Report On

Database Management System

<u>Submitted To</u>

Prof. Gyaneshwor Dhungana

<u>Submitted By</u>

Sulav Baral

Roll no : 09 (Sec.A)

Symbol no: 80012181

INDEX

Experiment 02

TITLE: Execution of SQL Queries on a Database .

OBJECTIVE: To create a relational database table and perform various SQL queries to retrieve, filter, sort, and analyze data using the MySQL command-line interface.

THEORY:

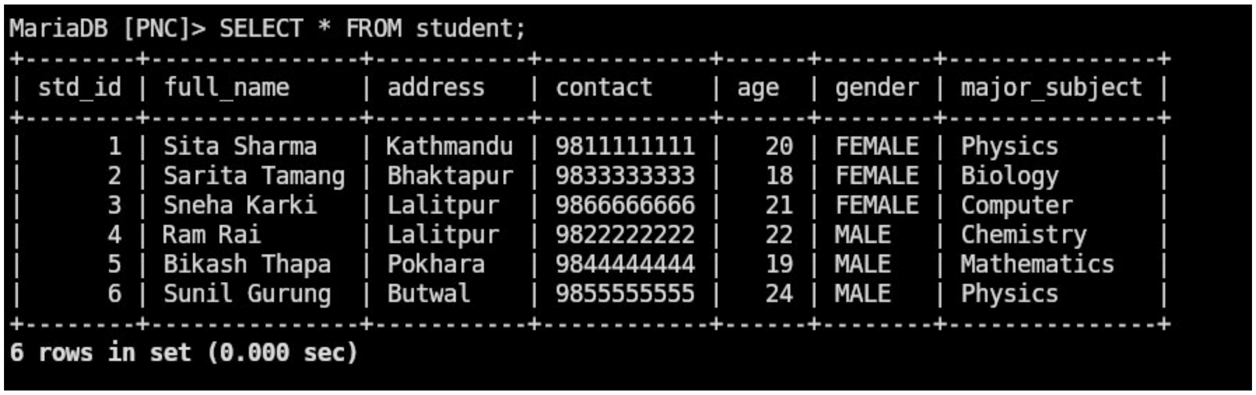
Structured Query Language (SQL) is the standard language used to manage and manipulate data in relational databases. SQL provides various commands for defining structures, inserting data, querying records, updating information, and performing administrative tasks. This lab focuses on:

- **DDL (Data Definition Language):** For creating and managing table structure using commands like CREATE TABLE.
- **DML (Data Manipulation Language):** For interacting with data using SELECT, INSERT, UPDATE, and DELETE.
- **Aggregate Functions:** Such as COUNT(), AVG(), DISTINCT, etc., for summarizing data.
- **Conditional and Pattern Matching Clauses:** Using WHERE, BETWEEN, LIKE, etc.
- **Sorting and Grouping:** Using ORDER BY and GROUP BY to organize data meaningfully.

SQL Queries and Outputs

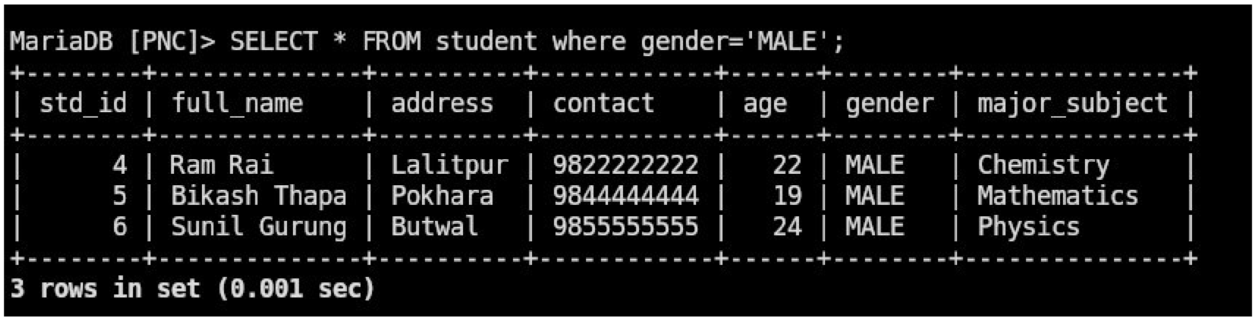
The following are the SQL queries executed during the lab using the MySQL command-line interface, along with their outputs.

1. Select all records



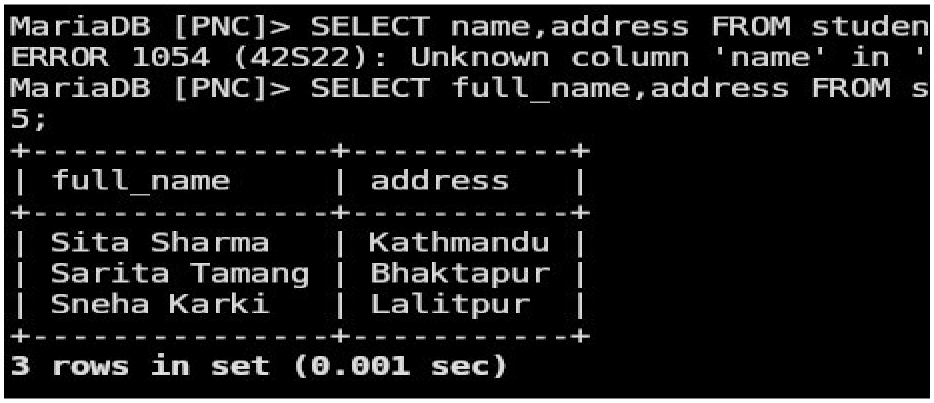
```
MariaDB [PNC]> SELECT * FROM student;
+--------+--------------+-----------+------------+------+--------+---------------+
| std_id | full_name    | address   | contact    | age  | gender | major_subject |
+--------+--------------+-----------+------------+------+--------+---------------+
|      1 | Sita Sharma  | Kathmandu | 9811111111 |   20 | FEMALE | Physics       |
|      2 | Sarita Tamang | Bhaktapur | 9833333333 |   18 | FEMALE | Biology       |
|      3 | Sneha Karki  | Lalitpur  | 9866666666 |   21 | FEMALE | Computer      |
|      4 | Ram Rai      | Lalitpur  | 9822222222 |   22 | MALE   | Chemistry     |
|      5 | Bikash Thapa | Pokhara   | 9844444444 |   19 | MALE   | Mathematics   |
|      6 | Sunil Gurung | Butwal    | 9855555555 |   24 | MALE   | Physics       |
+--------+--------------+-----------+------------+------+--------+---------------+
6 rows in set (0.000 sec)
```

2. Select records of all the males.



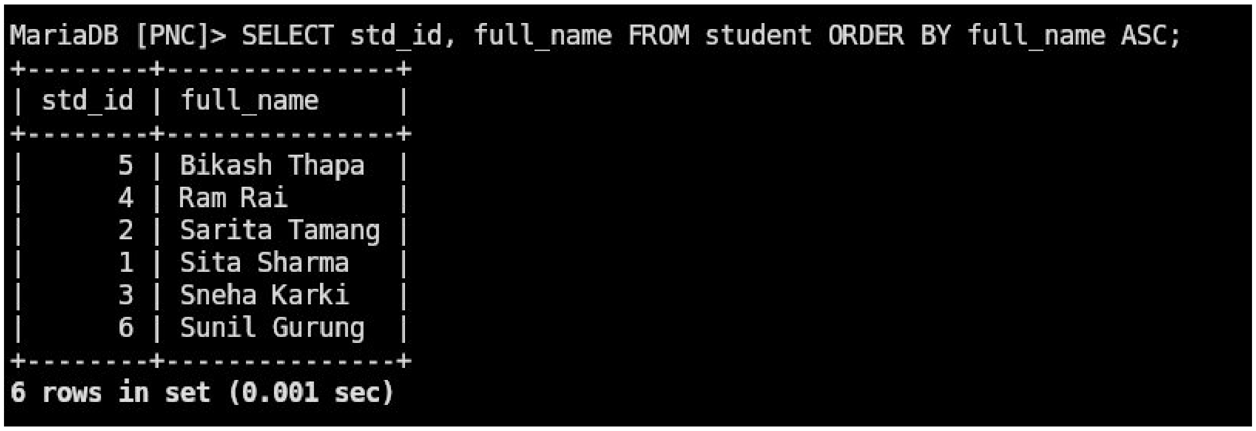
```
MariaDB [PNC]> SELECT * FROM student where gender='MALE';
+--------+--------------+----------+------------+------+--------+---------------+
| std_id | full_name    | address  | contact    | age  | gender | major_subject |
+--------+--------------+----------+------------+------+--------+---------------+
|      4 | Ram Rai      | Lalitpur | 9822222222 |   22 | MALE   | Chemistry     |
|      5 | Bikash Thapa | Pokhara  | 9844444444 |   19 | MALE   | Mathematics   |
|      6 | Sunil Gurung | Butwal   | 9855555555 |   24 | MALE   | Physics       |
+--------+--------------+----------+------------+------+--------+---------------+
3 rows in set (0.001 sec)
```

3. Select name and address of all the female whose age is between 15 and 25



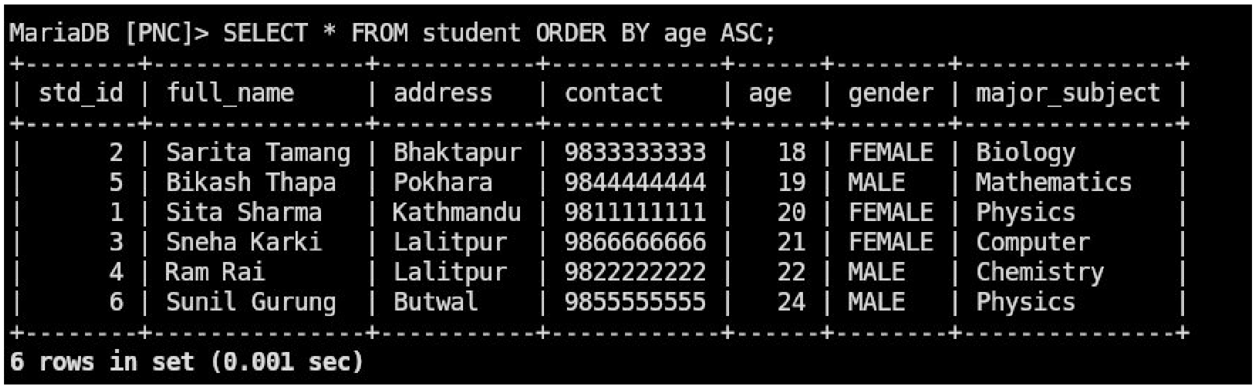
```
MariaDB [PNC]> SELECT name,address FROM studen
ERROR 1054 (42S22): Unknown column 'name' in '
MariaDB [PNC]> SELECT full_name,address FROM s
5;
+--------------+-----------+
| full_name    | address   |
+--------------+-----------+
| Sita Sharma  | Kathmandu |
| Sarita Tamang | Bhaktapur |
| Sneha Karki  | Lalitpur  |
+--------------+-----------+
3 rows in set (0.001 sec)
```

4. Display the stud_id and name of all the students in alphabetical order.



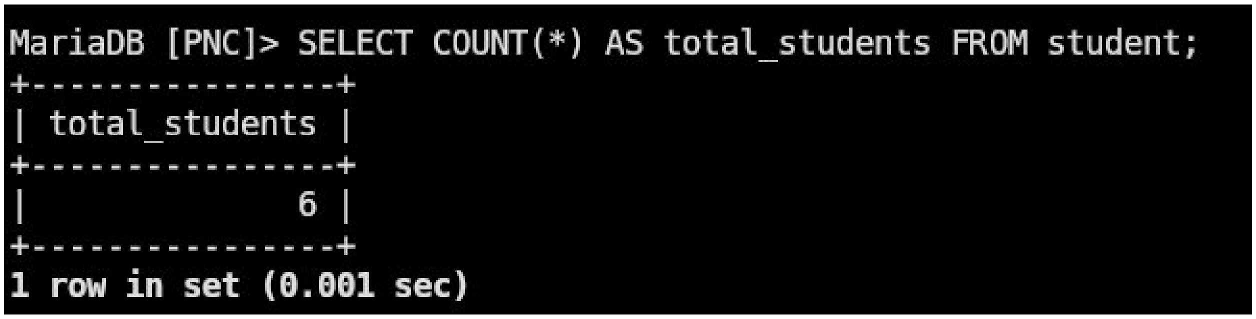
```
MariaDB [PNC]> SELECT std_id, full_name FROM student ORDER BY full_name ASC;
+--------+---------------+
| std_id | full_name     |
+--------+---------------+
|      5 | Bikash Thapa  |
|      4 | Ram Rai       |
|      2 | Sarita Tamang |
|      1 | Sita Sharma   |
|      3 | Sneha Karki   |
|      6 | Sunil Gurung  |
+--------+---------------+
6 rows in set (0.001 sec)
```

