CONDITIONAL Statement:

1. Write a PL/SQL block using **IF** to determine the output of the following code:

```
Declare
        num1 number:= 10;
        num2 number:= 20;
BEGIN
        if num1 > num2 then
        dbms_output.put_line('num1 small');
        end if;
        dbms_output.put_line('I am Not in if');
end;
```

2. Write a PL/SQL block using **IF..Then..Else** to determine the output of the following code:

```
   Declare
        num1 number:= 10;
        num2 number:= 20;
BEGIN
        if num1 < num2 then
                dbms_output.put_line('i am in if block');
        ELSE
                dbms_output.put_line('i am in else Block');
        end if;
                dbms_output.put_line('i am not in if or else Block');
        end;
```

Output:

```
        i'm in if Block
        i'm not in if and not in else Block
```

3. Write a PL/SQL block using **NESTED..IF..Then** to determine the output of the following code:

```
Declare

        num1 number:= 10;

        num2 number:= 20;

        num3 number:= 20;

BEGIN

        if num1 < num2 then

        dbms_output.put_line('num1 small num2');

                if num1 < num3 then

                dbms_output.put_line('num1 small num3 also');

                 end if;

        end if;

        dbms_output.put_line('after end if');

        end;
```

Output:

```
        num1 small num2
        num1 small num3 also
        after end if
```

4. Write a PL/SQL block using **IF..Then..ELSEIF..LADDER** to determine the output of the following code:

```
        Declare

        num1 number:= 10;

        num2 number:= 20;

        BEGIN

        if num1 < num2 then

                dbms_output.put_line('num1 small');

        ELSEIF num1 = num2 then

                dbms_output.put_line('both equal');
        ELSE
                dbms_output.put_line('num2 greater');
        end if;
        dbms_output.put_line('after end if');
end;
```

Output:-

```
        num1 small
        after end if
```

5. Write a PL/SQL block using **EXIT** to determine the output of the following code:

```
DECLARE
  counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('This is iteration number ' || counter);
    IF counter = 3 THEN
      EXIT;
    END IF;
    counter := counter + 1;
  END LOOP;
END;
/
```

Output:

```
Statement processed.
This is iteration number 1
This is iteration number 2
This is iteration number 3
```

6. Write a PL/SQL block using **EXIT..When..** to determine the output of the following code:

```
DECLARE
  counter NUMBER := 1;  -- Initialization of the counter variable
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('I LIKE DBMS PROGRAMMING');
    counter := counter + 1;  -- Increment the counter
    EXIT WHEN counter > 5;  -- Exit the loop when counter exceeds 5
  END LOOP;
END;
/
```

Output:

```
Statement processed.
I LIKE DBMS PROGRAMMING should be displayed as per the iteration number.
```

CASE Statement

7. Write a PL/SQL block using **CASE** to determine the grade of a student based on marks:

    a. ≥90 → "Excellent"

    b. 75–89 → "Good"

    c. 60–74 → "Average"

    d. <60 → "Poor"

```
DECLARE
   p_marks NUMBER := 82; -- Declare and initialize the student's marks
   v_grade VARCHAR2(20); -- Variable to store the calculated grade
BEGIN

   CASE
      WHEN p_marks >= 90 THEN
         v_grade := 'Excellent';
      WHEN p_marks BETWEEN 75 AND 89 THEN
         v_grade := 'Good';
```

```
        WHEN p_marks BETWEEN 60 AND 74 THEN
            v_grade := 'Average';
        WHEN p_marks < 60 THEN
            v_grade := 'Poor';
        ELSE
            v_grade := 'Invalid Marks'; -- Handle cases where marks are outside
expected range
    END CASE;

    DBMS_OUTPUT.PUT_LINE('Marks: ' || p_marks || ', Grade: ' || v_grade);
END;
/
```

