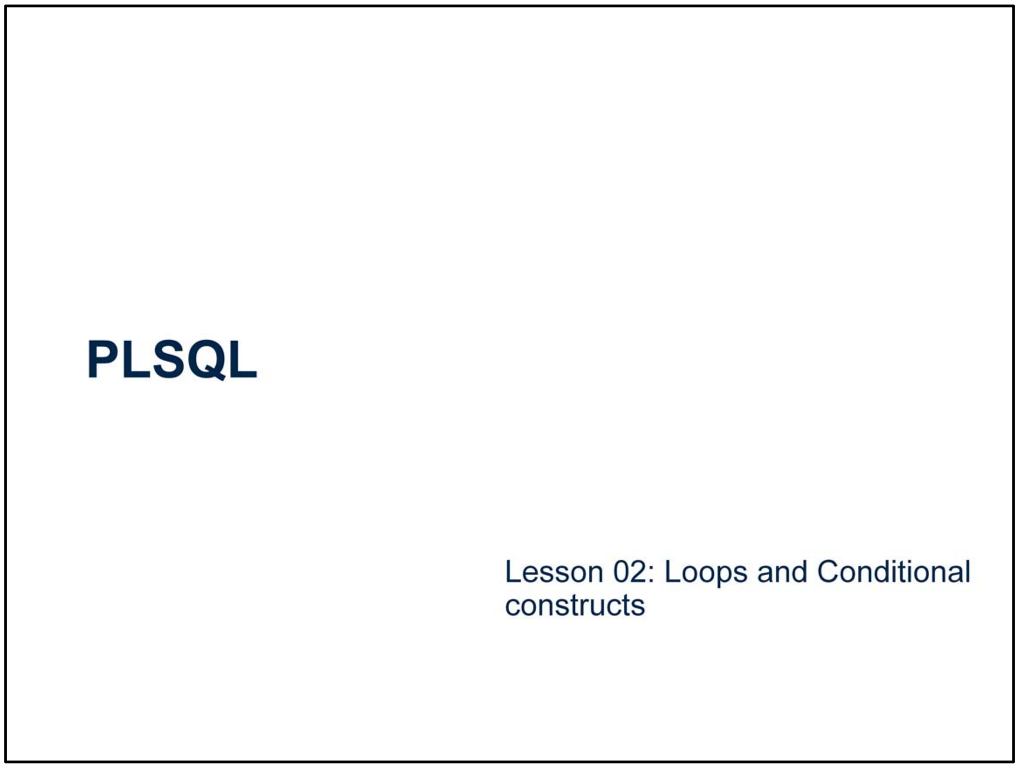
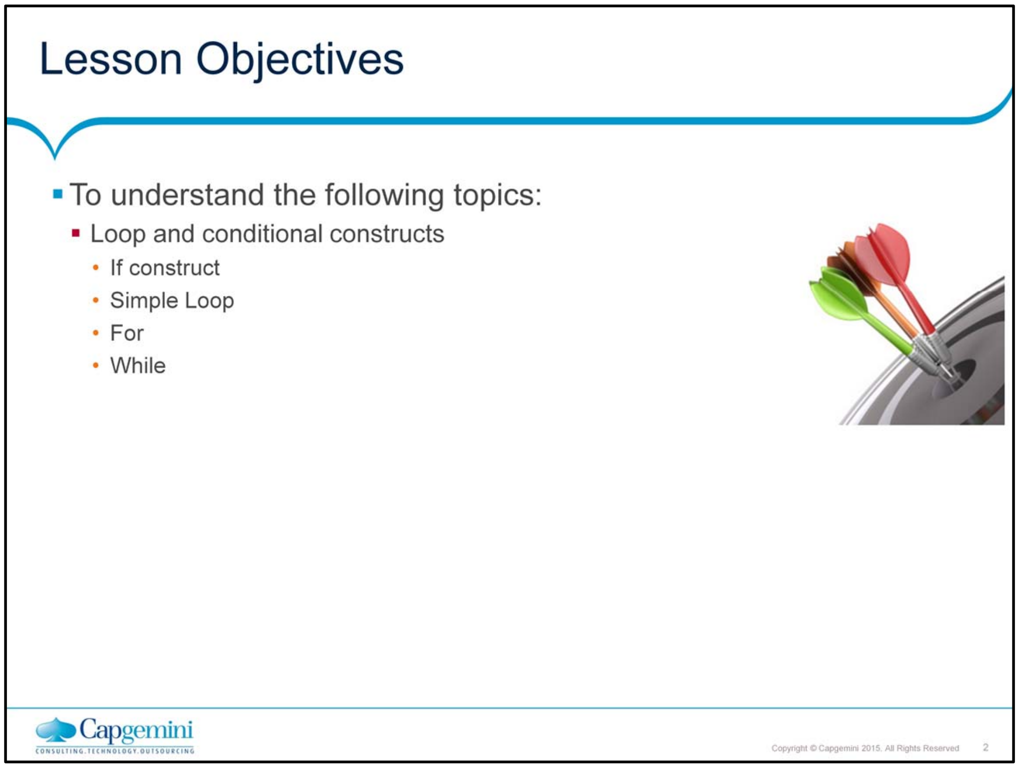


