***Advanced DBMS LAB***

***CIE 310P***

Faculty name: Mr. Varun Goel Student name: Vedant Nagar Designation: Assistant Professor Roll No.:03614812721

Semester: VI

Group: FSD-2c

Department: CST

A red circle with a white circle with a red center with a white circle with a red center with a white circle with a red center with a white circle with a red center with a white circle

Description automatically generated

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110086

**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

**MISSION**

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

**Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

**Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

**Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

**Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

**Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi- disciplinary approach.

**Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successfully entrepreneurs and responsible citizens.

**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**Computer Science and Technology**

**Vision**

To stand as a beacon of excellence dedicated to fostering education and research in Information Technology & Engineering, ultimately cultivating ethical-driven technocrats and visionary entrepreneurs.

**Mission**

To create an environment promoting both technical and professional advancement, producing graduates who are globally competent and industry-ready professionals, researchers, and entrepreneurs to solve real-world problems and societal issues.

**Program Specific Outcomes (PSOs)**

PSO1: Skilled in identifying appropriate data structures and algorithms to develop, implement, and validate efficient solutions for practical and research-based challenges.

PSO 2 Proficient in thriving in diverse programming and meeting technical challenges presented by professional organizations.

PSO 3: Able to acquire knowledge in various Information Technology and Engineering domains and utilize it for a successful career, entrepreneurship, and advanced education

**Program Educational Objectives (PEOs)**

PEO1: Our graduates will embark on prosperous careers in the field of information technology and engineering or excel in advanced academic pursuits.

PEO2: Our graduates will possess the capacity to ingeniously address complex real-world challenges through the application of Information technology and engineering principles.

PEO3: Our graduates will exemplify unwavering professionalism, showcasing robust interpersonal and collaborative skills while adhering to the highest ethical standards.

PEO4: Our graduates will adeptly identify research opportunities, engage in lifelong learning, and excel as entrepreneurial innovators.

**CO of the lab subject and its mapping with PO AND PSO**

**Course outcomes**

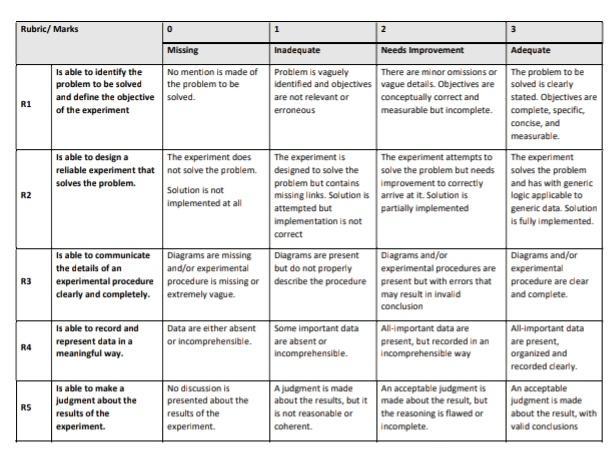
|  |  |
| --- | --- |
| **CIE-310P** | **Advanced DBMS Lab** |
| **CIE-310P.1** | Discuss the concept of creation of tables using constraints, Horization and Vertical fragmentation. |
| **CIE-310P.2** | Illustrate the concept related to nested and correlated subqueries. |
| **CIE-310P.3** | Analyze the queries based on views |
| **CIE-310P.4** | Evaluate performance of creation of procedures, triggers on tables and views. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CIE-310P** | **Advanced DBMS (CO-PO mapping)** | | | | | | | | | | | |
|  | **PO01** | **PO02** | **PO03** | **PO04** | **PO05** | **PO06** | **PO07** | **PO08** | **PO09** | **PO10** | **PO11** | **PO12** |
| **CIE-310P.1** | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 2 | 3 |
| **CIE-310P.2** | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 2 | 3 |
| **CIE-310P.3** | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 2 | 3 |
| **CIE-310P.4** | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 2 | 3 |
| **CIE-310P** | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 2 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **CIE-310P** | **(CO-PSO mapping)** | | |
|  | **PSO1** | **PSO2** | **PSO3** |
| **CIE-310P.1** | 3 | 2 | 3 |
| **CIE-310P.2** | 3 | 2 | 3 |
| **CIE-310P.3** | 3 | 2 | 3 |
| **CIE-310P.4** | 3 | 2 | 3 |
| **CIE-310P** | 3 | 2 | 3 |