8. Write a PL/SQL block using **CASE** to calculate bonus for employees:
    a. Manager → 20% of salary
    b. Developer → 15% of salary
    c. Clerk → 10% of salary
    d. Others → 5% of salary

```
DECLARE
  v_job        VARCHAR2(20);
  v_salary     NUMBER := 50000;  -- example salary, you can change this or fetch
from table
  v_bonus      NUMBER;
BEGIN
  -- Example job role, you can change this or fetch from table
  v_job := 'Developer';
  -- Calculate bonus using CASE
  v_bonus := CASE
               WHEN v_job = 'Manager' THEN v_salary * 0.20
               WHEN v_job = 'Developer' THEN v_salary * 0.15
```

```
           WHEN v_job = 'Clerk' THEN v_salary * 0.10
           ELSE v_salary * 0.05
          END;
  DBMS_OUTPUT.PUT_LINE('Job Role: ' || v_job);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || v_salary);
  DBMS_OUTPUT.PUT_LINE('Bonus: ' || v_bonus);
END;
/
```

FOR Loop

9. Write a PL/SQL program using a **FOR loop** to print the multiplication table of a given number (e.g., 5).

```
SET SERVEROUTPUT ON;
DECLARE
   v_number NUMBER := 5; -- The number for which to print the
multiplication table
BEGIN
   DBMS_OUTPUT.PUT_LINE('Multiplication Table of ' || v_number);
   DBMS_OUTPUT.PUT_LINE('---------------------------');
   FOR i IN 1..10 LOOP
      DBMS_OUTPUT.PUT_LINE(v_number || ' * ' || i || ' = ' ||
(v_number * i));
   END LOOP;
END;
/
```

10. Write a PL/SQL block that uses a **FOR loop** to calculate the sum of the first 100 natural numbers.

```
DECLARE
  v_sum NUMBER := 0; -- Declare a variable to store the sum, initialized
to 0
BEGIN
  -- Loop from 1 to 100 (inclusive)
  FOR i IN 1..100 LOOP
    v_sum := v_sum + i; -- Add the current number to the sum
  END LOOP;

  -- Display the final sum
  DBMS_OUTPUT.PUT_LINE ('The sum of the first 100 natural numbers is:
' || v_sum);
END;
/
```

11. Write a PL/SQL block that uses a **Nested FOR loop with continues iteration** to calculate the sum of the first 100 natural numbers.

```
DECLARE
  outer_counter NUMBER := 1;
  inner_counter NUMBER := 1;
BEGIN
  FOR outer_counter IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE('Outer Loop - Iteration ' || outer_counter);
        FOR inner_counter IN 1..2 LOOP
      DBMS_OUTPUT.PUT_LINE('Inner Loop - Iteration ' || inner_counter);
    END LOOP;
  END LOOP;
```

```
END;
/
```

Output:

```
Statement processed.
Outer Loop - Iteration 1
Inner Loop - Iteration 1
Inner Loop - Iteration 2
Outer Loop - Iteration 2
Inner Loop - Iteration 1
Inner Loop - Iteration 2
Outer Loop - Iteration 3
Inner Loop - Iteration 1
Inner Loop - Iteration 2
```

WHILE Loop

12. Write a PL/SQL program using a **WHILE loop** to reverse a given number.

```
SET SERVEROUTPUT ON;
DECLARE
  v_number       NUMBER := 12345; -- The number to be reversed
  v_reversed_num  NUMBER := 0;   -- Stores the reversed number
  v_remainder     NUMBER;        -- Stores the remainder after division
BEGIN
  DBMS_OUTPUT.PUT_LINE('Original number: ' || v_number);
  WHILE v_number > 0 LOOP
    -- Get the last digit (remainder when divided by 10)
    v_remainder := MOD(v_number, 10);
    -- Build the reversed number
    v_reversed_num := (v_reversed_num * 10) + v_remainder;
```

```
          -- Remove the last digit from the original number
          v_number := TRUNC(v_number / 10);
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('Reversed number: ' || v_reversed_num);
      END;
      /
```

13. Write a PL/SQL block that uses a **WHILE loop** to find the factorial of a given number.

```
DECLARE
 p_number    NUMBER := 5; -- The number for which to calculate the factorial
 v_factorial NUMBER := 1; -- Variable to store the calculated factorial
 v_counter   NUMBER;    -- Counter for the loop
BEGIN
 -- Initialize the counter with the input number
 v_counter := p_number;
 -- Handle edge case for 0! which is 1
 IF p_number = 0 THEN
  v_factorial := 1;
 ELSE
  -- Loop while the counter is greater than 0
  WHILE v_counter > 0 LOOP
   v_factorial := v_factorial * v_counter; -- Multiply factorial by current counter
value
   v_counter   := v_counter - 1;         -- Decrement the counter
  END LOOP;
 END IF;
 -- Display the result
```

```
  DBMS_OUTPUT.PUT_LINE('Factorial of ' || p_number || ' is: ' || v_factorial);
END;
/
```

