GOVERNMENT POLYTECHNIC SHEOHAR

LABORATORY MANUAL

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Ⅳ SEMESTER

DATABASE MANAGEMENT SYSTEM

SUBJECT CODE: P2418403

LIST OF EXPERIMENTS

1	Database installation (such as MySQL, MariaDB)	CO-4
2	Design table structure	CO-4
3	Apply integrity constraints	CO-4, CO-2
4	Use DML commands.	CO-4
5	Apply relational algebraic operations	CO-4
6	Write statements to demonstrate the use of aggregate functions	CO-4
7	Perform join operations	CO-4
8	Write Correlated and Nested Query	CO-4
9	Write TCL Queries	CO-4
10	Implement Views to perform following operations: a. Create views. b. Insert, modify and delete records through views. c. c. Delete the views.	CO-5
11	Create Indexes, Sequences, and Synonyms in SQL.	CO-5

PRACTICAL OUTCOMES

CO	Statement	Revised Bloom Taxonomy
1	Illustrate the fundamental concepts of Database, Database System and Database Management System.	Analyze
2	Use the concepts of E-R Modeling, Keys and constraints to design a database	Apply
3	Normalize/De-normalize the database to optimize its performance	Apply
4	Use Structured Query Language (SQL) for database manipulation	Apply
5	Create and use schema objects such as View, Index, Synonyms and Sequence to optimize database performance.	Apply

PROGRAM OUTCOMES (POS)

PO1 - Basic and Discipline specific knowledge: Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the engineering problems.

PO2 - Problem analysis: Identify and analyse well-defined engineering problems using codified standard methods.

PO3 - Design/ development of solutions: Design solutions for well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

PO4 - Engineering Tools, Experimentation and Testing: Apply modern engineering tools and appropriate technique to conduct standard tests and measurements.

PO5 - Engineering practices for society, sustainability and environment: Apply appropriate technology in context of society, sustainability, environment and ethical practices.

PO6 - Project Management: Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well- defined engineering activities.

PO7 - Life-long learning: Ability to analyze individual needs and engage in updating in the context of technological changes.

PROGRAM SPECIFIC OUTCOMES

PSO1: Apply the fundamentals of computer science, including algorithms and programming, to analyze complex problems and develop robust solutions.

PSO2: Design and implement computer science solutions with emphasis on energy efficiency, sustainability, and eco-friendly technologies.

MAPPING

S. No.	CO-Statement	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PSO1	PSO2
CO1	Illustrate the fundamental concepts of Database, Database System and Database Management System.	1						1	2	
CO2	Use the concepts of E-R Modeling, Keys and constraints to design a database		1		1				2	1
CO3	Normalize/De-normalize the database to optimize its performance		2	2	1	1		1	2	2
CO4	Use Structured Query Language (SQL) for database manipulation	2	1	2	1	1			2	1
CO5	Create and use schema objects such as View, Index, Synonyms and Sequence to optimize database performance.	1	1	2	1	1			2	1

EXPERIMENT NO: 1

INSTALLATION AND CONFIGURATION OF DATABASE SOFTWARE (MYSQL / MARIADB)

AIM: - To install and configure database management software such as MySQL or MariaDB on a computer system.

APPARATUS / TOOLS REQUIRED

- Computer System
- Internet Connection
- Operating System (Windows / Linux/Mac)
- Web Browser
- MySQL or MariaDB Installer

THEORY: -

A **Database Management System (DBMS)** is software used to store, manage, and retrieve data efficiently. MySQL and MariaDB are popular open-source relational database management systems that use Structured Query Language (SQL).

- **MySQL** is widely used for web and enterprise applications.
- **MariaDB** is a community-developed fork of MySQL and is fully compatible with it.

These databases allow users to create databases, tables, insert records, and perform queries securely and efficiently.

INSTALLATION STEPS: -

WINDOWS (MySQL / MariaDB)

- Open any web browser.
- Visit the official MySQL or MariaDB website.
- Download MySQL Installer for Windows or MariaDB Windows Installer.
- Run the downloaded installer file.
- Click Next and accept the license agreement.

- Select Developer Default or Server Only setup type.
- Allow the installer to download required components.
- Choose Standalone Server configuration.
- Keep the default port number 3306.
- Set the root password and note it down.
- Configure the database server as a Windows Service.
- Click Execute to complete installation.
- Open MySQL/MariaDB Command Line Client.
- Login using username root and the password.
- Verify installation using a test SQL command.

LINUX (MySQL / MariaDB)

- Open the **Terminal**.
- Update the package list using: sudo apt update
- Install MySQL or MariaDB server: sudo apt install mysql-server
- Or sudo apt install mariadb-server
- Start the database service: sudo systemctl start mysql
- Enable service at boot: sudo systemctl enable mysql
- Secure the installation: sudo mysql_secure_installation
- Set the **root password** when prompted.
- Login to database server: mysql -u root -p
- Enter the root password.
- Verify installation using SQL commands.

macOS (MySQL / MariaDB)

- Open a web browser.
- Visit the official MySQL or MariaDB website.
- Download the macOS DMG installer.
- Open the downloaded .dmg file.
- Run the installer package.
- Follow on-screen installation instructions.
- Set the root password during installation.

- Complete the installation process.
- Open Terminal.
- Login to MySQL/MariaDB using: mysql -u root -p
- Enter the root password.
- Verify installation using SQL commands.

INSTALLATION VERIFICATION COMMAND

SHOW DATABASES;

```
C:\Users\HP>mysql --version
mysql  Ver 8.0.44 for Win64 on x86_64 (MySQL Community Server - GPL)

C:\Users\HP>mysql -u root -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 8.0.44 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| amit               |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.06 sec)

mysql>
```

RESULT: -

MySQL / MariaDB database software was successfully installed and configured, and basic database creation commands were executed successfully.

EXPERIMENT NO: 2

DESIGN AND CREATE A DATABASE TABLE STRUCTURE USING MYSQL / MARIADB.

AIM: - To design and create a database table with appropriate fields, data types, and constraints using SQL.

APPARATUS / TOOLS REQUIRED

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

A database table is a structured collection of related data organized in rows and columns. Each column represents a field with a specific data type, while each row represents a record.

Designing a proper table structure is important for data integrity, efficiency, and scalability. SQL provides the CREATE TABLE command to define table structures along with constraints such as PRIMARY KEY, NOT NULL, UNIQUE, and AUTO_INCREMENT.