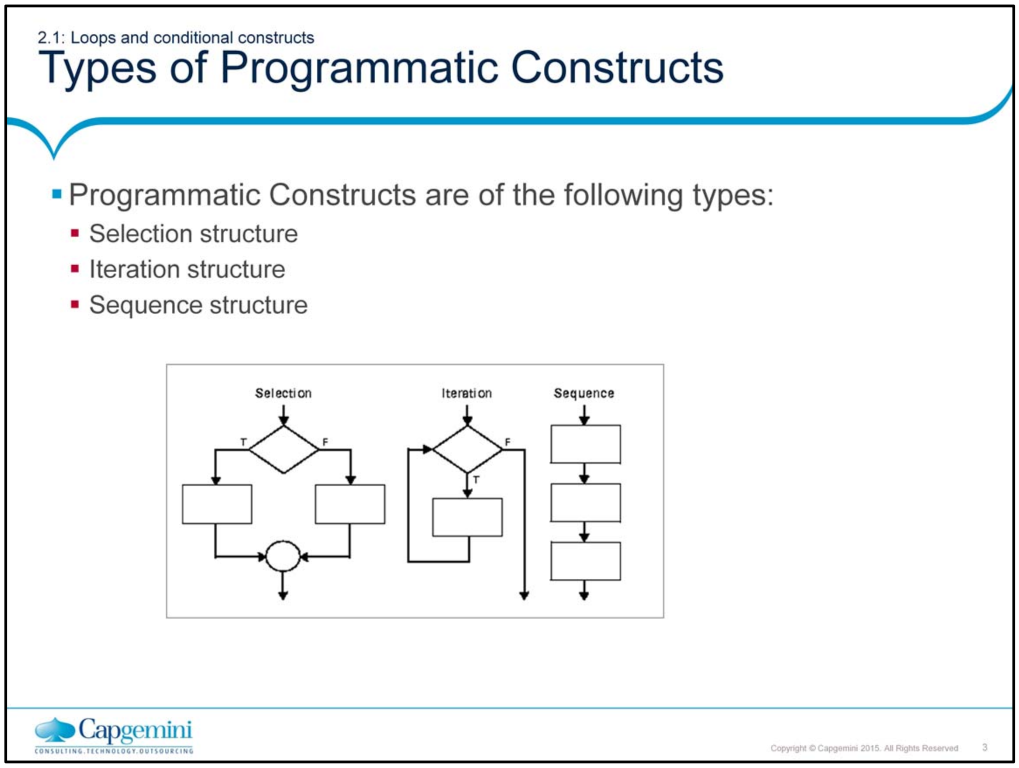
PLSQL

Lesson 02: Loops and Conditional constructs



Lesson Objectives

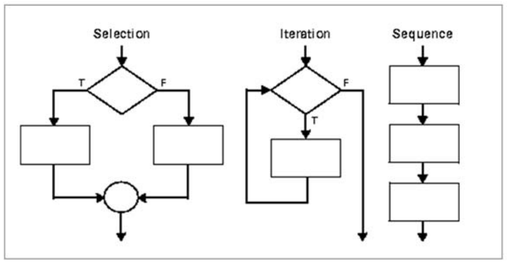
- To understand the following topics:
  - Loop and conditional constructs
    - If construct
    - Simple Loop
    - For
    - While



2.1: Loops and conditional constructs

Types of Programmatic Constructs

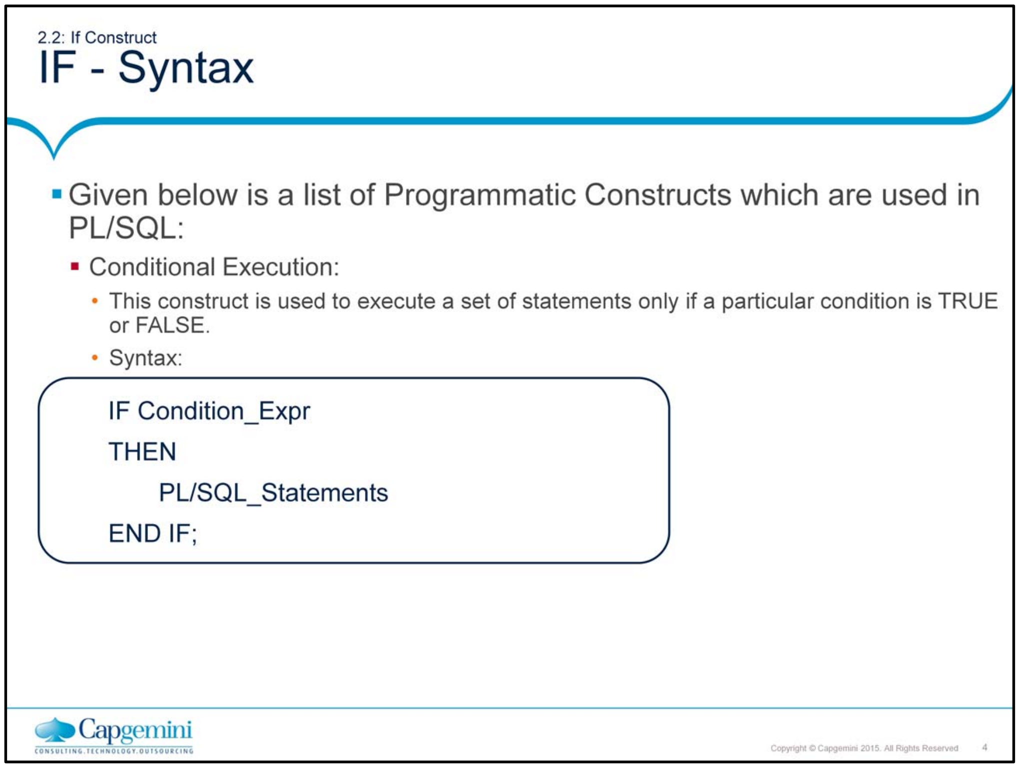
- Programmatic Constructs are of the following types:
  - Selection structure
  - Iteration structure
  - Sequence structure



Programming Constructs:
The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is TRUE or FALSE.

> A condition is any variable or expression that returns a Boolean value (TRUE or FALSE).

The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.
The sequence structure simply executes a sequence of statements in the order in which they occur.



2.2: If Construct

IF - Syntax

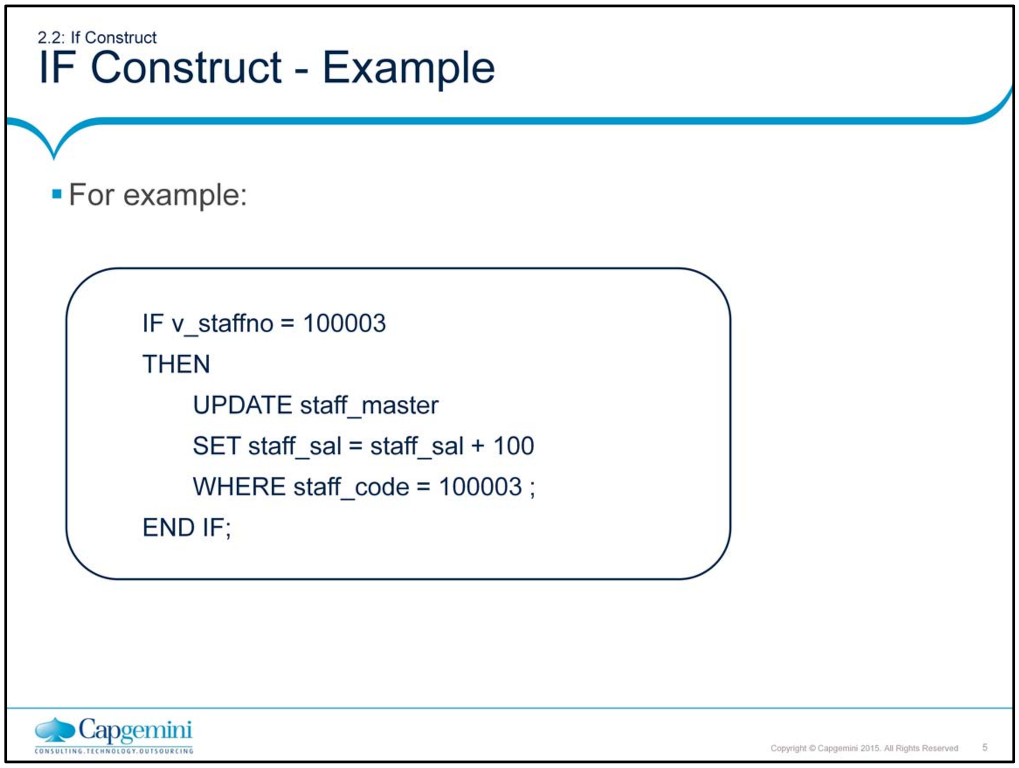
- Given below is a list of Programmatic Constructs which are used in PL/SQL:
  - Conditional Execution:
    - This construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.
    - Syntax:

```
IF Condition_Expr
THEN
      PL/SQL_Statements
END IF;
```

Programmatic Constructs (contd.)
Conditional Execution:
Conditional execution is of the following type:

IF-THEN-END IF
IF-THEN-ELSE-END IF
IF-THEN-ELSIF-END IF

Conditional Execution construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.



