6 Writing Control Structures

Objectives

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

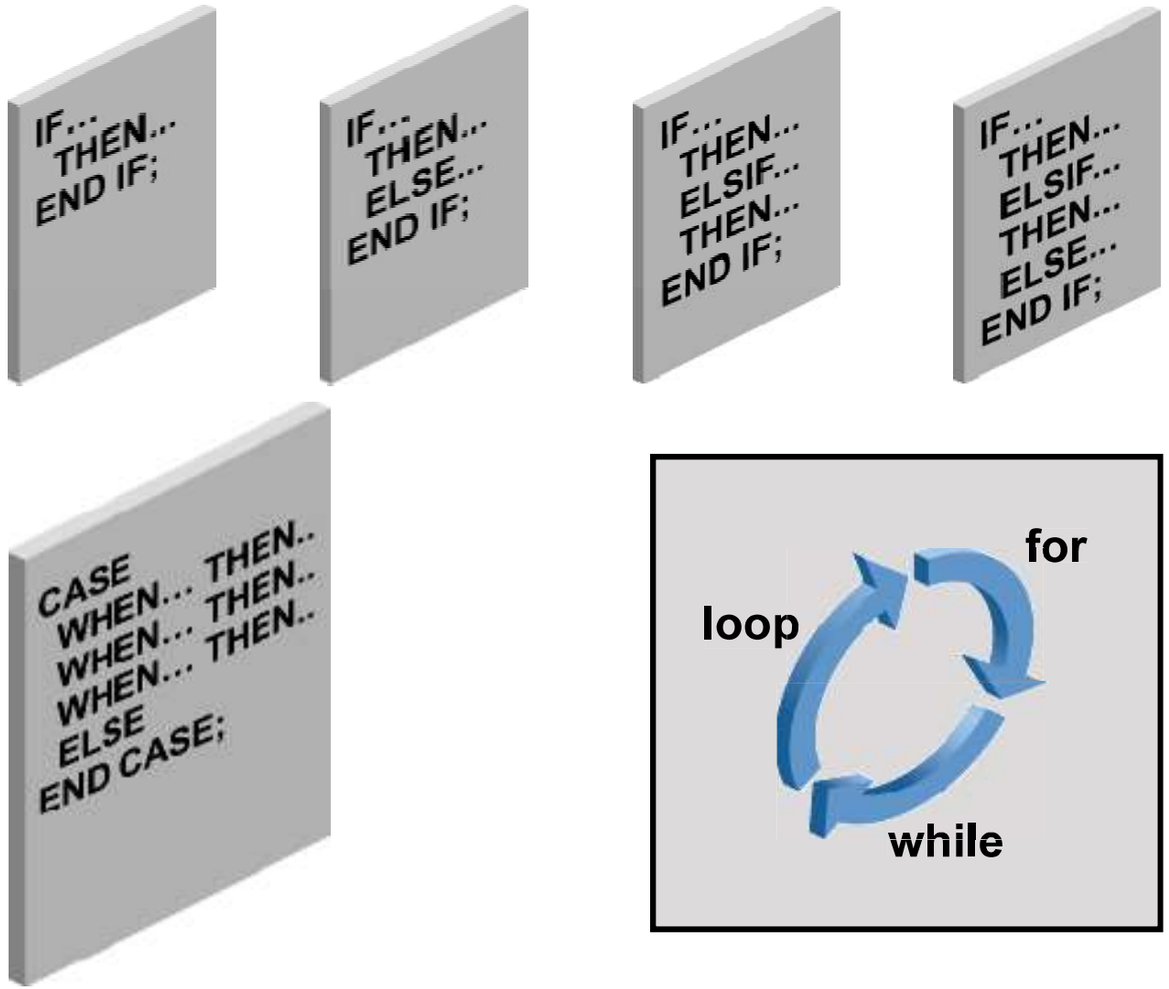
- Identify the uses and types of control structures
- Construct an `IF` statement
- Use `CASE` statements and `CASE` expressions
- Construct and identify loop statements
- Use guidelines when using conditional control structures

Course Roadmap

PLSQL

- Lesson 6: Writing Control Statements — You are here!
- Lesson 7: Working with Composite DataTypes
- Lesson 8: Using Explicit Cursors
- Lesson 9: Exception Handling
- Lesson 10: Stored Procedures and Functions

Controlling Flow of Execution



Agenda

- Using `IF` statements
- Using `CASE` statements and `CASE` expressions
- Constructing and identifying loop statements

- Syntax:

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

Simple IF Statement

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

```
anonymous block completed
```

IF THEN ELSE Statement

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

```
anonymous block completed
 I am not a child
```

IF ELSIF ELSE Clause

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

```
anonymous block completed
 I am in my thirties
```

NULL Value in IF Statement

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

```
anonymous block completed
 I am not a child
```

Agenda

- Using `IF` statements
- Using `CASE` statements and `CASE` expressions
- Constructing and identifying loop statements