PROCEDURE:

- Start the MySQL / MariaDB server.
- Open MySQL / MariaDB Command Line Client.
- Login using username and password.
- Create or select a database using SQL command.
- Define table fields with suitable data types.
- Apply necessary constraints to the table.
- Execute the CREATE TABLE command.
- Verify the table structure using DESCRIBE command.

PROGRAM (CODE): -

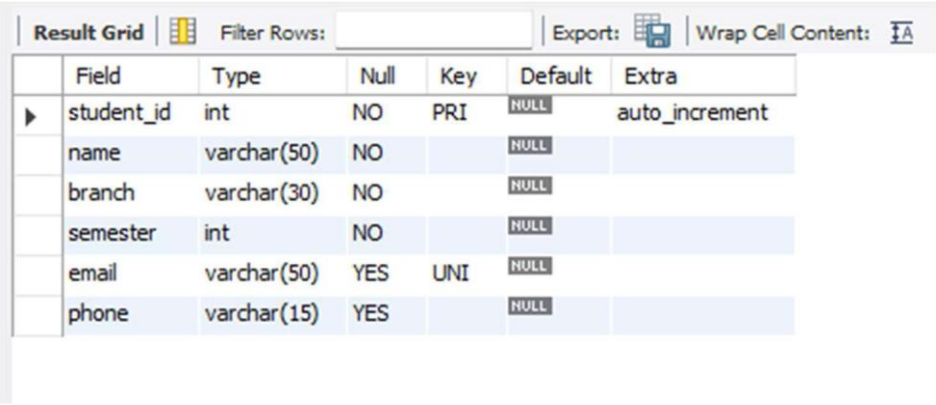
Create Database

```
CREATE DATABASE college;
USE college;
```

Create Table (Student Table)

```
CREATE TABLE student (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    branch VARCHAR(30) NOT NULL,
    semester INT NOT NULL,
    email VARCHAR(50) UNIQUE,
    phone VARCHAR(15)
);
```

OUTPUT: -



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
student_id	int	NO	PRI	NULL	auto_increment
name	varchar(50)	NO		NULL	
branch	varchar(30)	NO		NULL	
semester	int	NO		NULL	
email	varchar(50)	YES	UNI	NULL	
phone	varchar(15)	YES		NULL	

RESULT: -

The table structure was successfully designed and created using SQL with appropriate data types and constraints in MySQL / MariaDB.

EXPERIMENT NO: 3

APPLY INTEGRITY CONSTRAINTS ON A DATABASE TABLE USING SQL.

AIM: - To apply integrity constraints on a database table in order to maintain data accuracy, consistency, and reliability.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

Integrity constraints are rules enforced on database tables to ensure the validity and consistency of data. These constraints prevent invalid data entry and maintain relationships between tables.

Common integrity constraints include:

- **PRIMARY KEY**: Ensures unique identification of records.
- **NOT NULL**: Prevents null values in a column.
- **UNIQUE**: Ensures all values in a column are unique.
- **CHECK**: Ensures values meet specific conditions.
- **FOREIGN KEY**: Maintains referential integrity between tables.
- **DEFAULT**: Assigns default values when no value is provided.

PROCEDURE: -

- Start the MySQL / MariaDB server.
- Open MySQL / MariaDB Command Line Client.
- Login using valid credentials.
- Select the required database.
- Create tables with integrity constraints.
- Insert valid and invalid records to test constraints.

- Observe system responses.
- Verify table structure and constraints.

PROGRAM(CODE): -

Create Database

```
CREATE DATABASE college;
USE college;
```

Create Table with Integrity Constraints

```
CREATE TABLE student (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    branch VARCHAR(30) NOT NULL,
    semester INT,
    email VARCHAR(50) UNIQUE,
    phone VARCHAR(15),
    admission_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    CHECK (semester BETWEEN 1 AND 8)
);
```

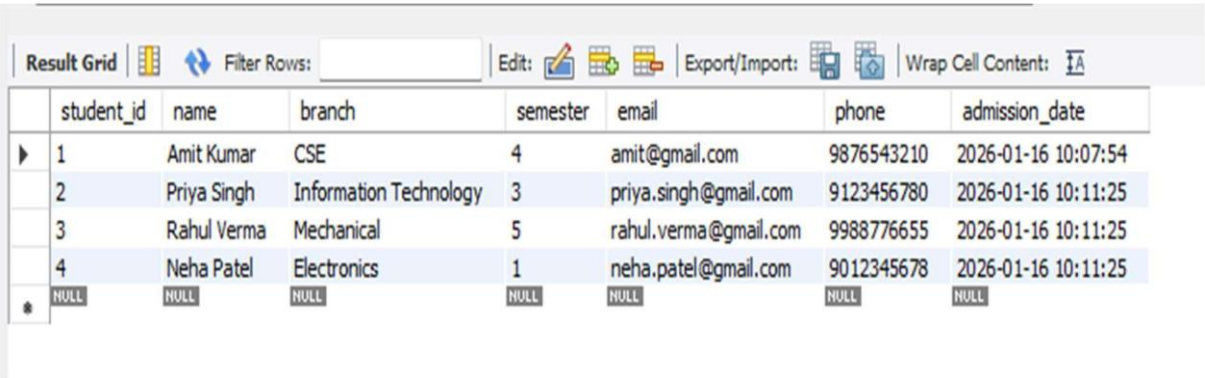
Insert Valid Record

```
INSERT INTO student (name, branch, semester, email, phone)
VALUES
('Priya Singh', 'Information Technology', 3, 'priya.singh@gmail.com', '9123456780'),
('Rahul Verma', 'Mechanical', 5, 'rahul.verma@gmail.com', '9988776655'),
('Neha Patel', 'Electronics', 1, 'neha.patel@gmail.com', '9012345678');
```

Check Output

```
Select * from student;
```

OUTPUT: -



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	name	branch	semester	email	phone	admission_date
1	Amit Kumar	CSE	4	amit@gmail.com	9876543210	2026-01-16 10:07:54
2	Priya Singh	Information Technology	3	priya.singh@gmail.com	9123456780	2026-01-16 10:11:25
3	Rahul Verma	Mechanical	5	rahul.verma@gmail.com	9988776655	2026-01-16 10:11:25
4	Neha Patel	Electronics	1	neha.patel@gmail.com	9012345678	2026-01-16 10:11:25
NULL	NULL	NULL	NULL	NULL	NULL	NULL

RESULT: -

Integrity constraints were successfully applied on the database table.

