**DBMS [Day – 2]**

UID: 24MCI10204

Name: Rahul Saxena

Branch: 24MCA – AI & ML

**Question 1:** A car rental company maintains the following PostgreSQL tables:

* Customers(customer\_id, name, email, city)
* Cars(car\_id, model, type, daily\_rate)
* Rentals(rental\_id, customer\_id, car\_id, rental\_date, return\_date, total\_amount)

Tasks:

1. Data Retrieval and Join Queries:
   * Write an SQL query to display the customer name, car model, rental date, return date, and total amount for all rentals.
   * Modify the query to only show rentals where the total\_amount is greater than 1000, and sort the results by total\_amount in descending order.
2. Aggregation and Grouping:
   * Write a query to display the number of rentals and total revenue generated from each car model.
   * Modify it to show only those car models that have generated revenue greater than 5000.
3. Join Challenge:

Write a query using a LEFT JOIN to list all customers and their most recent rental date (if any). Show NULL for customers who haven’t rented yet.

**Code:**

**Table Creation:**

CREATE TABLE Customers (

customer\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100),

city VARCHAR(50)

);

CREATE TABLE Cars (

car\_id SERIAL PRIMARY KEY,

model VARCHAR(100),

type VARCHAR(50),

daily\_rate DECIMAL(10, 2)

);

CREATE TABLE Rentals (

rental\_id SERIAL PRIMARY KEY,

customer\_id INT REFERENCES Customers(customer\_id),

car\_id INT REFERENCES Cars(car\_id),

rental\_date DATE,

return\_date DATE,

total\_amount DECIMAL(10, 2)

);

**Inserting Values**

INSERT INTO Customers (name, email, city) VALUES

('Alice Johnson', 'alice.johnson@example.com', 'New York'),

('Bob Smith', 'bob.smith@example.com', 'Los Angeles'),

('Charlie Brown', 'charlie.brown@example.com', 'Chicago'),

('Diana Prince', 'diana.prince@example.com', 'San Francisco');

INSERT INTO Cars (model, type, daily\_rate) VALUES

('Toyota Camry', 'Sedan', 50.00),

('Honda CRV', 'SUV', 70.00),

('Ford Mustang', 'Sports', 120.00),

('Tesla Model 3', 'Electric', 150.00);

INSERT INTO Rentals (customer\_id, car\_id, rental\_date, return\_date, total\_amount) VALUES

(1, 1, '2025-06-01', '2025-06-05', 250.00),

(2, 3, '2025-06-10', '2025-06-15', 720.00),

(3, 2, '2025-06-20', '2025-06-25', 350.00),

(1, 4, '2025-06-28', '2025-07-02', 600.00),

(4, 3, '2025-06-15', '2025-06-20', 900.00),

(2, 4, '2025-07-01', '2025-07-05', 1200.00);

**Data Retrieval and Join Queries**

**A)** SELECT c.name AS customer\_name, ca.model AS car\_model, r.rental\_date, r.return\_date, r.total\_amount

FROM Rentals r

JOIN Customers c ON r.customer\_id = c.customer\_id

JOIN Cars ca ON r.car\_id = ca.car\_id;

**B)** SELECT c.name AS customer\_name, ca.model AS car\_model, r.rental\_date, r.return\_date, r.total\_amount

FROM Rentals r

JOIN Customers c ON r.customer\_id = c.customer\_id

JOIN Cars ca ON r.car\_id = ca.car\_id

WHERE r.total\_amount > 1000

ORDER BY r.total\_amount DESC;

**Aggregation and Grouping**

**A)** SELECT ca.model, COUNT(r.rental\_id) AS number\_of\_rentals, SUM(r.total\_amount) AS total\_revenue

FROM Rentals r

JOIN Cars ca ON r.car\_id = ca.car\_id

GROUP BY ca.model;

**B)** SELECT ca.model, COUNT(r.rental\_id) AS number\_of\_rentals, SUM(r.total\_amount) AS total\_revenue

FROM Rentals r

JOIN Cars ca ON r.car\_id = ca.car\_id

GROUP BY ca.model

HAVING SUM(r.total\_amount) > 5000;