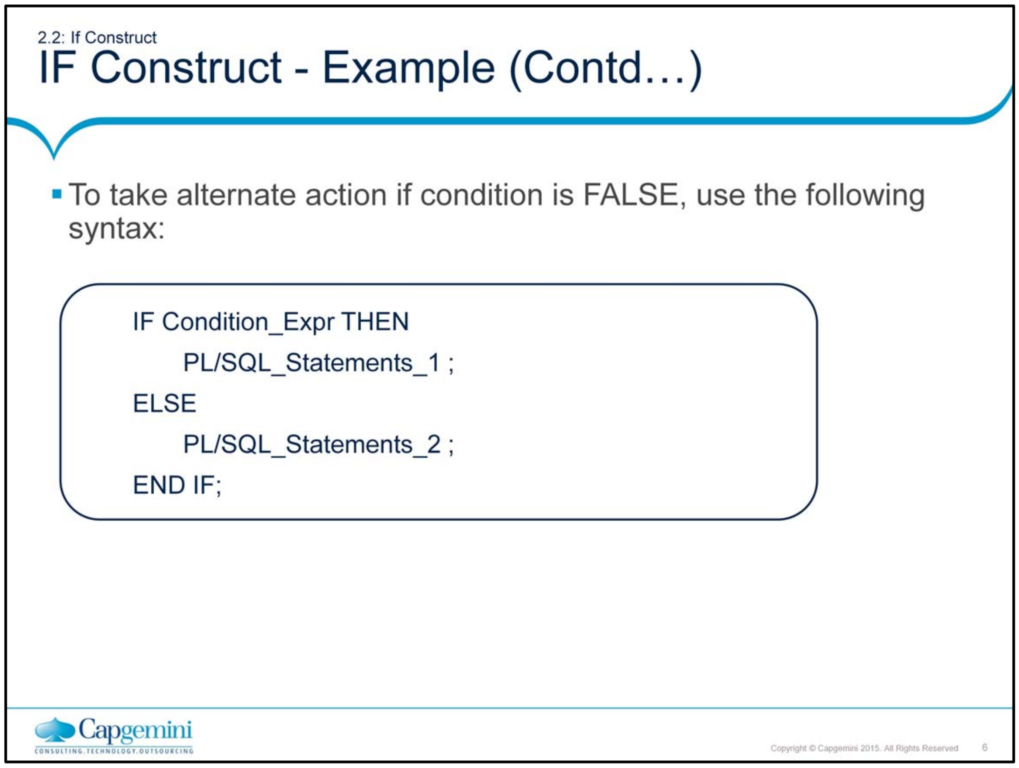
2.2: If Construct

IF Construct - Example

- For example:

```
IF v_staffno = 100003
THEN
      UPDATE staff_master
      SET staff_sal = staff_sal + 100
      WHERE staff_code = 100003 ;
END IF;
```

Programmatic Constructs (contd.)
Conditional Execution (contd.):
As shown in the example in the slide, when the condition evaluates to TRUE, the PL/SQL statements are executed, otherwise the statement following END IF is executed.
UPDATE statement is executed only if value of v_staffno variable equals 100003.
PL/SQL allows many variations for the IF – END IF construct.



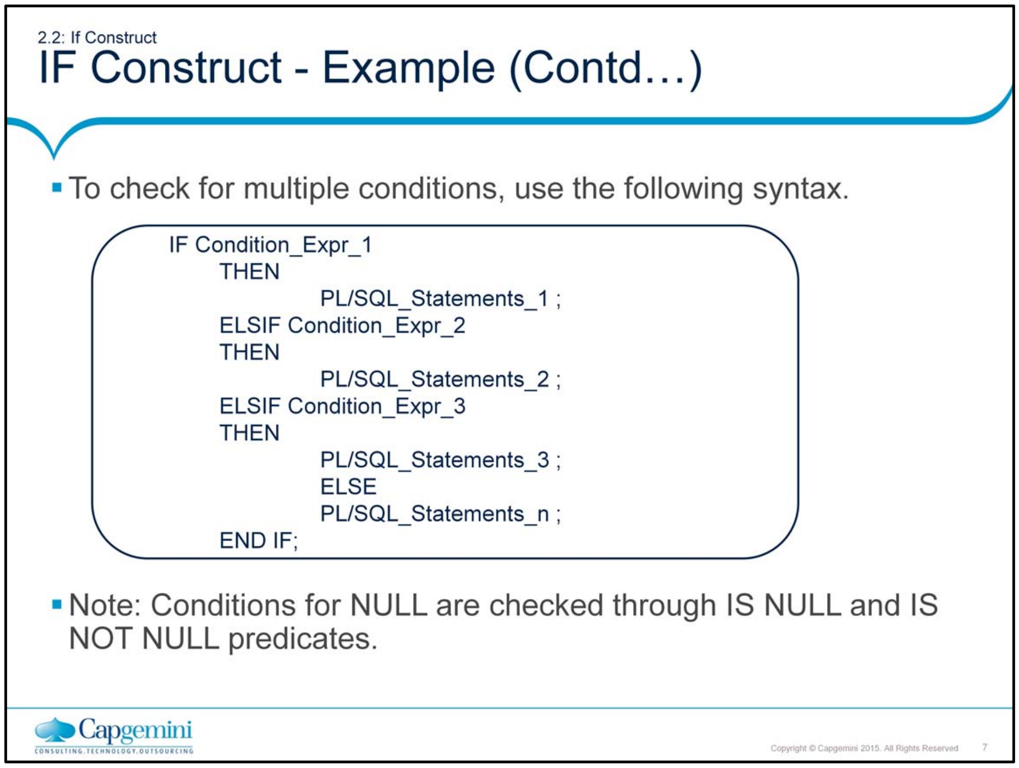
2.2: If Construct

IF Construct - Example (Contd…)

- To take alternate action if condition is FALSE, use the following syntax:

```
IF Condition_Expr THEN
    PL/SQL_Statements_1 ;
ELSE
    PL/SQL_Statements_2 ;
END IF;
```

Programmatic Constructs (contd.)
Conditional Execution (contd.):
Note:
When the condition evaluates to TRUE, the PL/SQL_Statements_1 is executed, otherwise PL/SQL_Statements_2 is executed.
The above syntax checks only one condition, namely Condition_Expr.



2.2: If Construct

IF Construct - Example (Contd…)

- To check for multiple conditions, use the following syntax.

```
IF Condition_Expr_1
    THEN
            PL/SQL_Statements_1 ;
    ELSIF Condition_Expr_2
    THEN
            PL/SQL_Statements_2 ;
    ELSIF Condition_Expr_3
    THEN
            PL/SQL_Statements_3 ;
            ELSE
            PL/SQL_Statements_n ;
    END IF;
```

- Note: Conditions for NULL are checked through IS NULL and IS NOT NULL predicates.

