Experiment 2: Advanced SQL Queries

Prerequisites

Basic knowledge of SQL (SELECT, WHERE, GROUP BY, JOIN)

Lab Objective

To understand and implement advanced SQL querying techniques for retrieving complex and meaningful information from relational databases.

Learning Outcomes

By completing this experiment, students will be able to:

- Perform complex JOIN operations across multiple tables.
- Write nested and correlated subqueries.
- Use Common Table Expressions (CTEs) for structured query writing.
- Apply window functions for analytical operations.
- Use aggregation functions with HAVING clause.
- Combine results using SQL set operators.

Introduction

Advanced SQL queries are used in real-world applications to extract analytical and relational insights from databases. These queries allow users to perform multi-table operations, nested data filtering, ranking, summarization, and result combination.

This experiment focuses on strengthening students' ability to write optimized and structured SQL queries beyond basic data retrieval.

Key Concepts Covered

- **Complex JOINs:** INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN
- **Subqueries:** Nested queries and correlated subqueries
- **Common Table Expressions (CTE):** Using WITH clause for modular queries
- **Window Functions:** RANK(), ROW_NUMBER(), SUM() OVER(), etc.
- **Aggregations:** GROUP BY with HAVING clause
- **Set Operators:** UNION, INTERSECT, EXCEPT

Experiment Steps

1. Implement Complex JOINs

Retrieve data from multiple related tables and compare results of different join types.

Example – INNER JOIN

```
SELECT s.name, c.course_name

FROM Students s

INNER JOIN Enrollments e ON s.student_id = e.student_id

INNER JOIN Courses c ON e.course_id = c.course_id;
```

Example – LEFT JOIN

```
SELECT s.name, c.course_name

FROM Students s

LEFT JOIN Enrollments e ON s.student_id = e.student_id

LEFT JOIN Courses c ON e.course_id = c.course_id;
```

Observe the difference in results between INNER and LEFT JOIN.

---

2. Work with Subqueries

Write nested queries to filter data based on calculated values.

Example – Nested Subquery

```
SELECT name
FROM Students
WHERE student_id IN (
   SELECT student_id
   FROM Enrollments
   WHERE marks > (SELECT AVG(marks) FROM Enrollments)
);
```

Example – Correlated Subquery

```
SELECT name
FROM Students s
WHERE EXISTS (
   SELECT 1
   FROM Enrollments e
   WHERE e.student_id = s.student_id
);
```

---

3. Use Common Table Expressions (CTEs)

Organize complex queries into readable and reusable blocks.

Example – CTE

```
WITH HighScorers AS (
   SELECT student_id, marks
   FROM Enrollments
   WHERE marks > 80
)
SELECT s.name, h.marks
FROM HighScorers h
JOIN Students s ON s.student_id = h.student_id;
```

---

4. Apply Window Functions

Perform ranking and analytical calculations without grouping rows.

Example – Ranking Students

```
SELECT student_id, marks,
RANK() OVER (ORDER BY marks DESC) AS Rank_Position
FROM Enrollments;
```

Example – Running Total

```
SELECT student_id, marks,
SUM(marks) OVER (ORDER BY student_id) AS Running_Total
```

```
FROM Enrollments;
```

---

5. Perform Aggregation with HAVING

Group records and apply conditions on aggregated results.

Example

```
SELECT course_id, AVG(marks) AS Average_Marks

FROM Enrollments

GROUP BY course_id

HAVING AVG(marks) > 75;
```

---

6. Use Set Operators

Combine results from multiple queries.

Example – UNION

```
SELECT department FROM Students

UNION

SELECT course_name FROM Courses;
```

Example – INTERSECT (if supported)

```
SELECT student_id FROM Students

INTERSECT

SELECT student_id FROM Enrollments;
```

How It Works

- Advanced SQL queries enable structured and analytical data retrieval.
- JOINs establish relationships between multiple tables.
- Subqueries and CTEs allow layered filtering and modular query design.
- Window functions provide powerful analytical capabilities without collapsing rows.
- Aggregations summarize grouped data.
- Set operators combine and compare multiple result sets.

Submission Instructions

Students must submit:

- Screenshot of JOIN query results.
- Output of a nested and correlated subquery.
- Output demonstrating window function usage.
- Aggregation with HAVING result.
- Result of a set operator query.

Conclusion

This experiment enhances students' SQL proficiency by introducing advanced querying techniques widely used in enterprise systems, reporting tools, and data analytics applications. Mastery of these concepts prepares students for database development and analytical roles.