

# Oracle Academy

## Introduction to Database Programming with PL/SQL

### Instructor Resource Guide

#### INSTRUCTOR NOTES FOR SLIDES

---

##### **General Note On Infinite Loops:**

When a user executes a PL/SQL block in APEX that contains an infinite loop, there is no way for the user to stop the loop. It can only be stopped by the DBA who oversees the Academy database. If the loop does not contain DML statements, closing the browser window and reopening it will allow the user to continue coding. If the block contains a FOR UPDATE clause, the affected table(s) will remain locked until released by the DBA, which may take a day or more.

This is important throughout the practices, but especially in Section 4 where students learn about looping structures for the first time, and are therefore more likely to make mistakes.

#### **SECTION 4 LESSON 1 – Conditional Control: IF Statements**

##### Slide 1: Conditional Control: IF Statements

No instructor notes for this slide

##### Slide 2: What Will I Learn?

No instructor notes for this slide

##### Slide 3: Why Learn It?

No instructor notes for this slide

##### Slide 4: Tell Me / Show Me – Controlling the Flow of Execution

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.

The CASE statement contains alternative courses of action in a block based on one condition. They are different in that they can be used outside of a PLSQL block in a SQL statement.

LOOPS are control structures that allow iteration of statements. Loop control structures are repetition statements that enable you to execute statements in a PLSQL block repeatedly. There are three types of loop control structures supported by PL/SQL, **BASIC**, **FOR**, and **WHILE**.

##### Slide 5: Tell Me / Show Me – IF Statement

The examples in this and the next slide are written in natural language-like “pseudocode” to help students to understand the concepts.

Please note that the ELSIF clause does not have a 'E' after the 'S'. Next note, that there can be many ELSIF clauses attached to the IF statement. Also, note that a IF statement must be before any and all ELSIF and ELSE clauses, these clauses cannot be by themselves, ELSIF and ELSE clauses belong to a IF statement. Finally, please note that a END IF clause must complete each and every IF statement.

All *statements* following the THEN clause but before the ELSE, ELSIF, or ENDIF need to be completed with a semi-colon.

Slide 6: Tell Me / Show Me – CASE Expressions

No instructor notes for this slide

Slide 7: Tell Me / Show Me – LOOP Control Structures

No instructor notes for this slide

Slide 8: Tell Me / Show Me – IF Statements

No instructor notes for this slide

Slide 9: Tell Me / Show Me – IF Statements (continued)

No instructor notes for this slide

Slide 10: Tell Me / Show Me – IF Statements (continued)

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.

**Point out that ELSIF (not ELSEIF!) is one word but END IF; is two words.**

Slide 11: Tell Me / Show Me – IF Statements (continued)

No instructor notes for this slide

Slide 12: Tell Me / Show Me – Simple IF Statements

Another example:

**DECLARE**

**v\_myage NUMBER := 15;**

**BEGIN**

**IF (v\_myage < 16) THEN**

**DBMS\_OUTPUT.PUT\_LINE('Cannot drive');**

**END IF;**

**END;**

Statement returns TRUE because v\_myage is not less than 16. Therefore, the control reaches the THEN clause and *Cannot drive* is displayed on the screen.

Slide 13: Tell Me / Show Me – IF THEN ELSE Statement

Ask students what output would be produced by this block.

Answer: I am not a child.

Another example:

```
DECLARE
  v_myage NUMBER := 16;
BEGIN
  IF (v_myage < 16) THEN
    DBMS_OUTPUT.PUT_LINE('Cannot drive');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Can drive');
  END IF;
END;
```

The ELSE clause is added to the code in this example. The condition has changed and, therefore, it now 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 and, therefore, the control moves to the ELSE statement.

Slide 14: Tell Me / Show Me – IF THEN ELSE Clause

Point out that ELSIF (not ELSEIF!) is one word but END IF is two words.

Another example:

```
DECLARE
  v_myage NUMBER := 16;
BEGIN
  IF v_myage < 15 THEN
    DBMS_OUTPUT.PUT_LINE('Cannot drive');
  ELSIF v_myage = 15 THEN
    DBMS_OUTPUT.PUT_LINE('Can drive with older driver in car');
  ELSIF v_myage = 16 THEN
    DBMS_OUTPUT.PUT_LINE('Can drive');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Can drive');
  END IF;
END;
```

In example above, the IF statement now contains multiple ELSIF clauses and an ELSE clause. Notice that the ELSIF clauses add additional conditions. As with the IF, each ELSIF condition is followed by a THEN clause, which is executed if the condition returns TRUE.

Slide 15: Tell Me / Show Me – IF THEN ELSE Clause (continued)

The IF clause now contains multiple ELSIF clauses and an ELSE clause. Observe that the ELSIF clauses can have conditions unlike the ELSE clause. The condition for ELSIF should be followed by the THEN clause that is executed if the condition of the ELSIF clause returns TRUE. When you have multiple ELSIF clauses, if the first condition is FALSE or NULL, the control shifts to the next ELSIF clause.