5. Display the records of students sorted by age in ascending order.



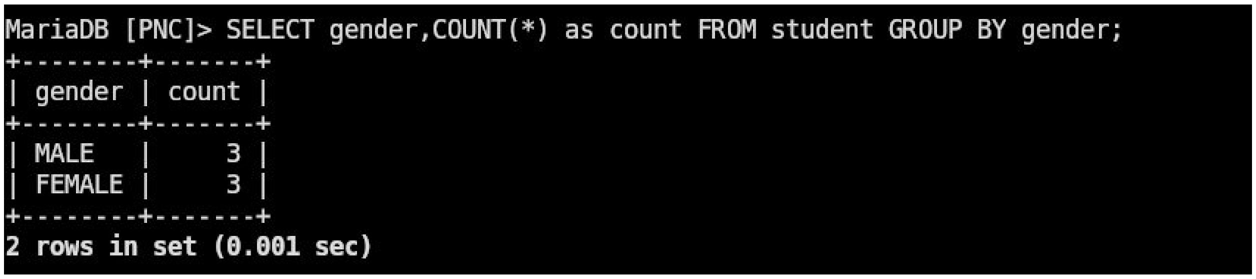
```
MariaDB [PNC]> SELECT * FROM student ORDER BY age ASC;
+--------+---------------+-----------+------------+------+--------+---------------+
| std_id | full_name     | address   | contact    | age  | gender | major_subject |
+--------+---------------+-----------+------------+------+--------+---------------+
|      2 | Sarita Tamang | Bhaktapur | 9833333333 |   18 | FEMALE | Biology       |
|      5 | Bikash Thapa  | Pokhara   | 9844444444 |   19 | MALE   | Mathematics   |
|      1 | Sita Sharma   | Kathmandu | 9811111111 |   20 | FEMALE | Physics       |
|      3 | Sneha Karki   | Lalitpur  | 9866666666 |   21 | FEMALE | Computer      |
|      4 | Ram Rai       | Lalitpur  | 9822222222 |   22 | MALE   | Chemistry     |
|      6 | Sunil Gurung  | Butwal    | 9855555555 |   24 | MALE   | Physics       |
+--------+---------------+-----------+------------+------+--------+---------------+
6 rows in set (0.001 sec)
```

6. Display the total number of students.



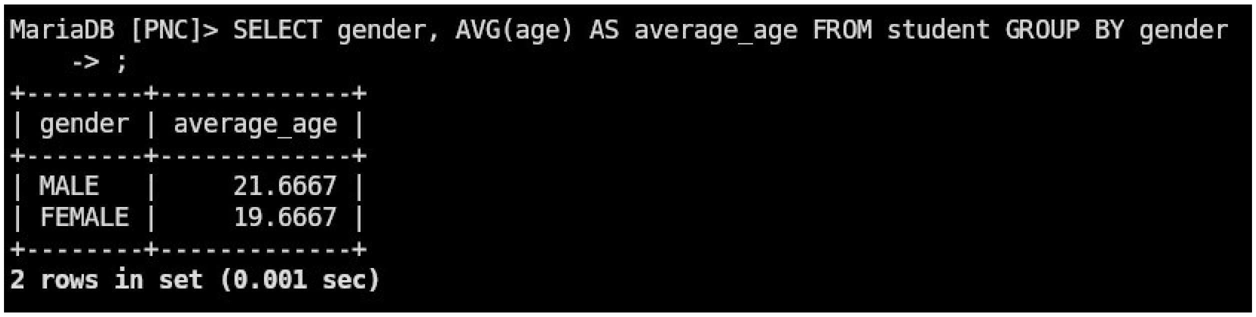
```
MariaDB [PNC]> SELECT COUNT(*) AS total_students FROM student;
+----------------+
| total_students |
+----------------+
|              6 |
+----------------+
1 row in set (0.001 sec)
```

7. Display the number of male and female students.



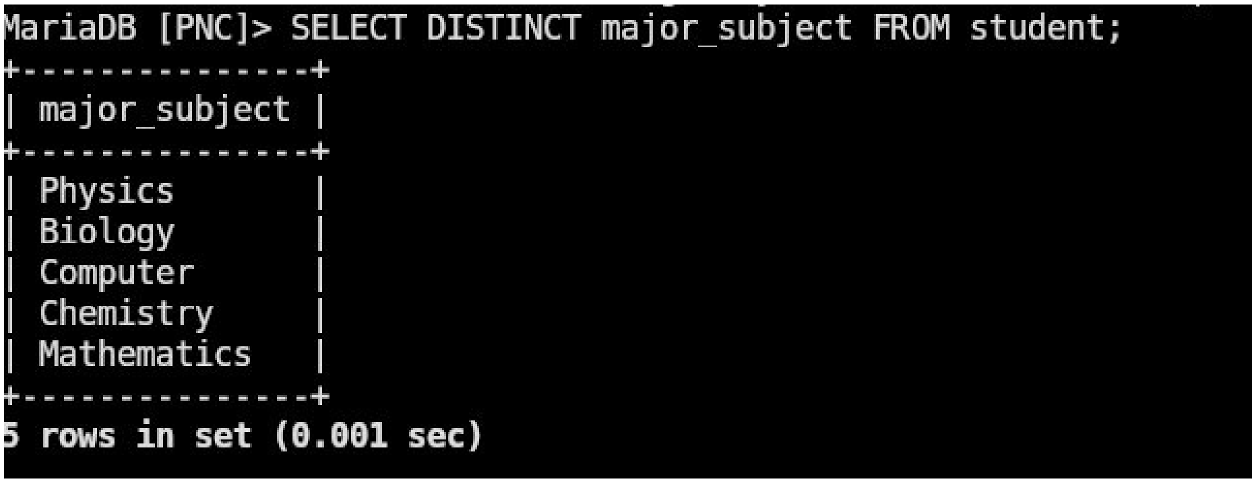
```
MariaDB [PNC]> SELECT gender,COUNT(*) as count FROM student GROUP BY gender;
+--------+-------+
| gender | count |
+--------+-------+
| MALE   |     3 |
| FEMALE |     3 |
+--------+-------+
2 rows in set (0.001 sec)
```

8. Display the average age of all the male and female students.



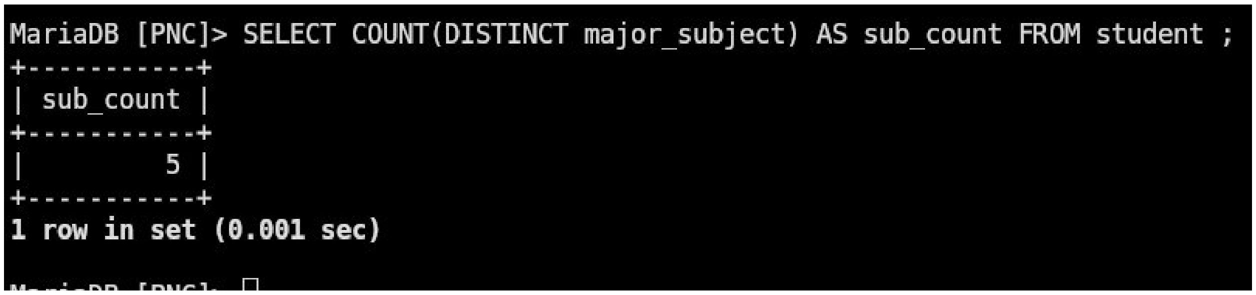
```
MariaDB [PNC]> SELECT gender, AVG(age) AS average_age FROM student GROUP BY gender
    -> ;
+--------+-------------+
| gender | average_age |
+--------+-------------+
| MALE   |     21.6667 |
| FEMALE |     19.6667 |
+--------+-------------+
2 rows in set (0.001 sec)
```

9. Display the distinct major subjects.



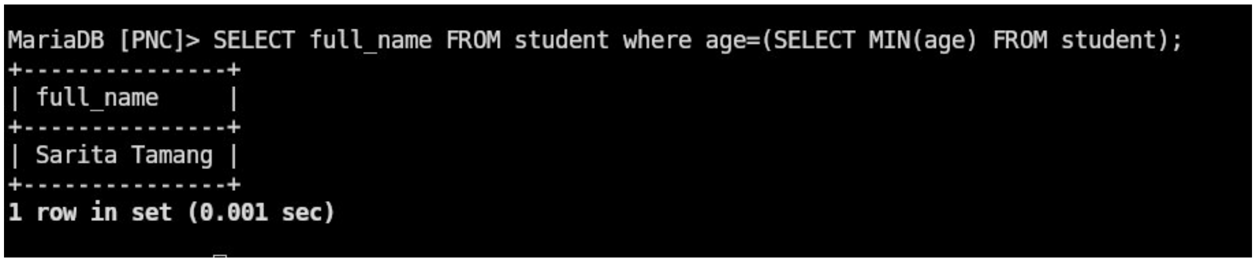
```
MariaDB [PNC]> SELECT DISTINCT major_subject FROM student;
+---------------+
| major_subject |
+---------------+
| Physics       |
| Biology       |
| Computer      |
| Chemistry     |
| Mathematics   |
+---------------+
5 rows in set (0.001 sec)
```

10. Display the number of major subjects taken by students.



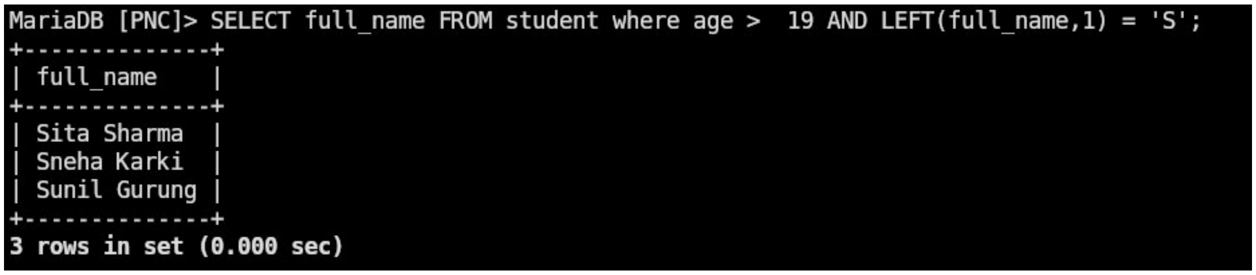
```
MariaDB [PNC]> SELECT COUNT(DISTINCT major_subject) AS sub_count FROM student ;
+-----------+
| sub_count |
+-----------+
|         5 |
+-----------+
1 row in set (0.001 sec)
```

11. Display the name of youngest student



```
MariaDB [PNC]> SELECT full_name FROM student where age=(SELECT MIN(age) FROM student);
+--------------+
| full_name    |
+--------------+
| Sarita Tamang |
+--------------+
1 row in set (0.001 sec)
```

12. Name of all the students whose name starts from 's' and are above 19 years



```
MariaDB [PNC]> SELECT full_name FROM student where age >  19 AND LEFT(full_name,1) = 'S';
+-------------+
| full_name   |
+-------------+
| Sita Sharma  |
| Sneha Karki  |
| Sunil Gurung |
+-------------+
3 rows in set (0.000 sec)
```

CONCLUSION

In this experiment, various SQL queries were executed on a relational database to understand how data can be efficiently stored, retrieved, and analyzed. The use of conditional clauses, sorting, and aggregate functions demonstrated the power of SQL in handling structured data.

Experiment 01

TITLE: Design ER Model for the following databases: Library System and Online Store.