**Rubrics Evaluation**

**Computer Science and Technology Department**



**Advanced DBMS Lab**

**PRACTICAL RECORD**

**PAPER CODE :** CIE 310P

University Roll No. :03614812721

Branch : CST

Section/ Group : FSD-2c

**PRACTICAL DETAILS**

1. Experiments according to the list provided by GGSIPU

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment No.** | **Date** | **Experiment Name** | **Marks (0-3)** | | | | | **Total Marks (15)** | **Signature** |
|  |  |  | **R1** | **R2** | **R3** | **R4** | **R5** |  |  |
| 1. |  |  |  |  |  |  |  |  |  |
| 2. |  |  |  |  |  |  |  |  |  |
| 3. |  |  |  |  |  |  |  |  |  |
| 4. |  |  |  |  |  |  |  |  |  |
| 5. |  |  |  |  |  |  |  |  |  |
| 6. |  |  |  |  |  |  |  |  |  |
| 7. |  |  |  |  |  |  |  |  |  |
| 8. |  |  |  |  |  |  |  |  |  |
| 9. |  |  |  |  |  |  |  |  |  |
| 10. |  |  |  |  |  |  |  |  |  |

1. Experiments beyond the list provided by GGSIPU

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment No.** | **Date** | **Experiment Name** | **Marks (0-3)** | | | | | **Total Marks(15)** | **Signature** |
|  |  |  | **R1** | **R2** | **R3** | **R4** | **R5** |  |  |
| 1. |  |  |  |  |  |  |  |  |  |
| 2. |  |  |  |  |  |  |  |  |  |
| 3. |  |  |  |  |  |  |  |  |  |
| 4. |  |  |  |  |  |  |  |  |  |

# Experiment 1

**Aim –** Creation of tables using all types of constraints

**Querry -** CREATE TABLE Employees ( EmployeeID INT PRIMARY KEY,

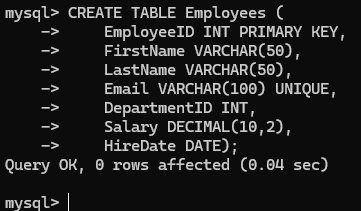
FirstName VARCHAR(50), LastName VARCHAR(50),

Email VARCHAR(100) UNIQUE,

DepartmentID INT, Salary DECIMAL(10,2),

HireDate DATE);

**Output –**



# Experiment 2

**Aim –** Queries based on set of operators with respect to horizontal fragmentation of table

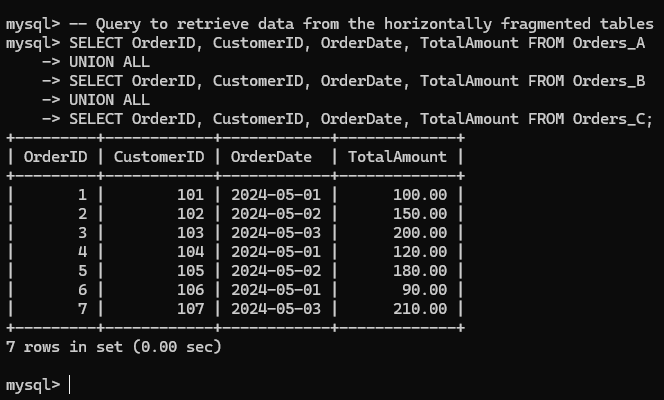
**Querry –**

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_A UNION ALL

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_B UNION ALL

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_C;

**Output –**



# Experiment 3

**Aim –** Queries based on join of more that two tables with respect to horizontal fragments of tables.

**Querry –**

**Union** - SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_A

UNION

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_B UNION

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_C;

**Intersect -** SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_A

INTERSECT

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_B INTERSECT

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_C;