Slide 16: Tell Me / Show Me – IF Statement With Multiple Expressions

Ask the students what would happen if you were to change the AND to OR in the IF statement.

Answer: the message would print because v\_myfirstname = 'Christopher' is TRUE even though 31 < 11 is FALSE. Condition\_1 OR condition\_2 evaluates to TRUE if at least one of the individual conditions is true.

Another example:

```
DECLARE  
  v_myage    NUMBER    := 16;  
  v_myfirstname VARCHAR2(12) := 'Christopher';  
BEGIN  
  IF v_myfirstname = 'Christopher' AND v_myage < 16 THEN  
    DBMS_OUTPUT.PUT_LINE('Christopher cannot drive');  
  END IF;  
END;
```

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

Slide 17: Tell Me / Show Me – NULL Values in IF Statements

Ask the students what would happen if you were to change

```
IF v_myage < 11  
to  
IF v_myage < 11 OR v_myage is NULL;
```

Answer: “I am a child” would print because the first condition would evaluate to TRUE because the value of v\_myage is NULL.

Slide 18: Tell Me / Show Me – Handling Nulls

No instructor notes for this slide

Slide 19: Tell Me / Show Me – Handling Nulls (continued)

Remind students that NULL means the absence of a value; it does not mean zero. In the slide example, IF x != y becomes (in pseudocode) “If 5 is not equal to something-missing” – this is not TRUE or FALSE, because “something-missing” cannot be compared with anything else.

Slide 20: Tell Me / Show Me – Handling Nulls (continued)

No instructor notes for this slide

Slide 21: Tell Me / Show Me – Guidelines for Using IF Statements

No instructor notes for this slide

Slide 22: Tell Me / Show Me – Terminology

**IF** – Statement that enables PL/SQL to perform actions selectively based on conditions.

**Condition** – An expression with a TRUE or FALSE value that is used to make a decision.

**CASE** – An expression that determines a course of action based on conditions and can be used outside of a PL/SQL block in a SQL statement.

**LOOP Control structures** – Repetition statements that enable you to execute statements in a PL/SQL block repeatedly.

Slide 23: Summary

No instructor notes for this slide

Slide 24: Try It / Solve It

No instructor notes for this slide

## SECTION 4 LESSON 2 – Conditional Control: CASE Statements

### Slide 1: Conditional Control: CASE Statements

No instructor notes for this slide

### Slide 2: What Will I Learn?

No instructor notes for this slide

### Slide 3: Why Learn It?

Nearly everything that can be done with CASE can also be done with IF, but CASE statements and expressions are shorter and easier to read than the equivalent IF statements.

### Slide 4: Tell Me / Show Me – Using a CASE Statement

No instructor notes for this slide

### Slide 5: Tell Me / Show Me – Using a CASE Statement (continued)

Point out that a CASE statement must end with END CASE; **not** END;

### Slide 6: Tell Me / Show Me – CASE Statements: A Second Example

In this CASE statement, the phrase “WHEN 108” means: when v\_mngid **is equal to** 108. You can use CASE statements to test for non-equality conditions such as <, >, BETWEEN ... AND, and so on. These are called **Searched CASE** statements. But they are not much shorter or simpler than their equivalent IF statements.

### Slide 7: Tell Me / Show Me – Using a CASE Expression

No instructor notes for this slide

### Slide 8: Tell Me / Show Me – Using a CASE Expression (continued)

Point out that v\_out\_var := CASE ..... END; is a single PL/SQL statement, ending with END; (not END CASE;)

### Slide 9: Tell Me / Show Me – CASE Expressions

No Instructor notes for this slide

### Slide 10: Tell Me / Show Me – CASE Expressions: A Second Example

#### **CASE Expressions: Example**

In the example in the slide, the CASE expression uses the value in the v\_grade variable as the expression. The CASE expression returns the value of the v\_appraisal variable based on the value of the v\_grade variable.

Ask students what output would be produced if the v\_grade variable were initialized to ‘C’ instead of ‘A’.

Slide 11: Tell Me / Show Me – CASE Expressions: A Third Example

Answer: ‘Same value’ will be displayed, not ‘Middle value’. WHEN v\_in\_var compares the variable with itself, and the condition: v\_in\_var = v\_in\_var is always TRUE (unless v\_in\_var is null).

Slide 12: Tell Me / Show Me – Searched CASE Expressions

Searched CASE expressions are more flexible, allowing non-equality conditions (and compound conditions) to be tested and different variables to be used in different WHEN clauses.

Slide 13: Tell Me / Show Me – Searched CASE Expressions: An Example

Often we have a choice of using a searched or non-searched CASE expression. The example in the slide could be written as:

```
v_appraisal :=  
CASE v_grade  
  WHEN 'A' THEN 'Excellent'  
  WHEN 'B' THEN 'Good'  
  WHEN 'C' THEN 'Good'  
  ELSE 'No such grade'  
END CASE;
```

However, searched CASE expressions are more flexible, allowing non-equality conditions (and compound conditions) to be tested and different variables to be used in different WHEN clauses

Slide 14: Tell Me / Show Me – How are CASE expressions Different from CASE Statements?