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;
```

CASE Expressions: Example

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

Searched CASE Expressions

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

CASE Statement

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

Handling Nulls

When you are 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.

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

Boolean Expressions or Logical Expression?

- 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)

Agenda

- Using `IF` statements
- Using `CASE` statements and `CASE` expressions
- Constructing and identifying loop statements

Iterative Control: LOOP Statements

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

Basic Loops

Syntax:

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

Basic Loop: Example

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

WHILE Loops

•Syntax:

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

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

WHILE Loops: Example

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

FOR Loops

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

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

FOR Loops: Example

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

FOR Loop Rules

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

Suggested Use of 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.

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.

Nested Loops and Labels: Example

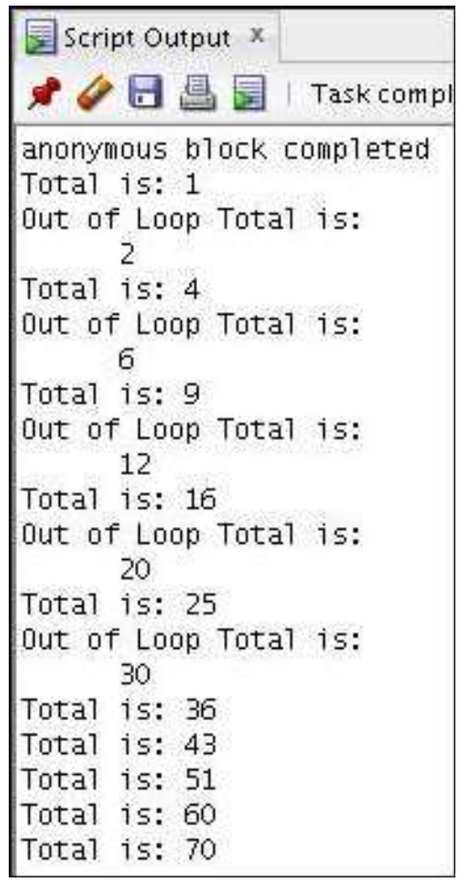
```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;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

PL/SQL CONTINUE Statement

- Definition
  - Adds the functionality to begin the next loop iteration
  - Provides programmers with the ability to transfer control to the next iteration of a loop
  - Uses parallel structure and semantics to the `EXIT` statement
- Benefits
  - Eases the programming process
  - May provide a small performance improvement over the previous programming workarounds to simulate the `CONTINUE` statement

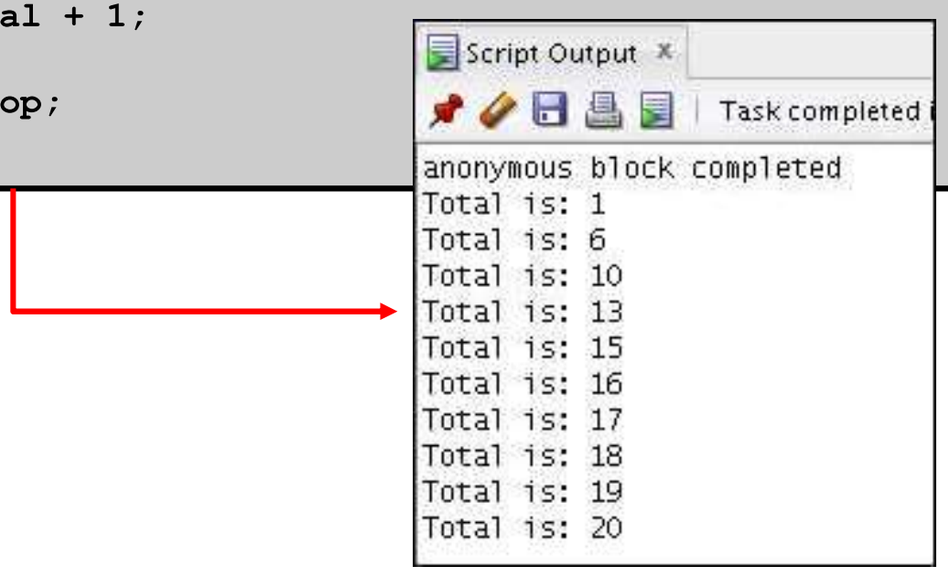
PL/SQL CONTINUE Statement: Example 1

```
DECLARE
  v_total SIMPLE_INTEGER := 0;
BEGIN
  FOR i IN 1..10 LOOP
1   v_total := v_total + i;
    dbms_output.put_line
     ('Total is: '|| v_total);
    CONTINUE WHEN i > 5;
    v_total := v_total + i;
2   dbms_output.put_line
     ('Out of Loop Total is:
      '|| v_total);
  END LOOP;
END;
/
```



PL/SQL CONTINUE Statement: Example 2

```
DECLARE
 v_total NUMBER := 0;
BEGIN
 <<BeforeTopLoop>>
 FOR i IN 1..10 LOOP
   v_total := v_total + 1;
   dbms_output.put_line
     ('Total is: ' || v_total);
   FOR j IN 1..10 LOOP
     CONTINUE BeforeTopLoop WHEN i + j > 5;
     v_total := v_total + 1;
   END LOOP;
 END LOOP BeforeTopLoop;
END;
```



There are three types of loops: basic, `FOR`, and `WHILE`.

a. True
b. False

Summary

In this lesson, you should have learned 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` statement
- `CONTINUE` statement

Practice 6: Overview

This practice covers the following topics:

- Performing conditional actions by using `IF` statements
- Performing iterative steps by using `LOOP` structures