EXPERMENT NO: 4

USE OF DATA MANIPULATION LANGUAGE (DML) COMMANDS IN SQL

AIM: - To use Data Manipulation Language (DML) commands to insert, update, delete, and retrieve data from a database table.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

Data Manipulation Language (DML) commands are used to manage and manipulate data stored in database tables. These commands allow users to insert new records, update existing records, delete records, and retrieve data.

Common DML commands include:

- **INSERT** – Adds new records to a table
- **SELECT** – Retrieves records from a table
- **UPDATE** – Modifies existing records
- **DELETE** – Removes records from a table

DML commands operate on table data and do not affect the table structure.

PROCEDURE: -

- Start the MySQL / MariaDB server.
- Open MySQL / MariaDB Command Line Client.
- Login using valid username and password.
- Select the required database.
- Use INSERT command to add records.
- Use SELECT command to display records.
- Use UPDATE command to modify records.

- Use DELETE command to remove records.
- Observe the results of each command.

PROGRAM (CODE): -

Select Database

USE college;

Insert Valid Record

INSERT INTO student (name, branch, semester, email, phone)

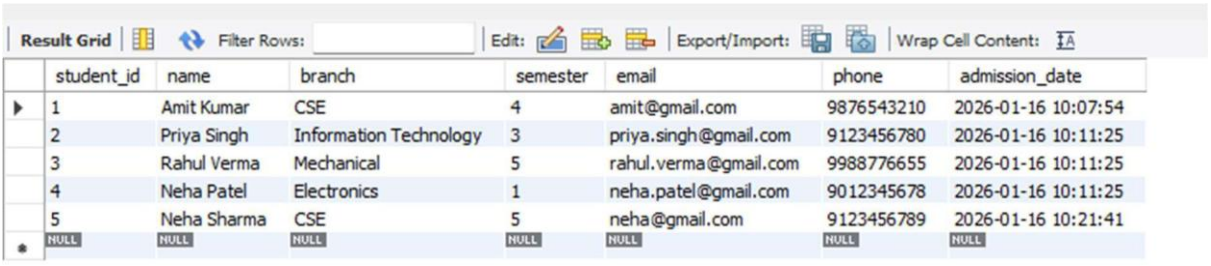
VALUES

('Priya Singh', 'Information Technology', 3, 'priya.singh@gmail.com', '9123456780'),

('Rahul Verma', 'Mechanical', 5, 'rahul.verma@gmail.com', '9988776655'),

('Neha Patel', 'Electronics', 1, 'neha.patel@gmail.com', '9012345678');

('Neha Sharma', 'CSE', 5, 'neha@gmail.com', '9123456789');



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

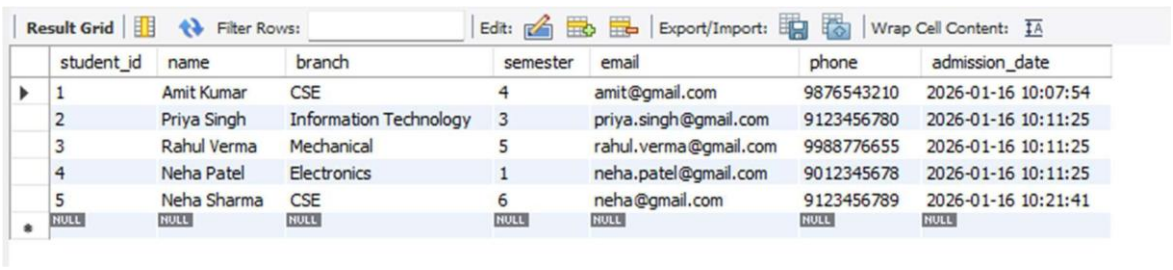
student_id	name	branch	semester	email	phone	admission_date
1	Amit Kumar	CSE	4	amit@gmail.com	9876543210	2026-01-16 10:07:54
2	Priya Singh	Information Technology	3	priya.singh@gmail.com	9123456780	2026-01-16 10:11:25
3	Rahul Verma	Mechanical	5	rahul.verma@gmail.com	9988776655	2026-01-16 10:11:25
4	Neha Patel	Electronics	1	neha.patel@gmail.com	9012345678	2026-01-16 10:11:25
5	Neha Sharma	CSE	5	neha@gmail.com	9123456789	2026-01-16 10:21:41
NULL	NULL	NULL	NULL	NULL	NULL	NULL

UPDATE Command

UPDATE student

SET semester = 6

WHERE name = 'Neha Sharma';



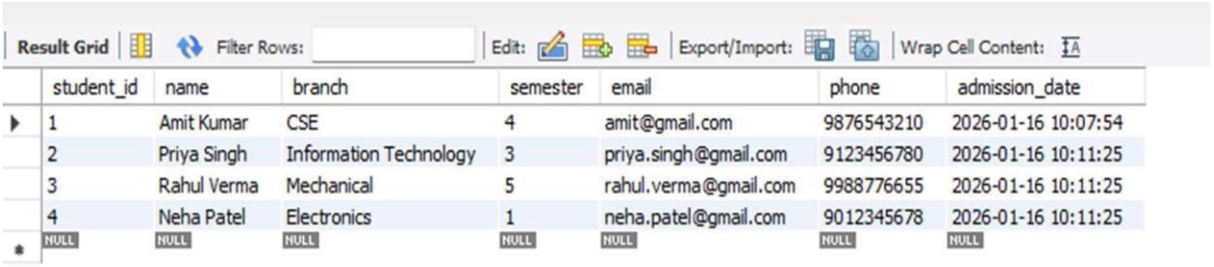
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	name	branch	semester	email	phone	admission_date
1	Amit Kumar	CSE	4	amit@gmail.com	9876543210	2026-01-16 10:07:54
2	Priya Singh	Information Technology	3	priya.singh@gmail.com	9123456780	2026-01-16 10:11:25
3	Rahul Verma	Mechanical	5	rahul.verma@gmail.com	9988776655	2026-01-16 10:11:25
4	Neha Patel	Electronics	1	neha.patel@gmail.com	9012345678	2026-01-16 10:11:25
5	Neha Sharma	CSE	6	neha@gmail.com	9123456789	2026-01-16 10:21:41
NULL	NULL	NULL	NULL	NULL	NULL	NULL