CASE expressions are actually functions, which always return exactly one value, just like PL/SQL stored functions (CREATE OR REPLACE FUNCTION ...) which students will learn later in this course.

Slide 15: Tell Me / Show Me – How are CASE expressions Different from CASE Statements?  
(continued)

Point out that a CASE expression only has one terminating semicolon at the END;, while a CASE statement has a semicolon after each executable statement in a WHEN clause.

Slide 16: Tell Me / Show Me – Logic Tables

You can build a simple Boolean condition by combining number, character, or 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:

- 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.

Slide 17: Tell Me / Show Me – 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

Slide 18: Tell Me / Show Me – Terminology

**CASE expression** – An expression that selects a result and returns it into a variable.

**CASE statement** – A block of code that performs actions based on conditional tests.

**Logic Tables** – Shows the results of all possible combinations of two conditions.

Slide 19: Summary

No instructor notes for this slide

Slide 20: Try It / Solve It

No instructor notes for this slide



## SECTION 4 LESSON 3 – Iterative Control: Basic Loops

### Slide 1: Iterative Control: Basic Loops

No instructor notes for this slide

### Slide 2: What Will I Learn?

No instructor notes for this slide

### Slide 3: Why Learn It?

Students will study FOR loops and WHILE loops in the next lesson.

### Slide 4: Tell Me / Show Me –Iterative Control: LOOP Statements

Each loop is structured for a specific purpose. These loops are used to write code to handle all situations (problems). Loops can repeat one statement, a group of statements, and/or a block. Loops have a scope and loop variables have a life.

### Slide 5: Tell Me / Show Me – Basic Loop

No instructor notes for this slide

### Slide 6: Tell Me / Show Me – Basic Loop (continued)

No instructor notes for this slide

### Slide 7: Tell Me / Show Me – Basic Loop (continued)

Ask students what would happen if no EXIT statement were included. Answer: the loop would execute 99 times, inserting 99 rows into the table. The 100th iteration would try to increment the counter to 100 and cause an error because V\_COUNTER is declared as NUMBER(2).

### Slide 8: Tell Me / Show Me – Basic Loops: The EXIT Statement

Students may not have seen the POWER function before. Demonstrate it briefly by:

```
SELECT POWER(3,2), POWER(3,3), POWER(3,4)
FROM dual;
```

### Slide 9: Tell Me / Show Me – Basic Loops: The EXIT Statement (continued)

You can issue EXIT either as an action within an IF statement or as a stand-alone statement within the loop.

### Slide 10: Tell Me / Show Me – Basic Loops: The EXIT WHEN Statement (continued)

EXIT WHEN v\_counter > 10; -- Is logically identical to (but is much neater than):

```
IF v_counter > 10 THEN
    EXIT;
END IF;
```

**The EXIT WHEN statement is a conditional expression that gives clear direction(s) to the code. It is the better way to check for the iterations to continue or the end of the iterations.**

*Slide 11: Tell Me / Show Me – Terminology*

**Basic (Infinite) Loop** – Encloses a sequence of statements between the keywords LOOP and END LOOP and must execute at least once.

**EXIT** – Statement to terminate a loop.

*Slide 12: Summary*

No instructor notes for this slide

*Slide 13: Try It / Solve It*

No instructor notes for this slide

## SECTION 4 LESSON 4 – Iterative Control: WHILE and FOR Loops

### Slide 1: Iterative Control: WHILE and FOR Loops

No instructor notes for this slide

### Slide 2: What Will I Learn?

No instructor notes for this slide

### Slide 3: Why Learn It?

No instructor notes for this slide

### Slide 4: Tell Me / Show Me – WHILE Loops

The WHILE loop is a conditional loop that continues to execute as long as the condition in the loop control evaluates to TRUE. Since the WHILE loop depends on a condition and is not fixed, use the WHILE loop when you don't know in advance the number of times a loop must execute.

### Slide 5: Tell Me / Show Me – WHILE Loops (continued)

In some programming languages (but not PL/SQL) you can also code an UNTIL loop, which continues to loop until the controlling condition becomes TRUE. This is not needed in PL/SQL because we can simply code:

```
WHILE NOT condition LOOP
```

```
...
```

```
END LOOP
```

### Slide 6: Tell Me / Show Me – WHILE Loops (continued)

No instructor notes for this slide

### Slide 7: Tell Me / Show Me – WHILE Loops (continued)

No instructor notes for this slide

### Slide 8: Tell Me / Show Me – FOR Loops

No instructor notes for this slide

### Slide 9: Tell Me / Show Me – FOR Loops (continued)

For example:

1. FOR i IN 1..5 LOOP ...; -- successive values 1,2,3,4,5
2. FOR i IN REVERSE 1..5 LOOP ...; -- successive values 5,4,3,2,1
3. FOR i IN REVERSE 5..1 LOOP ...; -- will cause an error.