Programmatic Constructs (contd.)
Conditional Execution (contd.):

```
DECLARE
    D VARCHAR2(3): = TO_CHAR(SYSDATE, ‘DY’)
BEGIN
    IF  D= ‘SAT’  THEN
        DBMS_OUTPUT.PUT_LINE(‘ENJOY YOUR
WEEKEND’);
    ELSIF D= ‘SUN’ THEN
        DBMS_OUTPUT.PUT_LINE(‘ENJOY YOUR
WEEKEND’);
    ELSE
        DBMS_OUTPUT.PUT_LINE(‘HAVE A NICE DAY’);
    END IF;
END;
```

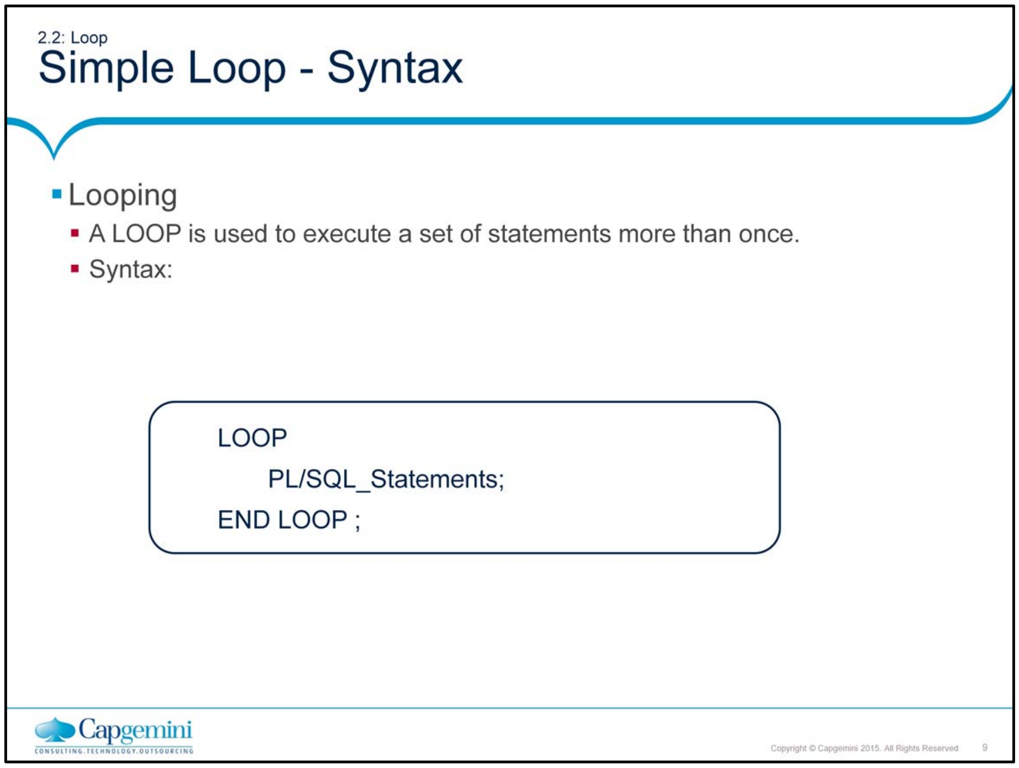
```
IF Condition_Expr_1 THEN

PL/SQL_Statements_1 ;
ELSIF Condition_Expr_2 THEN

PL/SQL_Statements_2 ;
ELSIF Condition_Expr_3 THEN
        Null;
END IF;
```

Programmatic Constructs (contd.)
Conditional Execution (contd.):
As every condition must have at least one statement, NULL statement can be used as filler.
NULL command does nothing.
Sometimes NULL is used in a condition merely to indicate that such a condition has been taken into consideration, as well. So your code will resemble the code as given below:

Conditions for NULL are checked through IS NULL and IS NOT NULL predicates.

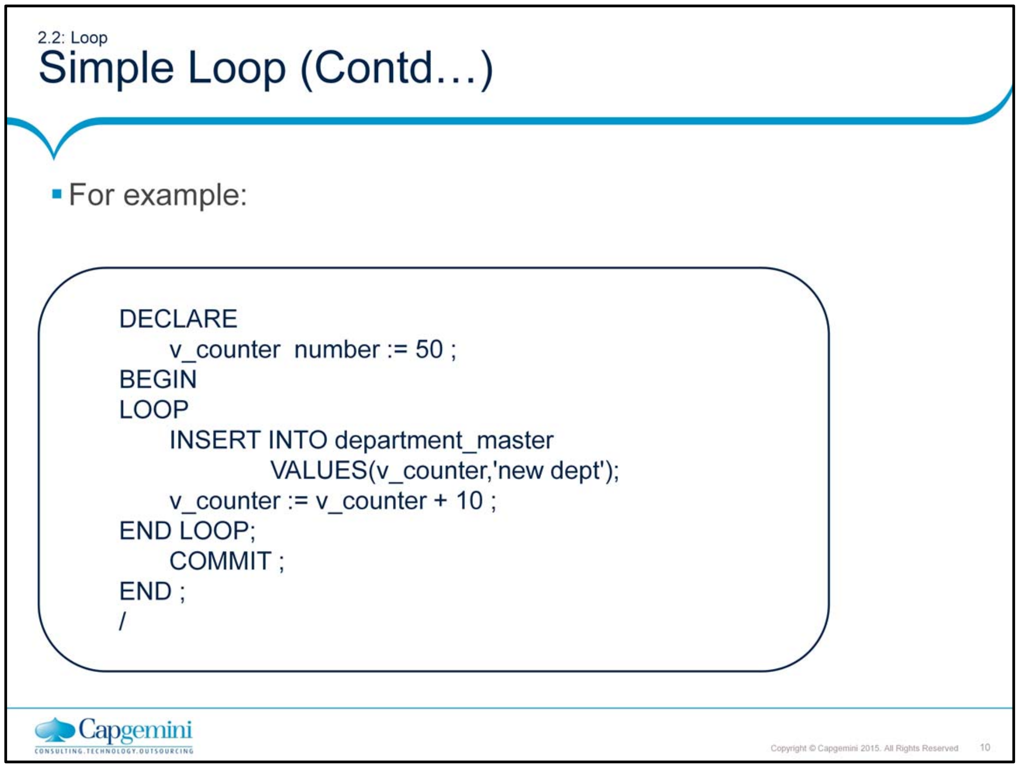


2.2: Loop

Simple Loop - Syntax

- Looping
  - A LOOP is used to execute a set of statements more than once.
  - Syntax:

```
LOOP
      PL/SQL_Statements;
END LOOP ;
```



