“Heaven's Light is Our Guide”

Rajshahi University of Engineering & Technology

Department of Electrical and Computer Engineering

Course Code: ECE- 2216

Course Title: Data Base Systems Sessional

Lab Report No: 01

Submitted By	**Submitted To**
Sohayel Ahmed Shakkhor Roll: 2110022 Registration No: 1076 Session: 2021-2022 Department of Electrical and Computer Engineering, Rajshahi University of Engineering & Technology	Oishi Jyoti Assistant Professor Department of Electrical and Computer Engineering, Rajshahi University of Engineering & Technology

Date of Submission: October 01, 2024

2.1 Experiment No: 02

2.2 Name of the Experiment:

Database Query using MySQL.

2.3 Theory:

SQL (Structured Query Language) is a robust tool used to manage and manipulate relational databases. It enables users to execute various operations, such as retrieving, inserting, updating, and deleting data. SQL plays a crucial role in database querying by offering a standardized approach to handling structured data. SQL queries can range from simple data retrieval tasks to more complex functions, including filtering, grouping, and aggregating data. The language is divided into several categories, such as Data Query Language (DQL), Data Manipulation Language (DML), Data Definition Language (DDL), and Data Control Language (DCL), providing comprehensive control over database management.

One of the key strengths of SQL is its ability to handle complex data operations with straightforward commands. Basic commands like SELECT, UPDATE, INSERT, and DELETE allow users to directly manipulate data, while more advanced operations such as JOIN, GROUP BY, HAVING, and ORDER BY help organize and filter data efficiently. Common aggregate functions like SUM(), AVG(), and COUNT() are used to gain insights from datasets. Additionally, SQL supports subqueries and conditional statements, enabling more sophisticated analysis. This versatility makes SQL a vital tool in fields such as business intelligence, web development, and scientific research.

2.4 Objectives:

- To Learn about Aggregate Functions.
- Learing about Subqueries and how to Use them.

2.5 Tasks:

Students Table

student_id	student_name	age	GPA	department	year_of_admission	fees_paid	credits_earned	enrollment_status
1	Eleven	21	3.8	Engineering	2021	10000	120	active
2	Dustin	22	3.9	Science	2020	9000	110	active
3	Will	19	3.4	Business	2022	8500	95	active
4	Mike	23	3.7	Science	2021	9500	115	inactive
5	Max	20	3.5	Engineering	2020	12000	130	active
6	Eddie	22	4.0	Arts	2019	8000	140	active
7	Billy	24	2.9	Engineering	2022	5000	60	active
8	Alexei	25	3.2	Business	2018	7500	100	inactive
9	Steve	21	3.8	Science	2021	10500	120	active
10	Robin	20	3.6	Engineering	2022	11000	125	active
11	Lucas	18	2.7	Engineering	2023	4000	50	active
12	Nancy	23	3.9	Business	2019	9500	135	active

Task:

1. Find students who are older than 20 and have a GPA above the average GPA of all students
2. Find the top 5 students with the highest fees paid, ordered by GPA (in descending order) as a tiebreaker
3. List students who belong to the "Engineering" department, have a GPA greater than 3.5, and are enrolled after 2020
4. Find students who are not active (i.e., enrollment_status = 'inactive') and have not paid any fees (fees_paid = 0)
5. Calculate the total fees paid and average GPA for each department, but only for departments with more than 10 students

2.6 Query & Output:

Creating a New Table and Inserting data:

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    age INT,
    GPA DECIMAL(3, 2),
    department VARCHAR(50),
    year_of_admission INT,
```

```
    fees_paid INT,
    credits_earned INT,
    enrollment_status VARCHAR(20)
);

INSERT INTO Students (student_id, student_name, age, GPA, department, year_of_admission,
fees_paid, credits_earned, enrollment_status)
VALUES
(1, 'Eleven', 21, 3.8, 'Engineering', 2021, 10000, 120, 'active'),
(2, 'Dustin', 22, 3.9, 'Science', 2020, 9000, 110, 'active'),
(3, 'Will', 19, 3.4, 'Business', 2022, 8500, 95, 'active'),
(4, 'Mike', 23, 3.7, 'Science', 2021, 9500, 115, 'inactive'),
(5, 'Max', 20, 3.5, 'Engineering', 2020, 12000, 130, 'active'),
(6, 'Eddie', 22, 4.0, 'Arts', 2019, 8000, 140, 'active'),
(7, 'Billy', 24, 2.9, 'Engineering', 2022, 5000, 60, 'active'),
(8, 'Alexei', 25, 3.2, 'Business', 2018, 7500, 100, 'inactive'),
(9, 'Steve', 21, 3.8, 'Science', 2021, 10500, 120, 'active'),
(10, 'Robin', 20, 3.6, 'Engineering', 2022, 11000, 125, 'active'),
(11, 'Lucas', 18, 2.7, 'Engineering', 2023, 4000, 50, 'active'),
(12, 'Nancy', 23, 3.9, 'Business', 2019, 9500, 135, 'active');
```

Output:

	student_id	student_name	age	GPA	department	year_of_admission	fees_paid	credits_earned	enrollment_status
Edit Copy Delete	1	Eleven	21	3.80	Engineering	2021	10000	120	active
Edit Copy Delete	2	Dustin	22	3.90	Science	2020	9000	110	active
Edit Copy Delete	3	Will	19	3.40	Business	2022	8500	95	active
Edit Copy Delete	4	Mike	23	3.70	Science	2021	9500	115	inactive
Edit Copy Delete	5	Max	20	3.50	Engineering	2020	12000	130	active
Edit Copy Delete	6	Eddie	22	4.00	Arts	2019	8000	140	active
Edit Copy Delete	7	Billy	24	2.90	Engineering	2022	5000	60	active
Edit Copy Delete	8	Alexei	25	3.20	Business	2018	7500	100	inactive
Edit Copy Delete	9	Steve	21	3.80	Science	2021	10500	120	active
Edit Copy Delete	10	Robin	20	3.60	Engineering	2022	11000	125	active
Edit Copy Delete	11	Lucas	18	2.70	Engineering	2023	4000	50	active
Edit Copy Delete	12	Nancy	23	3.90	Business	2019	9500	135	active

Task1:

Find students who are older than 20 and have a GPA above the average GPA of all students.

Query:

