NATIONAL ENGINEERING COLLEGE, K.R. NAGAR, KOVILPATTI – 628 503
(An Autonomous Institution, Affiliated to Anna University – Chennai)

DEPARTMENT OF	**COMPUTER SCIENCE AND ENGINEERING**
	INFORMATION TECHNOLOGY
	ARTIFICAL INTELLIGENCE AND DATA SCIENCE

Even Semester - 2024-2025

Laboratory Practice - 06

CO7: DESIGN AND IMPLEMENT SQL QUERIES FOR DATA MANIPULATION

Lab Instruction Sheet

Exercise: 06 To explore the use of subqueries and the GROUP BY clause in SQL for retrieving and analyzing data. This exercise focuses on applying nested queries to fetch specific results and using grouping techniques to perform aggregate computations effectively.

Concept:

GROUP BY Clause

The GROUP BY clause is used in SQL to group rows that have the same values in specified columns into aggregated summary results.

- It is commonly used with aggregate functions like:
  - COUNT()
  - SUM()
  - AVG()
  - MAX()
  - MIN()
- It allows you to perform calculation on group of data rather than individual rows.

Syntax of GROUP BY

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name;
```

Example of GROUP BY

- **Find the total marks obtained by students in each department:**

```
SELECT department, SUM(marks) AS total_marks
FROM students
GROUP BY department;
```

HAVING Clause

- The HAVING clause is used to filter groups of data after they have been grouped using GROUP BY.
- It is similar to the WHERE clause but works after aggregation.
- You cannot use WHERE with aggregate functions, so HAVING is used instead.

Syntax of HAVING:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Example of HAVING

- **Find departments where the total marks of students exceed 250:**

```
SELECT department, SUM(marks) AS total_marks
FROM students
GROUP BY department
HAVING SUM(marks) > 250;
```

Key Differences: WHERE vs. HAVING

Feature	WHERE Clause	HAVING Clause
Used with	Individual rows	Groups of rows
Works with aggregate functions?	No	Yes
Filtering order	Before grouping	After grouping

Subqueries & Nested Subqueries

- A subquery is a query that is nested inside another SQL query.
- It is enclosed in parentheses () and can return:
  - **Single values** (scalar subquery)
  - **Multiple rows** (row subquery)
  - **Entire tables** (table subquery)
- It is used to fetch intermediate results that the main query uses.

Types of Subqueries

1. **Single-row Subquery** → Returns a single value (used with =, <, >, etc.).
2. **Multi-row Subquery** → Returns multiple values (used with IN, ANY, ALL).
3. **Correlated Subquery** → Uses values from the outer query and executes for each row.

Syntax of a Subquery

```
SELECT column_name
FROM table_name
WHERE column_name OPERATOR (SELECT column_name FROM table_name WHERE
condition);
```

Example of a Single-Row Subquery:

- **Find students who have marks greater than the average marks of all students:**

```
SELECT name, marks
FROM students
WHERE marks > (SELECT AVG(marks) FROM students);
```

Nested Subquery

- A nested subquery is a subquery that contains another subquery inside it.
- It is useful when multiple levels of filtering are needed.

Example of a Nested Subquery

- **Find students whose marks are greater than the minimum marks obtained by students in the Computer Science department:**

```
SELECT name, marks
FROM students
WHERE marks > (SELECT MIN(marks)
               FROM students
               WHERE department = 'Computer Sci');
```

Example of a Correlated Subquery

- **Find students who have the highest marks in their department:**

