Chapter 3

Control Structures

Control Structures

- Three types of PL/SQL control structures:

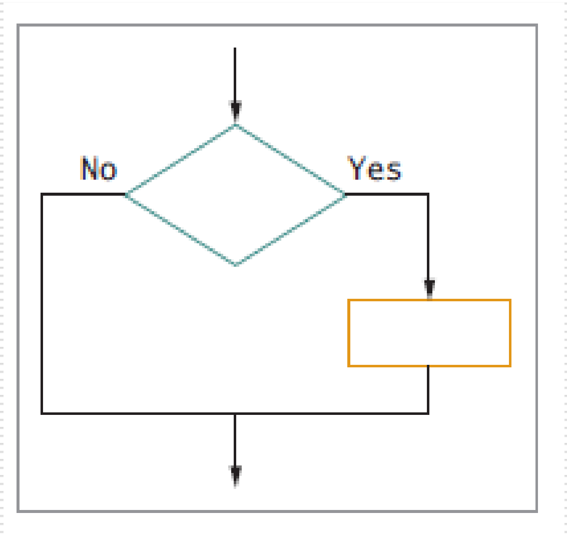
  - IF Statements
  - CASE Statements
  - Loop control structures
    - Basic Loops
    - WHILE loops
    - FOR loops
    - Nested loops

IF Statements

- IF Statements
  - Simple IF
  - IF/THEN/ELSE
  - IF/THEN/ELSIF/ELSE

Simple IF Statement

- Action is provided for only one outcome



Simple IF Statement
IF/END IF

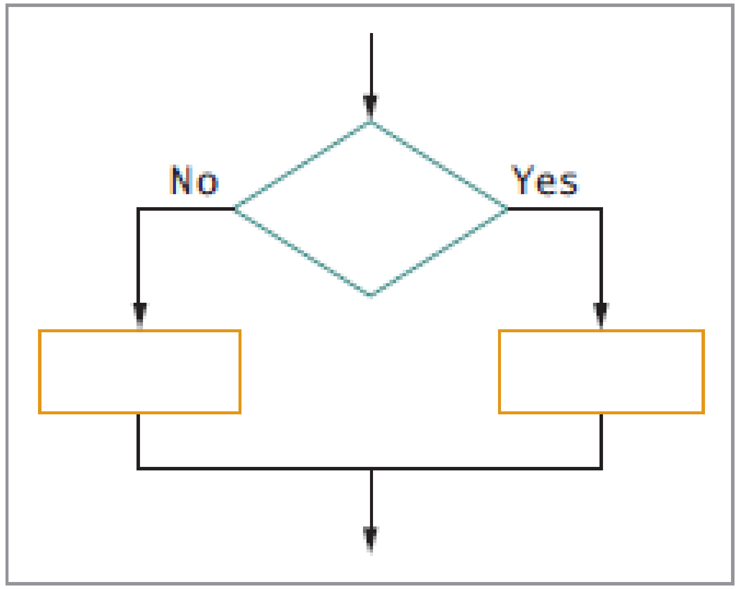
```
DECLARE
  v_myage NUMBER := 21;
BEGIN
  IF v_myage > 19 THEN
    DBMS_OUTPUT.PUT_LINE(' I am an adult ');
  END IF;
END;
```

Simple IF Statement
Boolean Variable

```
DECLARE
  v_active BOOLEAN := TRUE;
BEGIN
  IF v_active THEN
    DBMS_OUTPUT.PUT_LINE('Customer is active');
  END IF;
END;
```

IF/THEN/ELSE Statement

- Provides an action for two possible outcomes



IF/THEN/ELSE Statement

```
DECLARE
  v_myage NUMBER := 31;
BEGIN
  IF v_myage < 13 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE    -- default clause
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
```

IF/ELSIF Statement

MYAGE	Message
0-12	I am a child
13-19	I am a teenager
20-29	I am in my twenties
30-39	I am in my thirties
40 or older	I am always young

IF/THEN/ELSIF/ELSE Statement-WRONG WAY

```
DECLARE
  v_myage NUMBER := :Enter_age;
BEGIN
  IF v_myage >= 0 AND v_myage <= 12 THEN
    DBMS_OUTPUT.PUT_LINE('I am a child');
  ELSIF v_myage >= 13 AND v_myage <= 19 THEN
    DBMS_OUTPUT.PUT_LINE('I am a teenager');
  ELSIF v_myage >= 20 AND v_myage <= 29 THEN
    DBMS_OUTPUT.PUT_LINE('I am in my twenties');
  ELSIF v_myage >= 30 AND v_myage <= 39 THEN
    DBMS_OUTPUT.PUT_LINE('I am in my thirties');
  ELSE
    DBMS_OUTPUT.PUT_LINE('I am always young ');
  END IF;
END;
```

IF/THEN/ELSIF/ELSE Statement

```
DECLARE
  v_myage NUMBER := :Enter_age;
BEGIN
  IF v_myage < 13 THEN
    DBMS_OUTPUT.PUT_LINE('I am a child');
  ELSIF v_myage < 20 THEN -- ELSIF clause
    DBMS_OUTPUT.PUT_LINE('I am a teenager');
  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');
  ELSIF v_myage >= 40 THEN -- No ELSE clause
    DBMS_OUTPUT.PUT_LINE('I am always young ');
  END IF;  -- END IF is two words
END;
```