### Slide 10: Tell Me / Show Me – FOR Loops (continued)

No instructor notes for this slide

Slide 11: Tell Me / Show Me – FOR Loops (continued)

Students may ask: what if we want to increase (or decrease) the loop counter by 2 each time instead of by 1, giving successive values 2,4,6 instead of 1,2,3 ?

We can do this by declaring a second variable explicitly and setting it to twice the loop counter. For example:

```
DECLARE
    v_counter    INTEGER;
    ...
BEGIN
    FOR i IN 1..3 LOOP
        v_counter := i * 2;
        ...
    END LOOP;
```

Slide 12: Tell Me / Show Me – FOR Loops: Guidelines

The scope of the counter is automatically restricted to the loop alone.

A FOR loop is used within the code when the beginning and ending value of the loop is known. FOR loops are the staple loop of programming. It is easy to use and understand.

Slide 13: Tell Me / Show Me – FOR Loops

No instructor notes for this slide

Slide 14: Tell Me / Show Me – Guidelines For Using 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, then 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.

**Note:** All loops allow the use an EXIT WHEN condition allowing the loop to terminate before the WHILE condition is false or the upper bound of the FOR loop is reached

Remember that all the variables used within the loop blocks have a scope. The specific loop needed is determined by the problem being solved.

Slide 15: Tell Me / Show Me – Terminology

**WHILE Loop** – Repeats a sequence of statements until the controlling condition is no longer TRUE.

**FOR Loop** – Repeats a sequence of statements until a set number of iterations are fulfilled.

Slide 16: Summary

No instructor notes for this slide

Slide 17: Try It / Solve It

No instructor notes for this slide

## SECTION 4 LESSON 5 – Iterative Control: Nested Loops

### Slide 1: Iterative Control: Nested Loops

No instructor notes for this slide

### Slide 2: What Will I Learn?

No instructor notes for this slide

### Slide 3: Why Learn It?

No instructor notes for this slide

### Slide 4: Tell Me / Show Me – Nested Loops

Consider entering and executing this block so that students can clearly see the successive iterations.

The slide example shows a FOR loop nested inside another FOR loop. But (for example) we could nest a WHILE loop inside a FOR loop; or a FOR loop inside a basic loop; or .... All combinations are allowed.

### Slide 5: Tell Me / Show Me – Nested Loops (continued)

Explain that when V\_INNER\_DONE = 'YES', PL/SQL exits the inner loop but the outer loop continues executing. What if we want to exit the outer loop (ie both loops) while still within the inner loop?

### Slide 6: Tell Me / Show Me – Loop Labels

Loop labels can help to improve readability even when they are not needed. But they are needed only if we want to exit an outer loop from within an inner loop.

### Slide 7: Tell Me / Show Me – Loops Labels (continued)

No instructor notes for this slide

### Slide 8: Tell Me / Show Me – Loop Labels (continued)

The loop labels are not strictly needed in this example, but they do make the code more readable.

### Slide 9: Tell Me / Show Me – Nested Loops and Labels

In this example, the outer\_loop label is necessary because we reference it in an EXIT statement within the inner loop. The inner\_loop label is not strictly necessary. The label names are included after the END LOOP statements for clarity.

### Slide 10: Summary

No instructor notes for this slide

### Slide 11: Try It / Solve It

No instructor notes for this slide

## PRACTICE SOLUTIONS

---

### SECTION 1 LESSON 1 – Conditional Control: IF Statements .

#### Terminology

1. **IF** Statement that enables PL/SQL to perform actions selectively based on conditions.
2. **LOOP** Control structures – Repetition statements that enable you to execute statements in a PL/SQL block repeatedly.
3. **Condition** An expression with a TRUE or FALSE value that is used to make a decision.
4. **CASE** An expression that determines a course of action based on conditions and can be used outside a PL/SQL block in a SQL statement.

#### Try It/Solve It

1. What is the purpose of a conditional control structure in PL/SQL?

**A conditional control structure is used to change the logical flow of statements within a PL/SQL block.**

2. List the three categories of control structures in PL/SQL.

**IF statements**

**CASE expressions**

**LOOP control structures**

3. List the keywords that can be part of an IF statement.

**IF, THEN, ELSE, ELSIF, and END IF**

4. List the keywords that are a required part of an IF statement.

**IF, THEN, and END IF**

5. Write a PL/SQL block to find the population of a given country in the wf\_countries table. Display a message indicating whether the population is greater than or less than 1 billion (1,000,000,000). Test your block twice using India (country\_id=91) and United Kingdom (country\_id=44). India's population should be greater than 1 billion, while United Kingdom's should be less than 1 billion.

```
DECLARE
  v_country_id  wf_countries.country_id%TYPE := 91; -- or 44
  v_population  wf_countries.population%TYPE;
BEGIN
  SELECT population INTO v_population
    FROM wf_countries
   WHERE country_id = v_country_id;
  IF v_population > 1000000000
  THEN
    DBMS_OUTPUT.PUT_LINE('Greater than 1 billion');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Less than 1 billion');
  END IF;
END;
```

6. Modify the code from the previous exercise so that it handles all the following cases:
- A. Population is greater than 1 billion.
  - B. Population is greater than 0.
  - C. Population is 0.
  - D. Population is null. (Display: No data for this country.)