DELETE Command

```
DELETE FROM student

WHERE name = 'Neha Sharma';
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	name	branch	semester	email	phone	admission_date
1	Amit Kumar	CSE	4	amit@gmail.com	9876543210	2026-01-16 10:07:54
2	Priya Singh	Information Technology	3	priya.singh@gmail.com	9123456780	2026-01-16 10:11:25
3	Rahul Verma	Mechanical	5	rahul.verma@gmail.com	9988776655	2026-01-16 10:11:25
4	Neha Patel	Electronics	1	neha.patel@gmail.com	9012345678	2026-01-16 10:11:25
NULL	NULL	NULL	NULL	NULL	NULL	NULL

RESULT: -

The DML commands were successfully executed to insert, retrieve, update, and delete records from the database table.

EXPERMENT NO: 5

APPLICATION OF RELATIONAL ALGEBRAIC OPERATIONS

AIM: - To apply relational algebraic operations on database relations and understand their effect on data retrieval.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

Relational Algebra is a procedural query language used to retrieve data from relational databases. It works on relations (tables) and produces new relations as output. Each operation takes one or more relations as input and returns another relation.

Common relational algebra operations include:

- **Selection (σ)** – Selects rows based on a condition
- **Projection (π)** – Selects specific columns
- **Union (∪)** – Combines tuples of two relations
- **Set Difference (−)** – Retrieves tuples present in one relation but not in another
- **Cartesian Product (×)** – Combines all tuples of two relations
- **Join (⋈)** – Combines related tuples from two relations

These operations form the foundation of SQL query processing.

PROCEDURE: -

- Start the MySQL / MariaDB server.
- Open MySQL Command Line Client.
- Login using valid credentials.
- Select the required database.
- Apply relational algebra operations using equivalent SQL queries.
- Observe and verify the results of each operation.

PROGRAM (CODE): -

Select Database

USE college;

Insert Valid Record

INSERT INTO student (name, branch, semester, email, phone)

VALUES

('Priya Singh', 'Information Technology', 3, 'priya.singh@gmail.com', '9123456780'),

('Rahul Verma', 'Mechanical', 5, 'rahul.verma@gmail.com', '9988776655'),

('Neha Patel', 'Electronics', 1, 'neha.patel@gmail.com', '9012345678');

('Neha Sharma', 'CSE', 5, 'neha@gmail.com', '9123456789');

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	name	branch	semester	email	phone	admission_date
1	Amit Kumar	CSE	4	amit@gmail.com	9876543210	2026-01-16 10:07:54
2	Priya Singh	Information Technology	3	priya.singh@gmail.com	9123456780	2026-01-16 10:11:25
3	Rahul Verma	Mechanical	5	rahul.verma@gmail.com	9988776655	2026-01-16 10:11:25
4	Neha Patel	Electronics	1	neha.patel@gmail.com	9012345678	2026-01-16 10:11:25
5	Neha Sharma	CSE	5	neha@gmail.com	9123456789	2026-01-16 10:21:41
NULL	NULL	NULL	NULL	NULL	NULL	NULL

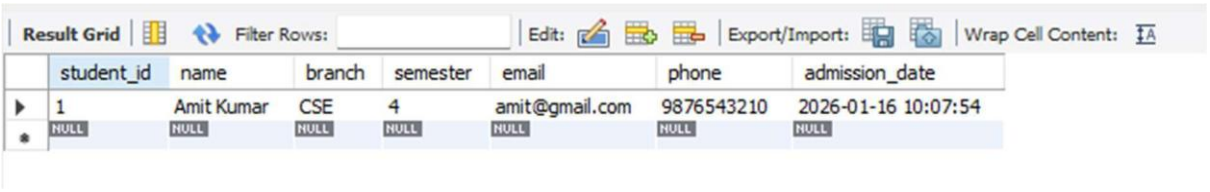
1. Selection (σ)

Relational Algebra:

σ(branch = 'CSE')(student)

SELECT * FROM student

WHERE branch = 'CSE';



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

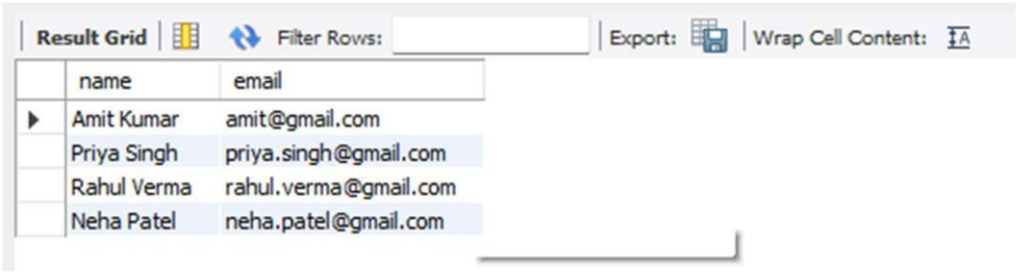
student_id	name	branch	semester	email	phone	admission_date
1	Amit Kumar	CSE	4	amit@gmail.com	9876543210	2026-01-16 10:07:54
NULL	NULL	NULL	NULL	NULL	NULL	NULL

2. Projection (π)

Relational Algebra:

π(name, email)(student)

SELECT name, email FROM student;



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	email
Amit Kumar	amit@gmail.com
Priya Singh	priya.singh@gmail.com
Rahul Verma	rahul.verma@gmail.com
Neha Patel	neha.patel@gmail.com

3. Union (∪)

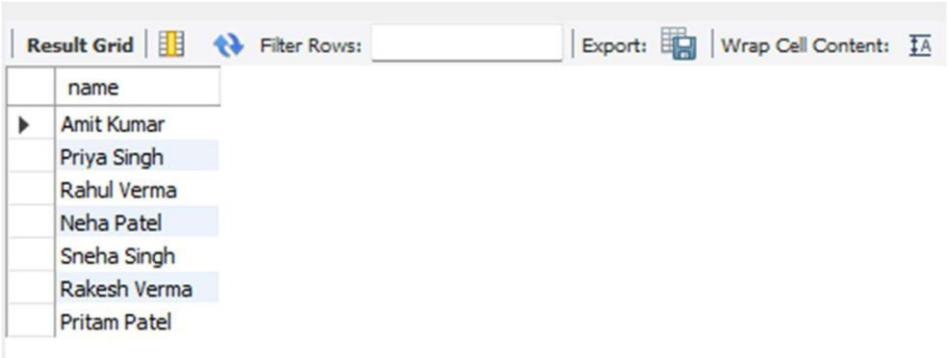
Relational Algebra:

student1 ∪ student2

SELECT name FROM student1

UNION

SELECT name FROM student2;



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
Amit Kumar
Priya Singh
Rahul Verma
Neha Patel
Sneha Singh
Rakesh Verma
Pritam Patel

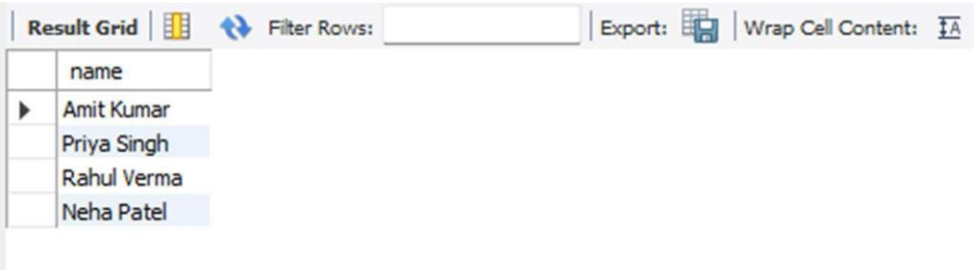
4. Set Difference (−)

Relational Algebra:

student1 − student2

SELECT name FROM student1

WHERE name NOT IN (SELECT name FROM student2);



5. Cartesian Product (×)

Relational Algebra:

student × department

SELECT * FROM student, department;

RESULT: -

Relational algebraic operations such as selection, projection, union, difference, Cartesian product, and join were successfully applied, and their results were verified using SQL queries.

EXPERMENT NO: 6

DEMONSTRATION OF AGGREGATE FUNCTIONS IN SQL

AIM: - To demonstrate the use of SQL aggregate functions such as COUNT, SUM, AVG, MAX, and MIN on a database table.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

Aggregate functions in SQL are used to perform calculations on a set of values and return a single summarized result. These functions are commonly used in database queries to analyze data stored in tables.

Common SQL aggregate functions include:

- **COUNT()** – Returns the number of rows in a table.
- **SUM()** – Returns the total sum of a numeric column.
- **AVG()** – Calculates the average value of a column.
- **MAX()** – Returns the maximum value from a column.
- **MIN()** – Returns the minimum value from a column.

Aggregate functions are often used with GROUP BY and HAVING clauses to summarize data based on categories.

Example: In an employee table, aggregate functions can be used to find the total salary, highest salary, lowest salary, and average salary.

PROCEDURE: -

1. Open the database software (MySQL or MariaDB).
2. Create a database and select it.
3. Create a table (e.g., Employee table).
4. Insert sample records into the table.
5. Execute SQL queries using aggregate functions.
6. Observe the output results.

PROGRAM (CODE): -

Create Table