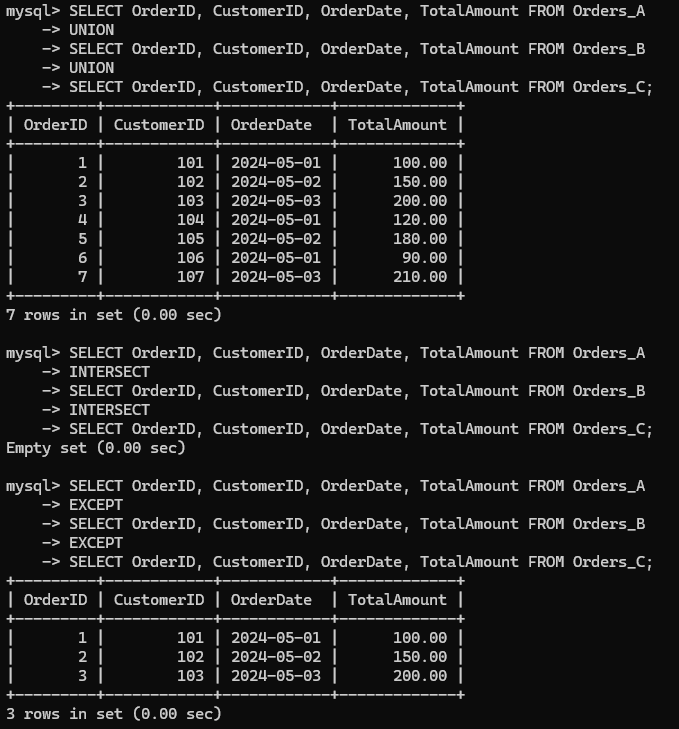
**Except -** SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_A

INTERSECT

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_B INTERSECT

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders\_C;

**Output –**



# Experiment 4

**Aim –** Queries based on nested subqueries.

**Querry –**

## SELECT -

SELECT CustomerID,

(SELECT COUNT(\*) FROM Orders WHERE Orders.CustomerID = Customers.CustomerID) AS TotalOrders

FROM Customers;

## WHERE –

SELECT OrderID, CustomerID, OrderDate, TotalAmount FROM Orders

WHERE TotalAmount > (SELECT AVG(TotalAmount) FROM Orders);

## FROM –

SELECT CustomerID, CustomerName FROM (

SELECT CustomerID, COUNT(\*) AS TotalOrders FROM Orders

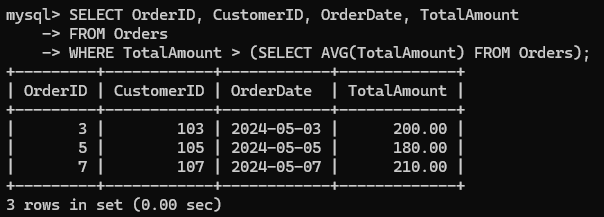
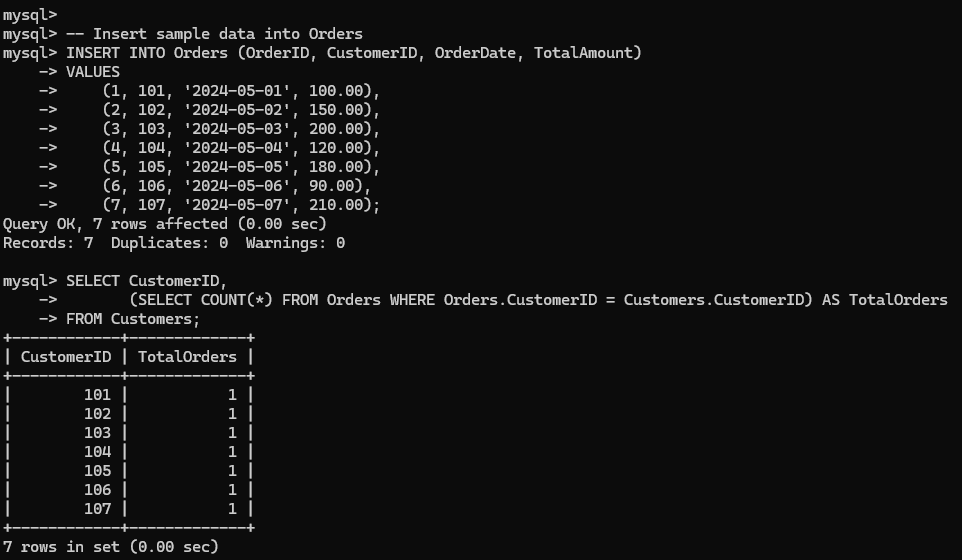
GROUP BY CustomerID

) AS OrderCounts

JOIN Customers ON Customers.CustomerID = OrderCounts.CustomerID

WHERE TotalOrders > (SELECT AVG(TotalOrders) FROM (SELECT CustomerID, COUNT(\*) AS TotalOrders FROM Orders GROUP BY CustomerID) AS AvgOrderCounts);

**Output –**



# Experiment 5

**Aim –** Queries based on corelated subqueries.

**Querry –**

## WHERE –

SELECT CustomerID, CustomerName FROM Customers C

WHERE EXISTS ( SELECT 1

FROM Orders O

WHERE O.CustomerID = C.CustomerID AND O.TotalAmount > 200

);