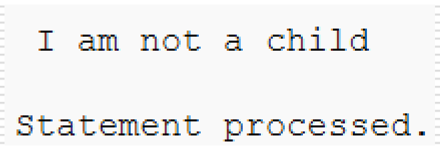
Logical Operators within IF

- Logical operators (AND, OR, NOT) enable multiple conditions to be checked

```
IF v_state_code = 'VA' OR v_state_code = 'PA' THEN
  v_tax_amount := v_sub_total * .06;
ELSE
  v_tax_amount := v_sub_total * .04;
END IF;
```

NULL Values in IF Statements

```
DECLARE
  v_myage NUMBER;  -- initial value is NULL
BEGIN
  IF v_myage < 13 THEN  -- returns NULL
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE    -- control goes to the ELSE statement
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
```



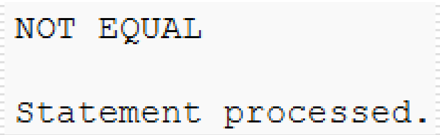
```
 I am not a child

Statement processed.
```

Handling NULLs

- Simple comparisons involving NULLs always return NULL
- Applying the logical operator NOT to a NULL returns NULL
- In conditional control statements, if a condition returns NULL, it behaves just like a FALSE, and the associated sequence of statements is not executed

Handling NULLs

```
DECLARE
  v_a CHAR(1) := NULL;
  v_b CHAR(1) := NULL;
BEGIN
  IF v_a = v_b  THEN   -- yields NULL, not TRUE and the
    DBMS_OUTPUT.PUT_LINE('EQUAL'); -- sequence of statements
  ELSE                             -- are not executed
    DBMS_OUTPUT.PUT_LINE('NOT EQUAL');
  END IF;
END;
```



```
NOT EQUAL

Statement processed.
```

Guidelines for Using IF Statements

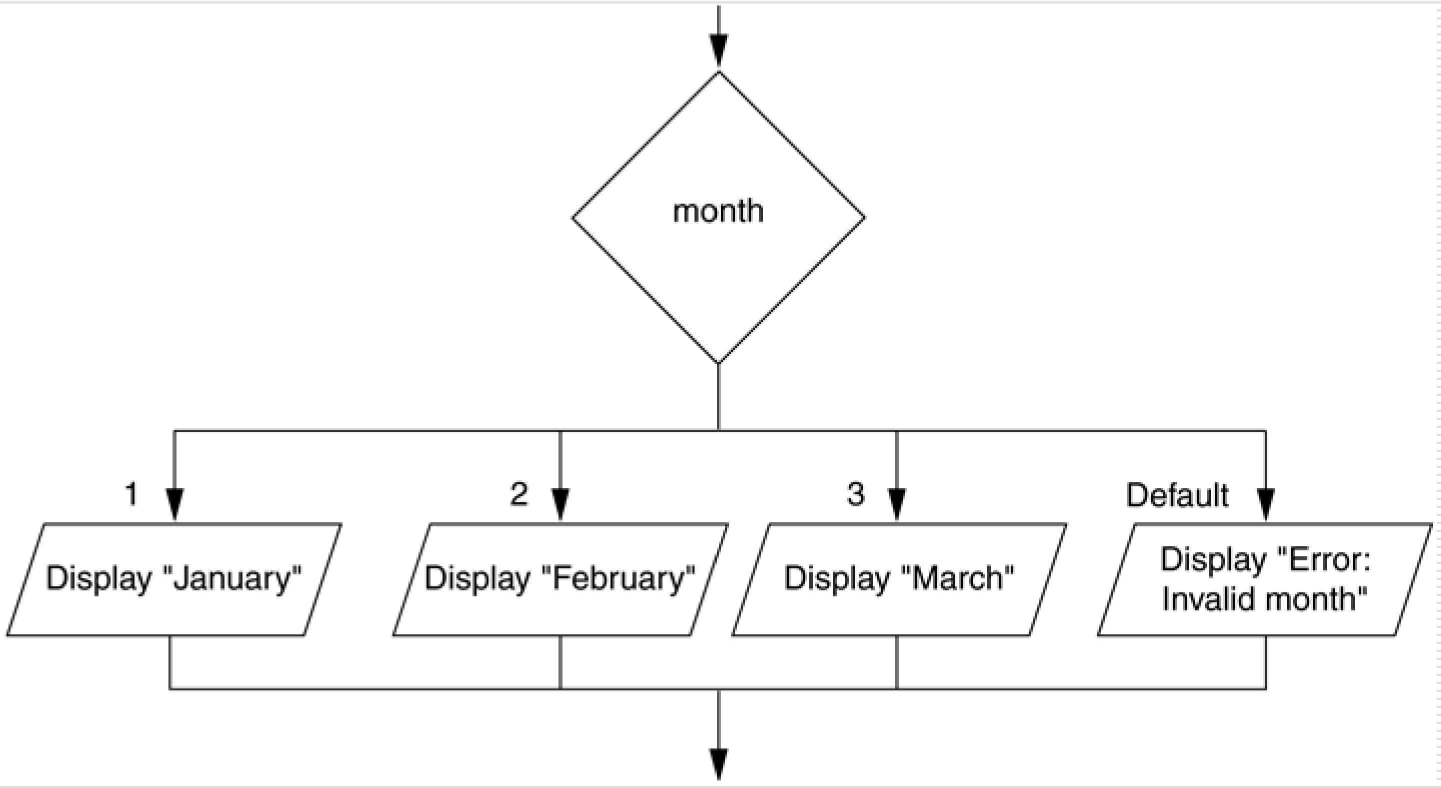
- Remember the spelling of the keywords:
  - ELSIF is one word
  - END IF is two words