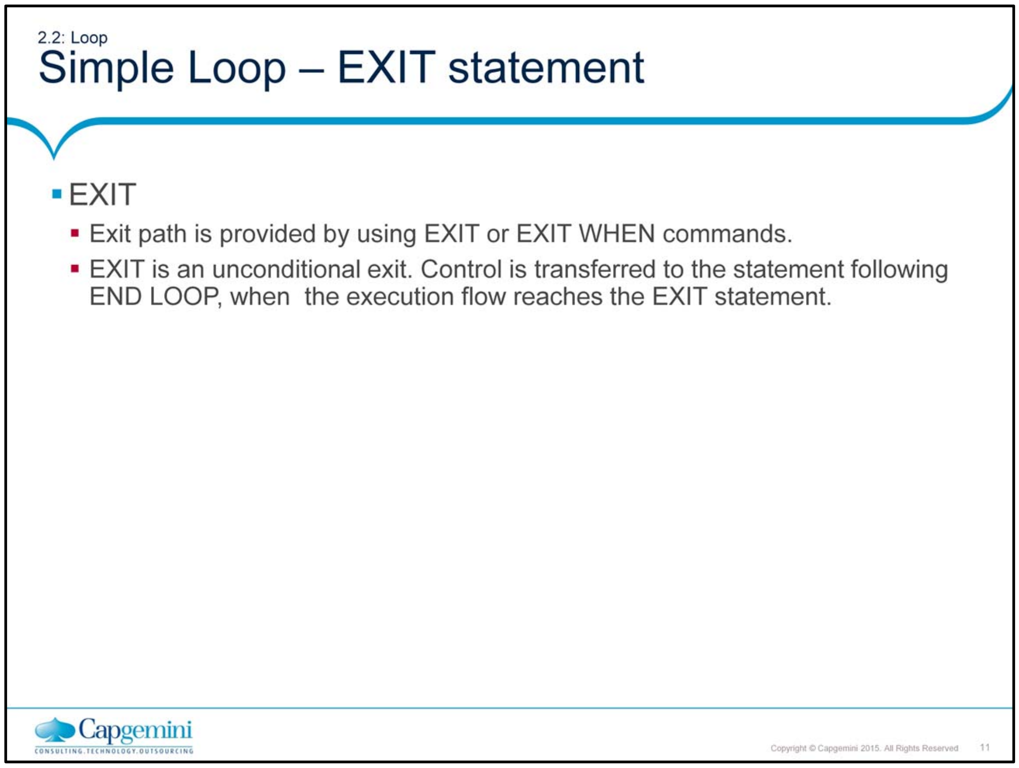
2.2: Loop

Simple Loop (Contd…)

- For example:

```
DECLARE
      v_counter  number := 50 ;
BEGIN
LOOP
      INSERT INTO department_master
                  VALUES(v_counter,'new dept');
      v_counter := v_counter + 10 ;
END LOOP;
      COMMIT ;
END ;
/
```

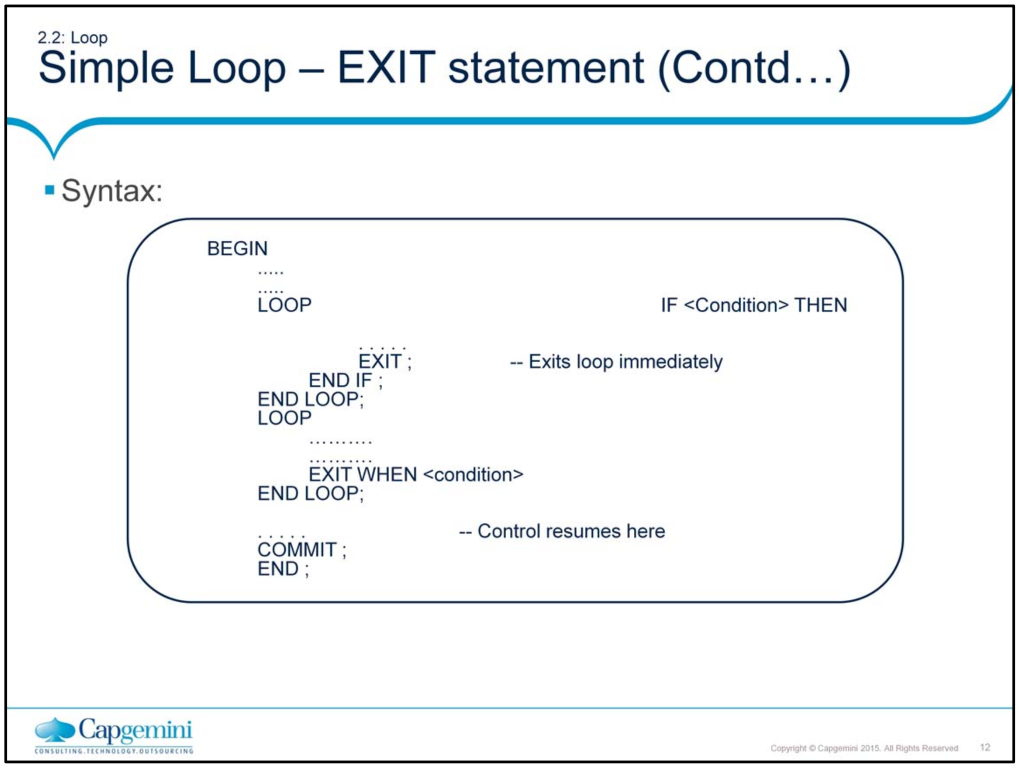
Programmatic Constructs (contd.)
Looping
The example shown in the slide is an endless loop.
When LOOP ENDLOOP is used in the above format, then an exit path must necessarily be provided. This is discussed in the following slide.



2.2: Loop

Simple Loop – EXIT statement

- EXIT
  - Exit path is provided by using EXIT or EXIT WHEN commands.
  - EXIT is an unconditional exit. Control is transferred to the statement following END LOOP, when the execution flow reaches the EXIT statement.



2.2: Loop

Simple Loop – EXIT statement (Contd…)