## SELECT –

SELECT CustomerID, CustomerName, (SELECT COUNT(\*)

FROM Orders O

WHERE O.CustomerID = C.CustomerID) AS TotalOrders FROM Customers C;

## HAVING –

SELECT CustomerID, CustomerName FROM Customers C

WHERE (

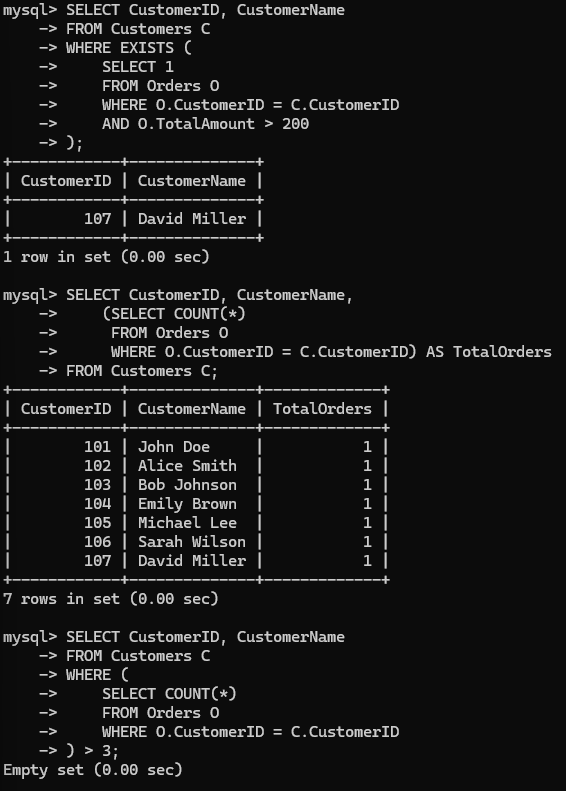
SELECT COUNT(\*)

FROM Orders O

WHERE O.CustomerID = C.CustomerID

) > 3;

**Output –**



# Experiment 6

**Aim –** Creation of views based on multiple tables.

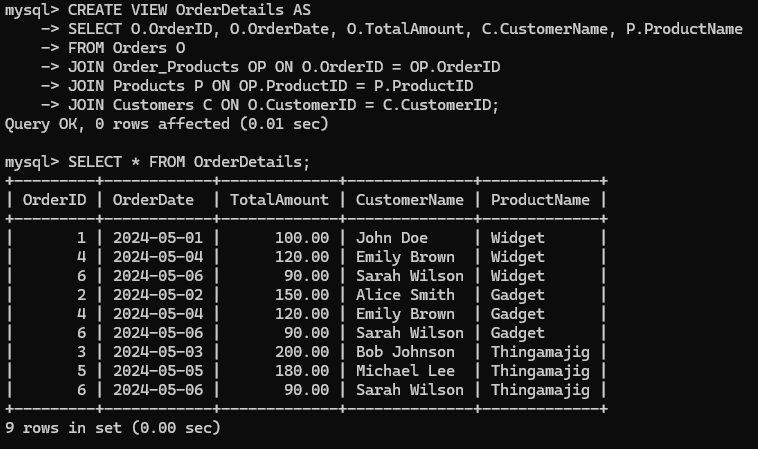
**Querry -** -- Create a view to combine information from Orders, Customers, and Products

CREATE VIEW OrderDetails AS

SELECT O.OrderID, O.OrderDate, O.TotalAmount, C.CustomerName, P.ProductName

FROM Orders O

JOIN Customers C ON O.CustomerID = C.CustomerID JOIN Products P ON O.ProductID = P.ProductID; **Output –**