```
CREATE TABLE Employee (
   emp_id INT,
   emp_name VARCHAR(50),
   salary INT
);
```

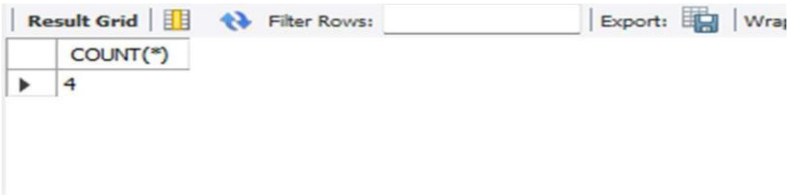
Insert Records

```
INSERT INTO Employee VALUES (1,'Rahul',30000);
INSERT INTO Employee VALUES (2,'Aman',25000);
INSERT INTO Employee VALUES (3,'Priya',40000);
INSERT INTO Employee VALUES (4,'Neha',35000);
```

✓	9	10:21:23	INSERT INTO Employee VALUES (1,'Rahul',300...	1 row(s) affected	0.016 sec
✓	10	10:21:23	INSERT INTO Employee VALUES (2,'Aman',25000)	1 row(s) affected	0.015 sec
✓	11	10:21:23	INSERT INTO Employee VALUES (3,'Priya',40000)	1 row(s) affected	0.000 sec
✓	12	10:21:23	INSERT INTO Employee VALUES (4,'Neha',35000)	1 row(s) affected	0.016 sec

Aggregate Function Queries

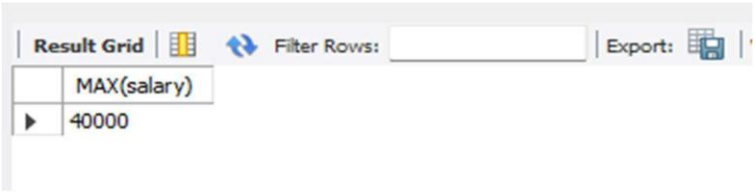
SELECT COUNT(*) FROM Employee;



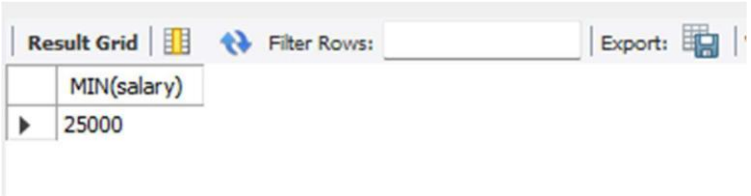
SELECT SUM(salary) FROM Employee;

SELECT AVG(salary) FROM Employee;

SELECT MAX(salary) FROM Employee;



SELECT MIN(salary) FROM Employee;



RESULT: -

The SQL aggregate functions (COUNT, SUM, AVG, MAX, MIN) were executed successfully and used to summarize data stored in the database table.

EXPERIMENT NO: 7

PERFORM JOIN OPERATIONS IN SQL

AIM: - To perform different SQL Join operations such as INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN on database tables.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

A JOIN operation in SQL is used to combine rows from two or more tables based on a related column between them. Joins are important in relational databases because data is often stored in multiple tables.

Common types of SQL joins:

1. INNER JOIN

Returns records that have matching values in both tables.

2. LEFT JOIN (LEFT OUTER JOIN)

Returns all records from the left table and matched records from the right table. If no match exists, NULL values are returned.

3. RIGHT JOIN (RIGHT OUTER JOIN)

Returns all records from the right table and matched records from the left table.

4. FULL JOIN

Returns all records when there is a match in either left or right table.

Join operations are widely used in relational database management systems to retrieve related data from multiple tables.

PROCEDURE: -

1. Open the database software (MySQL or MariaDB).
2. Create two tables (for example Employee and Department).
3. Insert sample records into both tables.
4. Use SQL JOIN queries to combine the tables.
5. Execute the queries and observe the results.

PROGRAM (CODE): -

Create Employee Table

