Ex.No.6

PL/SQL STATEMENTS

AIM

To implement a PL/SQL program to retrieve and display data from a table using looping and exception handling, ensuring error-free execution..

PL/SQL Control Structures:

1. **Simple IF-THEN Statement**

```
SQL> DECLARE
  2     n NUMBER;
  3  BEGIN
  4     n := &n;  -- Take user input
  5
  6     IF n > 0 THEN
  7        DBMS_OUTPUT.PUT_LINE('Given number is greater than ZERO'); -- Use
straight single quotes
  8     END IF;
  9  END;
 10  /
Enter value for n: 5
old   4:    n := &n;  -- Take user input
new   4:    n := 5;  -- Take user input
Given number is greater than ZERO
PL/SQL procedure successfully completed.
```

2.Simple IF-THEN-ELSE Statement

```
SQL> DECLARE
  2     n NUMBER;
  3  BEGIN
  4     n := &n;  -- Take user input
  5
  6    IF n > 0 THEN
 7      DBMS_OUTPUT.PUT_LINE('Given number is Greater than ZERO'); -- Use straight
single quotes
 8    ELSE
 9      DBMS_OUTPUT.PUT_LINE('Given number is Less than ZERO');
10     END IF;
11  END;
12  /
Enter value for n: 4
old   4:    n := &n;  -- Take user input
new   4:    n := 4;  -- Take user input
Given number is Greater than ZERO
PL/SQL procedure successfully completed.
```

3.Nested IF-THEN-ELSE Statements

```
SQL> DECLARE
  2    n number;
  3    BEGIN
  4    n:=&n;
  5    IF n > 0 THEN
  6    Dbms_output.put_line('Given number is Greater than ZERO');
  7    ELSIF n = 0 THEN
  8    Dbms_output.put_line('Given number is Equal to ZERO');
  9    ELSE
 10     Dbms_output.put_line('Given number is Less than ZERO');
 11     END IF;
 12     END;
 13  /
Enter value for n: 0
old   4:       n:=&n;
new   4:       n:=0;
Given number is Equal to ZERO
```

```
PL/SQL procedure successfully completed.
```

4. IF-THEN-ELSIF Statement

```
SQL> DECLARE
  2    n number;
  3    BEGIN
  4    n:=&n;
  5    IF n > 0 THEN
  6    Dbms_output.put_line('Given number is Greater than ZERO');
  7    ELSIF n = 0 THEN
  8    Dbms_output.put_line('Given number is Equal to ZERO');
  9    ELSE
 10    Dbms_output.put_line('Given number is Less than ZERO');
 11    END IF;
 12    END;
 13  /
Enter value for n: 0
old   4:       n:=&n;
new   4:       n:=0;
Given number is Equal to ZERO
PL/SQL procedure successfully completed.
```

***5.*Extended IF-THEN Statement**

```
SQL> DECLARE
  2    grade CHAR(1);  -- Declare the variable
  3  BEGIN
  4    grade := 'B';  -- Assign a value to the variable
  5
  6    IF grade = 'A' THEN
  7      DBMS_OUTPUT.PUT_LINE('Excellent');
  8    ELSIF grade = 'B' THEN
  9      DBMS_OUTPUT.PUT_LINE('Very Good');
```

```
 10    ELSIF grade = 'C' THEN
 11       DBMS_OUTPUT.PUT_LINE('Good');
 12    ELSIF grade = 'D' THEN
 13       DBMS_OUTPUT.PUT_LINE('Fair');
 14    ELSIF grade = 'F' THEN
 15       DBMS_OUTPUT.PUT_LINE('Poor');
 16    ELSE
 17       DBMS_OUTPUT.PUT_LINE('No such grade');
 18    END IF;
 19  END;
 20  /
Very Good
PL/SQL procedure successfully completed.
```

***6.*Simple CASE Statement**

```
 SQL> DECLARE
  2    grade CHAR(1);
  3    BEGIN
  4    grade := 'B';
  5    CASE grade
  6    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
  7    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
  8    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
  9    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
 10     WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
 11     ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
 12     END CASE;
 13     END;
 14     /
Very Good


PL/SQL procedure successfully completed.
```

***7.* Searched CASE Statement**

```
SQL> DECLARE
  2  grade CHAR(1);
  3    BEGIN
  4    grade := 'B';
  5    CASE
  6    WHEN grade = 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
  7    WHEN grade = 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
  8    WHEN grade = 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
  9    WHEN grade = 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
 10    WHEN grade = 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
 11    ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
 12    END CASE;
 13    END;
 14    /
Very Good

PL/SQL procedure successfully completed.
```