# Experiment 7

**Aim –** Queries based on views.

**Querry -**

**SELECT** – SELECT \* FROM OrderDetails;

**FILTERING** - SELECT \* FROM OrderDetails WHERE OrderDate > '2024- 05-01';

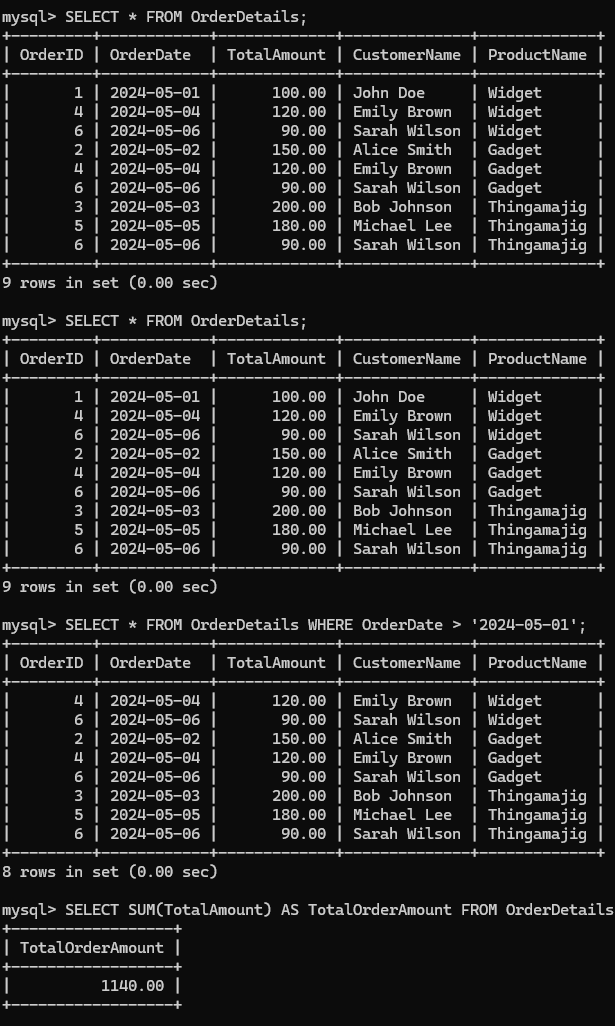
**AGGREGATION** - SELECT SUM(TotalAmount) AS TotalOrderAmount FROM OrderDetails;

**JOIN** - SELECT OD.OrderID, OD.OrderDate, OD.TotalAmount, OD.CustomerName, C.Email

FROM OrderDetails OD

JOIN Customers C ON OD.CustomerID = C.CustomerID;

**Output –**



# Experiment 8

**Aim –** Creation of procedures using cursor with exception handling.

**Querry -** DELIMITER /

CREATE PROCEDURE ProcessOrders() BEGIN

DECLARE done BOOLEAN DEFAULT FALSE;

DECLARE order\_id INT;

DECLARE total\_amount DECIMAL(10, 2);

-- Declare cursor for selecting orders DECLARE cur\_orders CURSOR FOR

SELECT OrderID, TotalAmount FROM Orders;

-- Declare handler for SQL exceptions

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

-- Open cursor OPEN cur\_orders;

-- Start processing orders read\_loop: LOOP

-- Fetch next order

FETCH cur\_orders INTO order\_id, total\_amount;

-- Check if no more rows to fetch IF done THEN

LEAVE read\_loop; END IF;

-- Process the order (example: print order details)

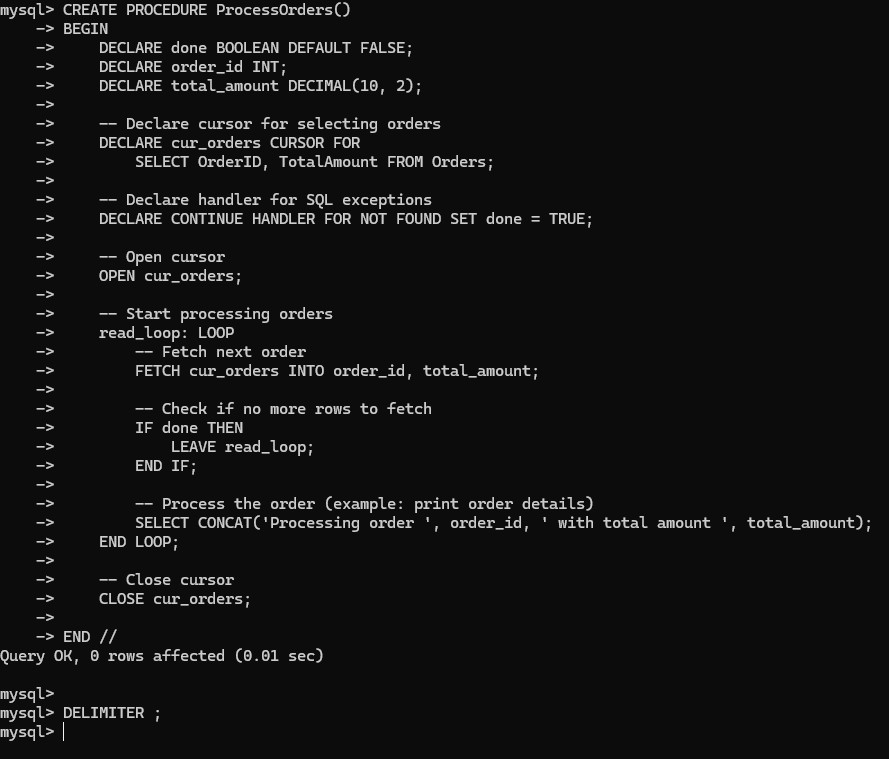
SELECT CONCAT('Processing order ', order\_id, ' with total amount ', total\_amount);