```
CREATE TABLE Employee (
    emp_id INT,
    emp_name VARCHAR(50),
    dept_id INT
);
```

Create Department Table

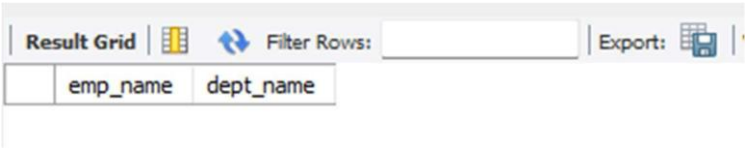
```
CREATE TABLE Department (
    dept_id INT,
    dept_name VARCHAR(50)
);
```

Insert Data

```
INSERT INTO Employee VALUES (1,'Rahul',101);
INSERT INTO Employee VALUES (2,'Aman',102);
INSERT INTO Employee VALUES (3,'Priya',103);
INSERT INTO Department VALUES (101,'HR');
INSERT INTO Department VALUES (102,'IT');
INSERT INTO Department VALUES (104,'Finance');
```

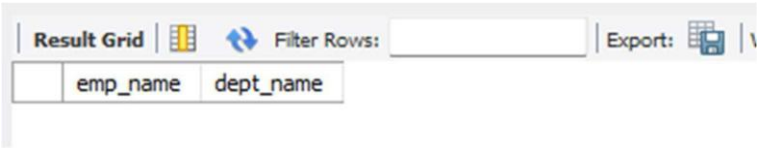
INNER JOIN

```
SELECT Employee.emp_name, Department.dept_name
FROM Employee
INNER JOIN Department
ON Employee.dept_id = Department.dept_id;
```



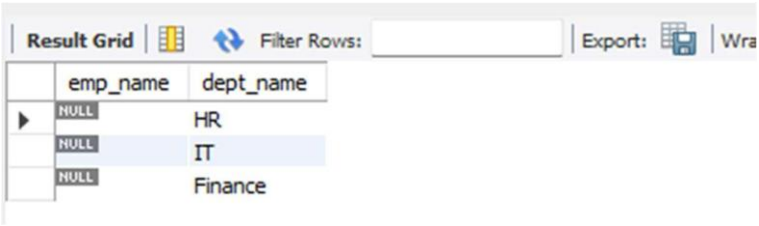
LEFT JOIN

```
SELECT Employee.emp_name, Department.dept_name
FROM Employee
LEFT JOIN Department
ON Employee.dept_id = Department.dept_id;
```



RIGHT JOIN

```
SELECT Employee.emp_name, Department.dept_name
FROM Employee
RIGHT JOIN Department
ON Employee.dept_id = Department.dept_id;
```



RESULT: -

Different SQL join operations were executed successfully to combine data from multiple tables and retrieve related information

EXPERIMENT NO: 8

DEMONSTRATION OF CORRELATED AND NESTED QUERIES IN SQL

AIM: - To write and execute correlated queries and nested queries in SQL to retrieve data from database tables.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

In SQL, queries can be written inside another query to perform complex data retrieval. These are called subqueries.

Two important types of subqueries are:

1. Nested Query

A nested query (subquery) is a query written inside another SQL query.
The inner query executes first, and its result is used by the outer query.

Example: Finding employees whose salary is greater than the average salary.

2. Correlated Query

A correlated query is a type of subquery where the inner query depends on the outer query.
The inner query is executed once for each row processed by the outer query.

Correlated queries are often used when comparison with rows of the outer table is required.

Applications:

- Data filtering

- Complex comparisons between table rows
- Conditional data retrieval

PROCEDURE: -

1. Open the database software (MySQL or MariaDB).
2. Create a table (for example Employee).
3. Insert sample records into the table.
4. Write and execute SQL queries using nested queries and correlated queries.
5. Observe the output.

PROGRAM / CODE: -

Create Table

```
CREATE TABLE Employee (
    emp_id INT,
    emp_name VARCHAR(50),
    salary INT,
    dept_id INT
);
```

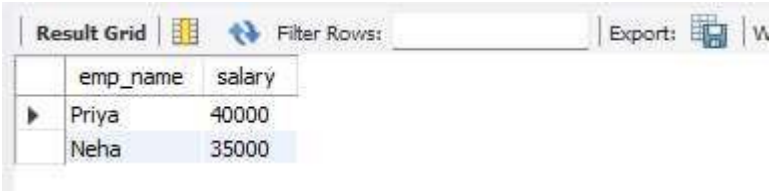
Insert Data

```
INSERT INTO Employee VALUES (1,'Rahul',30000,101);
INSERT INTO Employee VALUES (2,'Aman',25000,102);
INSERT INTO Employee VALUES (3,'Priya',40000,101);
INSERT INTO Employee VALUES (4,'Neha',35000,103);
```

Nested Query Example

Find employees whose salary is greater than the average salary.

```
SELECT emp_name, salary
FROM Employee
WHERE salary > (SELECT AVG(salary) FROM Employee);
```



emp_name	salary
Priya	40000
Neha	35000

Correlated Query Example

Find employees whose salary is greater than the average salary of their department.