CASE Statement

CASE Statement

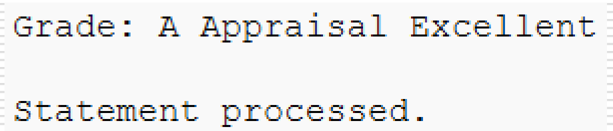
- Similar to IF statement
- A CASE statement can contain many PL/SQL statements
- CASE statements end with END CASE
- Three types:
  - Basic CASE Statement
  - CASE Expression
  - Searched CASE

Basic CASE Statement



Basic CASE Statements Using a Selector

```
DECLARE
  v_grade CHAR(1) := :Enter_grade;
  v_appraisal VARCHAR2(20);
BEGIN
  CASE v_grade    -- selector
    WHEN 'A' THEN v_appraisal := 'Excellent';
    WHEN 'B' THEN v_appraisal := 'Very Good';
    WHEN 'C' THEN v_appraisal := 'Good';
    ELSE v_appraisal := 'No such grade';
  END CASE;       -- End with END CASE
  DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal ‘
                                  || v_appraisal);
END;
```



```
Grade: A Appraisal Excellent

Statement processed.
```

Basic CASE Statement Using a Selector

- Uses a selector

```
DECLARE
   v_deptid   departments.department_id%TYPE;
   v_deptname departments.department_name%TYPE;
   v_emps     NUMBER;
   v_mngid    departments.manager_id%TYPE := 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;
```

CASE Expression

- A CASE expression is different from a CASE statement because it selects one of a number of results and assigns it to a variable
- A CASE expression ends with END not END CASE.

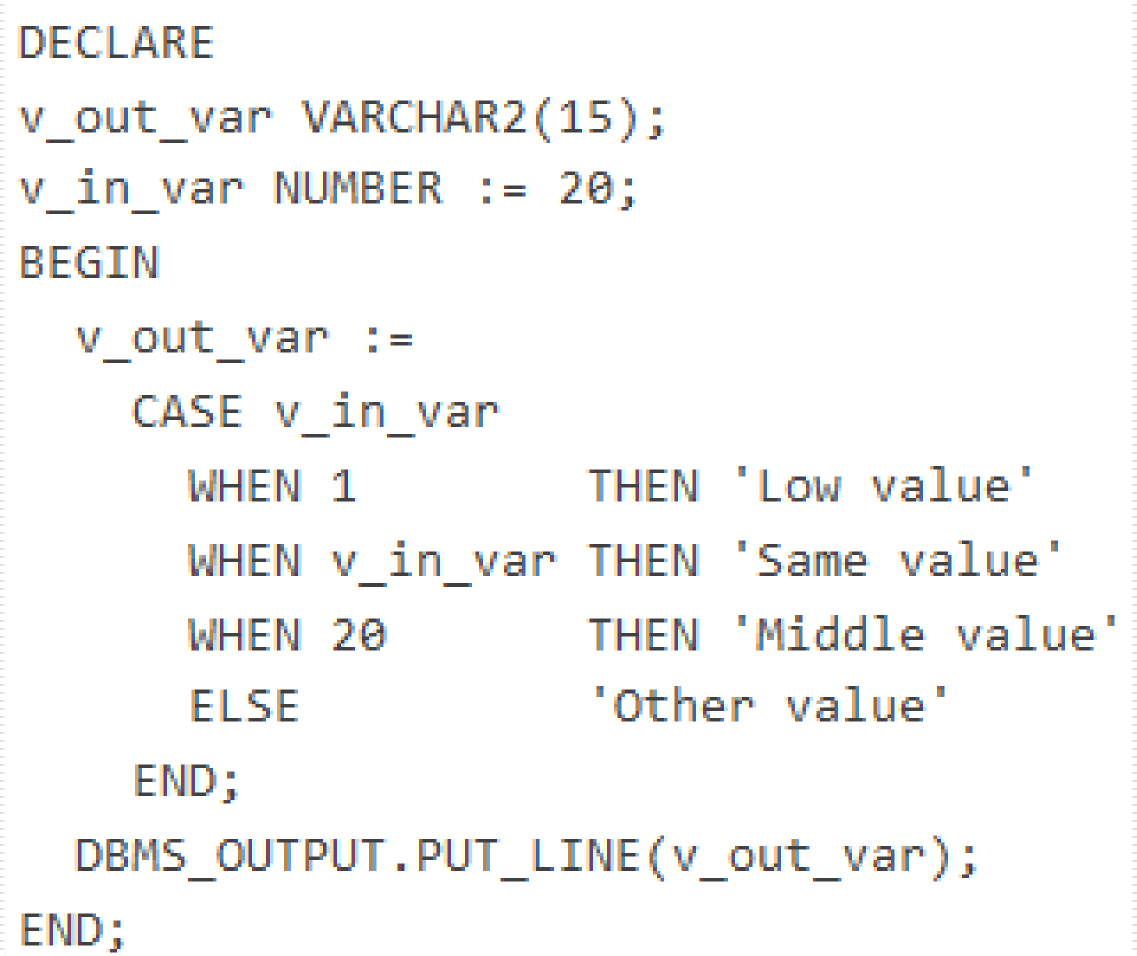
CASE Expression

```
DECLARE
  v_grade CHAR(1) := :Enter_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;       -- End with END
  DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal ' || v_appraisal);
END;
```

