13.
Demonstrate Control structures in PL/SQL

Control structures in PL/SQL

PL/SQL supports various control structures

1. **Conditional control statements**
   - IF statement
2. **Iterative statements**
   - Simple loop
   - While loop
   - For loop
   - Goto

IF statement

- IF statement allows you to either execute or skip a sequence of statements, depending on a condition.
- IF statement has following forms:

IF THEN

IF THEN ELSE

IF THEN ELSIF

IF THEN

1. IF-THEN statement

Syntax:

IF <condition> **THEN**

Set of Statements;

END IF;

- Condition is a Boolean expression that always evaluates to TRUE or FALSE.
- If the condition evaluates to TRUE, the statements after THEN execute. Otherwise, the IF statement does nothing.

IF THEN statement

```
SQL>DECLARE
        v1   NUMBER := 20;
      BEGIN
        IF v1 > 10 THEN
        DBMS_OUTPUT.PUT_LINE( ' v1 is ' || v1);
        END IF;
      END;
      /
```

OUTPUT:

V1 is 5.

IF THEN ELSE statement

2. IF THEN ELSE statement

```
IF  condition THEN
        statements;
ELSE
       else_statements;
END IF;
```

IF THEN ELSE statement

```
SQL>DECLARE
        v1   NUMBER := 2;
    BEGIN
      IF v1 > 10 THEN
            DBMS_OUTPUT.PUT_LINE( ' FROM IF v1 is ‘|| v1);
      ELSE
        DBMS_OUTPUT.PUT_LINE( ' FROM ELSE v1 is ‘|| v1);
      END IF;
    END;
    /
```

OUTPUT:

FROM ELSE V1 is 2.

IF THEN ELSIF statement

SYNTAX:

```
IF    condition_1 THEN
      statements_1
ELSIF   condition_2 THEN
      statements_2
ELSIF   condition_3 THEN
      statements_3
……..
ELSE
      else_statements
END IF;
```

IF THEN ELSIF statement

```
DECLARE  a number  := 10;
BEGIN
        IF ( a = 10 ) THEN
            dbms_output.put_line('Value of a is 10' );
        ELSIF ( a = 20 ) THEN
            dbms_output.put_line('Value of a is 20' );
        ELSIF ( a = 30 ) THEN
            dbms_output.put_line('Value of a is 30' );
        ELSE
            dbms_output.put_line('None of the values is matching');
        END IF;
            dbms_output.put_line('Exact value of a is: '|| a );
END;
/
```

OUTPUT: Value of a is 10

ENO	ENAME	SAL	JOB
1	A	1000	clerk
2	B	1000	clerk
3	C	2000	manager

Write a program to input employee no and update salary based on job.

clerk - 20% , manager -10%.

```
DECLARE
      V_ENO   NUMBER;
      V_JOB    EMP.JOB   %TYPE;
BEGIN
      V_ENO := 1;
      SELECT JOB INTO V_JOB FROM EMP WHERE ENO
    =V_ENO;
      IF  V_JOB ='CLERK'  THEN
      UPDATE  EMP SET SAL=SAL*1.2  WHERE ENO= V_ENO;
      ELSIF  V_JOB='MANAGER'  THEN
      UPDATE  EMP  SET  SAL = SAL*1.1  WHERE ENO= V_ENO;
      END  IF;
END;
```

LOOPS

- A loop statement allows us to execute a statement or group of statements multiple times.

LOOP statement

SYNTAX:

```
LOOP
        SET OF STATEMENTS;
EXIT WHEN  <condition>
END LOOP;
```

LOOP statement

```
SQL>DECLARE
        v1   NUMBER := 1;
    BEGIN
        LOOP
            DBMS_OUTPUT.PUT_LINE( ' v1 is ‘|| v1);
            V1 := V1+1;
        EXIT WHEN  V1>5;
        END LOOP;
    END;
    /
```

While loop

SYNTAX:

```
WHILE  condition
LOOP
statements;
END LOOP;
```

While loop

```
SQL>DECLARE
        v1   NUMBER := 1;
    BEGIN
        WHILE  V1< =5
        LOOP
         DBMS_OUTPUT.PUT_LINE( ' v1 is ‘|| v1);
         V1 := V1+1;
        END LOOP;
    END;
    /
```

While loop

```
SQL>DECLARE
        v1   NUMBER := 1;
    BEGIN
        WHILE  V1< =5
        LOOP
         DBMS_OUTPUT.PUT_LINE( ' v1 is ‘|| v1);
         V1 := V1+1;
          EXIT WHEN V1= 3;    -- breaks loop
        END LOOP;
    END;
    /
```

FOR Loop

SYNTAX:

```
FOR  index  IN  lower_bound .. upper_bound
LOOP
statements;
END LOOP;
```

FOR Loop

- **FOR** LOOP executes a sequence of statements a specified number of times.
- index is an implicit variable. It is local to the FOR LOOP statement. In other words, you cannot reference it outside the loop.
- Both lower_bound and upper_bound are INTEGERS.
- lower_bound is **less than** upper_bound.

FOR Loop

- **Index** is set to lower_bound, the statements in loop execute, and control returns to the top of the loop, where index is compared to upper_bound.
- If index is less than upper_bound, **index** is incremented by one, the statements execute, and control again returns to the top of the loop.
- When index is greater than upper_bound, the loop terminates, and control transfers to the statement after the FOR LOOP statement.

FOR Loop

```
BEGIN
    FOR  X   IN    1..5
    LOOP
        DBMS_OUTPUT.PUT_LINE( X );
    END LOOP;
END;
```

FOR LOOP with REVERSE keyword

```
FOR   index   IN    REVERSE    lower .. upper
LOOP
 statements;
 END LOOP;
```

FOR LOOP with REVERSE keyword

```
BEGIN
FOR x IN REVERSE 1..3
LOOP
DBMS_OUTPUT.PUT_LINE( x );
END LOOP;
END;
```

GOTO statement

- GOTO statement allows you to transfer control to a labeled block or statement.
- GOTO statement makes an unconditional jump from the GOTO to a specific statement label in the PL/SQL block.

Syntax:

GOTO label_name;

GOTO statement

- Label_name identifies the target statement and enclosed within the << >> symbols and must be followed by atleast one statement to execute.
- **Syntax**:

GOTO label_name;

..

..

<<label_name>>

Statement;

GOTO statement

```
DECLARE
      v NUMBER := 1;
 BEGIN
     dbms_output.put_line (‘ Demo on goto statement’ );
     <<xyz>>
     dbms_output.put_line (‘ value of v is  ' || V);
     IF v >= 5 THEN
     RETURN;
     END IF;
     v := v + 1;
     GOTO xyz;
END;
/
```

GOTO statement

```
BEGIN
    GOTO L2;
    <<L1>>
    DBMS_OUTPUT.PUT_LINE( 'Hello we are using GOTO' );
    GOTO L3;
    <<L2>>
    DBMS_OUTPUT.PUT_LINE( 'PL/SQL GOTO Demo' );
    GOTO L1;
    <<L3>>
    DBMS_OUTPUT.PUT_LINE(‘ good bye..' );
END;
```