Run your script using the following countries:

China (country\_id=86): Population is greater than 1 billion.

United Kingdom (country\_id=44): Population is greater than 0.

Antarctica (country\_id=672): Population is 0.

Europa Island (country\_id=15): There is no data for this country.



```

DECLARE
  v_country_id    wf_countries.country_id%TYPE
                := <Enter various values.>;
  v_population    wf_countries.population%TYPE;
BEGIN
  SELECT population INTO v_population
    FROM wf_countries
    WHERE country_id = v_country_id;
  IF v_population > 1000000000
  THEN
    DBMS_OUTPUT.PUT_LINE('Greater than 1 billion');
  ELSIF v_population > 0
  THEN
    DBMS_OUTPUT.PUT_LINE('Greater than 0');
  ELSIF v_population = 0
  THEN
    DBMS_OUTPUT.PUT_LINE('Population is 0');
  ELSIF v_population IS NULL
  THEN
    DBMS_OUTPUT.PUT_LINE('No data for this country');
  END IF;
END;

```

7. Examine the following code:

```

DECLARE
  v_country_id    wf_countries.country_name%TYPE := <a value>;
  v_ind_date      wf_countries.date_of_independence%TYPE;
  v_natl_holiday wf_countries.national_holiday_date%TYPE;
BEGIN
  SELECT date_of_independence, national_holiday_date
    INTO v_ind_date, v_natl_holiday
    FROM wf_countries
    WHERE country_id=v_country_id;
  IF v_ind_date IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('A');
  ELSIF v_natl_holiday IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('B');
  ELSIF v_natl_holiday IS NULL AND v_ind_date IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('C');
  END IF;
END;

```

A. What would print if the country has an independence date equaling NULL and a national holiday date equaling NULL?

**C**

- B. What would print if the country has an independence date equaling NULL and a national holiday date containing a value?

**B**

- C. What would print if the country has an independence date equaling a value and a national holiday date equaling NULL?

**A**

- D. Run a SQL script against the WF\_COUNTRIES table to determine whether the following countries have independence dates or national holiday dates, or both.

Country	Country_ID	Independence Date? Y/N	National Holiday Date? Y/N	Output should be
United States	1	4-Jul-1776	4-Jul	A
Iraq	964	28-Jun-2004		A
Antarctica	672			C
Spain	34		12-Oct	B

```
SELECT country_name, country_id, date_of_independence,  
       national_holiday_date  
FROM wf_countries  
WHERE country_id IN (1,964,672,34);
```

- E. Finally, run the above PL/SQL code using each of the above country ids as input. Check whether your output answers are correct.

8. Examine the following code. What output do you think it will produce?

```
DECLARE  
  v_num1  NUMBER(3) := 123;  
  v_num2  NUMBER;  
BEGIN  
  IF v_num1 <> v_num2 THEN  
    DBMS_OUTPUT.PUT_LINE('The two numbers are not equal');  
  ELSE  
    DBMS_OUTPUT.PUT_LINE('The two numbers are equal');  
  END IF;  
END;
```

Enter and run the script to check if your answer was correct.

Many students will opt for “The two numbers are not equal” because v\_num2 is not equal to 123, in fact it is NULL. But comparing a NULL with anything always yields NULL, therefore the condition v\_num1 <> v\_num2 is not TRUE.

Extension Exercise

1. Write a PL/SQL block to accept a year and check whether it is a leap year. For example, if the year entered is 1990, the output should be “1990 is not a leap year.”

Hint: The year should be exactly divisible by 4 but not divisible by 100, or it should be divisible by 400.

Test your solution with the following years:

1990	Not a leap year
2000	Leap year
1996	Leap year
1900	Not a leap year
1992	Leap year
1884	Leap year

**DECLARE**

**v\_year**     **NUMBER(4) := <enter a year>;**

**v\_remainder1** **NUMBER(5,2);**

**v\_remainder2** **NUMBER(5,2);**

**v\_remainder3** **NUMBER(5,2);**

**BEGIN**

**v\_remainder1 := MOD(v\_year,4);**

**v\_remainder2 := MOD(v\_year,100);**

**v\_remainder3 := MOD(v\_year,400);**

**IF ((v\_remainder1 = 0 AND v\_remainder2 <> 0 )**

**OR v\_remainder3 = 0) THEN**

**DBMS\_OUTPUT.PUT\_LINE(v\_year || ' is a leap year');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE(v\_year || ' is not a leap year');**

**END IF;**

**END;**

## SECTION 4 LESSON 2 – Conditional Control: Case Statements

### Terminology

1. **CASE expression** \_\_\_\_\_ An expression that selects a result and returns it into a variable.
2. **Logic Tables** \_\_\_\_\_ Shows the results of all possible combinations of two conditions.
3. **CASE statement** \_\_\_\_\_ A block of code that performs actions based on conditional tests.