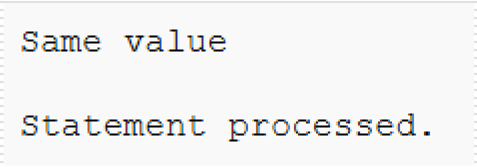
```
Grade: B Appraisal Very Good

Statement processed.
```

CASE Expression

- What will be displayed?



```
DECLARE
v_out_var VARCHAR2(15);
v_in_var NUMBER := 20;
BEGIN
  v_out_var :=
    CASE v_in_var
      WHEN 1        THEN 'Low value'
      WHEN v_in_var THEN 'Same value'
      WHEN 20       THEN 'Middle value'
      ELSE              'Other value'
    END;
  DBMS_OUTPUT.PUT_LINE(v_out_var);
END;
```



```
Same value

Statement processed.
```

CASE Expression

- CASE expressions end with END
- CASE expressions return a value into a variable

Searched CASE Expressions

- Has no selector
- Allows non-equality conditions, compound conditions, and different variables to be used in different WHEN clauses
- Returns a value into a variable

Searched CASE Expressions

```
DECLARE
  v_grade      CHAR(1) := 'A';
  v_appraisal VARCHAR2(20);
BEGIN
  v_appraisal :=
     CASE                     -- no selector here
       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;
```

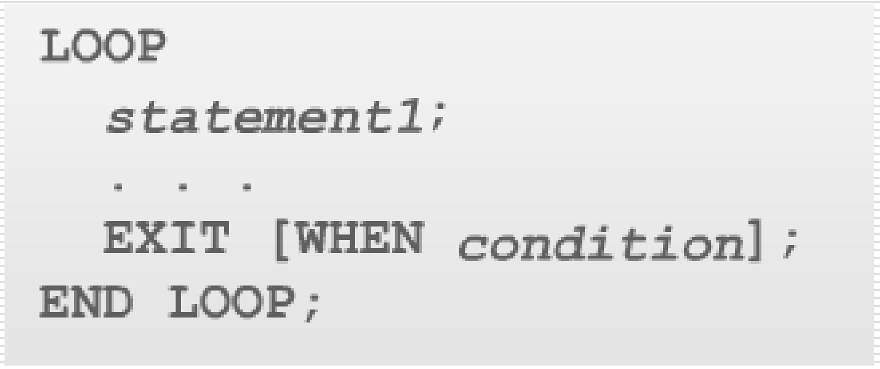
Looping

Loop Statements

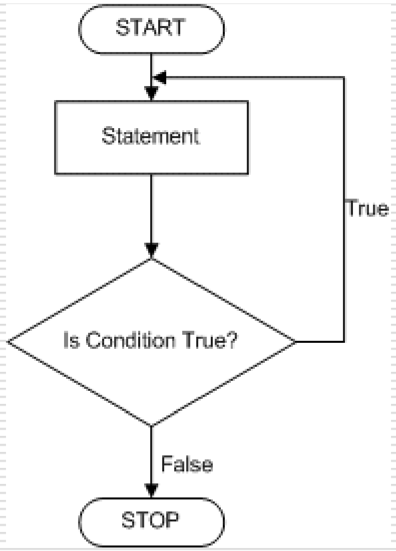
- Enables a statement or set of statements to be executed multiple times
- A loop must provide instructions to end the looping, or an ‘infinite’ loop will be produced
- Three types of loops:
  - Basic loop allows the execution of its statements at least once
  - WHILE loop performs repetitive actions based on a condition
  - FOR loop performs iterative actions based on a pre-determined counter

Basic Loop Statement

- Allows the execution of its statements at least once
- The EXIT condition is used to exit the loop
  - Optional WHEN clause



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



Basic Loop Statement

- When the EXIT statement is encountered, the condition in the WHEN clause is evaluated
  - If the condition yields TRUE, then the loop ends and control passes to the next statement after the loop
- Without the EXIT statement, the loop would be infinite

Basic Loop WHEN Statement - WRONG

- Three new location IDs for the country code of CA and the city of Montreal are inserted

```
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;
```

EXIT Within IF Statement

- EXIT statement
  - Can be specified as an action within an IF statement within the loop
  - No WHEN clause