END LOOP;

-- Close cursor CLOSE cur\_orders;

END // DELIMITER ;

**Output –**



# Experiment 9

**Aim –** Creation of triggers on tables.

**Querry –**

## TRIGGER FOR DELETE OPERATION-

DELIMITER //

CREATE TRIGGER after\_order\_delete AFTER DELETE ON Orders

FOR EACH ROW BEGIN

INSERT INTO Order\_Log (OrderID, LogMessage, LogDateTime) VALUES (OLD.OrderID, 'Order deleted', NOW());

END;

// DELIMITER ;

## TRIGGER FOR UPDATE OPERATIONS –

DELIMITER //

CREATE TRIGGER after\_order\_update AFTER UPDATE ON Orders

FOR EACH ROW BEGIN

INSERT INTO Order\_Log (OrderID, LogMessage, LogDateTime)

VALUES (NEW.OrderID, CONCAT('Order updated. TotalAmount changed to ', NEW.TotalAmount), NOW());

END;

// DELIMITER ;

## TRIGGER FOR INSERT OPERATIONS –

DELIMITER //

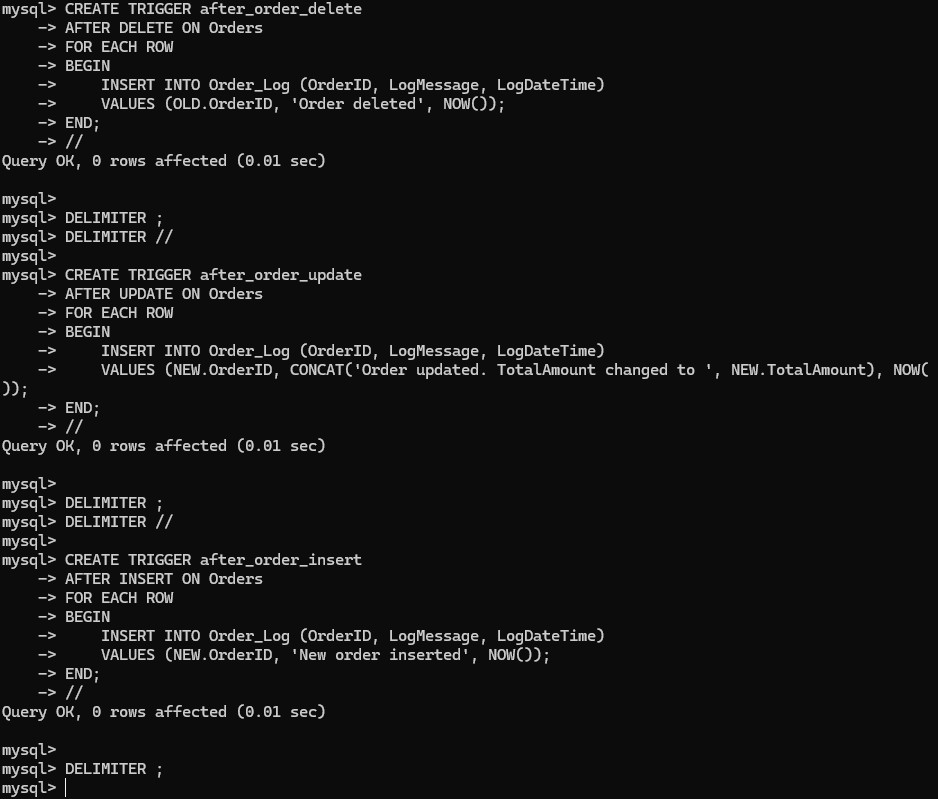
CREATE TRIGGER after\_order\_insert AFTER INSERT ON Orders

