

Boolean Conditions

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

Answers

1. TRUE
2. FALSE
3. NULL
4. FALSE

Iterative Control: LOOP Statements

- **Loops repeat a statement or sequence of statements multiple times.**
- **There are three loop types:**
  - **Basic loop**
  - **FOR loop**
  - **WHILE loop**

Iterative Control: LOOP Statements

PL/SQL provides a number of 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 second type of control structure. 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

Syntax:

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```


ORACLE

Basic Loops

The simplest form of a `LOOP` statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords `LOOP` and `END LOOP`. 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 Loops

Example

```
DECLARE
  countryid    locations.country_id%TYPE := 'CA';
  loc_id       locations.location_id%TYPE;
  counter      NUMBER(2) := 1;
  new_city     locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO loc_id FROM locations
  WHERE country_id = countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((loc_id + counter), new_city, countryid);
    counter := counter + 1;
    EXIT WHEN counter > 3;
  END LOOP;
END;
/
```

ORACLE

Basic Loops (continued)

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 at least once, even if the condition has been met upon entering the loop. This happens only if the condition is placed in the loop so that it is not checked until after these statements. 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.

WHILE Loops

Syntax:

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Use the WHILE loop to repeat statements while a condition is TRUE.

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.

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

```
DECLARE
  countryid   locations.country_id%TYPE := 'CA';
  loc_id      locations.location_id%TYPE;
  new_city    locations.city%TYPE := 'Montreal';
  counter     NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO loc_id FROM locations
  WHERE country_id = countryid;
  WHILE counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((loc_id + counter), new_city, countryid);
    counter := counter + 1;
  END LOOP;
END;
/
```


WHILE Loops (continued)

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

With each iteration through the WHILE loop, a counter (counter) is incremented. If the number of iterations is less than or equal to the number 3, then the code within the loop is executed and a row is inserted into the locations table. After the 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.

FOR Loops

- Use a FOR loop to shortcut the test for the number of iterations.
- Do not declare the counter; it is declared implicitly.
- 'lower_bound .. upper_bound' is required syntax.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```


ORACLE

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 PL/SQL performs.
In the syntax:

counter	Is an implicitly declared integer whose value automatically increases or decreases (decreases if the REVERSE keyword is used) by 1 on each iteration of the loop until the upper or lower bound is reached
REVERSE	Causes the counter to decrement with each iteration from the upper bound to the lower bound **Note:** The lower bound is still referenced first.
lower_bound	Specifies the lower bound for the range of counter values
upper_bound	Specifies the upper bound for the range of counter values

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 Loops

Example

```
DECLARE
  countryid   locations.country_id%TYPE := 'CA';
  loc_id      locations.location_id%TYPE;
  new_city    locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO loc_id
    FROM locations
    WHERE country_id = countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((loc_id + i), new_city, countryid );
  END LOOP;
END;
/
```

ORACLE

FOR Loops (continued)

You have already learned how to insert three new locations for the CA country code and the city Montreal by using the basic loop and the WHILE loop. This slide shows you how to achieve the same by using the FOR loop.

FOR Loops

Guidelines

- Reference the counter within the loop only; it is undefined outside the loop.
- Do not reference the counter as the target of an assignment.
- Neither loop bound should be NULL.

ORACLE

FOR Loops (continued)

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
  lower  NUMBER := 1;
  upper  NUMBER := 100;
BEGIN
  FOR i IN lower..upper LOOP
  ...
  END LOOP;
END;
/
```

Guidelines for Loops

- **Use the basic loop when the statements inside the loop must execute at least once.**
- **Use the `WHILE` loop if the condition must be evaluated at the start of each iteration.**
- **Use a `FOR` loop if the number of iterations is known.**

ORACLE

Guidelines for Loops

A basic loop allows 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 PL/SQL performs. Use a `FOR` loop if the number of iterations is predetermined.

Nested Loops and Labels

- You can nest loops to multiple levels.
- Use labels to distinguish between blocks and loops.
- Exit the outer loop with the `EXIT` statement that references the label.

ORACLE

Nested Loops and Labels

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

Label names follow the same rules as 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 optionally be included after the `END LOOP` statement for clarity.

Nested Loops and Labels

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    counter := counter+1;
  EXIT WHEN counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
/
```

ORACLE

Nested Loops and Labels (continued)

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.

Summary

In this lesson, you should have learned how to change the logical flow of statements by using the following control structures:

- **Conditional (`IF` statement)**
- **`CASE` expressions and `CASE` statements**
- **Loops:**
  - **Basic loop**
  - **`FOR` loop**
  - **`WHILE` loop**
- **`EXIT` statements**

ORACLE

Summary

A language can be called a programming language only if it provides control structures for the implementation of the business logic. These control structures are also used to control the flow of the program. PL/SQL is a programming language that integrates programming constructs with SQL.

A conditional control construct checks for the validity of a condition and performs an action accordingly. You use the `IF` construct to perform a conditional execution of statements.

An iterative control construct executes a sequence of statements repeatedly, as long as a specified condition holds `TRUE`. You use the various loop constructs to perform iterative operations.

Practice 5: Overview

This practice covers the following topics:

- **Performing conditional actions by using the `IF` statement**
- **Performing iterative steps by using the loop structure**

ORACLE

Practice 5: Overview

In this practice, you create PL/SQL blocks that incorporate loops and conditional control structures. The exercises test your understanding of writing various `IF` statements and `LOOP` constructs.

Practice 5

1. Execute the command in the file `lab_05_01.sql` to create the `messages` table. Write a PL/SQL block to insert numbers into the `messages` table.
   a. Insert the numbers 1 to 10, excluding 6 and 8.
   b. Commit before the end of the block.
   c. Execute a `SELECT` statement to verify that your PL/SQL block worked. You should see the following output.

RESULTS
1
2
3
4
5
7
9
10

8 rows selected.

2. Execute the script `lab_05_02.sql`. This script creates an `emp` table that is a replica of the `employees` table. It alters the `emp` table to add a new column, `stars`, of `VARCHAR2` data type and size 50. Create a PL/SQL block that inserts an asterisk in the `stars` column for every $1000 of the employee's salary. Save your script as `lab_05_02_soln.sql`.
   a. Use the `DEFINE` command to define a variable called `empno` and initialize it to 176.
   b. Start the declarative section of the block and pass the value of `empno` to the PL/SQL block through an *i*SQL*Plus substitution variable. Declare a variable `asterisk` of type `emp.stars` and initialize it to `NULL`. Create a variable `sal` of type `emp.salary`.
   c. In the executable section, write logic to append an asterisk (*) to the string for every $1000 of the salary amount. For example, if the employee earns $8000, the string of asterisks should contain eight asterisks. If the employee earns $12500, the string of asterisks should contain 13 asterisks.
   d. Update the `stars` column for the employee with the string of asterisks. Commit before the end of the block.

Practice 5 (continued)

e. Display the row from the emp table to verify whether your PL/SQL block has executed successfully.

f. Execute and save your script as `lab_05_02_soln.sql`. The output is shown below.

EMPLOYEE_ID	SALARY	STARS
176	8600	*********

3. Load the script `lab_04_04_soln.sql`, which you created in question 4 of Practice 4.

   a. Look for the comment "INCLUDE SIMPLE IF STATEMENT HERE" and include a simple `IF` statement to check if the values of `emp_id` and `emp_authorization` are the same.

   b. Save your script as `lab_05_03_soln.sql`.