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('The square of '
                ||v_counter||' is: '|| POWER(v_counter,2));
    v_counter := v_counter + 1;
    IF v_counter > 10 THEN
      EXIT;
    END IF;
  END LOOP;
END;
```

Basic Loop EXIT Rules

- The EXIT statement must be placed inside a loop
- If the EXIT condition is placed at the top of the loop (before any of the other executable statements) and that condition is initially true, then the loop exits and the other statements in the loop never execute
- A basic loop can contain multiple EXIT statemenst, BUT only one EXIT point is recommended

Basic Loop EXIT WHEN Statement

- Although the IF…THEN EXIT works to end a loop, the correct way to end a basic loop is with the EXIT WHEN statement
- If the WHEN clause evaluates to TRUE, the loop ends and control passes to the next statement following END LOOP

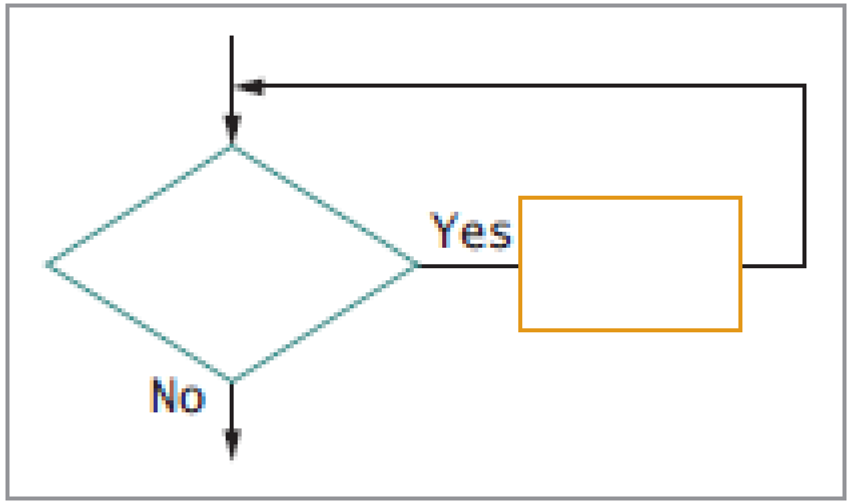
```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('Counter is ' || v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

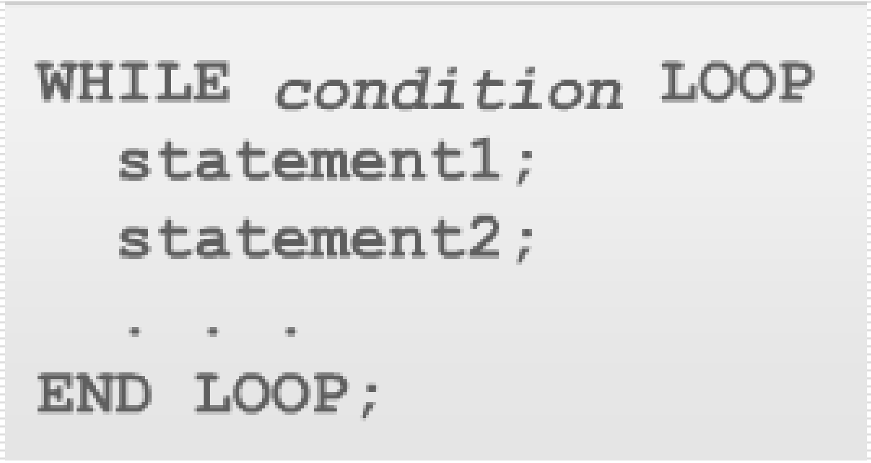
WHILE Loop

- Repeat a sequence of statements until the controlling condition is no longer TRUE
- Condition is evaluated at the start of each iteration
- Loop terminates when the condition is FALSE or NULL
- If the variables involved in the conditions do not change during the body of the loop, then the condition remains TRUE and the loop does not terminate
- If the condition is FALSE or NULL at the start of the loop, then the loop is not executed

WHILE Loop

- Condition is a Boolean variable or expression (TRUE, FALSE, or NULL)
- Statement can be one or more PL/SQL or SQL statements