```
SELECT name, department, marks
FROM students S1
WHERE marks = (SELECT MAX(marks)
               FROM students S2
               WHERE S1.department = S2.department);
```

Concept	Description	Example Query
GROUP BY	Groups data based on column values and performs aggregation	GROUP BY department
HAVING	Filters groups after aggregation	HAVING SUM(marks) > 250
Subquery	Query inside another query	WHERE marks > (SELECT AVG(marks) FROM students)
Nested Subquery	Subquery inside a subquery	WHERE marks > (SELECT MIN(marks) FROM students WHERE department = 'CS')
Correlated Subquery	Uses outer query values inside subquery	WHERE marks = (SELECT MAX(marks) FROM students S2 WHERE S1.department = S2.department)

Problem Description:

Problem 01:

The students table contains information about students, their courses, marks, and departments.

Table Structure:

Column Name	Data Type	Description
student_id	INT (PK)	Unique ID for each student
name	VARCHAR(50)	Student's full name
department	VARCHAR(30)	Department of the student
course	VARCHAR(50)	Course enrolled
marks	INT	Marks obtained
age	INT	Age of the student
city	VARCHAR(50)	City of residence

Sample Data

student_id	name	department	course	marks	age	city
1	Alice	Computer Sci	DBMS	85	20	New York
2	Bob	Computer Sci	DBMS	78	21	Chicago
3	Charlie	Electronics	Networks	92	22	Boston
4	David	Computer Sci	Algorithms	88	20	New York
5	Emma	Mechanical	Thermodynamics	75	23	Dallas
6	Frank	Electronics	Circuits	83	22	Boston
7	Grace	Mechanical	Mechanics	80	21	Chicago
8	Henry	Computer Sci	DBMS	90	20	New York
9	Isabella	Electronics	Networks	88	22	Boston
10	Jack	Mechanical	Mechanics	70	23	Dallas

Practice Queries:

1. Find the average marks obtained by students in each department, and display only those departments where the average marks are greater than 80.
2. Count the number of students in each city and display only those cities where more than 2 students are residing.
3. Find the number of students in each course and display only those courses where more than one student is enrolled.
4. Find the total number of students in each department and display only those departments having more than 2 students.
5. Find the highest marks in each course and display only those courses where the highest marks are greater than 85.
6. Count the number of students enrolled in each course and display only those courses where the number of students is greater than 1

7. Find departments where the sum of students' marks exceeds 250.

Subquery:

8. Find the students who scored more than the average marks of all students.
9. Retrieve the names of students who are in the same department as 'Alice'.
10. Find the students who are the top scorers in their respective departments.
11. Display the average marks of students from each city but only include cities where the average marks are above 80.
12. Find the students whose marks are greater than the average marks of students in their own department.
13. Retrieve the names of students who are older than the youngest student in the 'Computer Science' department.
14. List students who live in cities where at least one student has scored above 90.
15. Retrieve students who have the same marks as the highest marks obtained in any department.

Problem 02:

Consider the Ticket booking scenario, the following tables were used:

Movie

Column	Data Type	Constraints
id	int	Primary Key
title	varchar(128)	
genre	varchar(32)	
release_year	int	
rating	decimal	
theater_id	int	Foreign Key (References theaters.id)

Theaters

Column	Data Type	Constraints
id	int	Primary Key
Name	varchar(64)	
City	varchar(32)	
country	varchar(32)	
rating	decimal	

Bookings

Column	Data Type	Constraints
id	int	Primary Key
user_name	varchar(64)	
movie_id	int	Foreign Key (References movies.id)
theater_id	int	Foreign Key (References theaters.id)
tickets	int	
booking_date	date	

Queries:

1. Select the names of all theaters that are located in the same city where any theater screened a movie in the year 2022.
2. Show the name and number of tickets booked by users who have booked more tickets than the average number of tickets booked for action movies.
3. Select the names and genres of all movies that have a rating above 8.0 and are screened only in theaters with a rating above 9.0.

Problem 03: Consider the Library Scenario

Books Table

Column	Data Type	Constraints
Id	int	Primary Key
title	varchar(128)	
author	varchar(64)	
genre	varchar(32)	
year	int	
library_id	int	Foreign Key (References libraries.id)

Libraries

Column	Data Type	Constraints
Id	Int	Primary Key
name	varchar(64)	
city	varchar(32)	
country	varchar(32)	
rating	decimal	

Members

Column	Data Type	Constraints
Id	Int	Primary Key
name	varchar(64)	
membership_type	varchar(32)	(e.g., Regular, Premium)
age	Int	
books_borrowed	Int	
library_id	int	Foreign Key (References libraries.id)

Queries:

1. Select the names of all libraries that are located in the same city where any library issued a book in the year 2020.
2. Show the name and membership type of library members who have borrowed more books than the average number of books borrowed by premium members.
3. Select the names of books and their authors that belong to libraries with a rating of 8.5 or higher and were published before the year 2000.