```
SELECT emp_name, salary, dept_id
FROM Employee E1
WHERE salary > (
    SELECT AVG(salary)
    FROM Employee E2
    WHERE E1.dept_id = E2.dept_id
);
```

emp_name	salary	dept_id
Priya	40000	101

RESULT: -

The Nested Query and Correlated Query were successfully executed to retrieve data based on complex conditions from the database table.

EXPERIMENT NO: 9

DEMONSTRATION OF TRANSACTION CONTROL LANGUAGE (TCL) QUERIES

AIM: - To write and execute Transaction Control Language (TCL) commands such as COMMIT, ROLLBACK, and SAVEPOINT in SQL.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

Transaction Control Language (TCL) commands are used to manage transactions in a database. A transaction is a group of SQL operations executed as a single unit of work.

TCL ensures that database operations follow the ACID properties (Atomicity, Consistency, Isolation, Durability).

Common TCL commands include:

1. COMMIT

The **COMMIT** command saves all changes made during the transaction permanently in the database.

2. ROLLBACK

The ROLLBACK command cancels all changes made in the current transaction and restores the database to its previous state.

3. SAVEPOINT

The SAVEPOINT command creates a point inside a transaction so that partial rollback can be performed.

4. RELEASE SAVEPOINT

This command removes a previously defined savepoint.

TCL commands are used to control data consistency and integrity in database transactions.

PROCEDURE: -

1. Open the database software (MySQL or MariaDB).
2. Create a table and insert sample data.
3. Perform update or delete operations.
4. Use SAVEPOINT to create a transaction point.
5. Use ROLLBACK to undo changes if required.
6. Use COMMIT to permanently save the changes.
7. Observe the results.

PROGRAM / CODE: -

Create Table

```
CREATE TABLE Employee (
    emp_id INT,
    emp_name VARCHAR(50),
    salary INT
);
```

Insert Data

```
INSERT INTO Employee VALUES (1,'Rahul',30000);
INSERT INTO Employee VALUES (2,'Aman',25000);
INSERT INTO Employee VALUES (3,'Priya',40000);
```

TCL Commands

Create Savepoint

SAVEPOINT sp1;

Update Record

UPDATE Employee SET salary = 35000 WHERE emp_id = 2;

Rollback to Savepoint

ROLLBACK TO sp1;

Commit Changes

COMMIT;

Action Output

#	Time	Action	Message	Duration / Fetch
42	11:00:05	SELECT emp_name, salary FROM Employe WHE...	2 row(s) returned	0.000 sec / 0.000 sec
43	11:01:00	SELECT emp_name, salary, dept_id FROM Emplo...	1 row(s) returned	0.000 sec / 0.000 sec
44	11:14:33	CREATE TABLE Employ (emp_id INT, emp...	0 row(s) affected	0.047 sec
45	11:15:08	INSERT INTO Employ VALUES (1,'Rahul',30000)	1 row(s) affected	0.016 sec
46	11:15:08	INSERT INTO Employ VALUES (2,'Aman',25000)	1 row(s) affected	0.016 sec
47	11:15:08	INSERT INTO Employ VALUES (3,'Priya',40000)	1 row(s) affected	0.000 sec
48	11:15:22	SAVEPOINT sp1	0 row(s) affected	0.000 sec
49	11:15:37	UPDATE Employee SET salary = 35000 WHERE ...	3 row(s) affected Rows matched: 3 Changed: 3 ...	0.015 sec
50	11:15:58	ROLLBACK TO sp1	Error Code: 1305. SAVEPOINT sp1 does not exist	0.000 sec
51	11:16:14	UPDATE Employ SET salary = 35000 WHERE e...	1 row(s) affected Rows matched: 1 Changed: 1 ...	0.031 sec
52	11:16:18	ROLLBACK TO sp1	Error Code: 1305. SAVEPOINT sp1 does not exist	0.000 sec
53	11:16:35	COMMIT	0 row(s) affected	0.016 sec

RESULT: -

The TCL commands (COMMIT, ROLLBACK, SAVEPOINT) were successfully executed to control and manage database transactions.

EXPERIMENT NO: 10

IMPLEMENTATION OF VIEWS IN SQL

AIM: - To create views and perform insert, update, delete operations through views, and finally delete the view.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

A View in SQL is a virtual table created from the result of a SQL query. It does not store data physically; instead, it displays data from one or more tables.

Views are used for:

- Simplifying complex queries
- Providing data security
- Restricting access to specific columns or rows
- Presenting data in a customized format

Operations that can be performed on views include:

1. Create View

A view is created using the CREATE VIEW statement.

2. Insert / Update / Delete through View

Data can be inserted, modified, or deleted through a view if the view is created from a single table and follows certain conditions.

3. Delete View

A view can be removed using the DROP VIEW statement.

Views help in improving data abstraction and security in database systems.

PROCEDURE: -

1. Open the database software (MySQL or MariaDB).
2. Create a table (for example Employee).
3. Insert sample records into the table.
4. Create a view from the table.
5. Perform insert, update, and delete operations through the view.
6. Display the results.
7. Delete the view using DROP VIEW command.

PROGRAM / CODE: -

```
CREATE TABLE Employee (
   emp_id INT,
   emp_name VARCHAR(50),
   salary INT
);
```

Insert Records

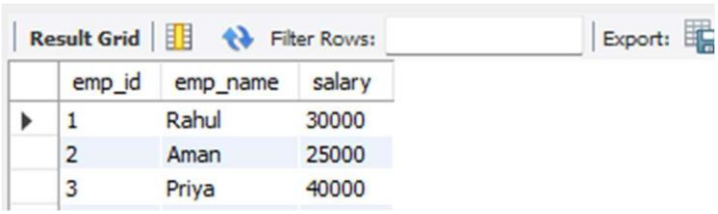
```
INSERT INTO Employee VALUES (1,'Rahul',30000);
INSERT INTO Employee VALUES (2,'Aman',25000);
INSERT INTO Employee VALUES (3,'Priya',40000);
```

Create View