OBJECTIVE: To design a conceptual database schema using the Entity-Relationship (ER) model for a Library System and an Online Store, identifying key entities, their attributes, and the relationships between them.

THEORY:

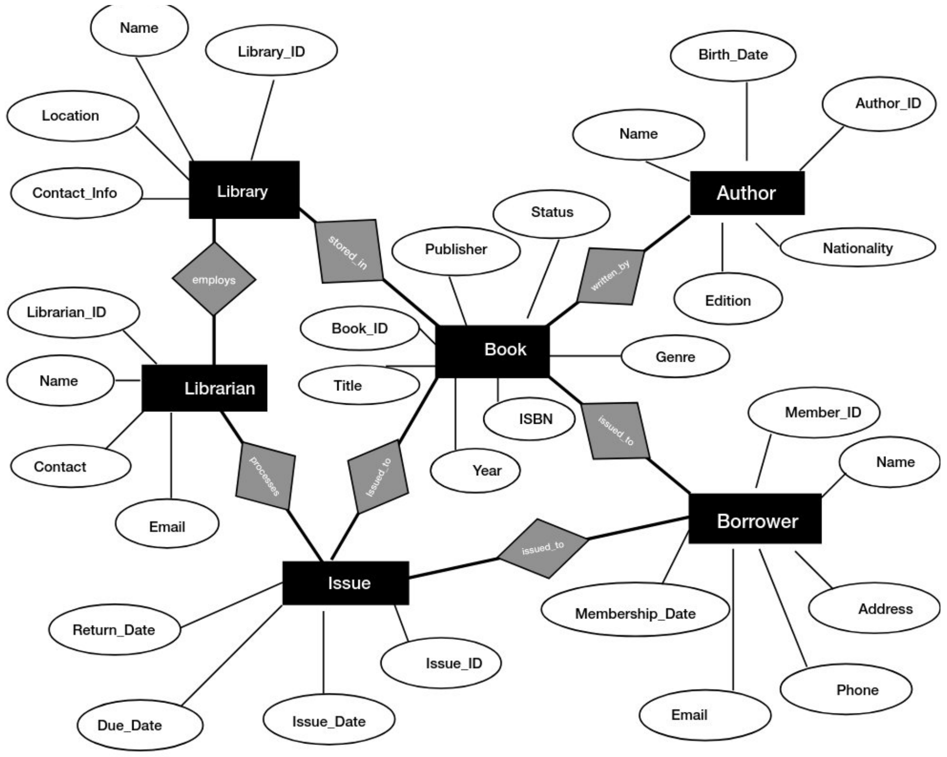
The Entity-Relationship (ER) model is a high-level conceptual data model. It is used to describe the data and relationships between data. An ER model consists of three main components:

- **Entities:** Real-world objects or concepts that can be distinguished from other objects (e.g., Book, Member, Customer).
- **Attributes:** Properties or characteristics of an entity (e.g., Book_id, title, author_name).
- **Relationships:** Associations between entities (e.g., a Member borrows a Book).

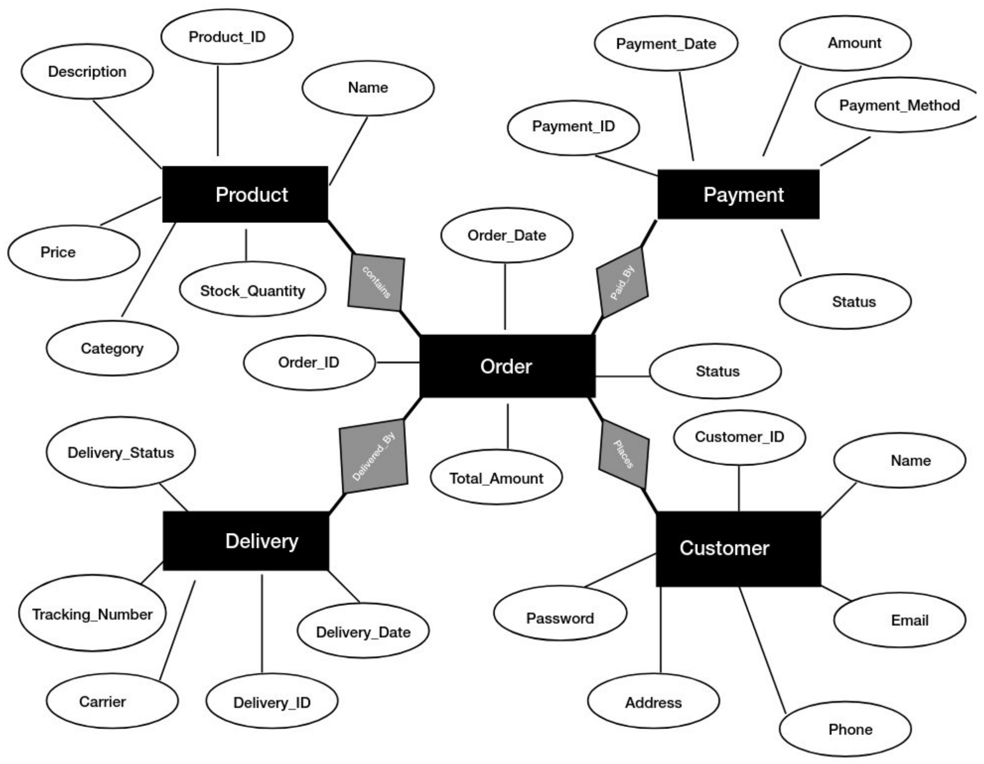
ER diagrams use specific symbols to represent these components: rectangles for entities, ovals for attributes, and diamonds for relationships. The cardinalities (one-to-one, one-to-many, many-to-many) of relationships are also indicated.

ER model design:

1. For Library system



2. For Online store



CONCLUSION

This experiment successfully identified the key entities, attributes, and relationships for both a Library System and an Online Store. The structured descriptions serve as a foundational blueprint for designing the database and implementing the schema.

Lab 3

TITLE: To study different data types in MySQL.

THEORY: In MySQL, data types define the kind of values a column can store in a database table. Choosing the correct data type is very important because it affects storage requirements, performance, and data accuracy. MySQL provides several categories of data types:

1. Numeric Data Types
Used for storing numbers (integers and decimals).

- Integer Types
  - TINYINT → very small integers (-128 to 127)
  - SMALLINT → small integers (-32,768 to 32,767)
  - MEDIUMINT → medium integers (-8 million to 8 million approx.)
  - INT / INTEGER → standard integer (-2 billion to 2 billion)
  - BIGINT → very large integers
- Decimal & Floating Types
  - DECIMAL(M, D) or NUMERIC(M, D) → exact values (good for money)
  - FLOAT → approximate single-precision floating-point numbers
  - DOUBLE / REAL → approximate double-precision floating-point numbers

2. Date and Time Data Types
Used to store dates, times, and timestamps.

- DATE → stores date in format YYYY-MM-DD
- TIME → stores time in format HH:MM:SS
- DATETIME → stores both date and time
- TIMESTAMP → stores date & time (with automatic updates for current time)
- YEAR → stores year in 2-digit or 4-digit format

3. String (Character) Data Types
Used to store text, characters, or binary data.

- Fixed-length strings
  - CHAR(n) → stores fixed-length strings (always uses n characters)
- Variable-length strings
  - VARCHAR(n) → stores variable-length strings (saves space if text length varies)
- Text types (for large text storage)
  - TINYTEXT → up to 255 characters
  - TEXT → up to 65,535 characters
  - MEDIUMTEXT → up to 16 million characters
  - LONGTEXT → up to 4 billion characters
- Binary types (for images, files, etc.)
  - BLOB (Binary Large Object) with variations:

        - TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
- Others
    - ENUM('val1', 'val2', …) → predefined set of values (choose one)
    - SET('val1', 'val2', …) → predefined set of values (choose multiple)

4. Boolean Data Type

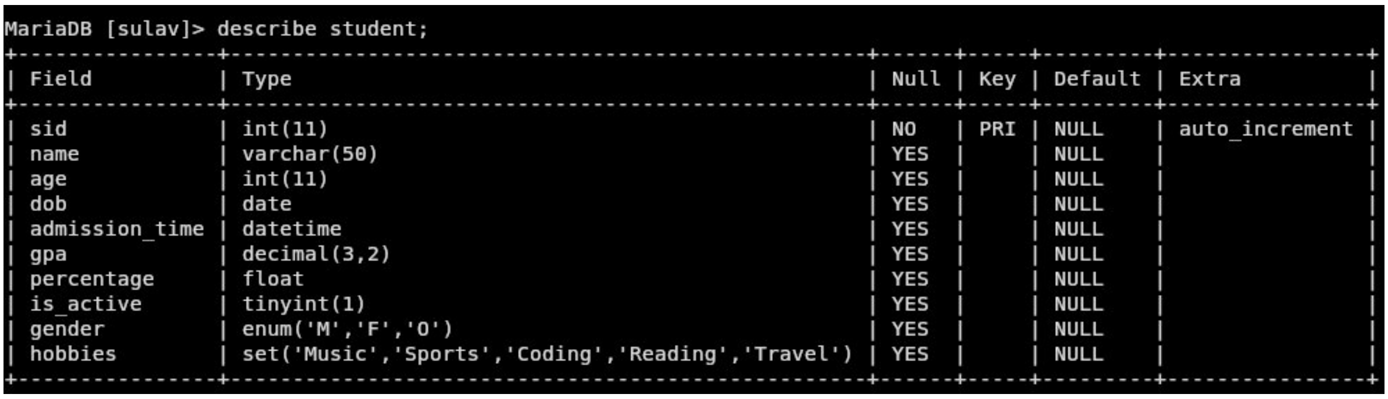
- MySQL doesn't have a direct BOOLEAN type.
- Instead, BOOLEAN is treated as a synonym for TINYINT(1), where 0 = FALSE and 1 = TRUE

Key Points:

- Always select the most efficient data type (saves memory & improves performance).
- Use DECIMAL for exact precision (e.g., financial data) instead of FLOAT.
- For strings, use CHAR when values have a fixed length and VARCHAR when lengths vary.
- ENUM and SET help restrict values to predefined options.
- Date and time types are best for chronological data handling instead of storing as strings.

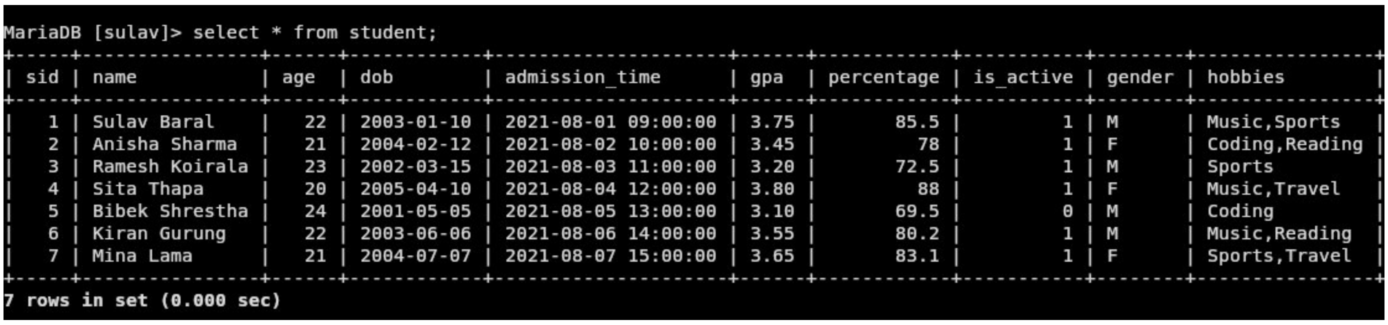
QUERIES:

Database with a table named student with multiple datatypes:



```
MariaDB [sulav]> describe student;
+----------------+------------------------------------------------------+------+-----+---------+----------------+
| Field          | Type                                                 | Null | Key | Default | Extra          |
+----------------+------------------------------------------------------+------+-----+---------+----------------+
| sid            | int(11)                                              | NO   | PRI | NULL    | auto_increment |
| name           | varchar(50)                                          | YES  |     | NULL    |                |
| age            | int(11)                                              | YES  |     | NULL    |                |
| dob            | date                                                 | YES  |     | NULL    |                |
| admission_time | datetime                                             | YES  |     | NULL    |                |
| gpa            | decimal(3,2)                                         | YES  |     | NULL    |                |
| percentage     | float                                                | YES  |     | NULL    |                |
| is_active      | tinyint(1)                                           | YES  |     | NULL    |                |
| gender         | enum('M','F','O')                                    | YES  |     | NULL    |                |
| hobbies        | set('Music','Sports','Coding','Reading','Travel')    | YES  |     | NULL    |                |
+----------------+------------------------------------------------------+------+-----+---------+----------------+
```

Insert into table student:



```
MariaDB [sulav]> select * from student;
+-----+-----------------+------+------------+---------------------+------+------------+-----------+--------+-----------------+
| sid | name            | age  | dob        | admission_time      | gpa  | percentage | is_active | gender | hobbies         |
+-----+-----------------+------+------------+---------------------+------+------------+-----------+--------+-----------------+
|   1 | Sulav Baral     |   22 | 2003-01-10 | 2021-08-01 09:00:00 | 3.75 |       85.5 |         1 | M      | Music,Sports    |
|   2 | Anisha Sharma   |   21 | 2004-02-12 | 2021-08-02 10:00:00 | 3.45 |         78 |         1 | F      | Coding,Reading  |
|   3 | Ramesh Koirala  |   23 | 2002-03-15 | 2021-08-03 11:00:00 | 3.20 |       72.5 |         1 | M      | Sports          |
|   4 | Sita Thapa      |   20 | 2005-04-10 | 2021-08-04 12:00:00 | 3.80 |         88 |         1 | F      | Music,Travel    |
|   5 | Bibek Shrestha  |   24 | 2001-05-05 | 2021-08-05 13:00:00 | 3.10 |       69.5 |         0 | M      | Coding          |
|   6 | Kiran Gurung    |   22 | 2003-06-06 | 2021-08-06 14:00:00 | 3.55 |       80.2 |         1 | M      | Music,Reading   |
|   7 | Mina Lama       |   21 | 2004-07-07 | 2021-08-07 15:00:00 | 3.65 |       83.1 |         1 | F      | Sports,Travel   |
+-----+-----------------+------+------------+---------------------+------+------------+-----------+--------+-----------------+
7 rows in set (0.000 sec)
```

CONCLUSION:

Understanding MySQL data types is fundamental to effective database design. Each category—numeric, string, date/time, and special—serves a distinct purpose and offers specific advantages. Selecting the appropriate data type ensures efficient storage, accurate data representation, and optimal performance. This lab provides a foundational overview that supports deeper exploration into schema design, normalization, and query optimization.

Lab 4

Title: To study and implement the alter command in MYSQL.

THEORY:

The ALTER command in MySQL is **a** Data Definition Language (DDL) command.
It is used to modify the structure of an existing table without dropping and recreating it.

With ALTER TABLE, you can:

- Add new columns
- Modify existing columns (datatype, size, constraints)
- Rename a column
- Drop (delete) a column
- Rename the table itself
- Add or drop constraints (like PRIMARY KEY, FOREIGN KEY, UNIQUE)
- Change default values

Common Uses of ALTER TABLE

Action	Syntax and Example
Basic Syntax Change table schema	ALTER TABLE table_name Action…;

Add a column	ALTER TABLE table_name
	ADD COLUMN column_name datatype [constraints] [after/before col];
	
	ALTER TABLE employees
	ADD COLUMN phone VARCHAR(15);

Remove Column	ALTER TABLE table_name
	DROP COLUMN column_name;
	
	ALTER TABLE employees
	DROP COLUMN phone;

Modify a column	ALTER TABLE table_name
	MODIFY COLUMN column_name new_datatype [constraints];
	
	ALTER TABLE employees
	MODIFY COLUMN phone VARCHAR(20);

Change column name and properties	ALTER TABLE table_name CHANGE old_column_name new_column_name new_datatype [constraints]; ALTER TABLE employees CHANGE phone contact_number VARCHAR(20);

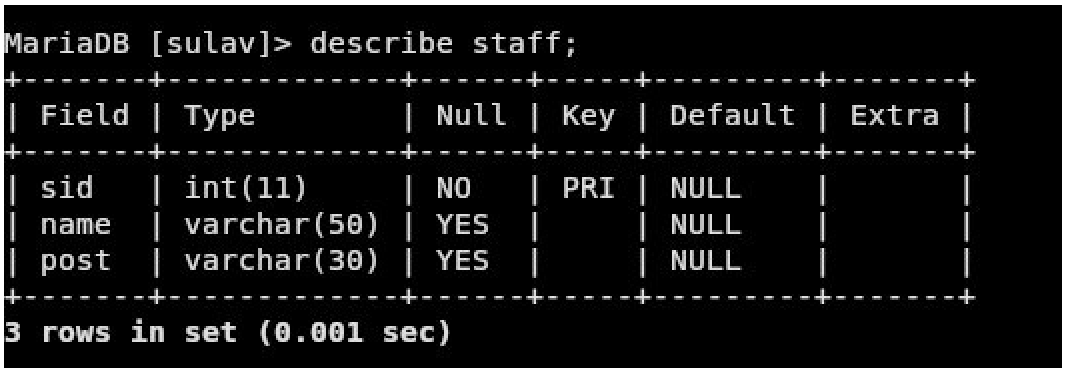
Change table name	ALTER TABLE old_table_name
	RENAME TO new_table_name;
	
	ALTER TABLE employees
	RENAME TO staff;

Add or remove primary key	ALTER TABLE table_name
	ADD PRIMARY KEY (column_name);
	
	ALTER TABLE table_name
	DROP PRIMARY KEY;

Perform multiple actions	ALTER TABLE employees ADD COLUMN age INT, DROP COLUMN phone;

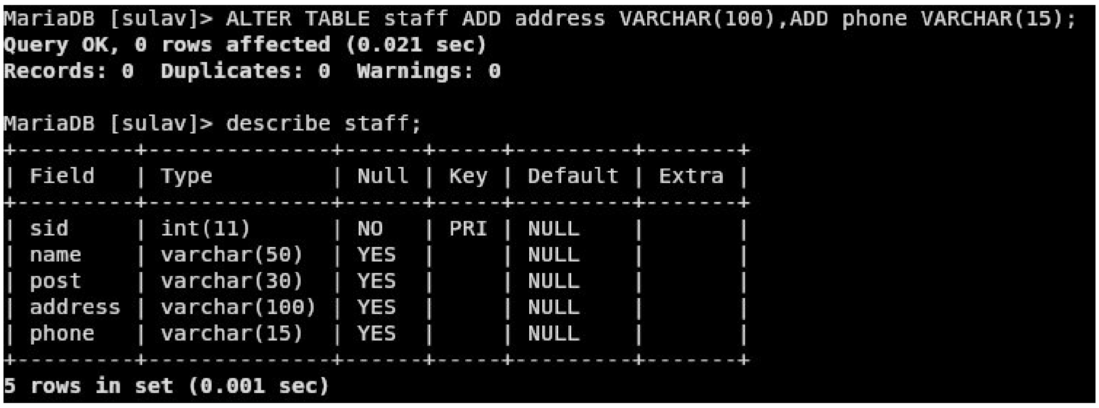
QUERIES:

Create database:



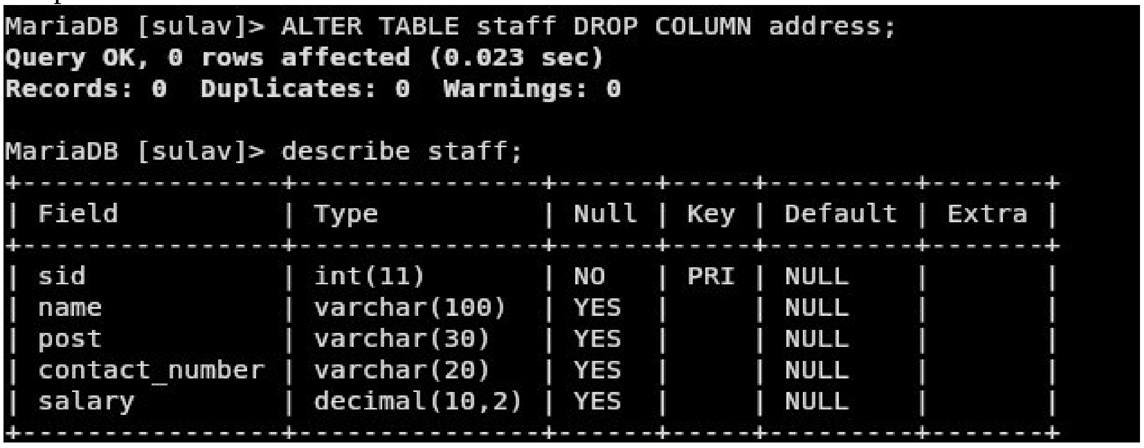
```
MariaDB [sulav]> describe staff;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| sid   | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(50) | YES  |     | NULL    |       |
| post  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.001 sec)
```

Add column:



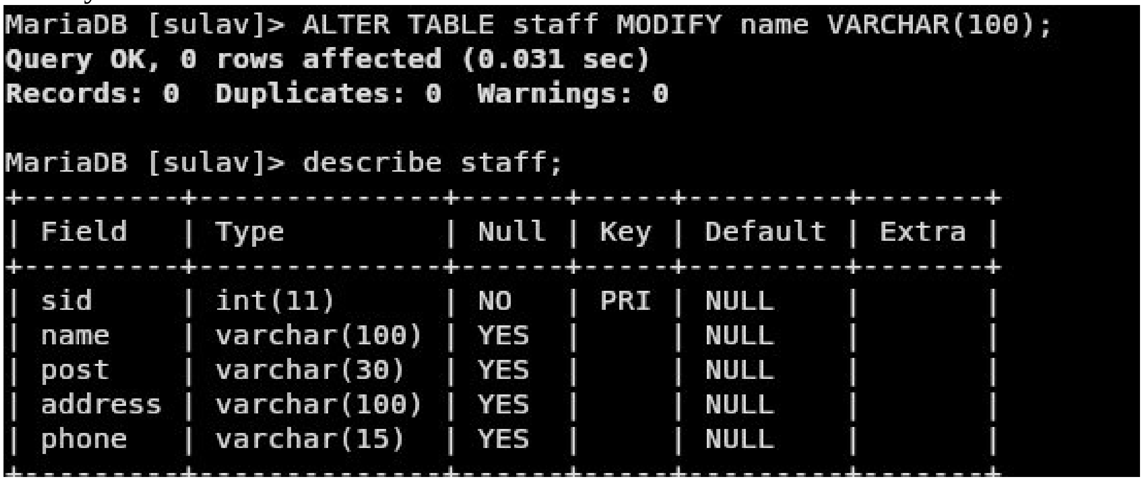
```
MariaDB [sulav]> ALTER TABLE staff ADD address VARCHAR(100),ADD phone VARCHAR(15);
Query OK, 0 rows affected (0.021 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> describe staff;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| sid     | int(11)      | NO   | PRI | NULL    |       |
| name    | varchar(50)  | YES  |     | NULL    |       |
| post    | varchar(30)  | YES  |     | NULL    |       |
| address | varchar(100) | YES  |     | NULL    |       |
| phone   | varchar(15)  | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
5 rows in set (0.001 sec)
```

Drop Column:



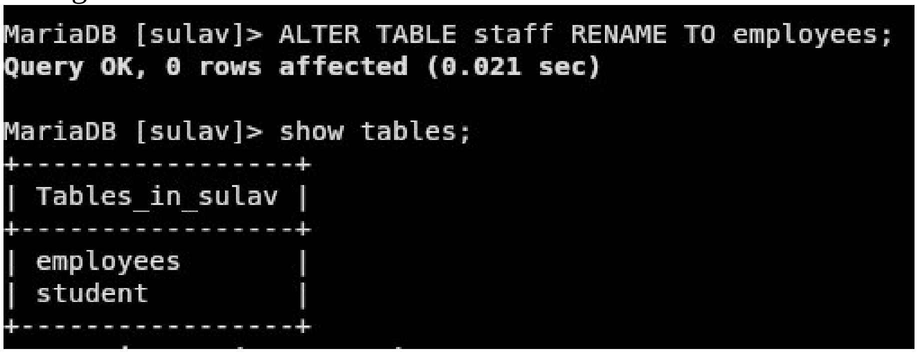
```
MariaDB [sulav]> ALTER TABLE staff DROP COLUMN address;
Query OK, 0 rows affected (0.023 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> describe staff;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| sid            | int(11)       | NO   | PRI | NULL    |       |
| name           | varchar(100)  | YES  |     | NULL    |       |
| post           | varchar(30)   | YES  |     | NULL    |       |
| contact_number | varchar(20)   | YES  |     | NULL    |       |
| salary         | decimal(10,2) | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
```