Functions

14. Write a **PL/SQL function** that accepts a student's roll number and returns their total marks from the StudentMarks table.

```
CREATE OR REPLACE FUNCTION GET_STUDENT_TOTAL_MARKS (
   p_roll_number IN NUMBER)
RETURN NUMBER
IS
   v_total_marks NUMBER;
BEGIN
   SELECT SUM(MARKS)
   INTO v_total_marks
   FROM Student_Marks
   WHERE ROLL_NUMBER = p_roll_number;
   RETURN v_total_marks;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      -- Handle the case where no marks are found for the given roll
number
      RETURN 0; -- Or raise an application error, or return NULL
   WHEN OTHERS THEN
      -- Handle other potential errors
      RAISE; -- Re-raise the exception
END;
```

/

15. Write a **function** that accepts an employee ID and returns their annual salary (monthly salary × 12).

```
def get_annual_salary(employee_id):
    """
    Accepts an employee ID and returns their annual salary (monthly salary * 12).
    Args:
        employee_id (str or int): The unique identifier for the employee.
    Returns:
        float: The annual salary of the employee.
               Returns None if the employee ID is not found.
    """
    # In a real-world scenario, this would involve querying a database
    # or an external system to retrieve the monthly salary based on the employee_id.
    employee_data = {
        "EMP001": {"monthly_salary": 5000.00},
        "EMP002": {"monthly_salary": 6500.00},
        "EMP003": {"monthly_salary": 4800.00},
    }
    if employee_id in employee_data:
        monthly_salary = employee_data[employee_id]["monthly_salary"]
        annual_salary = monthly_salary * 12
        return annual_salary
    else:
        print(f"Error: Employee with ID '{employee_id}' not found.")
        return None

# Example usage:
employee_id_1 = "EMP001"
annual_salary_1 = get_annual_salary(employee_id_1)
if annual_salary_1 is not None:
    print(f"The annual salary for employee {employee_id_1} is:
${annual_salary_1:.2f}")
employee_id_2 = "EMP004"
annual_salary_2 = get_annual_salary(employee_id_2)
if annual_salary_2 is not None:
    print(f"The annual salary for employee {employee_id_2} is:
${annual_salary_2:.2f}")
```

Procedures

16. Write a **PL/SQL procedure** to insert a new record into the Employees(emp_id, name, salary) table.

```
CREATE OR REPLACE PROCEDURE insert_employee (
    p_emp_id   IN NUMBER,
    p_name     IN VARCHAR2,
    p_salary   IN NUMBER)
IS
BEGIN
    INSERT INTO Employees (emp_id, name, salary)
    VALUES (p_emp_id, p_name, p_salary);
    COMMIT; -- Commit the transaction to make the changes permanent
```

```
   DBMS_OUTPUT.PUT_LINE('Employee ' || p_name || ' with ID ' ||
p_emp_id || ' inserted successfully.');
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || p_emp_id ||
' already exists.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' ||
SQLERRM);
      ROLLBACK; -- Rollback the transaction in case of other errors
END insert_employee;
/
```

17. Write a **procedure** that accepts an account number and an amount, and updates the balance after depositing the amount.

```
class BankAccount:
   def __init__(self):
      self.balance = 0
      print("Welcome to the Machine")

   def deposit(self):
      amount = float(input("Enter amount to be Deposited: "))
      self.balance += amount
      print("\nAmount Deposited:", amount)

   def withdraw(self):
      amount = float(input("Enter amount to be Withdrawn: "))
      if self.balance >= amount:
            self.balance -= amount
            print("\nYou Withdrew:", amount)
        else:
            print("\nInsufficient balance")

    def display(self):
        print("\nNet Available Balance =", self.balance)

# Driver code
if __name__ == "__main__":
    s = BankAccount()  # Create an object of BankAccount

    s.deposit()       # Deposit money
    s.withdraw()      # Withdraw money
    s.display()       # Display balance
```

Expected Output:

Output:

```
Welcome to the Machine
Enter amount to be Deposited: 1000
Amount Deposited: 1000.0
Enter amount to be Withdrawn: 500
You Withdrew: 500.0
Net Available Balance = 500.0
```