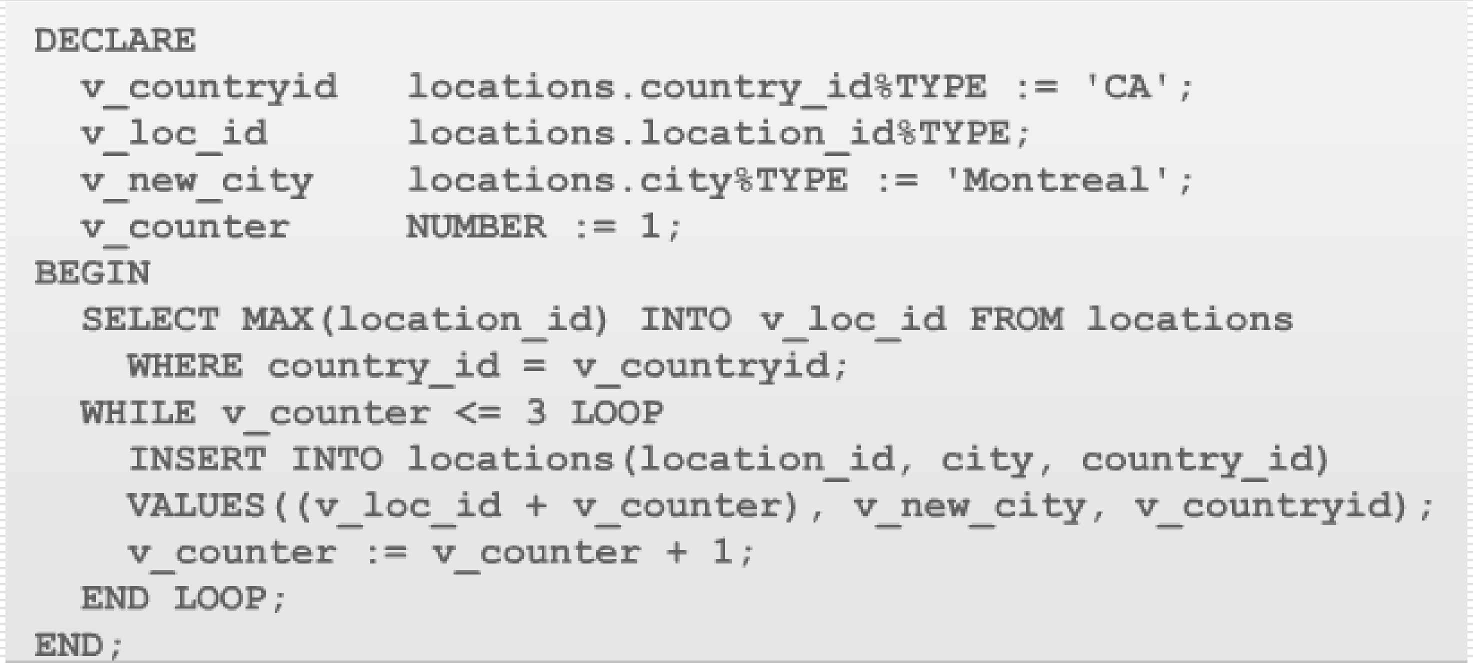




```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

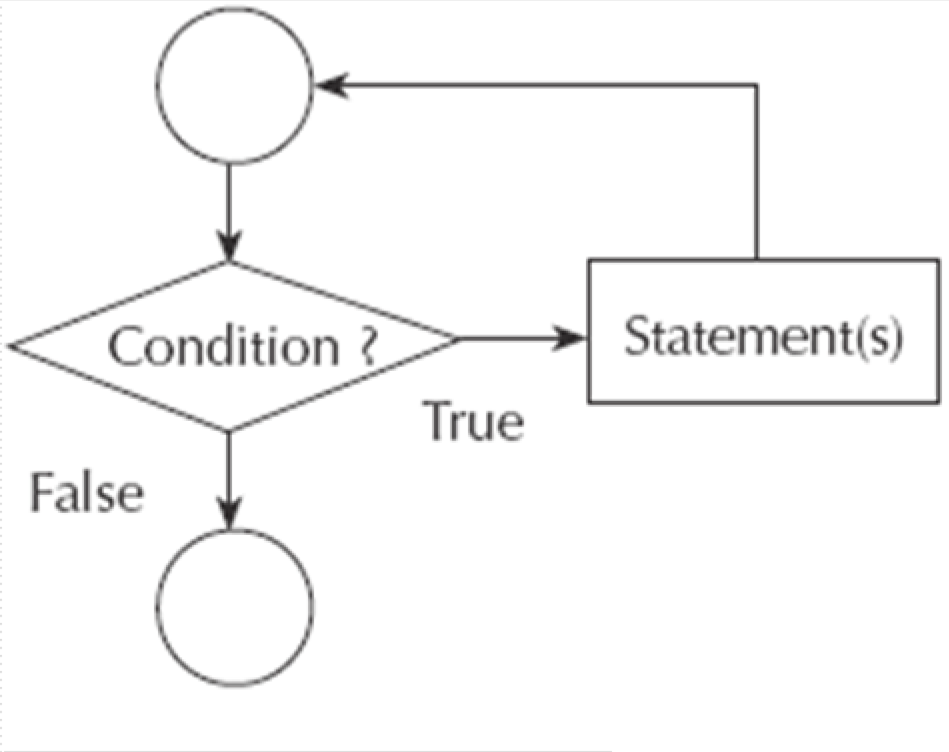
WHILE Loop - WRONG

- In this example, three new location IDs for Montreal, Canada, are inserted in the LOCATIONS table
- After the counter exceeds the number of new locations, the condition evaluates to FALSE and the loop is terminated

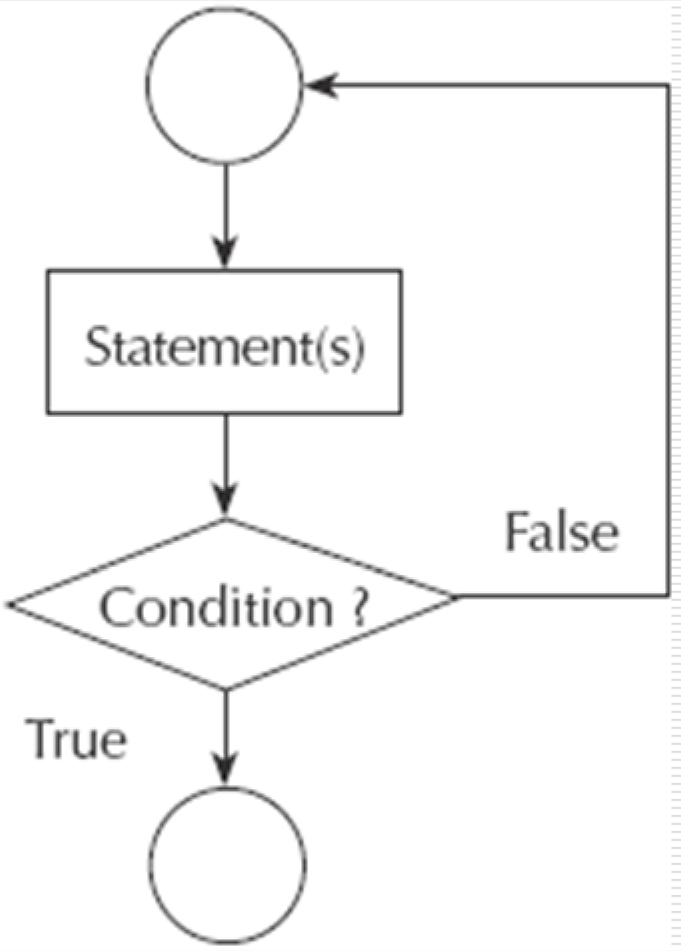