Modify a column:



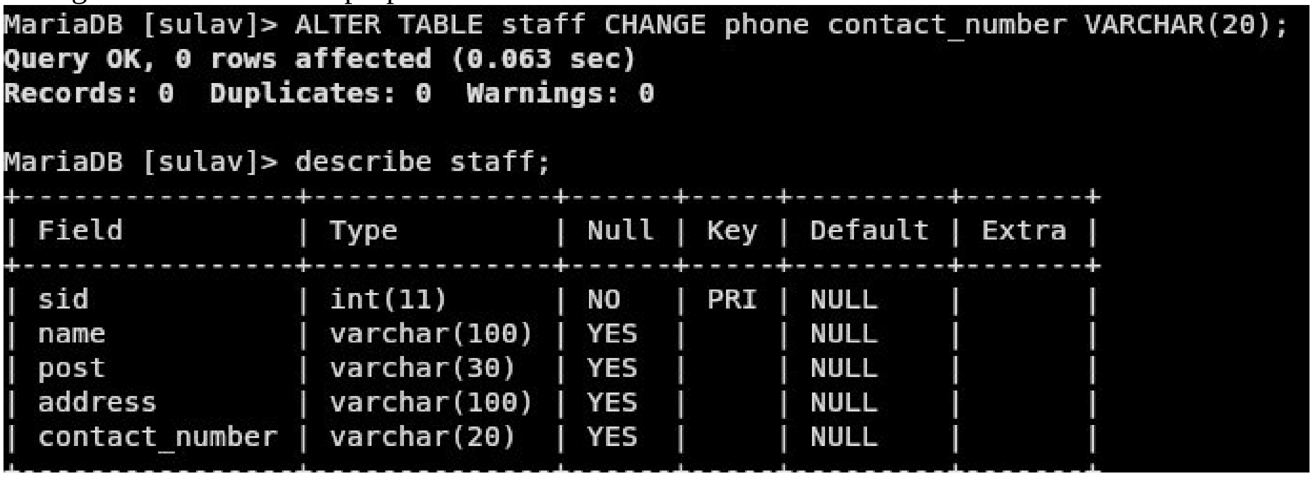
```
MariaDB [sulav]> ALTER TABLE staff MODIFY name VARCHAR(100);
Query OK, 0 rows affected (0.031 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> describe staff;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| sid     | int(11)      | NO   | PRI | NULL    |       |
| name    | varchar(100) | YES  |     | NULL    |       |
| post    | varchar(30)  | YES  |     | NULL    |       |
| address | varchar(100) | YES  |     | NULL    |       |
| phone   | varchar(15)  | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
```

Change table name:



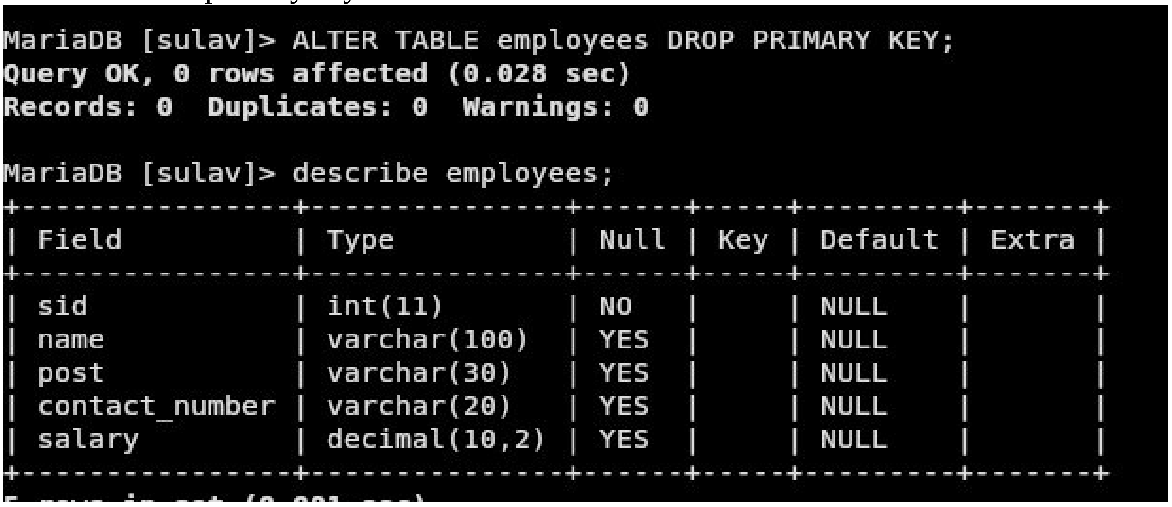
```
MariaDB [sulav]> ALTER TABLE staff RENAME TO employees;
Query OK, 0 rows affected (0.021 sec)

MariaDB [sulav]> show tables;
+-----------------+
| Tables_in_sulav |
+-----------------+
| employees       |
| student         |
+-----------------+
```

Change column name and properties:



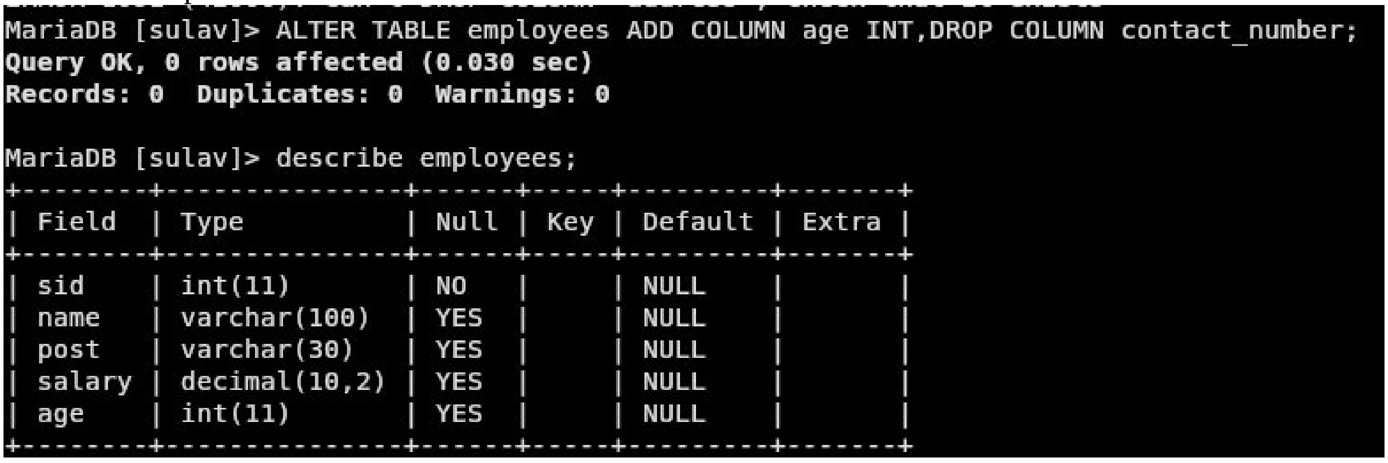
```
MariaDB [sulav]> ALTER TABLE staff CHANGE phone contact_number VARCHAR(20);
Query OK, 0 rows affected (0.063 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> describe staff;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| sid            | int(11)      | NO   | PRI | NULL    |       |
| name           | varchar(100) | YES  |     | NULL    |       |
| post           | varchar(30)  | YES  |     | NULL    |       |
| address        | varchar(100) | YES  |     | NULL    |       |
| contact_number | varchar(20)  | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
```

Add or remove primary key:



```
MariaDB [sulav]> ALTER TABLE employees DROP PRIMARY KEY;
Query OK, 0 rows affected (0.028 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> describe employees;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| sid            | int(11)       | NO   |     | NULL    |       |
| name           | varchar(100)  | YES  |     | NULL    |       |
| post           | varchar(30)   | YES  |     | NULL    |       |
| contact_number | varchar(20)   | YES  |     | NULL    |       |
| salary         | decimal(10,2) | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
```

Perform multiple actions:



```
MariaDB [sulav]> ALTER TABLE employees ADD COLUMN age INT,DROP COLUMN contact_number;
Query OK, 0 rows affected (0.030 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> describe employees;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| sid    | int(11)       | NO   |     | NULL    |       |
| name   | varchar(100)  | YES  |     | NULL    |       |
| post   | varchar(30)   | YES  |     | NULL    |       |
| salary | decimal(10,2) | YES  |     | NULL    |       |
| age    | int(11)       | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

CONCLUSION:

Hence, the ALTER command in MySQL provides a flexible way to modify table structures without losing existing data. It enables adding, modifying, or dropping columns, renaming tables or columns, and managing primary keys, making it essential for efficient and organized database management.

Lab 5

TITLE: To study and implement the DROP command in MYSQL.

THEORY: The DROP command in MySQL is a powerful SQL statement used to permanently delete database objects, including databases, tables, columns, indexes, and constraints. Unlike commands such as DELETE or TRUNCATE, which only remove data while keeping the table or structure intact, the DROP command completely removes both the object and all the data it contains, leaving no trace behind. This means that once executed, the operation is irreversible, and the deleted objects cannot be recovered unless a prior backup exists. The DROP command is widely used in database management for cleaning up obsolete or unused objects, restructuring databases, or freeing up storage space. Still, its destructive nature makes it extremely sensitive. Therefore, it requires careful planning, proper authorization, and caution before execution to prevent accidental loss of valuable data. It is an essential tool for database administrators, but it must always be used responsibly to maintain the integrity and stability of the database system.

Precautions:

1. The DROP command is irreversible; once executed, data cannot be retrieved unless a backup exists.
2. It helps in database maintenance, cleaning up unused objects, and redesigning structures.
3. Proper authorization and caution are required since accidental usage can lead to data loss.
4. Always ensure backups exist before performing drop operations, especially on important databases or tables.

Key Uses of the DROP Command:

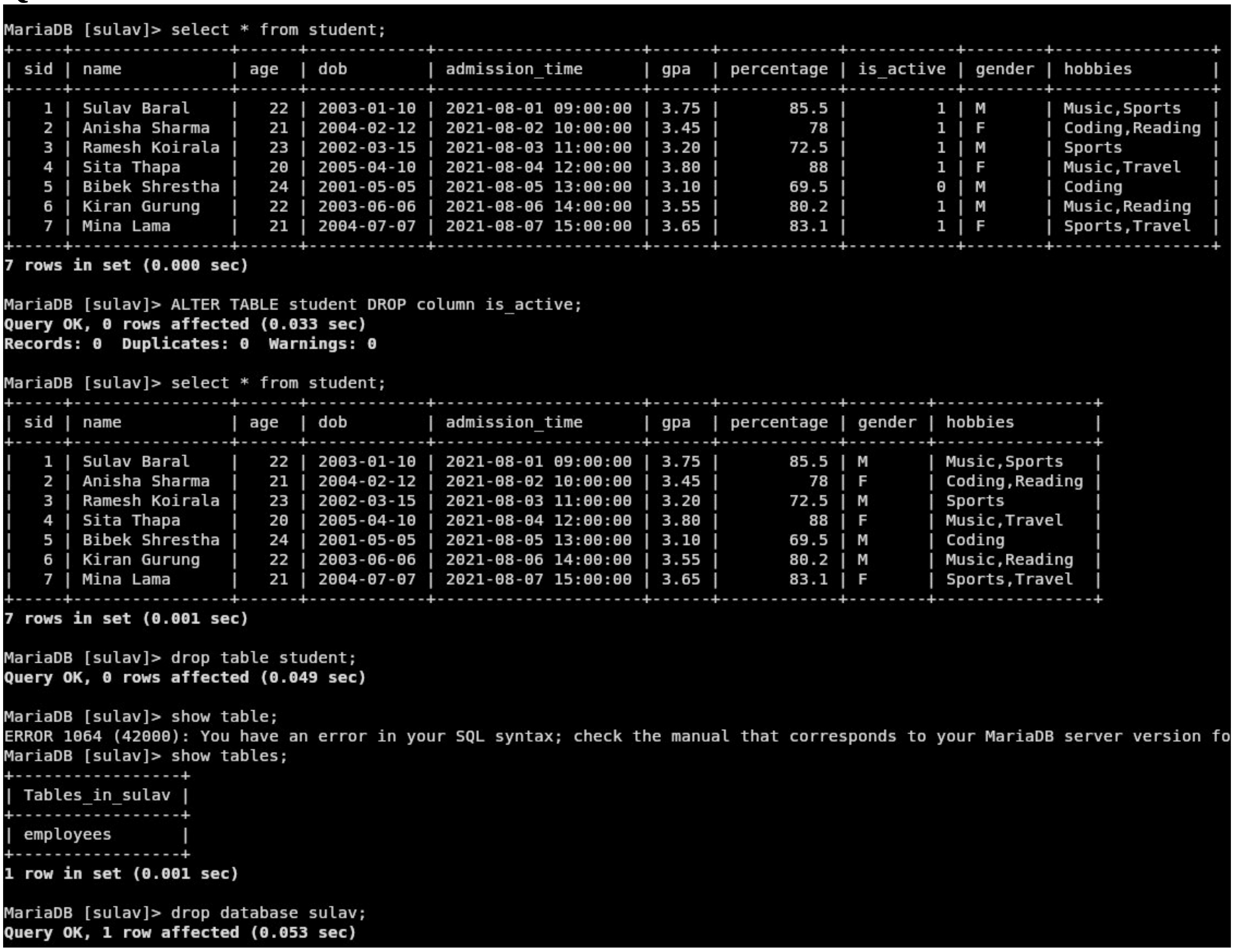
- Drop a Database: Removes an entire database including all tables, views, triggers, and procedures inside it.

DROP DATABASE database_name;

- Drop a Table: Deletes an entire table along with all its rows and structure. DROP TABLE table_name;
- Drop a Column: Removes a column from an existing table. ALTER TABLE table_name DROP

  COLUMN column_name;
- Drop an Index or Constraint: Deletes an index or a foreign key constraint from a table. DROP INDEX index_name ON table_name;

QUERIES:



```
MariaDB [sulav]> select * from student;
+-----+---------------+------+------------+---------------------+------+------------+-----------+--------+----------------+
| sid | name          | age  | dob        | admission_time      | gpa  | percentage | is_active | gender | hobbies        |
+-----+---------------+------+------------+---------------------+------+------------+-----------+--------+----------------+
|   1 | Sulav Baral   |   22 | 2003-01-10 | 2021-08-01 09:00:00 | 3.75 |       85.5 |         1 | M      | Music,Sports   |
|   2 | Anisha Sharma |   21 | 2004-02-12 | 2021-08-02 10:00:00 | 3.45 |         78 |         1 | F      | Coding,Reading |
|   3 | Ramesh Koirala |  23 | 2002-03-15 | 2021-08-03 11:00:00 | 3.20 |       72.5 |         1 | M      | Sports         |
|   4 | Sita Thapa    |   20 | 2005-04-10 | 2021-08-04 12:00:00 | 3.80 |         88 |         1 | F      | Music,Travel   |
|   5 | Bibek Shrestha |  24 | 2001-05-05 | 2021-08-05 13:00:00 | 3.10 |       69.5 |         0 | M      | Coding         |
|   6 | Kiran Gurung  |   22 | 2003-06-06 | 2021-08-06 14:00:00 | 3.55 |       80.2 |         1 | M      | Music,Reading  |
|   7 | Mina Lama     |   21 | 2004-07-07 | 2021-08-07 15:00:00 | 3.65 |       83.1 |         1 | F      | Sports,Travel  |
+-----+---------------+------+------------+---------------------+------+------------+-----------+--------+----------------+
7 rows in set (0.000 sec)

MariaDB [sulav]> ALTER TABLE student DROP column is_active;
Query OK, 0 rows affected (0.033 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> select * from student;
+-----+---------------+------+------------+---------------------+------+------------+--------+----------------+
| sid | name          | age  | dob        | admission_time      | gpa  | percentage | gender | hobbies        |
+-----+---------------+------+------------+---------------------+------+------------+--------+----------------+
|   1 | Sulav Baral   |   22 | 2003-01-10 | 2021-08-01 09:00:00 | 3.75 |       85.5 | M      | Music,Sports   |
|   2 | Anisha Sharma |   21 | 2004-02-12 | 2021-08-02 10:00:00 | 3.45 |         78 | F      | Coding,Reading |
|   3 | Ramesh Koirala |  23 | 2002-03-15 | 2021-08-03 11:00:00 | 3.20 |       72.5 | M      | Sports         |
|   4 | Sita Thapa    |   20 | 2005-04-10 | 2021-08-04 12:00:00 | 3.80 |         88 | F      | Music,Travel   |
|   5 | Bibek Shrestha |  24 | 2001-05-05 | 2021-08-05 13:00:00 | 3.10 |       69.5 | M      | Coding         |
|   6 | Kiran Gurung  |   22 | 2003-06-06 | 2021-08-06 14:00:00 | 3.55 |       80.2 | M      | Music,Reading  |
|   7 | Mina Lama     |   21 | 2004-07-07 | 2021-08-07 15:00:00 | 3.65 |       83.1 | F      | Sports,Travel  |
+-----+---------------+------+------------+---------------------+------+------------+--------+----------------+
7 rows in set (0.001 sec)

MariaDB [sulav]> drop table student;
Query OK, 0 rows affected (0.049 sec)

MariaDB [sulav]> show table;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version fo
MariaDB [sulav]> show tables;
+-----------------+
| Tables_in_sulav |
+-----------------+
| employees       |
+-----------------+
1 row in set (0.001 sec)

MariaDB [sulav]> drop database sulav;
Query OK, 1 row affected (0.053 sec)
```

CONCLUSION:

Hence, the DROP command in MySQL provides a powerful way to permanently remove databases, tables, or columns. It allows for efficient database maintenance and restructuring, but must be used carefully, as the deleted data and structures cannot be recovered without a backup.

Lab 6

TITLE: To query the given relational database in order to retrieve the required information by using the join operations.

THEORY: In relational databases, data is often distributed across multiple tables to reduce redundancy, maintain consistency, and improve data organization. To retrieve meaningful information that spans multiple tables, JOIN operations are used. JOINs allow combining rows from two or more tables based on a relationship between columns, enabling comprehensive and useful datasets.

Types of joins:

Inner join:

- Retrieves only the rows that have matching values in both tables.
- Ensures that the result includes only related data, ignoring unmatched rows.
- Useful when you need information that exists in both tables, such as students with enrolled courses.

Left join (left outer join):

- Returns all rows from the left table, along with the matching rows from the right table.
- If there is no match in the right table, the result will contain NULL values for the right table's columns.
- Useful for situations where you want to retain all data from the primary table, such as listing all students and showing course information where available.

Right join (right outer join):

- Returns all rows from the right table, along with the matching rows from the left table.
- Unmatched rows from the left table appear as NULL.
- Useful when the focus is on the secondary table, such as listing all courses and showing enrolled students if any.

Full outer join:

- Combines all rows from both tables. If a row has no match in the other table, NULL values are shown in the unmatched columns.
- Provides a complete dataset that includes both matched and unmatched data from all tables.
- Useful for comprehensive reports, such as showing all students and all courses, even if some students are not enrolled or some courses have no students.

Natural join:

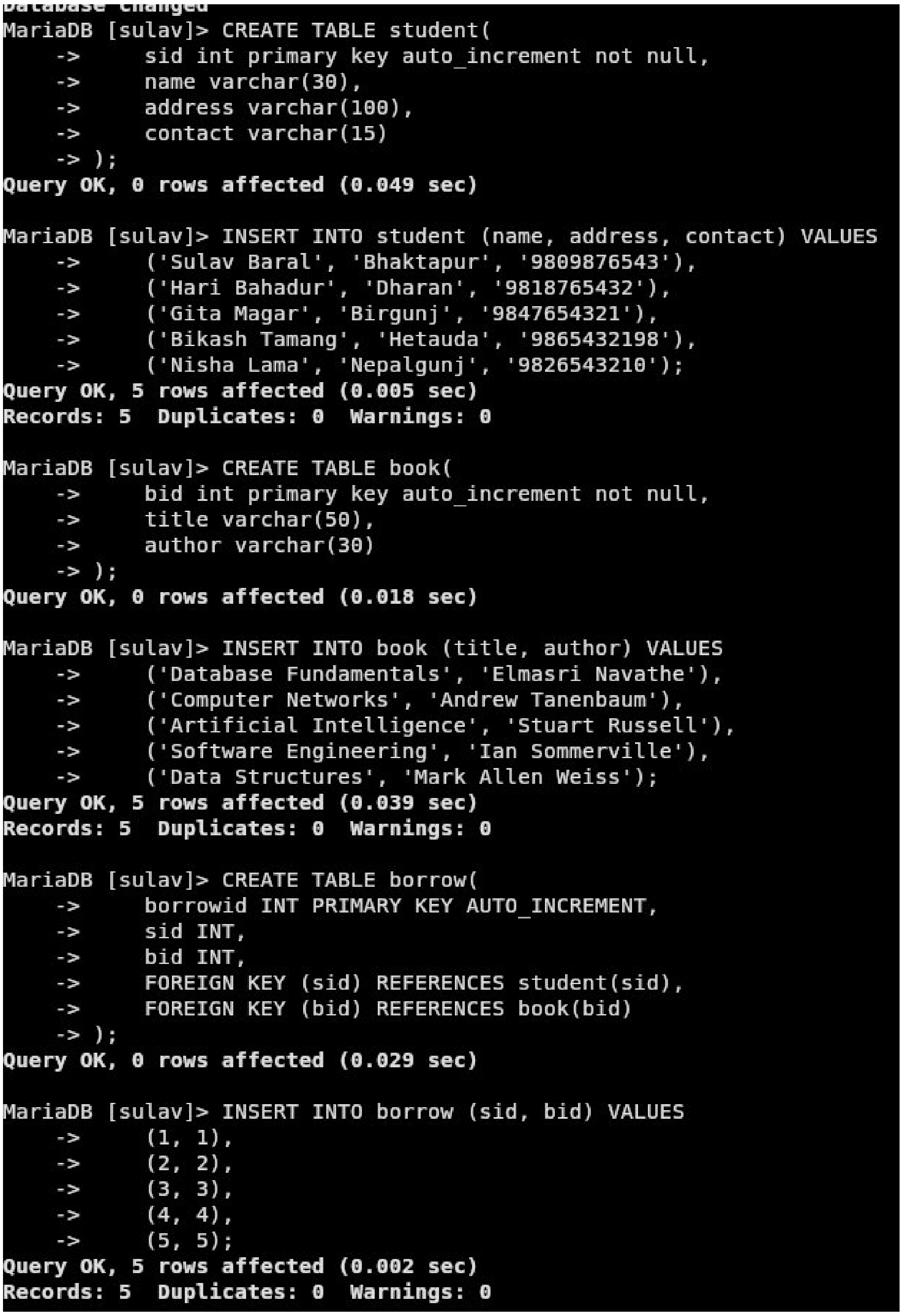
- Automatically joins tables based on columns that share the same name and compatible data types.
- Reduces the need to explicitly specify join conditions, simplifying queries when tables have standardized column names.
- Useful for quickly combining related tables without manually identifying the common columns.

Theta join:

- A generalized form of join where any comparison operator can be used (e.g., >, <, >=,
<=, <>), not just equality.
- Allows flexible conditions for combining tables based on a specific relationship between columns.
- Useful for complex queries, such as finding students with marks greater than the average of another table's values.

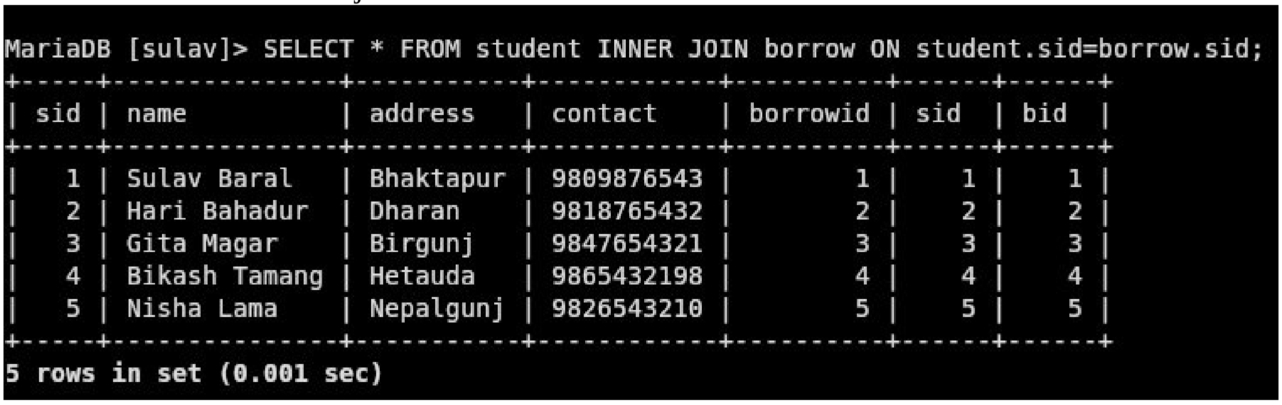
QUERIES:

Create database:



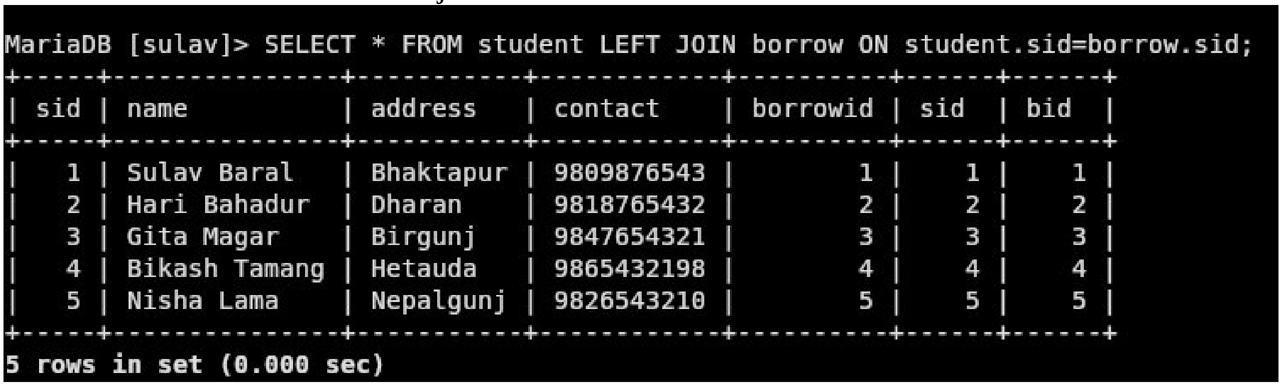
```
Database changed
MariaDB [sulav]> CREATE TABLE student(
    ->     sid int primary key auto_increment not null,
    ->     name varchar(30),
    ->     address varchar(100),
    ->     contact varchar(15)
    -> );
Query OK, 0 rows affected (0.049 sec)

