TASK 6 :

Procedures, Functions, and Loops in PL/SQL (Based on Online Food Ordering System)

Step 1: Ensure the Necessary Tables Exist

```
DROP TABLE OrderTable PURGE;
DROP TABLE Delivery PURGE;
DROP TABLE Menu_Item PURGE;

CREATE TABLE OrderTable (
Order_ID NUMBER PRIMARY KEY,
Cust_ID NUMBER,
Order_Date DATE,
Order_Total NUMBER(10,2),
Payment_Status VARCHAR2(20)
);
```

EXPECTED OUTPUT: Table created

```
CREATE TABLE Delivery (
Order_ID NUMBER PRIMARY KEY,
Delivery_Status VARCHAR2(20),
FOREIGN KEY (Order_ID) REFERENCES OrderTable(Order_ID)
);
```

EXPECTED OUTPUT: Table created

```
CREATE TABLE Menu_Item (
Item_ID NUMBER PRIMARY KEY,
Item_Name VARCHAR2(100),
Price NUMBER(10,2)
);
```

EXPECTED OUTPUT: Table created

```
INSERT INTO OrderTable VALUES (1, 101, TO_DATE('2024-02-01', 'YYYY-MM-DD'), 250.50, 'Pending');
```

OUTPUT: 1 row created

```
INSERT INTO OrderTable VALUES (2, 102, TO_DATE('2024-02-02', 'YYYY-MM-DD'), 400.75, 'Paid');
```

OUTPUT: 1 row created

```
INSERT INTO OrderTable VALUES (3, 103, TO_DATE('2024-02-03', 'YYYY-MM-DD'), 150.00, 'Pending');
```

OUTPUT: 1 row created

```
INSERT INTO Delivery VALUES (1, 'Pending');
```

OUTPUT: 1 row created

```
INSERT INTO Delivery VALUES (2, 'Delivered');
```

OUTPUT: 1 row created

```
INSERT INTO Delivery VALUES (3, 'Pending');
```

OUTPUT: 1 row created

INSERT INTO Menu_Item VALUES (1, 'Pizza', 500);

OUTPUT: 1 row created

INSERT INTO Menu_Item VALUES (2, 'Burger', 300);

OUTPUT: 1 row created

INSERT INTO Menu_Item VALUES (3, 'Pasta', 450);

OUTPUT: 1 row created

1. **Procedure to Update Payment Status**
   Step 1: Create a Procedure

```
CREATE OR REPLACE PROCEDURE Update_Payment_Status (
   p_Order_ID   IN NUMBER,
   p_New_Status IN VARCHAR2
) AS
BEGIN
   UPDATE OrderTable
   SET Payment_Status = p_New_Status
   WHERE Order_ID = p_Order_ID;

   COMMIT;

   DBMS_OUTPUT.PUT_LINE( 'Payment status updated successfully for Order ID: ' ||
p_Order_ID   );
END;
/
```

Expected Output:

Procedure created

Step 2: Execution

```
SET SERVEROUTPUT ON

BEGIN
        Update_Payment_Status(1, 'Paid');
END;
/
```

Expected Output:

```
Payment status updated successfully for Order ID: 1

Statement processed.
```

Query 2: Function to Calculate Total Revenue

Step 1: Create a Function

```
CREATE OR REPLACE FUNCTION Get_Total_Revenue
RETURN NUMBER
AS
   v_Total_Revenue NUMBER;
BEGIN
   SELECT SUM(Order_Total)
   INTO v_Total_Revenue
   FROM OrderTable;

   RETURN v_Total_Revenue;
END;
/
```

Expected Output:

Function created

Step 2: Execution

```
SELECT Get_Total_Revenue FROM dual;
```

Expected Output:

GET_TOTAL_REVENUE()
801.25

Query 3: Loop: Mark All Undelivered Orders as "Delayed"

```
DECLARE
   v_Order_ID   OrderTable.Order_ID%TYPE;
   CURSOR cur IS
      SELECT Order_ID
      FROM Delivery
      WHERE Delivery_Status = 'Pending';
BEGIN
   OPEN cur;
   LOOP
      FETCH cur INTO v_Order_ID;
      EXIT WHEN cur%NOTFOUND;

      UPDATE Delivery
      SET Delivery_Status = 'Delayed'
      WHERE Order_ID = v_Order_ID;

      DBMS_OUTPUT.PUT_LINE('Order ' || v_Order_ID || ' marked as Delayed.');
   END LOOP;
   CLOSE cur;

   COMMIT;
END;
/
```

Expected Output:

procedure successfully completed.

Order 1 marked as Delayed.

Order 3 marked as Delayed.

Query 4: Procedure to Get Order Details by Customer ID

Step 1: Create a Procedure

```
CREATE OR REPLACE PROCEDURE Get_Customer_Orders (
   p_Cust_ID IN NUMBER
) AS
BEGIN
   FOR order_rec IN (
      SELECT Order_ID, Order_Date, Order_Total, Payment_Status
      FROM OrderTable
      WHERE Cust_ID = p_Cust_ID
   ) LOOP
      DBMS_OUTPUT.PUT_LINE(
         'Order ID: ' || order_rec.Order_ID ||
         ', Date: ' || order_rec.Order_Date ||
         ', Total: ' || order_rec.Order_Total ||
         ', Status: ' || order_rec.Payment_Status
      );
   END LOOP;
END;
/
```

Expected Output:

```
Procedure created.
```

Step 2: Execution

```
BEGIN
Get_Customer_Orders(101);
END;
```

Expected Output:

```
Order ID: 1, Date: 2024-02-01, Total: 250.5, Payment: Paid
Statement processed.
```

Query 5: Procedure to Apply Discount on Menu Items

Step 1: Create a Procedure

```
CREATE OR REPLACE PROCEDURE Apply_Discount (
    discount_percent IN NUMBER
)
IS
BEGIN
    UPDATE Menu_Item
    SET Price = Price - (Price * discount_percent / 100);

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Discount Applied: ' || discount_percent || '%');
END;
/
```

Expected Output:

```
Procedure created.
```

Step 2: Execution

```
BEGIN
    Apply_Discount(10);
END;
/
```

Expected Output:

```
Discount Applied: 10%

Statement processed.
```