```
DECLARE
  v_countryid   locations.country_id%TYPE := 'CA';
  v_loc_id      locations.location_id%TYPE;
  v_new_city    locations.city%TYPE := 'Montreal';
  v_counter     NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
  WHILE v_counter <= 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;
```

WHILE versus Basic LOOP (until)



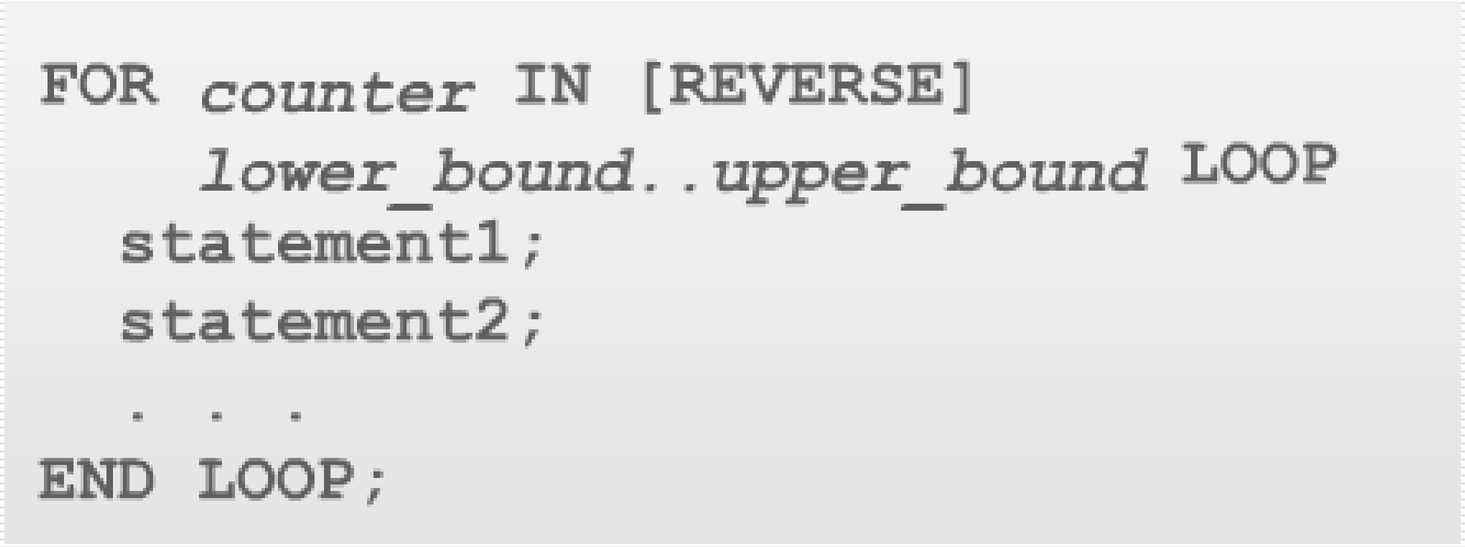
WHILE



LOOP

FOR Loop

- Use when the number of iterations is predetermined
- Counter is declared implicitly
- Counter can be referenced inside the loop



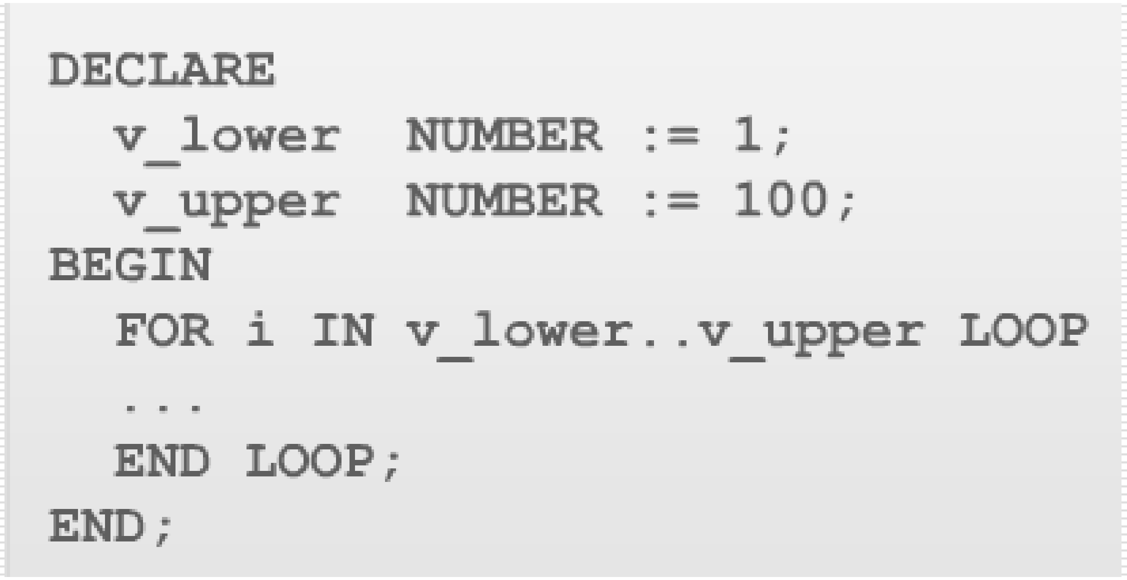
```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

