Writing Control Structures

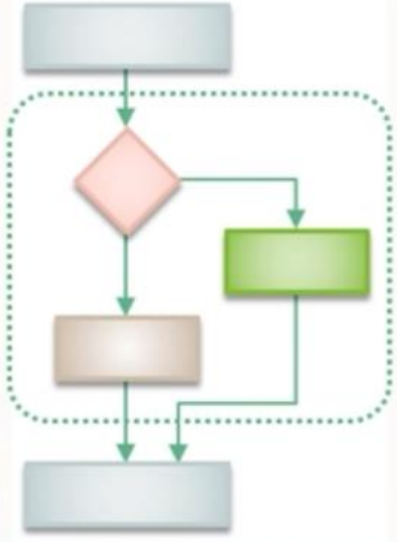
Objectives

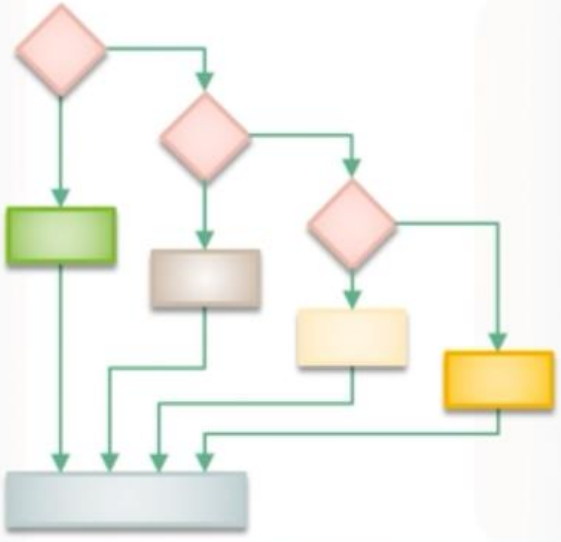
After completing this lesson, you should be able to do the following:

- 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

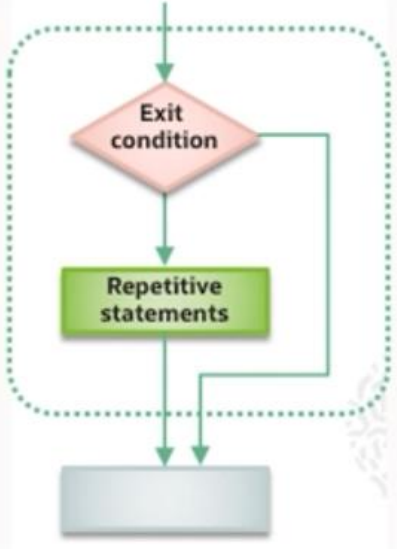
PL/SQL Control Structures

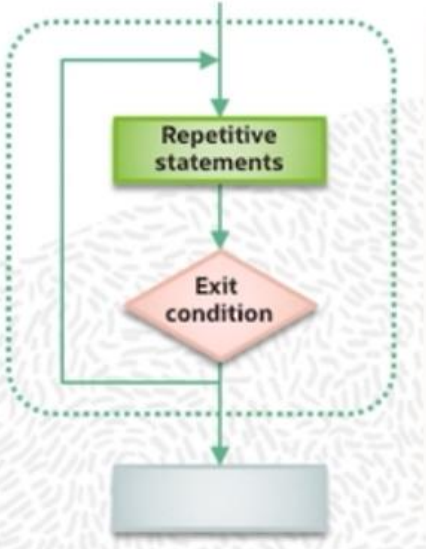
Conditional statements





Loop statements

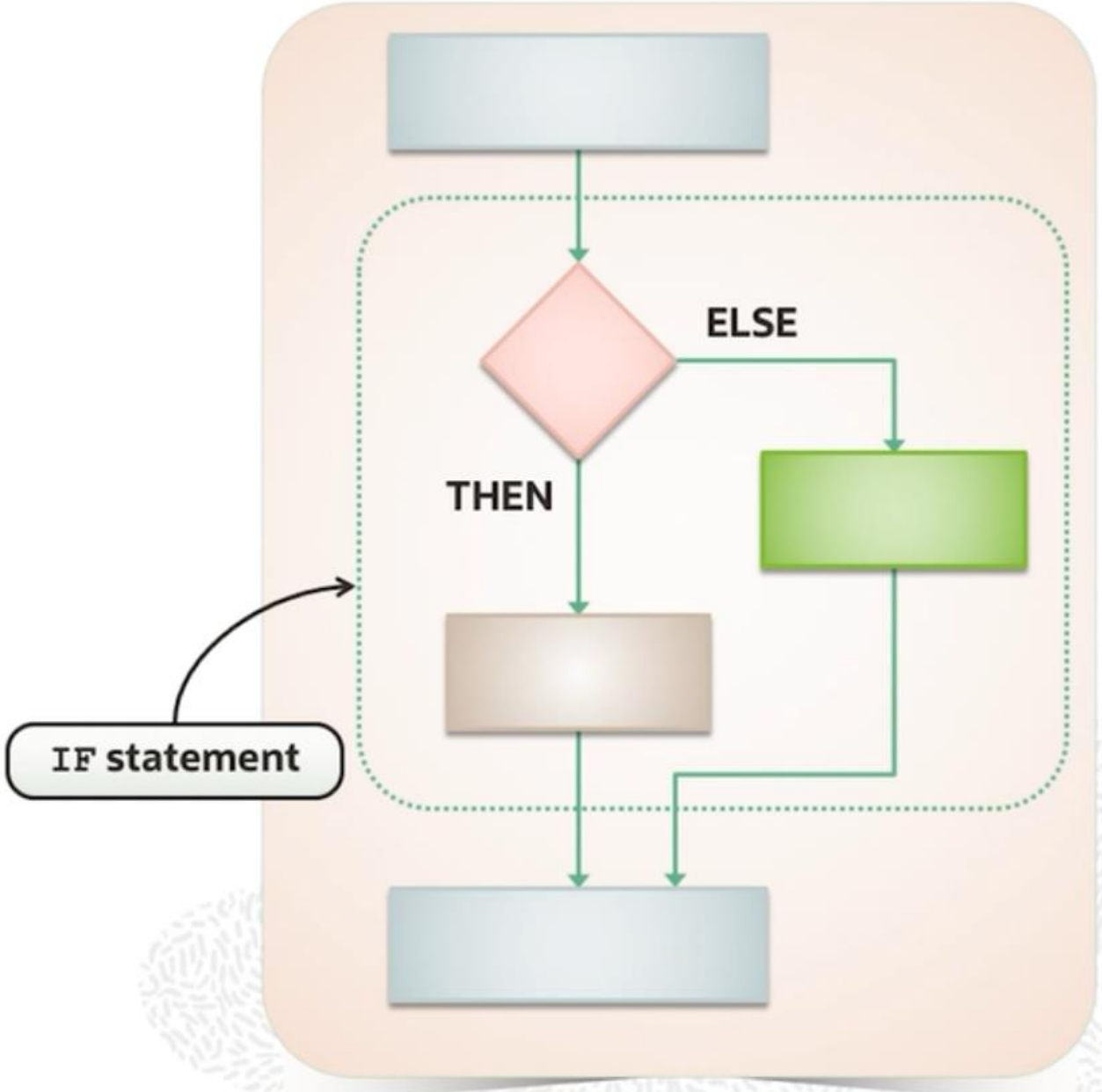




IF Statement

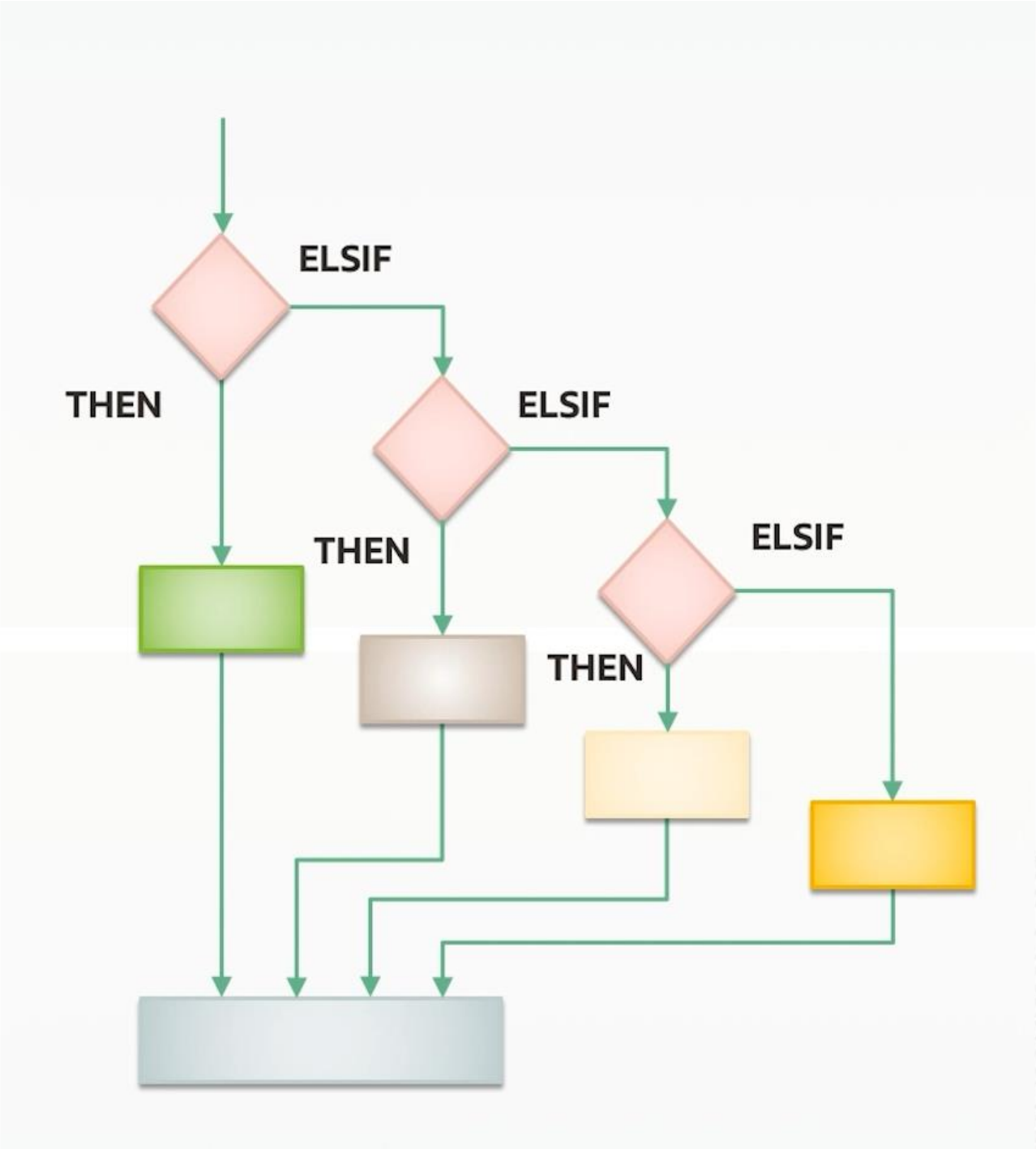
Syntax:

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



IF-ELSIF Statements

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



Simple IF Statement

```
DECLARE
  v_myage  NUMBER := 10;
BEGIN
  IF v_myage  < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
/
```

```
PL/SQL procedure successfully completed.

I am a child
```

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;
/
```

```
PL/SQL procedure successfully 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;
/
```

```
PL/SQL procedure successfully completed.

