Lab Manual - SQL Joins, Sub-Queries, and Views

Experiment No. 3

Title

SQL Queries – Joins, Sub-Queries, and Views

Objectives

- To understand and implement different types of **Joins** (INNER, LEFT, RIGHT, FULL, SELF).
- To design **Sub-queries** (single row, multiple row, correlated).
- To create and use **Views** for simplified data access.
- To perform at least **10 SQL queries** on a given case study using DML statements.

Problem Statement

*"Consider an **Online Bookstore** application. Write SQL queries that demonstrate various Joins, Sub-queries, and Views to retrieve meaningful information about customers, books, orders, and payments."*

Software and Hardware Requirements

- **Software:** MySQL / Oracle / PostgreSQL / SQL Server, SQL Workbench / pgAdmin / Oracle SQL Developer
- **Hardware:**
  - Processor: Intel i3 or higher
  - RAM: 4 GB or higher
  - Disk Space: 500 MB for DBMS installation
  - OS: Windows/Linux/Mac

Theory – Concept in Brief

Joins

- **INNER JOIN:** Returns matching rows from both tables.
- **LEFT JOIN:** Returns all rows from left table and matching from right.
- **RIGHT JOIN:** Returns all rows from right table and matching from left.
- **FULL OUTER JOIN:** Returns rows from both tables (where matches exist or not).
- **SELF JOIN:** A table joins with itself.

Sub-Queries

- **Single-row subquery:** Returns a single value.
- **Multiple-row subquery:** Returns multiple values (used with IN, ANY, ALL).
- **Correlated subquery:** Subquery depends on outer query.

Views

- Virtual tables created by storing SQL queries for reuse and simplified data access.

Algorithm

1. Start DBMS and connect to the database.
2. Use the normalized schema from the case study (Customer, Book, Orders, Order_Details, Payment).
3. Write SQL queries for:
   - All types of **joins**.
   - **Sub-queries** (single-row, multiple-row, correlated).
   - **Views**.
4. Execute queries and observe results.
5. Store queries and outputs for reporting.
6. End.

Flowchart : Create one yourself

Test Cases

Query No.	**Query Description**	**Expected Output**	**Status**
1	INNER JOIN: Customer & Orders	List of customers with their orders	Pass/Fail
2	LEFT JOIN: Books not ordered yet	List of books without sales	Pass/Fail
3	RIGHT JOIN: Orders with/without customer details	Order details	Pass/Fail
4	FULL OUTER JOIN: Customers and orders	All customers + all orders	Pass/Fail
5	SELF JOIN: Books by same author	Book pairs	Pass/Fail
6	Sub-query: Customer who placed highest order	Name of customer	Pass/Fail
7	Sub-query: Books priced above avg price	List of books	Pass/Fail
8	Correlated Sub-query: Orders above customer's avg spend	List of orders	Pass/Fail
9	Create View: Order summary	Virtual table with customer & total bill	Pass/Fail
10	Query View: Select top 5 customers by spending	List of 5 customers	Pass/Fail

---

Test Data Set

Customer

Cust_ID	**Name**	**Email**	**Phone**
1	Ramesh	ramesh@gmail.com	9876543210
2	Sneha	sneha@gmail.com	9998887777

Book

ISBN	Title	Author	Price
B101	DBMS Concepts	Korth	550
B102	SQL Fundamentals	Ram	350
B103	Data Structures	Weiss	450

Orders

Order_ID	Order_Date	Cust_ID
1001	2025-09-01	1
1002	2025-09-02	2

Order_Details

Order_ID	ISBN	Quantity
1001	B101	2
1001	B103	1
1002	B102	3

Payment

Pay_ID	Pay_Type	Amount	Order_ID
5001	UPI	1650	1001
5002	Card	1050	1002

Sample SQL Queries

1. INNER JOIN: Customers with their orders
```
SELECT c.Name, o.Order_ID, o.Order_Date
FROM Customer c
INNER JOIN Orders o ON c.Cust_ID = o.Cust_ID;
```

2. LEFT JOIN: Books not yet ordered
```
SELECT b.Title, od.Order_ID
```

```sql
FROM Book b
LEFT JOIN Order_Details od ON b.ISBN = od.ISBN
WHERE od.Order_ID IS NULL;
```

3. RIGHT JOIN: Orders and Customers

```sql
SELECT o.Order_ID, c.Name
FROM Customer c
RIGHT JOIN Orders o ON c.Cust_ID = o.Cust_ID;
```

4. FULL OUTER JOIN: Customers and Orders

```sql
SELECT c.Name, o.Order_ID
FROM Customer c
FULL OUTER JOIN Orders o ON c.Cust_ID = o.Cust_ID;
```

5. SELF JOIN: Books by same author

```sql
SELECT b1.Title AS Book1, b2.Title AS Book2
FROM Book b1, Book b2
WHERE b1.Author = b2.Author AND b1.ISBN <> b2.ISBN;
```

6. Sub-query: Customer with highest total order value

```sql
SELECT Name FROM Customer
WHERE Cust_ID = (SELECT Cust_ID FROM Orders o
JOIN Payment p ON o.Order_ID = p.Order_ID
GROUP BY Cust_ID ORDER BY SUM(p.Amount) DESC LIMIT 1);
```

7. Sub-query: Books priced above average

```sql
SELECT Title, Price FROM Book
WHERE Price > (SELECT AVG(Price) FROM Book);
```

8. Correlated Sub-query: Orders above customer’s avg spend

```sql
SELECT o.Order_ID, p.Amount FROM Orders o
JOIN Payment p ON o.Order_ID = p.Order_ID
WHERE p.Amount > (SELECT AVG(p2.Amount)
            FROM Orders o2 JOIN Payment p2
            ON o2.Order_ID = p2.Order_ID
            WHERE o2.Cust_ID = o.Cust_ID);
```

9. Create View: Order Summary

```
CREATE VIEW OrderSummary AS
SELECT c.Name, o.Order_ID, SUM(b.Price * od.Quantity) AS Total
FROM Customer c
JOIN Orders o ON c.Cust_ID = o.Cust_ID
JOIN Order_Details od ON o.Order_ID = od.Order_ID
JOIN Book b ON od.ISBN = b.ISBN
GROUP BY c.Name, o.Order_ID;
```

10. Query from View: Top 5 Customers by Spending

```
SELECT Name, SUM(Total) AS GrandTotal
FROM OrderSummary
GROUP BY Name
ORDER BY GrandTotal DESC
LIMIT 5;
```

Mathematical Model (if applicable)

- Let **C = {c1, c2, … cn}** be set of customers.
- Let **B = {b1, b2, … bn}** be set of books.
- Let **O = {o1, o2, … om}** be set of orders.
- **Relation R1 (Customer–Order):** C × O → mapping between customers and orders.
- **Relation R2 (Order–Book):** O × B → mapping between orders and books.
- **Function f:** TotalAmount(o) = Σ (Price(b) × Quantity(b)) for all b in order o.

Conclusion / Analysis

In this experiment, we successfully implemented **Joins (INNER, LEFT, RIGHT, FULL, SELF)**, **Sub-queries (single row, multiple row, correlated)**, and **Views** for the selected database application. These concepts enable **complex data retrieval, improved query modularity, and better readability** in SQL-based database systems.