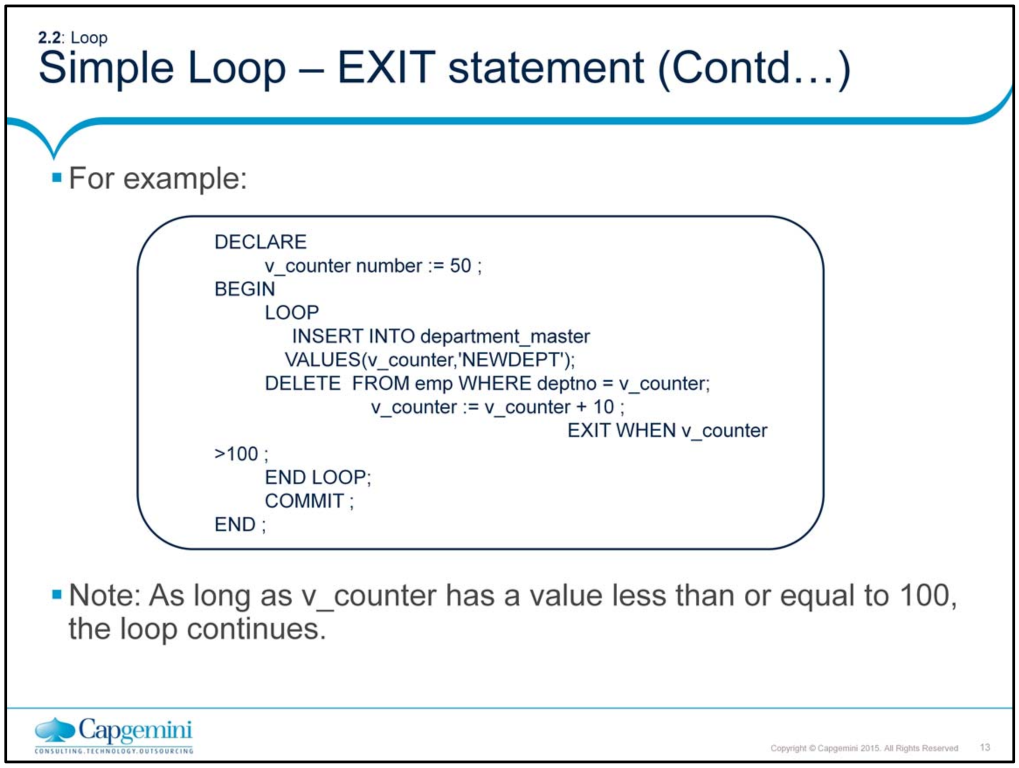
- Syntax:

```
BEGIN
    .....
    .....
    LOOP                                          IF <Condition> THEN

                 . . . . .
                 EXIT ;               -- Exits loop immediately
         END IF ;
    END LOOP;
    LOOP
         ……….
         ……….
         EXIT WHEN <condition>
    END LOOP;

    . . . . .                      -- Control resumes here
    COMMIT ;
    END ;
```

Note:
EXIT WHEN is used for conditional exit out of the loop.



2.2: Loop

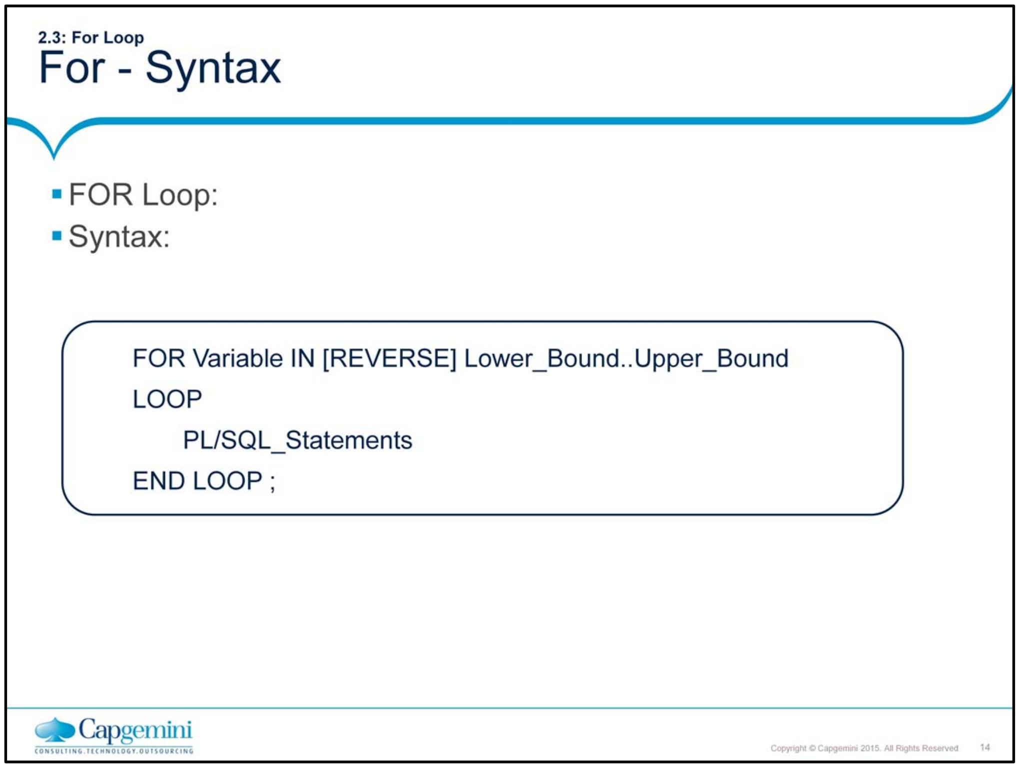
Simple Loop – EXIT statement (Contd…)

- For example:

```
DECLARE
        v_counter number := 50 ;
BEGIN
        LOOP
            INSERT INTO department_master
           VALUES(v_counter,'NEWDEPT');
        DELETE  FROM emp WHERE deptno = v_counter;
                        v_counter := v_counter + 10 ;
                                                      EXIT WHEN v_counter
>100 ;
        END LOOP;
        COMMIT ;
END ;
```

- Note: As long as v_counter has a value less than or equal to 100, the loop continues.

Note:
LOOP.. END LOOP can be used in conjunction with FOR and WHILE for better control on looping.



2.3: For Loop

For - Syntax

- FOR Loop:
- Syntax:

```
FOR Variable IN [REVERSE] Lower_Bound..Upper_Bound
LOOP
      PL/SQL_Statements
END LOOP ;
```

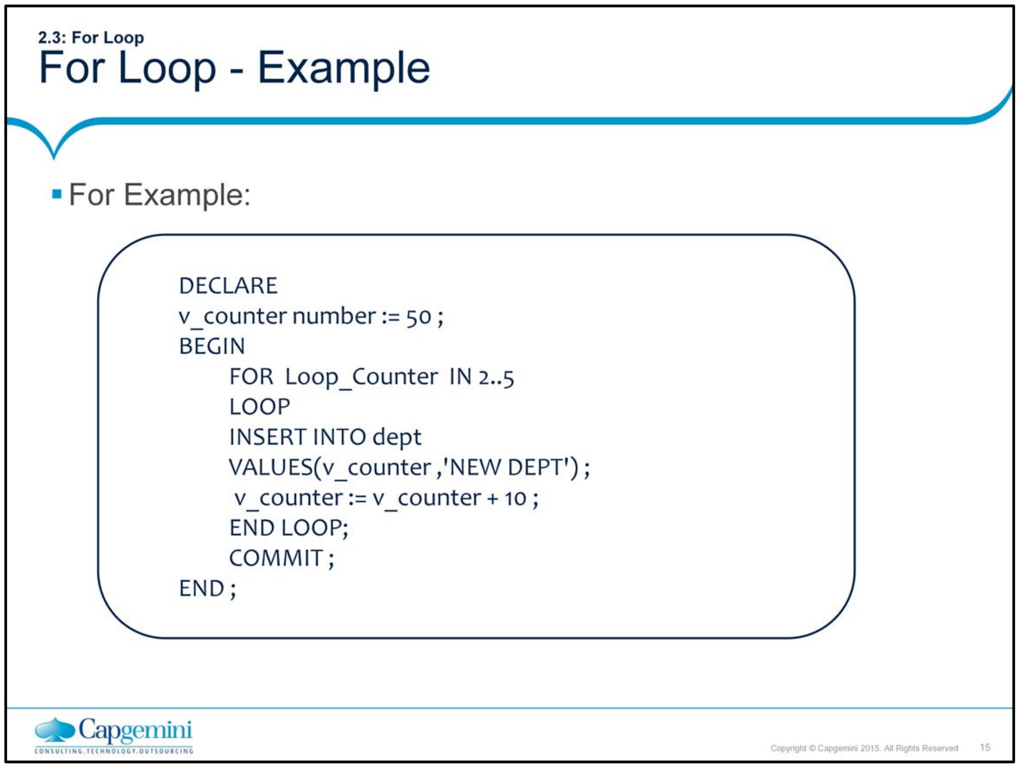
Programmatic Constructs (contd.)
FOR Loop:
FOR loop is used for executing the loop a fixed number of times. The number of times the loop will execute equals the following:

Upper_Bound - Lower_Bound + 1.