I am in my thirties
```

NULL Value an 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;
/
```

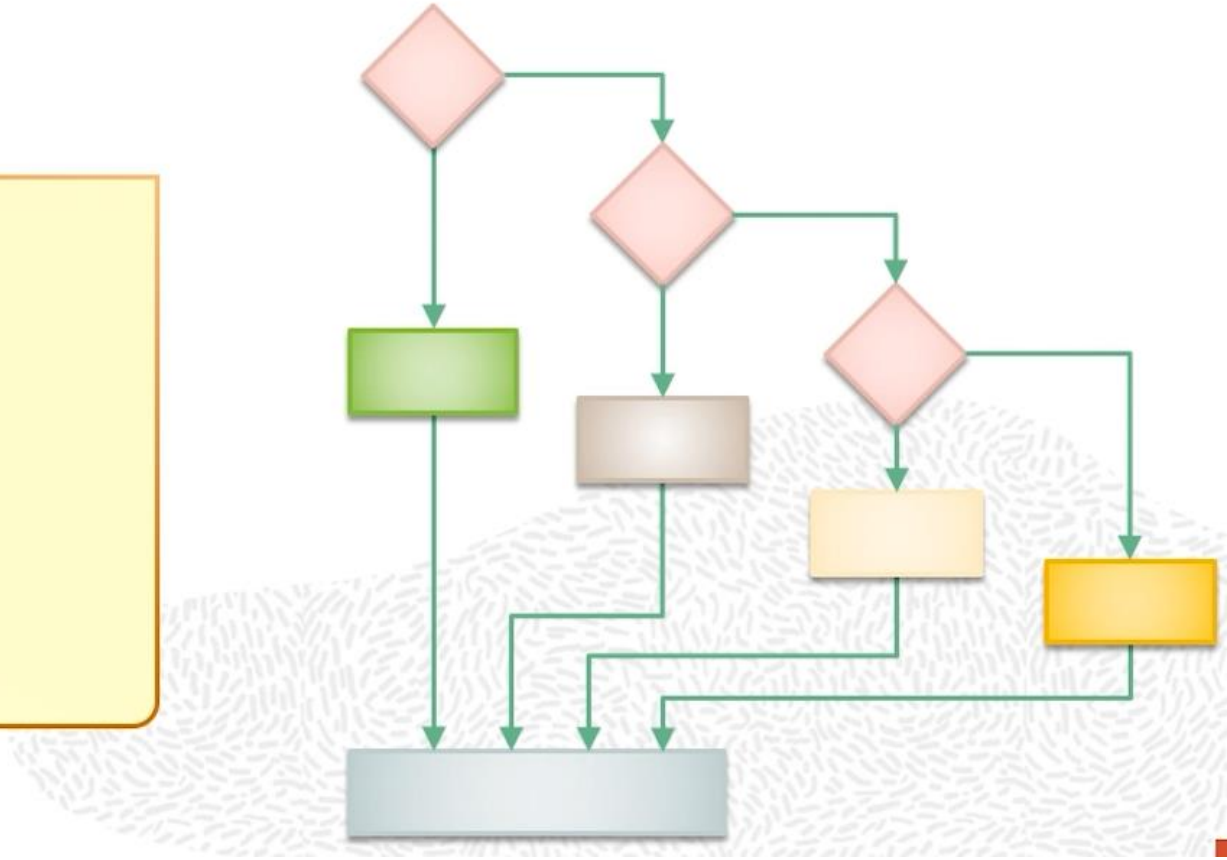
```
PL/SQL procedure successfully completed.

I am not a child
```