```
CREATE VIEW emp_view AS
SELECT emp_id, emp_name, salary
FROM Employee;
```

Display View

SELECT * FROM emp_view;


Result Grid | Filter Rows: | Export:

emp_id	emp_name	salary
1	Rahul	30000
2	Aman	25000
3	Priya	40000

Insert Record through View

INSERT INTO emp_view VALUES (4,'Neha',35000);

Modify Record through View

UPDATE emp_view

SET salary = 32000

WHERE emp_id = 1;

Delete Record through View

DELETE FROM emp_view

WHERE emp_id = 2;

Delete View

DROP VIEW emp_view;

DROP VIEW emp_view;

tput

Action Output

#	Time	Action	Message	Duration / Fetch
53	11:16:35	COMMIT	0 row(s) affected	0.016 sec
54	11:22:08	CREATE TABLE Emplo (emp_id INT, emp_...	0 row(s) affected	0.031 sec
55	11:22:50	INSERT INTO Emplo VALUES (1,'Rahul',30000)	1 row(s) affected	0.031 sec
56	11:22:50	INSERT INTO Emplo VALUES (2,'Aman',25000)	1 row(s) affected	0.000 sec
57	11:22:50	INSERT INTO Emplo VALUES (3,'Priya',40000)	1 row(s) affected	0.016 sec
58	11:23:18	INSERT INTO Emplo VALUES (3,'Priya',40000)	1 row(s) affected	0.015 sec
59	11:23:27	CREATE VIEW emp_view AS SELECT emp_id, e...	0 row(s) affected	0.016 sec
60	11:23:58	SELECT * FROM emp_view LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
61	11:25:02	INSERT INTO emp_view VALUES (4,'Neha',350...	1 row(s) affected	0.000 sec
62	11:25:37	UPDATE emp_view SET salary = 32000 WHERE...	1 row(s) affected Rows matched: 1 Changed: 1 ...	0.000 sec
63	11:26:43	DELETE FROM emp_view WHERE emp_id = 2	1 row(s) affected	0.016 sec
64	11:26:56	DROP VIEW emp_view	0 row(s) affected	0.000 sec

RESULT: -

The SQL View was created successfully, and insert, update, delete operations were performed through the view, and finally the view was deleted successfully.

EXPERIMENT NO: 11

IMPLEMENTATION OF VIEWS IN SQL

AIM: - In SQL databases, objects like indexes, sequences, and synonyms are used to improve performance and accessibility.

APPARATUS / TOOLS REQUIRED: -

- Computer System
- Operating System (Windows / Linux / macOS)
- MySQL or MariaDB Database Server
- MySQL / MariaDB Command Line Client or GUI Tool (MySQL Workbench)

THEORY: -

In SQL databases, objects like indexes, sequences, and synonyms are used to improve performance and accessibility.

1. Index

An Index is a database object used to speed up the retrieval of rows from a table.
It works similarly to an index in a book, allowing the database to find records faster without scanning the entire table.

Indexes are created using the CREATE INDEX statement.

Example use:

- Faster searching
- Faster sorting and filtering of data

2. Sequence

A Sequence is a database object used to generate unique numeric values automatically.
Sequences are commonly used to generate primary key values.

Each time the sequence is called, it produces the next number in the sequence.

3. Synonym

A Synonym is an alternative name given to a database object such as a table, view, or sequence.
It simplifies access to objects and improves readability of SQL queries.

For example, a long table name can be replaced with a short synonym.

PROCEDURE: -

1. Open the database software (MySQL or MariaDB).
2. Create a table in the database.
3. Create an index on a column of the table.
4. Create a sequence to generate automatic numbers.
5. Create a synonym for the table.
6. Execute queries to verify the objects created.

PROGRAM / CODE: -

Create Table

```
CREATE TABLE Employee (
    emp_id INT,
    emp_name VARCHAR(50),
    salary INT
);
```

Create Index

```
CREATE INDEX idx_emp_name
ON Employee(emp_name);
```

Create Sequence

CREATE SEQUENCE emp_seq

START WITH 1

INCREMENT BY 1;

Use Sequence

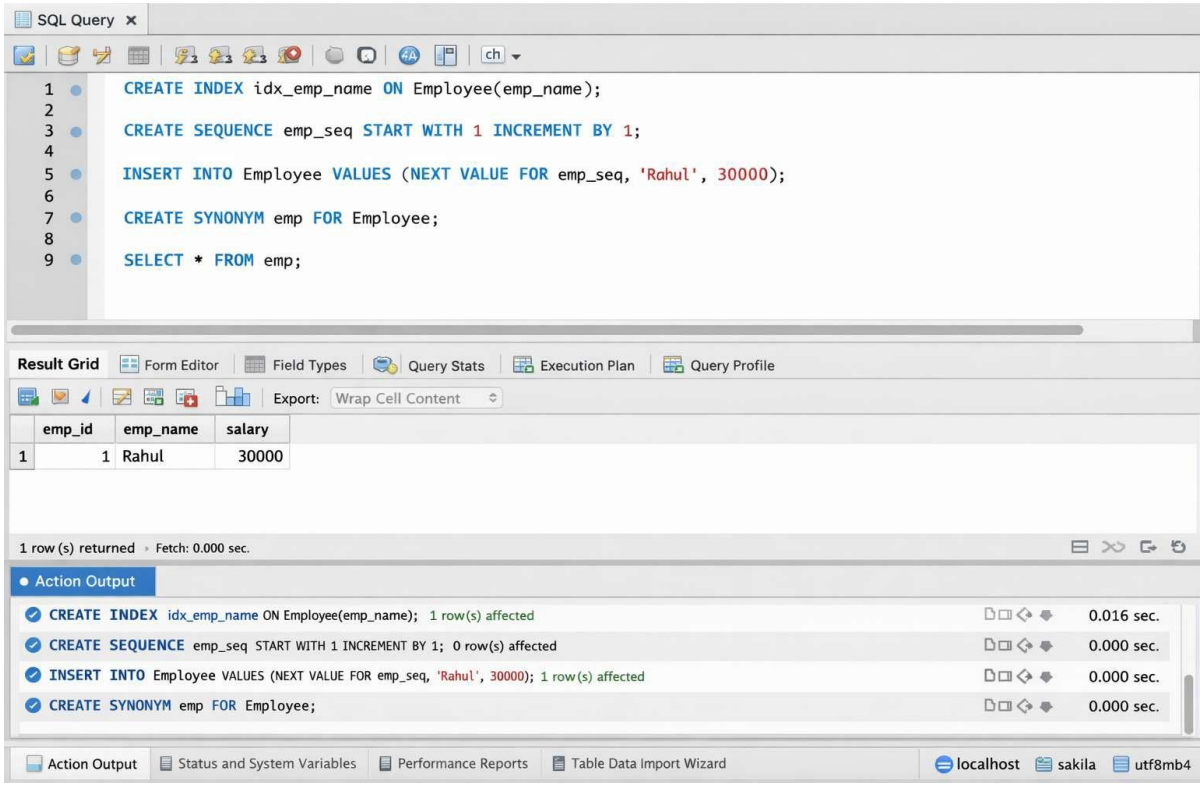
INSERT INTO Employee VALUES (NEXT VALUE FOR emp_seq,'Rahul',30000);

Create Synonym

CREATE SYNONYM emp FOR Employee;

Use Synonym

SELECT * FROM emp;



RESULT: -

The database objects Index, Sequence, and Synonym were created successfully and used for improving data retrieval and object referencing in SQL.