8.EXCEPTION Instead of ELSE Clause in CASE Statement

```
SQL> DECLARE
  2       grade CHAR(1);
  3       BEGIN
  4       grade := 'B';
  5       CASE
  6       WHEN grade = 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
  7       WHEN grade = 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
  8       WHEN grade = 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
  9       WHEN grade = 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
 10       WHEN grade = 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
 11       END CASE;
 12       EXCEPTION
 13       WHEN CASE_NOT_FOUND THEN
 14       DBMS_OUTPUT.PUT_LINE('No such grade');
 15       END;
 16       /
Very Good
PL/SQL procedure successfully completed.
```

9.WHILE-LOOP Statement

```
SQL> DECLARE
  2  A NUMBER;
  3  I NUMBER :=1;
  4  BEGIN
  5  A:=10;
  6  WHILE I<A LOOP
  7  DBMS_OUTPUT.PUT_LINE('VALUE :'||I);
  8  I:=I+1;
  9  END LOOP;
 10  END;
 11  /
VALUE :1
VALUE :2
VALUE :3
VALUE :4
VALUE :5
VALUE :6
VALUE :7
VALUE :8
VALUE :9

PL/SQL procedure successfully completed.
```

10.FOR-LOOP Statement

```
SQL> BEGIN
  2      FOR i IN 1..3 LOOP
  3      DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
  4      END LOOP;
  5      END;
  6      /
1
2
3

PL/SQL procedure successfully completed.
```

11. *Reverse FOR-LOOP Statement*

```
SQL> BEGIN
  2      FOR i IN REVERSE 1..3 LOOP
  3      DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
  4      END LOOP;
  5      END;
  6      /
3
2
1

PL/SQL procedure successfully completed.
```

***12.* Simple GOTO Statement**

```
SQL> DECLARE

        n NUMBER := 37;
        p VARCHAR2(30);

        BEGIN

        FOR j IN 2..ROUND(SQRT(n)) LOOP

        IF n MOD j = 0 THEN

         p := ' is NOT a prime number';

         GOTO print_now;  -- Jump to label

         END IF;

         END LOOP;

         p := ' is a prime number';

         DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);

         END;

         /

37 is a prime number

PL/SQL procedure successfully completed.
```

13: GOTO Statement to Branch to an Enclosing Block:

```
SQL> DECLARE
  2      v_last_name  VARCHAR2(25);
  3      v_emp_id     NUMBER(6) := 120;
  4      BEGIN
  5      <<get_name>>
  6      SELECT last_name INTO v_last_name FROM employees
  7      WHERE employee_id = v_emp_id;
  8      BEGIN
  9      DBMS_OUTPUT.PUT_LINE (v_last_name);
 10       v_emp_id := v_emp_id + 5;
 11       IF v_emp_id < 120 THEN
 12       GOTO get_name;
 13       END IF;
 14       END;
 15       END;
 16       /
Smith

PL/SQL procedure successfully completed.
```

***14.* Do…While Statement:**

```
SQL> declare
  2  n_num number := 1;
  3  begin
  4  loop
  5  dbms_output.put(n_num||', ');
  6  n_num := n_num + 1;
  7  exit when n_num > 5;
  8  end loop;
  9  dbms_output.put_line('Final: '||n_num);
 10  end;
 11  /
1, 2, 3, 4, 5, Final: 6

PL/SQL procedure successfully completed.
```

Factorial value

```
SQL> DECLARE
        v_num NUMBER := 5;  -- Input number
        v_fact NUMBER := 1; -- Stores factorial result
        v_counter NUMBER;   -- Counter variable
        BEGIN
        v_counter := v_num; -- Initialize counter
        LOOP
        v_fact := v_fact * v_counter;
        v_counter := v_counter - 1;
        EXIT WHEN v_counter = 0; -- Exit condition (DO-WHILE behavior)
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('Factorial of ' || v_num || ' is: ' || v_fact);
        END;
          /
Factorial of 5 is: 120
PL/SQL procedure successfully completed.
```

Prime Number Generation

```
SQL> DECLARE
          v_n NUMBER := 10; -- Number of prime numbers to generate
          v_count NUMBER := 0;
```

```
v_num NUMBER := 2;
v_is_prime BOOLEAN;
BEGIN
DBMS_OUTPUT.PUT_LINE('First ' || v_n || ' Prime Numbers:');
WHILE v_count < v_n LOOP
 v_is_prime := TRUE;
 FOR i IN 2 .. SQRT(v_num) LOOP
  IF v_num MOD i = 0 THEN
  v_is_prime := FALSE;
  EXIT;
  END IF;
  END LOOP;
  IF v_is_prime THEN
  DBMS_OUTPUT.PUT_LINE(v_num);
  v_count := v_count + 1;
  END IF;
  v_num := v_num + 1;
  END LOOP;
  END;
  /
```

```
First 10 Prime Numbers:
2
3
5
7
11
13
17
19
23
29
```

```
PL/SQL procedure successfully completed.
```