MariaDB [sulav]> INSERT INTO student (name, address, contact) VALUES
    ->     ('Sulav Baral', 'Bhaktapur', '9809876543'),
    ->     ('Hari Bahadur', 'Dharan', '9818765432'),
    ->     ('Gita Magar', 'Birgunj', '9847654321'),
    ->     ('Bikash Tamang', 'Hetauda', '9865432198'),
    ->     ('Nisha Lama', 'Nepalgunj', '9826543210');
Query OK, 5 rows affected (0.005 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [sulav]> CREATE TABLE book(
    ->     bid int primary key auto_increment not null,
    ->     title varchar(50),
    ->     author varchar(30)
    -> );
Query OK, 0 rows affected (0.018 sec)

MariaDB [sulav]> INSERT INTO book (title, author) VALUES
    ->     ('Database Fundamentals', 'Elmasri Navathe'),
    ->     ('Computer Networks', 'Andrew Tanenbaum'),
    ->     ('Artificial Intelligence', 'Stuart Russell'),
    ->     ('Software Engineering', 'Ian Sommerville'),
    ->     ('Data Structures', 'Mark Allen Weiss');
Query OK, 5 rows affected (0.039 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [sulav]> CREATE TABLE borrow(
    ->     borrowid INT PRIMARY KEY AUTO_INCREMENT,
    ->     sid INT,
    ->     bid INT,
    ->     FOREIGN KEY (sid) REFERENCES student(sid),
    ->     FOREIGN KEY (bid) REFERENCES book(bid)
    -> );
Query OK, 0 rows affected (0.029 sec)

MariaDB [sulav]> INSERT INTO borrow (sid, bid) VALUES
    ->     (1, 1),
    ->     (2, 2),
    ->     (3, 3),
    ->     (4, 4),
    ->     (5, 5);
Query OK, 5 rows affected (0.002 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

All information from inner join borrow:



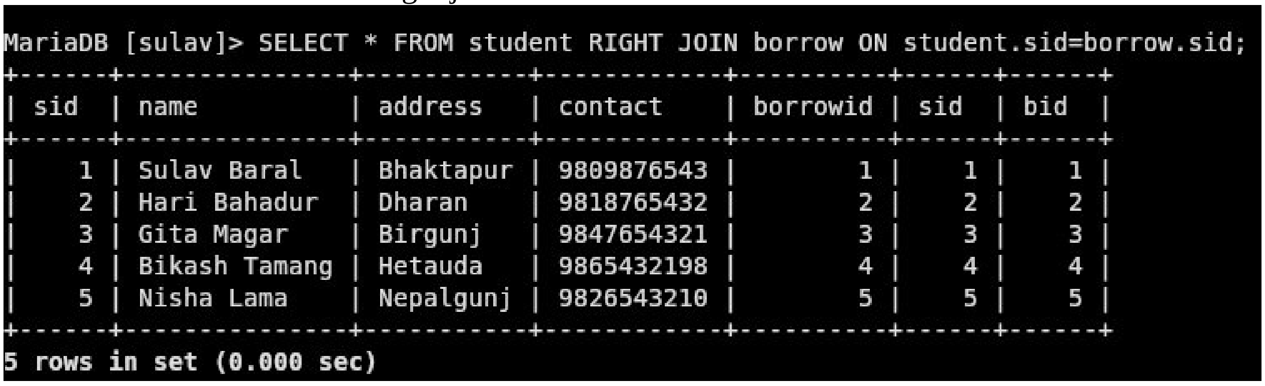
```
MariaDB [sulav]> SELECT * FROM student INNER JOIN borrow ON student.sid=borrow.sid;
+-----+--------------+-----------+------------+----------+------+------+
| sid | name         | address   | contact    | borrowid | sid  | bid  |
+-----+--------------+-----------+------------+----------+------+------+
|   1 | Sulav Baral  | Bhaktapur | 9809876543 |        1 |    1 |    1 |
|   2 | Hari Bahadur | Dharan    | 9818765432 |        2 |    2 |    2 |
|   3 | Gita Magar   | Birgunj   | 9847654321 |        3 |    3 |    3 |
|   4 | Bikash Tamang | Hetauda  | 9865432198 |        4 |    4 |    4 |
|   5 | Nisha Lama   | Nepalgunj | 9826543210 |        5 |    5 |    5 |
+-----+--------------+-----------+------------+----------+------+------+
5 rows in set (0.001 sec)
```

All information from student left join borrow:



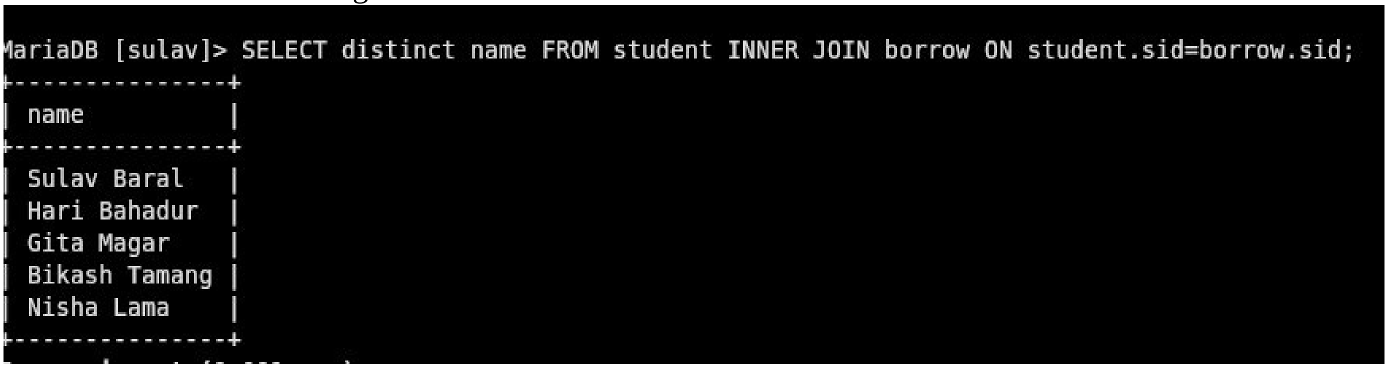
```
MariaDB [sulav]> SELECT * FROM student LEFT JOIN borrow ON student.sid=borrow.sid;
+-----+--------------+-----------+------------+----------+------+------+
| sid | name         | address   | contact    | borrowid | sid  | bid  |
+-----+--------------+-----------+------------+----------+------+------+
|   1 | Sulav Baral  | Bhaktapur | 9809876543 |        1 |    1 |    1 |
|   2 | Hari Bahadur | Dharan    | 9818765432 |        2 |    2 |    2 |
|   3 | Gita Magar   | Birgunj   | 9847654321 |        3 |    3 |    3 |
|   4 | Bikash Tamang | Hetauda  | 9865432198 |        4 |    4 |    4 |
|   5 | Nisha Lama   | Nepalgunj | 9826543210 |        5 |    5 |    5 |
+-----+--------------+-----------+------------+----------+------+------+
5 rows in set (0.000 sec)
```

All information from student right join borrow:



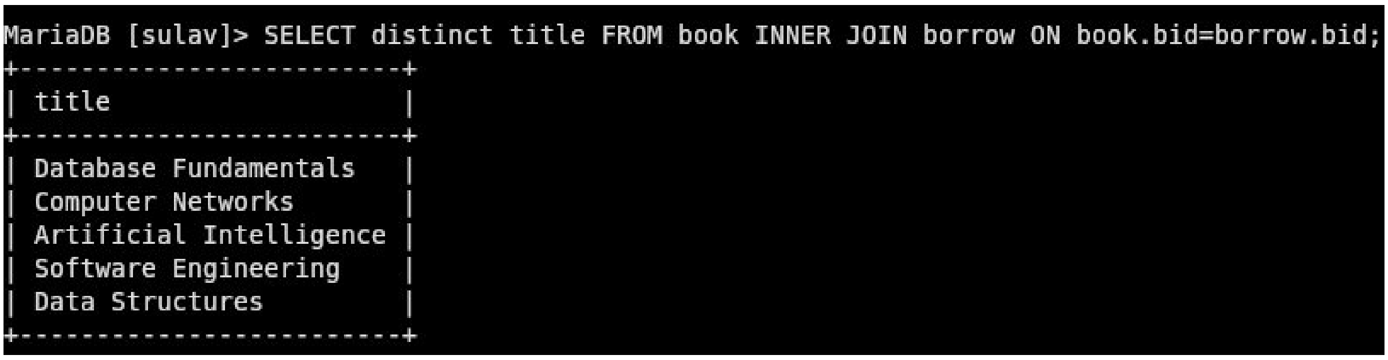
```
MariaDB [sulav]> SELECT * FROM student RIGHT JOIN borrow ON student.sid=borrow.sid;
+------+--------------+-----------+------------+----------+------+------+
| sid  | name         | address   | contact    | borrowid | sid  | bid  |
+------+--------------+-----------+------------+----------+------+------+
|    1 | Sulav Baral  | Bhaktapur | 9809876543 |        1 |    1 |    1 |
|    2 | Hari Bahadur | Dharan    | 9818765432 |        2 |    2 |    2 |
|    3 | Gita Magar   | Birgunj   | 9847654321 |        3 |    3 |    3 |
|    4 | Bikash Tamang | Hetauda  | 9865432198 |        4 |    4 |    4 |
|    5 | Nisha Lama   | Nepalgunj | 9826543210 |        5 |    5 |    5 |
+------+--------------+-----------+------------+----------+------+------+
5 rows in set (0.000 sec)
```