FOR EACH ROW BEGIN

INSERT INTO Order\_Log (OrderID, LogMessage, LogDateTime) VALUES (NEW.OrderID, 'New order inserted', NOW());

END;

// DELIMITER ;

**Output –**

# Experiment 10

**Aim –** Creation of triggers on views.

**Querry -** DELIMITER //

CREATE TRIGGER after\_employee\_insert AFTER INSERT ON Employees

FOR EACH ROW BEGIN

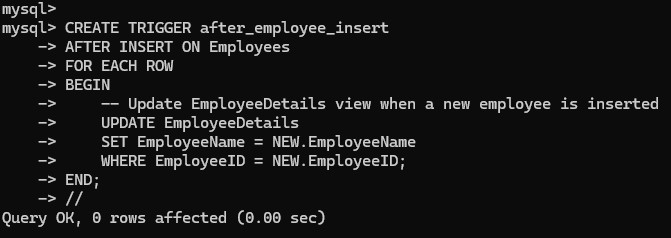
-- Update EmployeeDetails view when a new employee is inserted UPDATE EmployeeDetails

SET EmployeeName = NEW.EmployeeName WHERE EmployeeID = NEW.EmployeeID;

END;

// DELIMITER ;

**Output –**



**BEYOND SYLLABUS EXPERIMETS**

# Experiment 1

**Aim –** In Distributed Database : Implementation of Partitions using Range and List.

**Querry –**

**Range Partitioning (based on OrderDate):**

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT,

OrderDate DATE, CustomerID INT,

TotalAmount DECIMAL(10, 2), PRIMARY KEY (OrderID, OrderDate)

)

PARTITION BY RANGE (YEAR(OrderDate)) ( PARTITION p0 VALUES LESS THAN (2010), PARTITION p1 VALUES LESS THAN (2011), PARTITION p2 VALUES LESS THAN (2012), PARTITION p3 VALUES LESS THAN (2013)

);

**List Partitioning (based on Region):**

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT,

OrderDate DATE, CustomerID INT,

TotalAmount DECIMAL(10, 2), Region VARCHAR(50),

PRIMARY KEY (OrderID, Region), -- Include both OrderID and Region in the primary key

KEY (Region) -- Add an index on the partitioning column

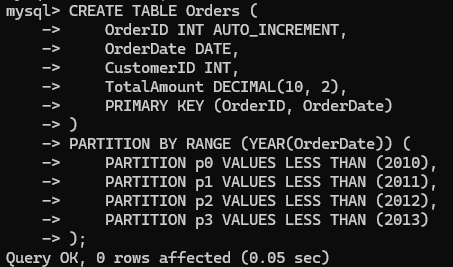
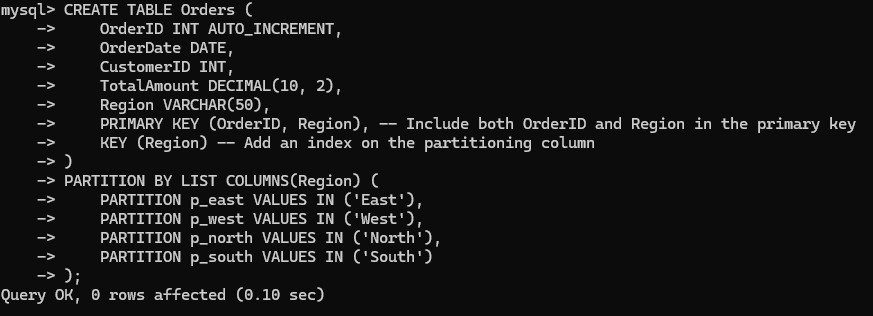
)

PARTITION BY LIST COLUMNS(Region) ( PARTITION p\_east VALUES IN ('East'), PARTITION p\_west VALUES IN ('West'),

PARTITION p\_north VALUES IN ('North'), PARTITION p\_south VALUES IN ('South')

);

**Output –**



# Experiment 2

**Aim –** Implementation of Inheritance in SQL.

**Querry –**

**Create the Parent Table:**

CREATE TABLE Person (

PersonID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50), LastName VARCHAR(50), Gender VARCHAR(10),

BirthDate DATE

);

**Create Child Tables:**

CREATE TABLE Employee (

EmployeeID INT AUTO\_INCREMENT PRIMARY KEY,

PersonID INT,

Position VARCHAR(50), Salary DECIMAL(10, 2),

FOREIGN KEY (PersonID) REFERENCES Person(PersonID)

);