### Try It/Solve It

1. Write a PL/SQL block:
  - A. To find the number of airports from the wf\_countries table for a supplied country\_name. Based on this number, display a customized message as follows:

# Airports	Message
0–100	There are 100 or fewer airports.
101–1,000	There are between 101 and 1,000 airports.
1001–1,0000	There are between 1,001 and 10,000 airports.
> 10,000	There are more than 10,000 airports.
No value in database	The number of airports is not available for this country.

Use a CASE statement. You can use the following code to get started:

```
DECLARE
  v_country_name wf_countries.country_name%TYPE :=
    '<country_name>';
  v_airports    wf_countries.airports%TYPE;
BEGIN
  SELECT airports INTO v_airports
    FROM wf_countries
   WHERE country_name = v_country_name;

  CASE
    WHEN ...
  END CASE;

END;
```

```

DECLARE
  v_country_name wf_countries.country_name%TYPE :=
    '<country_name>';
  v_airports    wf_countries.airports%TYPE;
BEGIN
  SELECT airports INTO v_airports
    FROM wf_countries
    WHERE country_name = v_country_name;

  CASE
    WHEN v_airports < 100 THEN
      DBMS_OUTPUT.PUT_LINE ('There 100 or fewer airports');
    WHEN v_airports < 1001 THEN
      DBMS_OUTPUT.PUT_LINE ('There are between 101 and 1000
        airports');
    WHEN v_airports < 10001 THEN
      DBMS_OUTPUT.PUT_LINE ('There are between 1001 and 10000
        airports');
    WHEN v_airports > 10000 THEN
      DBMS_OUTPUT.PUT_LINE ('There are more than 10000
        airports');
    ELSE
      DBMS_OUTPUT.PUT_LINE ('The number of airports is not
        listed for this country. ');
  END CASE;

END;

```

B. Test your code against the following data:

	No value	0–100	101–1,000	1,001–10,000	> 10,000
Canada				X	
Malaysia			X		
United States of America					X
Romania		X			
Japan			X		
Mongolia		X			
Navassa Island	X				

2. Write a PL/SQL block:

- A. To find the amount of coastline for a supplied country name. Use the wf\_countries table. Based on the amount of coastline for the country, display a customized message as follows:

Length of Coastline	Message
0	no coastline
<1000	a small coastline
<10000	a mid-range coastline
All other values	a large coastline

Use a CASE expression. Use the following code to get started:

```
DECLARE
    v_country_name      wf_countries.country_name%TYPE :=
                        '<country name>';
    v_coastline          wf_countries.coastline %TYPE;
    v_coastline_description VARCHAR2(50);

BEGIN
    SELECT coastline INTO v_coastline
    FROM wf_countries
    WHERE country_name = v_country_name;

    v_coastline_description :=
        CASE ...
        END;

    DBMS_OUTPUT.PUT_LINE('Country ' || v_country_name ||
        ' has ' || v_coastline_description);
END;
```

```

DECLARE
  v_country_name      wf_countries.country_name%TYPE :=
                        '<country name>';
  v_coastline         wf_countries.coastline %TYPE;
  v_coastline_description VARCHAR2(50);

BEGIN
  SELECT coastline INTO v_coastline
    FROM wf_countries
    WHERE country_name = v_country_name;

  v_coastline_description :=
  CASE
    WHEN v_coastline = 0
      THEN 'no coastline'
    WHEN v_coastline < 1000
      THEN 'a small coastline'
    WHEN v_coastline < 10000 THEN
      'a mid-range coastline'
    ELSE
      'a large coastline'
  END;

  DBMS_OUTPUT.PUT_LINE('Country ' || v_country_name ||
    ' has ' || v_coastline_description);
END;

```

B. Test your code against the following data:

	No coastline	Small coastline	Mid-range coastline	Large coastline
Canada				X
Jamaica			X	
Mongolia	X			
Ukraine			X	
Japan				X
Grenada		X		

3. Use a CASE statement:

A. Write a PL/SQL block to select the number of countries using a supplied currency name. If the number of countries is greater than 20, display “More than 20 countries”. If the number of countries is between 10 and 20, display “Between 10 and 20 countries”. If the number of countries is less than 10, display “Fewer than 10 countries”. Use a CASE statement.