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;
```



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;
/
```

```sql
SELECT
    last_name,
    job_id,
    salary,
case job_id
        WHEN 'AD_PRES'   THEN
            salary * 2
        WHEN 'AD_VP' then
            salary * 1.5
        ELSE
            salary *.5
    END AS raise_or_not
FROM
    employees;
```

```sql
SELECT
    last_name,
    job_id,
    salary,
case
        WHEN job_id='AD_PRES'   THEN
            salary * 2
        WHEN job_id='AD_VP' then
            salary * 1.5
        ELSE
            salary *.5
    END AS raise_or_not
FROM
    employees;
```

```sql
SELECT
    last_name,
    job_id,
    salary,
case
        WHEN job_id='AD_PRES'   THEN
            salary * 2
        WHEN job_id='AD_VP' and last_name='Kochhar' then
            salary * 1.5
        ELSE
            salary *.5
    END AS raise_or_not
FROM
    employees;
```

```sql
SELECT
    last_name,
    job_id,
    salary,
case --searched case
        WHEN job_id='AD_PRES'   THEN
            salary * 2
        WHEN job_id='AD_VP' and last_name='Kochhar' then
            salary * 1.5
        ELSE
            salary *.5
    END AS raise_or_not
FROM
    employees;
```

CASE Statement

If 108 matches with the value of v_mngid, an action is performed instead of just returning a value.

```
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;
/
```

The action defined can modify the database also.

Uses END CASE instead of END

```
PL/SQL procedure successfully completed.
You are working in the Finance department. There are 6 employees in this department
```

Handling Nulls

When you are working with `NULL` values, you can avoid some common mistakes by keeping the following rules in mind:

- Simple comparisons involving `NULL`s 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

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

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

TRUE takes precedence in an OR condition.

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

FALSE takes precedence in an AND condition.

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

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

```oracle_plsql
--basic loop always executes at least once
DECLARE
v_val number:=0;
begin
LOOP
exit when v_val > 10;
dbms_output.put_line(v_val );
v_val:=v_val+1;
END LOOP;
END;
```

```oracle_plsql
--while loop always checks before it enters a loop
DECLARE
    v_val NUMBER :=0;
begin
    WHILE v_val < 11 LOOP
        dbms_output.put_line(v_val);
        v_val:= v_val + 1;
END LOOP;
END;
```

```
1  --for loop executes a defined number of times
2  BEGIN
3      FOR i IN 1..10 LOOP
4          dbms_output.put_line(i);
5      END LOOP;
6  END;
```

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;
DBMS_OUTPUT.PUT_LINE(v_counter-1||' rows added.');
END;
/
```

Script Output

Task completed in 0.016 seconds

```
PL/SQL procedure successfully completed.

3 rows added.
```

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;
DBMS_OUTPUT.PUT_LINE(v_counter-1||' rows added.');
END;
/
```

Script Output

Task completed in 0.016 seconds

```
PL/SQL procedure successfully completed.

3 rows added.
```

```
1 --for loop executes a defined number of times
2 BEGIN
3     FOR i IN 1..10 LOOP
4         dbms_output.put_line(i);
5     END LOOP;
6 END;
```

Script Output

Task completed in 0.035 seconds

```
1
2
3
4
5
6
7
8
9
10


PL/SQL procedure successfully completed.
```

```
--for loop executes a defined number of times
BEGIN
    FOR i IN reverse 1..10 LOOP
        dbms_output.put_line(i);
    END LOOP;
END;
```

