**Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology**

**(Deemed to be University Estd. u/s 3 of UGC Act, 1956)**

**School of Computing**

**B.Tech. – Computer Science and Engineering**

**VTR UGE2021- (CBCS)**

Academic Year: 2025–2026

SUMMER SEMESTER - SS2526

Course Code : 10211CS207

Course Name: Database Management Systems

Slot No : S2L5

DBMS TASK - 6 REPORT

**Title:** Writing Join Queries, Equivalent, and/or Recursive Queries

**Submitted by:**

|  |  |  |
| --- | --- | --- |
| **VTUNO** | **REGISTER NUMBER** | **STUDENT NAME** |
| VTU29510 | 24UECS0249 | P RAM CHARAN |

**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-MMDD'), 250.50, 'Pending');

## **OUTPUT:** 1 row created

INSERT INTO OrderTable VALUES (2, 102, TO\_DATE('2024-02-02', 'YYYY-MMDD'), 400.75, 'Paid');

## **OUTPUT:** 1 row created

INSERT INTO OrderTable VALUES (3, 103, TO\_DATE('2024-02-03', 'YYYY-MMDD'), 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**

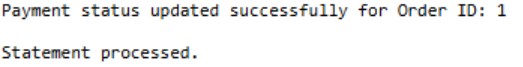
SET SERVEROUTPUT ON

BEGIN

Update\_Payment\_Status(1, 'Paid');

END; /

**Expected Output:**



**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; /

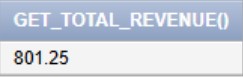
**Expected Output:**

Function created

**Step 2: Execution**

SELECT Get\_Total\_Revenue FROM dual;

**Expected Output:**



**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:**



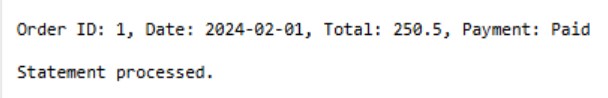
**Step 2: Execution**

BEGIN

Get\_Customer\_Orders(101);

END;

**Expected Output:**



**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 || '%');

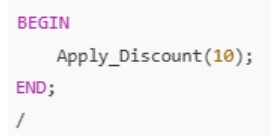
END;

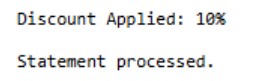
/

**Expected Output:**



**Step 2: Execution**



**Expected Output:**

# **RESULT:**Thus the program is executed and verified sucessfully