B. Test your code using the following data:

	Fewer than 10 countries	Between 10 and 20 countries	More than 20 countries
US Dollar		X	
Swiss franc	X		
Euro			X

**DECLARE**

```
v_currency_name    wf_currencies.currency_name%TYPE :=  
                    '<currency name>';  
v_no_of_countries  NUMBER(3);
```

**BEGIN**

```
SELECT currency_name, count(*)  
  INTO v_currency_name, v_no_of_countries  
  FROM wf_countries c, wf_currencies cu  
  WHERE c.currency_code = cu.currency_code  
  AND currency_name = v_currency_name  
  GROUP BY currency_name;
```

**CASE**

```
  WHEN v_no_of_countries > 20  
    THEN DBMS_OUTPUT.PUT_LINE ('More than 20 countries');  
  WHEN v_no_of_countries BETWEEN 10 AND 20  
    THEN DBMS_OUTPUT.PUT_LINE ('Between 10 and 20 countries');  
  WHEN v_no_of_countries < 10  
    THEN DBMS_OUTPUT.PUT_LINE ('Fewer than 10 countries');  
  END CASE;
```

**END;**



4. Examine the following code.

A. What do you think the output will be? Test your theory by running the code in Application Express.

```
DECLARE
  x BOOLEAN := FALSE;
  y BOOLEAN;
  v_color VARCHAR(20) := 'Red';
BEGIN
  IF (x OR y)
    THEN v_color := 'White';
  ELSE
    v_color := 'Black';
  END IF;
  DBMS_OUTPUT.PUT_LINE(v_color);
END;
```

**Black**

B. Change the declarations to x and y as follows. What do you think the output will be? Test your theory by running the code again.

```
x BOOLEAN ;
y BOOLEAN ;
```

**Black**

C. Change the declarations to x and y as follows. What do you think the output will be? Test your theory by running the code again.

```
x BOOLEAN := TRUE;
y BOOLEAN := TRUE;
```

**White**

D. Experiment with changing the OR condition to AND.

## SECTION 4 LESSON 3 – Iterative Control: Basic Loops

### Terminology

1. Basic Loop Encloses a sequence of statements between the keywords LOOP and END LOOP and must execute at least once.
2. EXIT Statement to terminate a loop.

### Try It/Solve It

1. What purpose does a loop serve in PL/SQL?

**A loop is used to execute statements repeatedly.**

2. List the types of loops in PL/SQL.

**PL/SQL has basic loops, FOR loops, and WHILE loops.**

3. What statement is used to explicitly end a loop?

### **The EXIT statement**

4. Write a PL/SQL block to display the country\_id and country\_name values from the WF\_COUNTRIES table for country\_id whose values range from 1 through 3. Use a basic loop. Increment a variable from 1 through 3. Use an IF statement to test your variable and EXIT the loop after you have displayed the first 3 countries.

### **DECLARE**

```
v_country_id wf_countries.country_id%TYPE := 1;  
v_country_name wf_countries.country_name%TYPE;
```

### **BEGIN**

#### **LOOP**

```
SELECT country_name INTO v_country_name  
FROM wf_countries  
WHERE country_id = v_country_id;  
DBMS_OUTPUT.PUT_LINE ('Country ID '|| v_country_id||' is '||  
v_country_name );  
v_country_id := v_country_id + 1;  
IF v_country_id > 3  
THEN EXIT;  
END IF;  
END LOOP;  
END;
```

5. Modify your solution to question 4 above, replacing the IF statement with an EXIT....WHEN statement.

**DECLARE**

```
v_country_id wf_countries.country_id%TYPE := 1;  
v_country_name wf_countries.country_name%TYPE;
```

**BEGIN**

**LOOP**

```
SELECT country_name INTO v_country_name  
FROM wf_countries  
WHERE country_id = v_country_id;  
DBMS_OUTPUT.PUT_LINE ('Country ID '|| v_country_id|| ' is '||  
v_country_name );  
v_country_id := v_country_id + 1;  
EXIT WHEN v_country_id > 3;  
END LOOP;
```

**END;**

6. Create a Messages Table and insert several rows into it:

- A. To create the messages table.

```
DROP TABLE messages;  
CREATE TABLE messages (results NUMBER(2));
```

- B. Write a PL/SQL block to insert numbers into the messages table. Insert the numbers 1 through 10, excluding 6 and 8.

**DECLARE**

```
v_number NUMBER(2) := 1;
```

**BEGIN**

**LOOP**

```
IF v_number = 6 OR v_number = 8 THEN
```

```
NULL;
```

```
ELSE
```

```
INSERT INTO messages(results)
```

```
VALUES (v_number);
```

```
END IF;
```

```
v_number := v_number + 1;
```

```
EXIT WHEN v_number > 10;
```

```
END LOOP;
```

```
COMMIT;
```

**END;**

C. Execute a SELECT statement to verify that your PL/SQL block worked.

**SELECT \* FROM messages;**

RESULTS
1
2
3
4
5
7
9
10

8 rows returned in 0.01 seconds

## SECTION 4 LESSON 4 – Iterative Control: WHILE and FOR Loops

### Terminology

1. WHILE Loop Repeats a sequence of statements until the controlling condition is no longer TRUE.
2. FOR Loop Repeats a sequence of statements until a set number of iterations have been completed.

### Try It/Solve It

1. Write a PL/SQL block to display the country\_id and country\_name values from the WF\_COUNTRIES table for country\_id whose values range from 51 through 55. Use a WHILE loop. Increment a variable from 51 through 55. Test your variable to see when it reaches 55. EXIT the loop after you have displayed the 5 countries.