Upper_Bound and Lower_Bound must be integers.
Upper_Bound must be equal to or greater than Lower_Bound.
Variables in FOR loop need not be explicitly declared.

Variables take values starting at a Lower_Bound and ending at a Upper_Bound.
The variable value is incremented by 1, every time the loop reaches the bottom.
When the variable value becomes equal to the Upper_Bound, then the loop executes and exits.

When REVERSE is used, then the variable takes values starting at Upper_Bound and ending at Lower_Bound.
Value of the variable is decremented each time the loop reaches the bottom.



2.3: For Loop

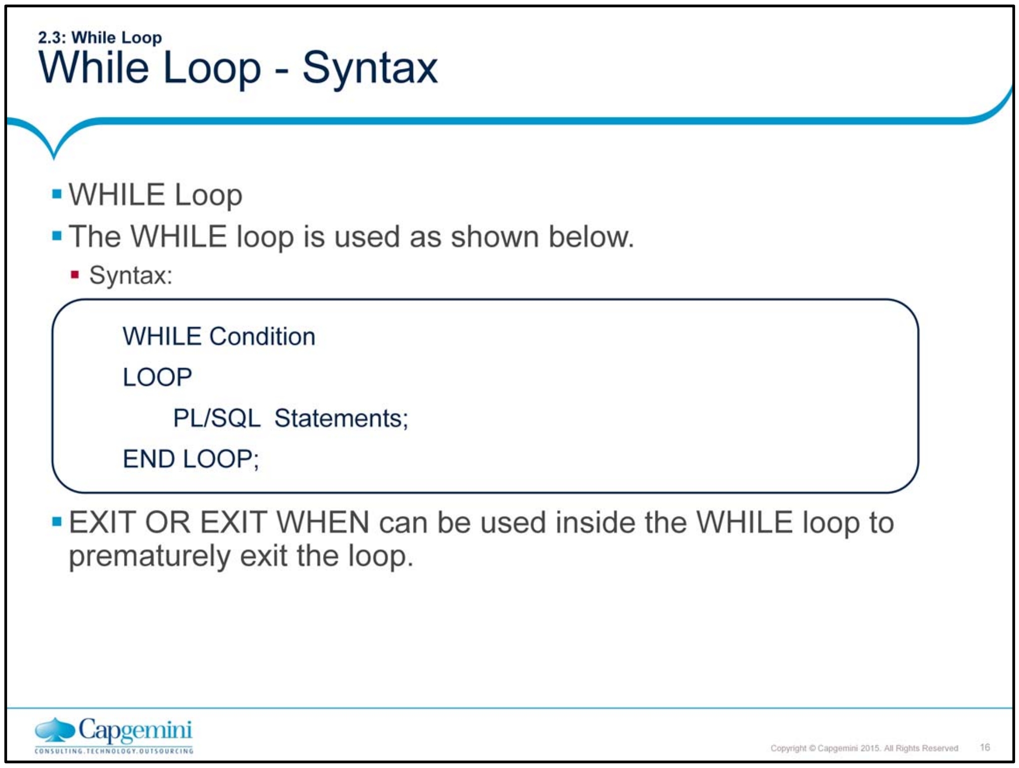
For Loop - Example

- For Example:

```
DECLARE
v_counter number := 50 ;
BEGIN
      FOR  Loop_Counter  IN 2..5
      LOOP
      INSERT INTO dept
      VALUES(v_counter ,'NEW DEPT') ;
       v_counter := v_counter + 10 ;
      END LOOP;
      COMMIT ;
END ;
```

15

Programmatic Constructs (contd.)
In the example in the above slide, the loop will be executed (5 - 2 + 1) = 4 times.
A Loop_Counter variable can also be used inside the loop, if required.
Lower_Bound and/or Upper_Bound can be integer expressions, as well.



2.3: While Loop

While Loop - Syntax

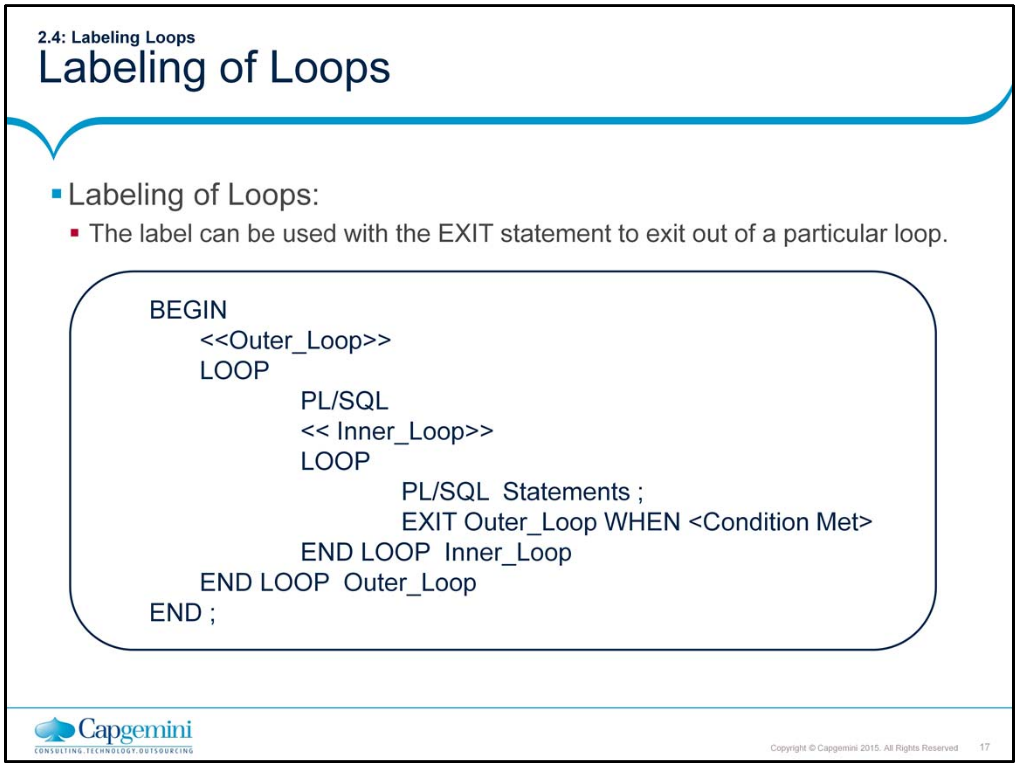
- WHILE Loop
- The WHILE loop is used as shown below.
  - Syntax:

```
WHILE Condition
LOOP
      PL/SQL  Statements;
END LOOP;
```

- EXIT OR EXIT WHEN can be used inside the WHILE loop to prematurely exit the loop.