```
SELECT *
FROM Students
```

```
WHERE age > 20
AND GPA > (SELECT AVG(GPA) FROM Students);
```

Output:

student_id	student_name	age	GPA	department	year_of_admission	fees_paid	credits_earned	enrollment_status
1	Eleven	21	3.80	Engineering	2021	10000	120	active
2	Dustin	22	3.90	Science	2020	9000	110	active
4	Mike	23	3.70	Science	2021	9500	115	inactive
6	Eddie	22	4.00	Arts	2019	8000	140	active
9	Steve	21	3.80	Science	2021	10500	120	active
12	Nancy	23	3.90	Business	2019	9500	135	active

Task 2:
Find the top 5 students with the highest fees paid, ordered by GPA (descending order) as a tiebreaker.

Query:

```
SELECT *
FROM Students
ORDER BY fees_paid DESC, GPA DESC
LIMIT 5;
```

Output:

student_id	student_name	age	GPA ▼ 2	department	year_of_admission	fees_paid ▼ 1	credits_earned	enrollment_status
5	Max	20	3.50	Engineering	2020	12000	130	active
10	Robin	20	3.60	Engineering	2022	11000	125	active
9	Steve	21	3.80	Science	2021	10500	120	active
1	Eleven	21	3.80	Engineering	2021	10000	120	active
12	Nancy	23	3.90	Business	2019	9500	135	active

Task 3:
List students in the "Engineering" department with GPA > 3.5 and enrolled after 2020.

```
SELECT *
FROM Students
WHERE department = 'Engineering'
AND GPA > 3.5
AND year_of_admission > 2020;
```

Output:

student_id	student_name	age	GPA	department	year_of_admission	fees_paid	credits_earned	enrollment_status
1	Eleven	21	3.80	Engineering	2021	10000	120	active
10	Robin	20	3.60	Engineering	2022	11000	125	active

Task 4:
Find students who are not active and have not paid any fees.

```
SELECT *
FROM Students
WHERE enrollment_status = 'inactive'
AND fees_paid = 0;
```

Output:

student_id	student_name	age	GPA	department	year_of_admission	fees_paid	credits_earned	enrollment_status

Task 5:
Calculate total fees paid and average GPA for each department, only for departments with more than 10 students.

```
SELECT department, SUM(fees_paid) AS total_fees, AVG(GPA) AS average_GPA
FROM Students
GROUP BY department
HAVING COUNT(student_id) > 10;
```

Output:

department	total_fees	average_GPA

2.7 Discussion:

In this experiment, we effectively demonstrated the application of various MySQL commands to query and manipulate data in a student database. We began by constructing a "Students" table with key attributes such as student ID, name, age, GPA, department, year of admission, fees paid, credits earned, and enrollment status. Throughout the experiment, we executed several tasks using SQL queries to analyze and retrieve specific data
For example, we employed conditional filtering to identify students older than 20 years with a GPA higher than the average, illustrating the use of subqueries and comparison operators. Additionally, we ranked students based on fees paid and utilized GPA as a tiebreaker to retrieve

the top five records, which showcased the functionality of the `ORDER BY` and `LIMIT` clauses.
Moreover, we utilized aggregation functions like `SUM()` and `AVG()` to compute the total fees paid and the average GPA per department, filtered by departments with more than 10 students. This task highlighted the importance of **grouping** and the `HAVING` clause.
These exercises provided valuable hands-on experience with Data Manipulation Language (DML) commands, such as `SELECT`, `ORDER BY`, and `GROUP BY`, emphasizing how SQL can be effectively used for advanced data querying and analysis.

2.8 References:

[1]
Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education

[2]
W3Schools. (n.d.). SQL Tutorial. Retrieved from https://www.w3schools.com/sql/