#### DECLARE

```
v_country_id wf_countries.country_id%TYPE := 51;  
v_country_name wf_countries.country_name%TYPE;
```

#### BEGIN

```
WHILE v_country_id <= 55 LOOP  
  SELECT country_name INTO v_country_name  
    FROM wf_countries  
   WHERE country_id = v_country_id;  
  DBMS_OUTPUT.PUT_LINE ('Country ID ' || v_country_id ||  
    ' is ' || v_country_name );  
  v_country_id := v_country_id + 1;  
END LOOP;  
END;
```

2. Write a PL/SQL block to display the country\_id and country\_name values from the WF\_COUNTRIES table for country\_id whose values range from 51 through 55 *in the reverse order*. Use a FOR loop.

#### DECLARE

```
v_country_name wf_countries.country_name%TYPE;
```

#### BEGIN

```
FOR loop_counter IN REVERSE 51..55 LOOP  
  SELECT country_name INTO v_country_name  
    FROM wf_countries  
   WHERE country_id = loop_counter;  
  DBMS_OUTPUT.PUT_LINE ('Country ID ' || loop_counter ||  
    ' is ' || v_country_name );  
END LOOP;  
END;
```

### 3. new\_emp table

A. Execute the following statements to build a new\_emp table.

```
DROP TABLE new_emps;  
CREATE TABLE new_emps AS SELECT * FROM employees;  
ALTER TABLE new_emps ADD stars VARCHAR2(50);
```

B. Create a PL/SQL block that inserts an asterisk in the stars column for every whole \$1000 of an employee's salary. For example, if an employee has salary of \$7800, the string "\*\*\*\*\*" would be inserted. Use the following code as a starting point.

```
DECLARE  
    v_empno      new_emps.employee_id%TYPE := <employee_id>;  
    v_asterisk    new_emps.stars%TYPE := NULL;  
    v_sal_in_thousands new_emps.salary%TYPE;  
BEGIN  
    SELECT NVL(TRUNC(salary/1000), 0) INTO v_sal_in_thousands  
        FROM new_emps WHERE employee_id = v_empno;  
  
    FOR ...  
  
    UPDATE new_emps SET stars = v_asterisk  
        WHERE employee_id = v_empno;  
  
END;
```

```
DECLARE  
    v_empno      new_emps.employee_id%TYPE := <employee_id>;  
    v_asterisk    new_emps.stars%TYPE := NULL;  
    v_sal_in_thousands new_emps.salary%TYPE;  
BEGIN  
    SELECT NVL(TRUNC(salary/1000), 0) INTO v_sal_in_thousands  
        FROM new_emps WHERE employee_id = v_empno;  
    FOR i IN 1..v_sal_in_thousands LOOP  
        v_asterisk := v_asterisk || '*';  
    END LOOP;  
    UPDATE new_emps SET stars = v_asterisk  
        WHERE employee_id = v_empno;  
    COMMIT;  
END;
```

- C. Test your code using employee\_ids 124 and 142.
- D. Execute a SQL statement to check your results.

```
SELECT employee_id,salary, stars  
FROM new_emps  
WHERE employee_id IN (124,142);
```

## SECTION 4 LESSON 5 – Iterative Control: Nested Loops

### Terminology

No new vocabulary for this lesson.

### Try It/Solve It

1. Write a PL/SQL block to produce a list of available vehicle license plate numbers. These numbers must be in the following format: NN-MMM, where NN is between 60 and 65, and MMM is between 100 and 110. Use nested FOR loops. The outer loop should choose numbers between 60 and 65. The inner loop should choose numbers between 100 and 110, and concatenate the two numbers together.

**DECLARE**

**v\_license\_number CHAR(6);**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE ('Available License Tags are: ');**

**FOR first\_number\_counter IN 60..65 LOOP**

**FOR second\_number\_counter IN 100..110 LOOP**

**v\_license\_number := first\_number\_counter || '-' ||  
second\_number\_counter;**

**DBMS\_OUTPUT.PUT\_LINE (v\_license\_number);**

**END LOOP;**

**END LOOP;**

**END;**



2. Modify your block from question 1 to calculate the sum of the two numbers on each iteration of the inner loop (for example, 62-107 sums to 169), and exit from the OUTER loop if the sum of the two numbers is greater than 172. Use loop labels. Test your modified code.

```
DECLARE
  v_license_number CHAR(6);

BEGIN
  DBMS_OUTPUT.PUT_LINE ('Available License Tags are: ');
  <<outer_loop>>
  FOR first_number_counter IN 60..65 LOOP
    <<inner_loop>>
    FOR second_number_counter IN 100..110 LOOP
      v_license_number := first_number_counter || '-' ||
                          second_number_counter;
      DBMS_OUTPUT.PUT_LINE (v_license_number);
      EXIT outer_loop WHEN
        first_number_counter + second_number_counter > 172;
    END LOOP inner_loop;
  END LOOP outer_loop;
END;
```

The last number outputted should be: 63-110.