Script Output

Task completed in 0.022 seconds

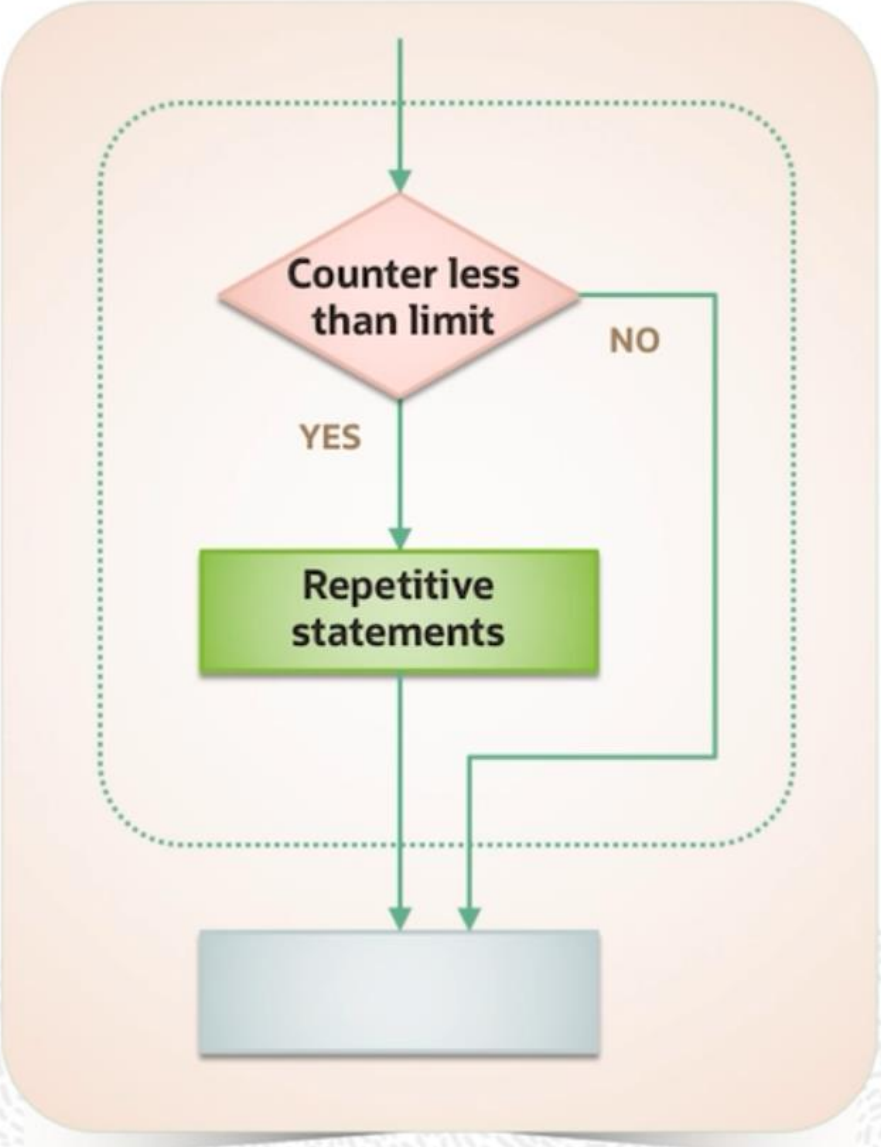
```
10
9
8
7
6
5
4
3
2
1
```

PL/SQL procedure successfully completed.

FOR Loops

- Use a FOR loop when you know 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;
/
```

Result in the LOCATIONS table

	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1901	(null)	(null)	Montreal	(null)	CA
2	1902	(null)	(null)	Montreal	(null)	CA
3	1903	(null)	(null)	Montreal	(null)	CA

`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.
- The loop bound should not 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       done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
/
```

PL/SQL `CONTINUE` Statement

- Definition
  - Adds functionality to begin the next loop iteration
  - Provides programmers with the ability to transfer control to the next iteration of a loop
- 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
    v_total := v_total + i;
    DBMS_OUTPUT.PUT_LINE
     ('Total is: '|| v_total);
    CONTINUE WHEN i > 5;
    v_total := v_total + i;
    DBMS_OUTPUT.PUT_LINE
     ('Out of Loop Total is:
      '|| v_total);
  END LOOP;
END;
/
```

1

2

Executed for each of the 10 iterations of the loop

Executed only for the first 5 iterations of the loop

Script Output

Task completed in 0.004 secon

```
PL/SQL procedure successfully completed.

 Total is : 1
Out of loop total is : 2
 Total is : 4
Out of loop total is : 6
 Total is : 9
Out of loop total is : 12
 Total is : 16
Out of loop total is : 20
 Total is : 25
Out of loop total is : 30
 Total is : 36
 Total is : 43
 Total is : 51
 Total is : 60
 Total is : 70
```

Quiz

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

a. True
b. False