FOR Loop

```
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

- The lower and upper bounds can be numeric literals



```
DECLARE
   v_lower   NUMBER := 1;
   v_upper   NUMBER := 100;
BEGIN
   FOR i IN v_lower..v_upper LOOP
   ...
   END LOOP;
END;
```

When to Use Loops

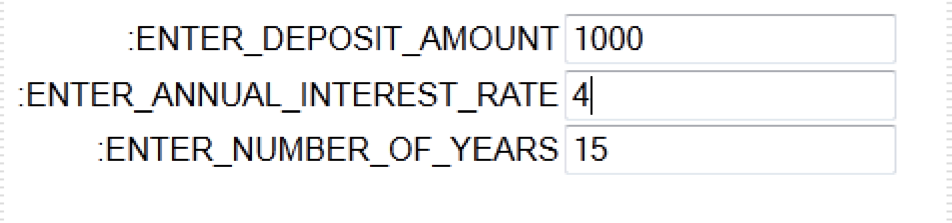
- Basic loop - when the statements inside the loop must execute at least once
- WHILE loop - if the condition has to be evaluated at the start of each iteration
- FOR loop - if the number of iterations is predetermined

Nested Loops

```
BEGIN
  FOR v_outerloop IN 1..3 LOOP
    FOR v_innerloop IN REVERSE 1..5 LOOP
     DBMS_OUTPUT.PUT_LINE('Outer loop is: ' ||
                          v_outerloop ||
                          ' and inner loop is: ' ||
                          v_innerloop);
    END LOOP;
  END LOOP;
END;
```

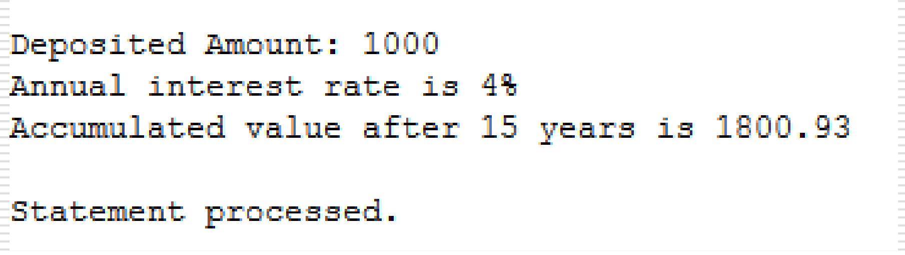
FOR LOOP Example

- Input: deposit amount, interest rate, years



:ENTER_DEPOSIT_AMOUNT	1000
:ENTER_ANNUAL_INTEREST_RATE	4
:ENTER_NUMBER_OF_YEARS	15

- Determine what the accumulated value would be on the deposited amount for the number of years at the interest rate



```
Deposited Amount: 1000
Annual interest rate is 4%
Accumulated value after 15 years is 1800.93

Statement processed.
```

FOR LOOP Example

```
DECLARE
  v_deposit_amount    INTEGER := :Enter_deposit_amount;
  v_interest_rate     INTEGER := :Enter_annual_interest_rate;
  v_years             INTEGER := :Enter_number_of_years;
  v_accumulated_value NUMBER(11,2)  := 0;
BEGIN
  v_accumulated_value := v_deposit_amount;
  FOR i IN 1..v_years LOOP
     v_accumulated_value := v_accumulated_value * (1 + v_interest_rate /100);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Deposited Amount: ' || v_deposit_amount);
  DBMS_OUTPUT.PUT_LINE('Annual interest rate is ' || v_interest_rate || '%');
  DBMS_OUTPUT.PUT_LINE('Accumulated value after ' || v_years || ' years is '
                                                  || v_accumulated_value);
END;
```

Q & A
Questions & Answers