Fibonacci Series

```
SQL> DECLARE
        v_n NUMBER := 10; -- Number of Fibonacci terms
        v_first NUMBER := 0;
        v_second NUMBER := 1;
        v_next NUMBER;
        v_counter NUMBER := 1;
        BEGIN
        DBMS_OUTPUT.PUT_LINE('Fibonacci Series:');
        DBMS_OUTPUT.PUT_LINE(v_first);
       DBMS_OUTPUT.PUT_LINE(v_second);
       WHILE v_counter <= v_n - 2 LOOP
        v_next := v_first + v_second;
        DBMS_OUTPUT.PUT_LINE(v_next);
        v_first := v_second;
        v_second := v_next;
        v_counter := v_counter + 1;
        END LOOP;
        END;
          /
Fibonacci Series:
0
1
1
2
3
5
8
13
21
```

```
34
PL/SQL procedure successfully completed.
```

Checking Palindrome

```
SQL> DECLARE
        v_num NUMBER := 121; -- Input number
        v_reverse NUMBER := 0;
        v_temp NUMBER;
        v_digit NUMBER;
        BEGIN
        v_temp := v_num;
        LOOP
        v_digit := MOD(v_temp, 10);
        v_reverse := (v_reverse * 10) + v_digit;
        v_temp := TRUNC(v_temp / 10);
        EXIT WHEN v_temp = 0; -- Exit condition (DO-WHILE behavior)
        END LOOP;IF v_reverse = v_num THEN
        DBMS_OUTPUT.PUT_LINE(v_num || ' is a Palindrome');
        ELSE
        DBMS_OUTPUT.PUT_LINE(v_num || ' is NOT a Palindrome');
         END IF;
       END;
        /
121 is a Palindrome
PL/SQL procedure successfully completed.
```

PL/SQL block for inserting rows into EMPDET table with the following Calculations:

```
SQL> CREATE TABLE EMPDET (ENO NUMBER PRIMARY KEY, NAME
VARCHAR2(50), DEPTNO NUMBER, BASIC NUMBER, HRA NUMBER, DA
NUMBER, PF NUMBER, NETPAY NUMBER);
```

```
Table created.

SQL> DECLARE
           ENO1 NUMBER := &ENO1;
           ENAME1 VARCHAR2(50) := '&ENAME1';
           DEPTNO1 NUMBER := &DEPTNO1;
           BASIC1 NUMBER := &BASIC1;
           HRA1 NUMBER;
           DA1 NUMBER;
           PF1 NUMBER;
           NETPAY1 NUMBER;
           BEGIN
           HRA1 := (BASIC1 * 50) / 100;
          DA1 := (BASIC1 * 20) / 100;
          PF1 := (BASIC1 * 7) / 100;
          NETPAY1 := BASIC1 + DA1 + HRA1 - PF1;
          INSERT INTO EMPDET (ENO, NAME, DEPTNO, BASIC, HRA, DA, PF, NETPAY)
          VALUES (ENO1, ENAME1, DEPTNO1, BASIC1, HRA1, DA1, PF1, NETPAY1);
         DBMS_OUTPUT.PUT_LINE('Employee record inserted successfully.');
         END;
          /
Enter value for eno1: 102
old   2:     ENO1 NUMBER := &ENO1;
new   2:     ENO1 NUMBER := 102;
Enter value for ename1: Alice
old   3:     ENAME1 VARCHAR2(50) := '&ENAME1';
new   3:     ENAME1 VARCHAR2(50) := 'Alice';
Enter value for deptno1: 20
old   4:     DEPTNO1 NUMBER := &DEPTNO1;
new   4:     DEPTNO1 NUMBER := 20;
```

```
Enter value for basic1: 60000
old   5:     BASIC1 NUMBER := &BASIC1;
new   5:     BASIC1 NUMBER := 60000;
Employee record inserted successfully.

PL/SQL procedure successfully completed.

SQL> DECLARE
        v_last_name  VARCHAR2(25);
        v_emp_id     NUMBER(6) := 120;
        BEGIN
        BEGIN
        BEGIN
       SELECT last_name INTO v_last_name FROM EMPLOYEES WHERE employee_id =
v_emp_id;
        DBMS_OUTPUT.PUT_LINE ('Employee Name: ' || v_last_name);
        EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee found with ID: ' || v_emp_id);
        RETURN;
        END;

          v_emp_id := v_emp_id - 5;
          IF v_emp_id > 100 THEN
          GOTO main_block;
          END IF;
          END;
          END;
           /
Employee Name: Smith
Employee Name: Johnson
Employee Name: Brown
No employee found with ID: 105
PL/SQL procedure successfully completed.
```

CONTENTS	**MARKS ALLOTED**	**MARKS OBTAINED**
Aim,Algorithm,SQL,PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

RESULT

Successfully implemented PL/SQL control structures, including conditional statements, loops, and exception handling. The program efficiently retrieves and processes data while ensuring error-free execution. Various PL/SQL constructs such as IF-THEN-ELSE, CASE statements, and LOOP structures were executed successfully, demonstrating robust procedural control in SQL operations.