**Join Challenge**

SELECT c.name, MAX(r.rental\_date) AS most\_recent\_rental\_date

FROM Customers c

LEFT JOIN Rentals r ON c.customer\_id = r.customer\_id

GROUP BY c.name

ORDER BY c.name;

**Question 2:** An online bookstore uses the following tables:

* Books(book\_id, title, author, price, stock\_quantity)
* Orders(order\_id, customer\_name, order\_date)
* OrderDetails(order\_id, book\_id, quantity)

When a customer places an order, the system must:

* Deduct the ordered quantity from the Books.stock\_quantity.
* Add a new record in the Orders table and related entries in OrderDetails.

Tasks:

1. Transactional Control:
   * Write a SQL script that does the following in a transaction:
     + Inserts a new order into Orders.
     + Inserts multiple books into OrderDetails (at least 2).
     + Updates the stock\_quantity of each ordered book by subtracting the quantity.
   * Use SAVEPOINT after the first book update and implement a conditional ROLLBACK TO SAVEPOINT if stock of the second book goes below zero.
2. Data Manipulation and Integrity:
   * Write a query to find books where stock\_quantity < 5, and update their price by increasing it by 10%.
   * Write a query to delete all books that have not been ordered (i.e., book\_id not present in any OrderDetails record).
3. Complex Filtering:
   * Retrieve a list of all books whose title contains the word “Data” (case-insensitive), price is between 200 and 500, and sort them by author name in ascending order.

**Code:**

**Table Creation:**

CREATE TABLE Books (

book\_id SERIAL PRIMARY KEY,

title VARCHAR(200),

author VARCHAR(100),

price DECIMAL(10,2),

stock\_quantity INT

);

CREATE TABLE Orders (

order\_id SERIAL PRIMARY KEY,

customer\_name VARCHAR(100),

order\_date DATE

);

CREATE TABLE OrderDetails (

order\_id INT REFERENCES Orders(order\_id),

book\_id INT REFERENCES Books(book\_id),

quantity INT,

PRIMARY KEY (order\_id, book\_id)

);

**Inserting Value:**

INSERT INTO Books (title, author, price, stock\_quantity) VALUES

('Data Structures in Python', 'Alice Smith', 350.00, 10),

('Learning SQL', 'Bob Jones', 250.00, 4),

('Data Science Essentials', 'Carol White', 450.00, 2),

('Modern Web Development', 'David Brown', 500.00, 8),

('Introduction to Algorithms', 'Eve Black', 600.00, 0),

('Database Systems', 'Frank Green', 300.00, 6);

**Transactional Control**

BEGIN;

INSERT INTO Orders (customer\_name, order\_date)

VALUES ('John Doe', CURRENT\_DATE)

RETURNING order\_id;

INSERT INTO OrderDetails (order\_id, book\_id, quantity) VALUES (1, 2, 2);

INSERT INTO OrderDetails (order\_id, book\_id, quantity) VALUES (1, 3, 3);

UPDATE Books SET stock\_quantity = stock\_quantity - 2 WHERE book\_id = 2;

SAVEPOINT after\_first\_update;

UPDATE Books SET stock\_quantity = stock\_quantity - 3 WHERE book\_id = 3;

DO $$

DECLARE

new\_stock INT;

BEGIN

SELECT stock\_quantity INTO new\_stock FROM Books WHERE book\_id = 3;

IF new\_stock < 0 THEN

RAISE NOTICE 'Stock below zero for book\_id 3. Rolling back to savepoint.';

ROLLBACK TO SAVEPOINT after\_first\_update;

DELETE FROM OrderDetails WHERE order\_id = 1 AND book\_id = 3;

END IF;

END $$;

COMMIT;

**Data Manipulation and Integrity**

**A)** UPDATE Books

SET price = price \* 1.10

WHERE stock\_quantity < 5;

**B)** DELETE FROM Books

WHERE book\_id NOT IN (SELECT DISTINCT book\_id FROM OrderDetails);

**Complex Filtering**

SELECT \*

FROM Books

WHERE

title ILIKE '%data%'

AND price BETWEEN 200 AND 500

ORDER BY author ASC;