CREATE TABLE Customer (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

PersonID INT,

MembershipLevel VARCHAR(50),

FOREIGN KEY (PersonID) REFERENCES Person(PersonID)

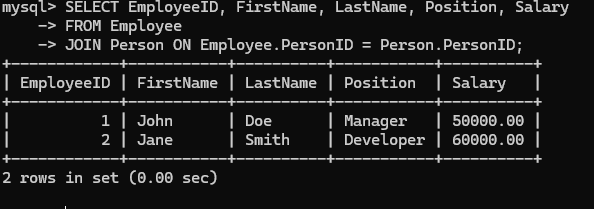
);

**Querying Data:**

SELECT EmployeeID, FirstName, LastName, Position, Salary FROM Employee

JOIN Person ON Employee.PersonID = Person.PersonID;

**Output –**



# Experiment 3

**Aim –** Implement Dynamic SQL.

**Querry –**

CREATE PROCEDURE SearchEmployees @FirstName VARCHAR(50) = NULL, @LastName VARCHAR(50) = NULL,

@DepartmentID INT = NULL AS

BEGIN

DECLARE @SQL NVARCHAR(MAX)

SET @SQL = 'SELECT EmployeeID, FirstName, LastName, DepartmentID FROM Employees WHERE 1=1'

IF @FirstName IS NOT NULL

SET @SQL = @SQL + ' AND FirstName = @FirstName'

IF @LastName IS NOT NULL

SET @SQL = @SQL + ' AND LastName = @LastName'

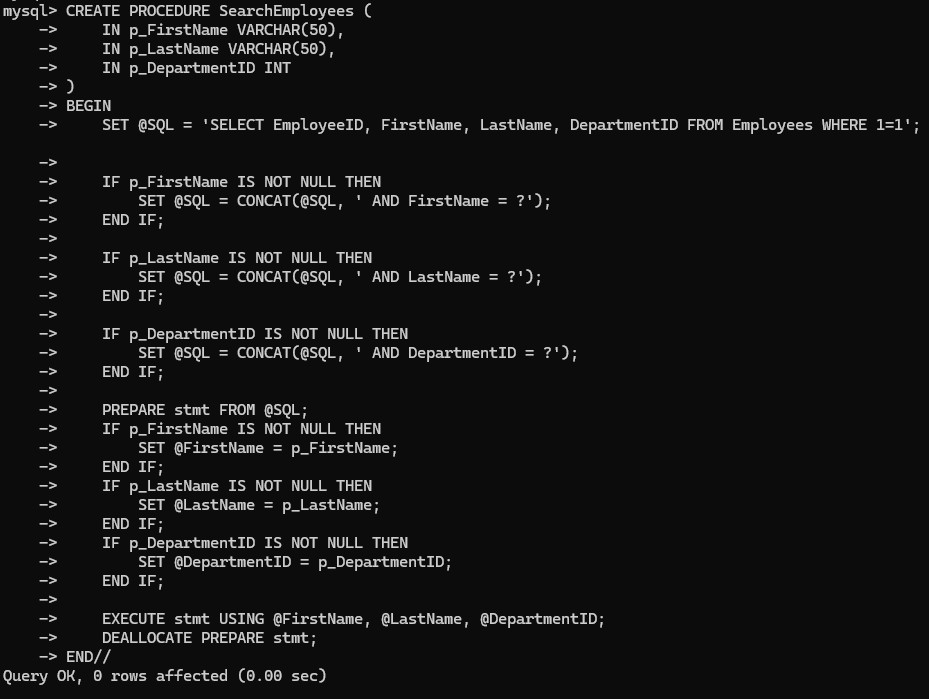
IF @DepartmentID IS NOT NULL

SET @SQL = @SQL + ' AND DepartmentID = @DepartmentID'

EXEC sp\_executesql @SQL, N'@FirstName VARCHAR(50), @LastName VARCHAR(50), @DepartmentID INT',

@FirstName, @LastName, @DepartmentID END

**Output –**



# Experiment 4

**Aim –** Implement functions in SQL

**Querry –**

DELIMITER //

CREATE FUNCTION CalculateTotalSalary(hourlyRate DECIMAL(10, 2), hoursWorked INT)

RETURNS DECIMAL(10, 2) DETERMINISTIC

BEGIN

DECLARE totalSalary DECIMAL(10, 2); SET totalSalary = hourlyRate \* hoursWorked; RETURN totalSalary;

END//

DELIMITER ;

**Output –**

