Lab Manual

Task 6: Procedures, Functions, and Loops in PL/SQL (Based on Online Food Ordering System)

Case Study: Online Food Ordering System

Objective:

The objective of this task is to **design, implement, and execute PL/SQL procedures, functions, and loops** to handle real-world business scenarios related to **an online food ordering system**. This will help in **automating transactions, improving database efficiency, and enforcing business rules** in a structured manner.

Step 1: Ensure the Necessary Tables Exist

Before running the procedures and functions, **create the required tables** in your Oracle Database.

```
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)
);

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

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

INSERT INTO OrderTable VALUES (1, 101, TO_DATE('2024-02-01', 'YYYY-MM-DD'),
250.50, 'Pending');
INSERT INTO OrderTable VALUES (2, 102, TO_DATE('2024-02-02', 'YYYY-MM-DD'),
400.75, 'Paid');
INSERT INTO OrderTable VALUES (3, 103, TO_DATE('2024-02-03', 'YYYY-MM-DD'),
150.00, 'Pending');

INSERT INTO Delivery VALUES (1, 'Pending');
INSERT INTO Delivery VALUES (2, 'Delivered');
INSERT INTO Delivery VALUES (3, 'Pending');
```

```
INSERT INTO Menu_Item VALUES (1, 'Pizza', 500);
INSERT INTO Menu_Item VALUES (2, 'Burger', 300);
INSERT INTO Menu_Item VALUES (3, 'Pasta', 450);
```

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

```
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

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:

```
1 row(s) updated.
```

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_Order_Details_By_Customer(1);  -- Replace '1' with any Customer ID
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.
```