Programmatic Constructs (contd.)
WHILE Loop:
Example:

```
DECLARE
    ctr number := 1;
BEGIN
    WHILE ctr <= 10
    LOOP
    dbms_output.put_line(ctr);
        ctr := ctr+1;
    END LOOP;
END;
/
```



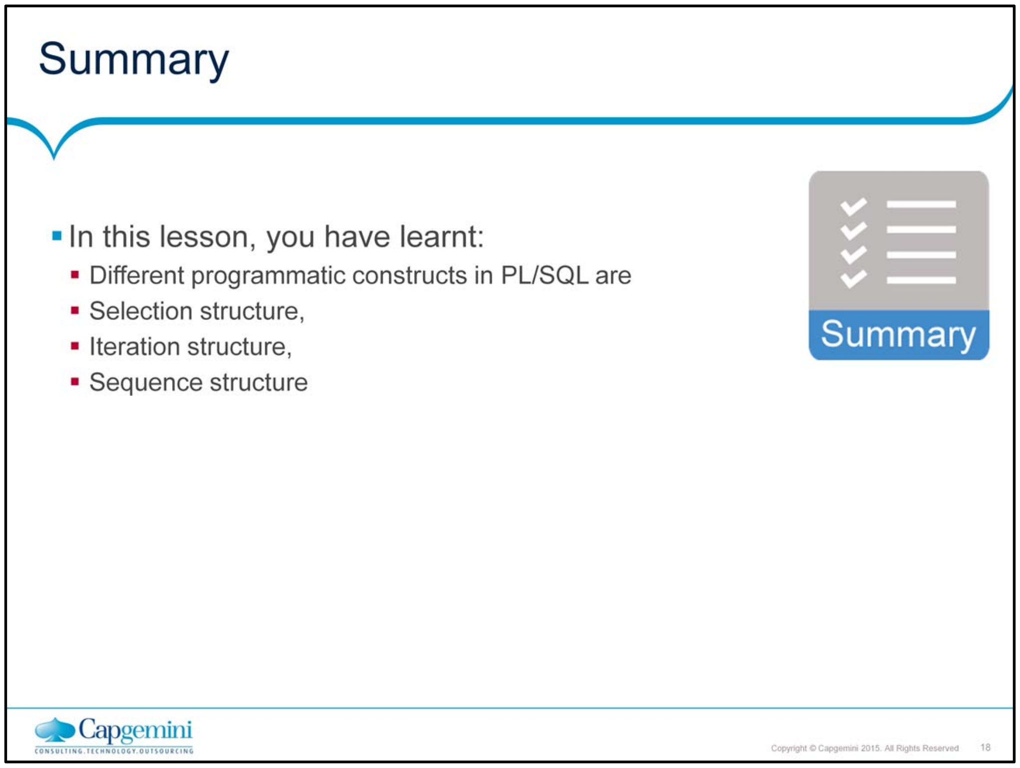
2.4: Labeling Loops

Labeling of Loops

- Labeling of Loops:
  - The label can be used with the EXIT statement to exit out of a particular loop.

```
BEGIN
      <<Outer_Loop>>
      LOOP
                  PL/SQL
                  << Inner_Loop>>
                  LOOP
                              PL/SQL  Statements ;
                              EXIT Outer_Loop WHEN <Condition Met>
                  END LOOP  Inner_Loop
      END LOOP  Outer_Loop
END ;
```

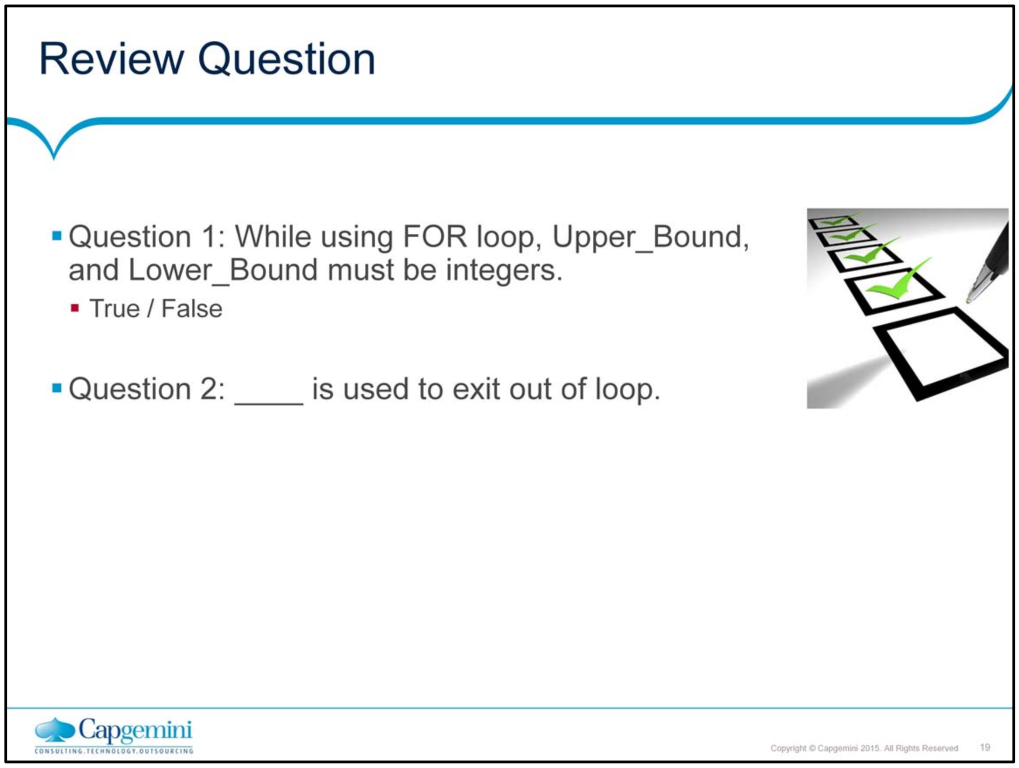
Programmatic Constructs (contd.)
Labeling of Loops:
Loops themselves can be labeled as in the case of blocks.
The label can be used with the EXIT statement to exit out of a particular loop.



Summary

- In this lesson, you have learnt:
  - Different programmatic constructs in PL/SQL are
  - Selection structure,
  - Iteration structure,
  - Sequence structure

Add the notes here.



Review Question

- Question 1: While using FOR loop, Upper_Bound, and Lower_Bound must be integers.
  - True / False
- Question 2: ____ is used to exit out of loop.

Add the notes here.