Name of students borrowing at least one book:



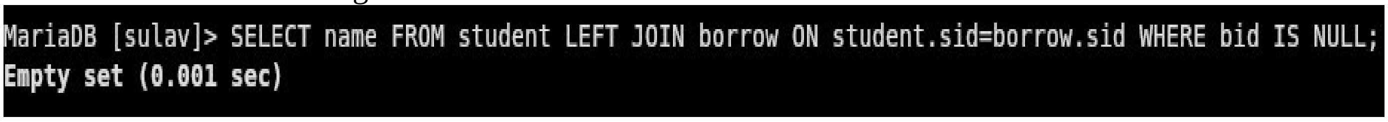
```
lariaDB [sulav]> SELECT distinct name FROM student INNER JOIN borrow ON student.sid=borrow.sid;
+---------------+
| name          |
+---------------+
| Sulav Baral   |
| Hari Bahadur  |
| Gita Magar    |
| Bikash Tamang |
| Nisha Lama    |
+---------------+
```

Title of all books that are borrowed:



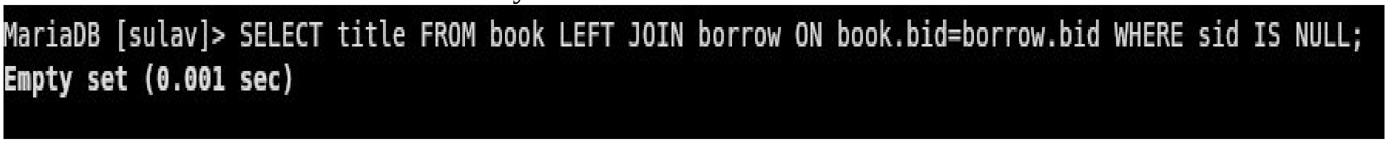
```
MariaDB [sulav]> SELECT distinct title FROM book INNER JOIN borrow ON book.bid=borrow.bid;
+------------------------+
| title                  |
+------------------------+
| Database Fundamentals   |
| Computer Networks       |
| Artificial Intelligence |
| Software Engineering    |
| Data Structures         |
+------------------------+
```

Name of students borrowing no books:



```
MariaDB [sulav]> SELECT name FROM student LEFT JOIN borrow ON student.sid=borrow.sid WHERE bid IS NULL;
Empty set (0.001 sec)
```

Title of all books that are not borrowed yet:



```
MariaDB [sulav]> SELECT title FROM book LEFT JOIN borrow ON book.bid=borrow.bid WHERE sid IS NULL;
Empty set (0.001 sec)
```

CONCLUSION:

Hence, JOIN operations are essential for querying relational databases, allowing data from multiple related tables to be combined and retrieved efficiently. By using INNER, LEFT, RIGHT, FULL OUTER, NATURAL, and THETA joins, meaningful and accurate information can be extracted, supporting effective data analysis, reporting, and decision-making. In our sample data, there were instances where every student borrowed at least one book.Therefore, we got to see an empty set as an output table.

Lab 7

TITLE: Create the following database schema and run SQL queries to retrieve the information as instructed.

Schema: employee(<u>eid</u>, name, post, salary, dob, gender, address, contact)

THEORY: A database is an organized collection of data that allows easy storage, retrieval, and management. Ithelps in maintaining large amounts of information in a structured format, making it easier to update,manipulate, and analyze. A Relational Database Management System (RDBMS) stores data in tables(relations) where relationships among tables are defined through keys. The schema defines the structureof the database, including tables, columns, data types, constraints, and relationships. For instance, theschema for an employee table may be:

employee(eid, name, post, salary, dob, gender, address, contact)

This table stores essential details about employees within an organization Importance of

SQL:

- Widely used in HR systems, banking, schools, inventory management, and business analytics.
- Ensures efficient data management, integrity, and security.
- Essential for roles like database developers, data analysts, and backend developers.

SQL (Structured Query Language) is the standard language used to manage and interact with relational databases. SQL enables the creation, modification, deletion, and querying of data. Its main categories include:

Data Definition Language (DDL): Used to define and alter the database structure.

CREATE – To create new tables or databases.

ALTER – To modify existing tables. DROP – To delete tables or databases. Example:

CREATE TABLE student (.................................);

Data Manipulation Language (DML): Used to manage the data stored in tables.

INSERT – To add new records.

UPDATE – To modify existing records. DELETE – To remove records.

Example: INSERT INTO student (name, class, …...) VALUES (….);

Data Query Language (DQL): Used to retrieve data from tables. SELECT – Retrieves information based on specified conditions. Example: SELECT name, class FROM student;

Use of Aliases:

Column and table aliases allow temporary renaming for better readability. Example:
SELECT s.name AS StudentName, s.post AS grade FROM student s; Here, s is the table alias, and StudentName and grade are column aliases.

Common SQL Functions and Clauses:

DISTINCT – Returns unique values only.

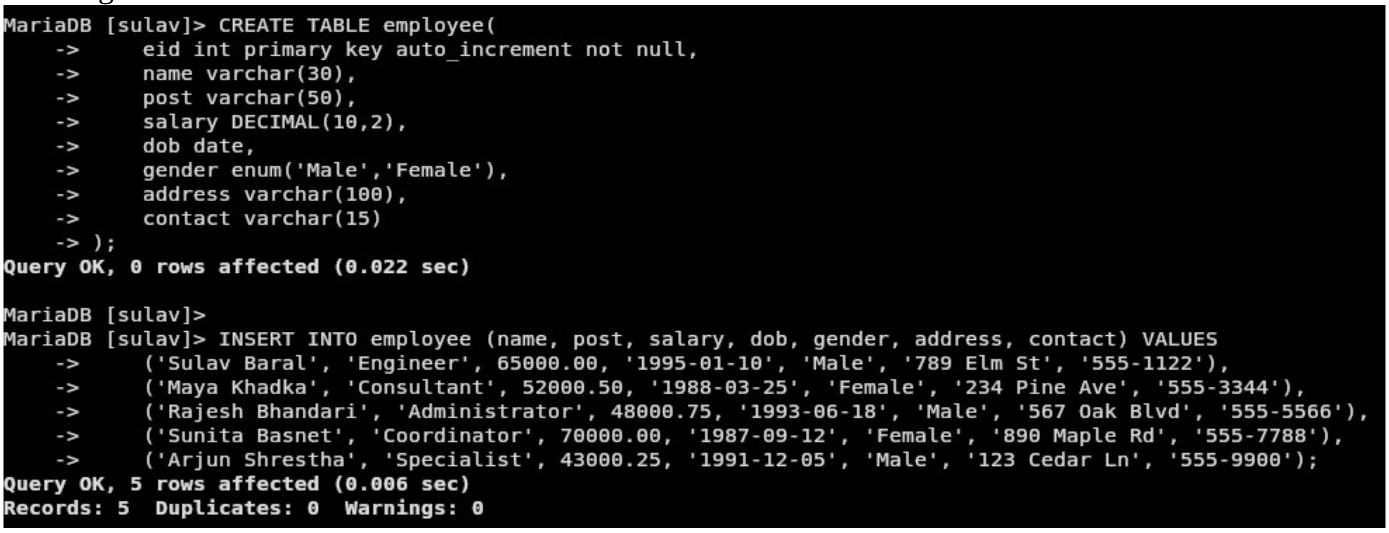
ORDER BY – Sorts results in ascending or descending order. BETWEEN – Filters values within a specified range.

NOT BETWEEN – Filters values outside a specified range.

Arithmetic operations (e.g., salary * 0.15) – Can be used to calculate taxes or bonuses.

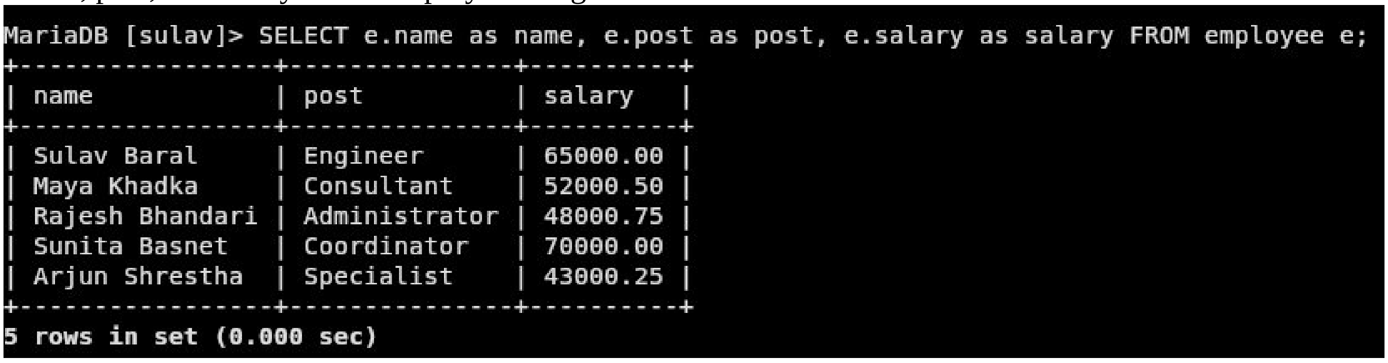
QUERIES:

Creating Database:



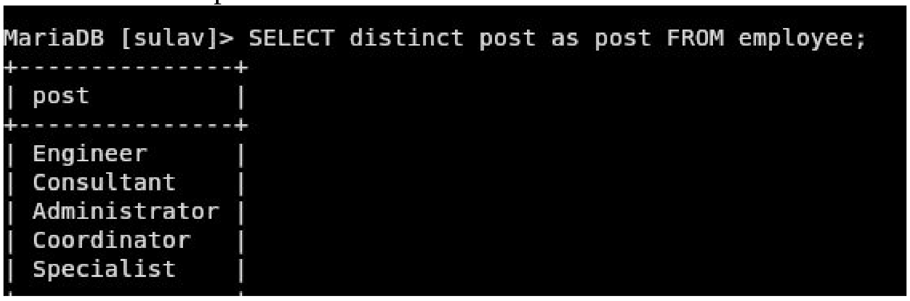
```
MariaDB [sulav]> CREATE TABLE employee(
    ->     eid int primary key auto_increment not null,
    ->     name varchar(30),
    ->     post varchar(50),
    ->     salary DECIMAL(10,2),
    ->     dob date,
    ->     gender enum('Male','Female'),
    ->     address varchar(100),
    ->     contact varchar(15)
    -> );
Query OK, 0 rows affected (0.022 sec)

MariaDB [sulav]>
MariaDB [sulav]> INSERT INTO employee (name, post, salary, dob, gender, address, contact) VALUES
    ->     ('Sulav Baral', 'Engineer', 65000.00, '1995-01-10', 'Male', '789 Elm St', '555-1122'),
    ->     ('Maya Khadka', 'Consultant', 52000.50, '1988-03-25', 'Female', '234 Pine Ave', '555-3344'),
    ->     ('Rajesh Bhandari', 'Administrator', 48000.75, '1993-06-18', 'Male', '567 Oak Blvd', '555-5566'),
    ->     ('Sunita Basnet', 'Coordinator', 70000.00, '1987-09-12', 'Female', '890 Maple Rd', '555-7788'),
    ->     ('Arjun Shrestha', 'Specialist', 43000.25, '1991-12-05', 'Male', '123 Cedar Ln', '555-9900');
Query OK, 5 rows affected (0.006 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Name, post, and salary of the employee using column alias and table alias:



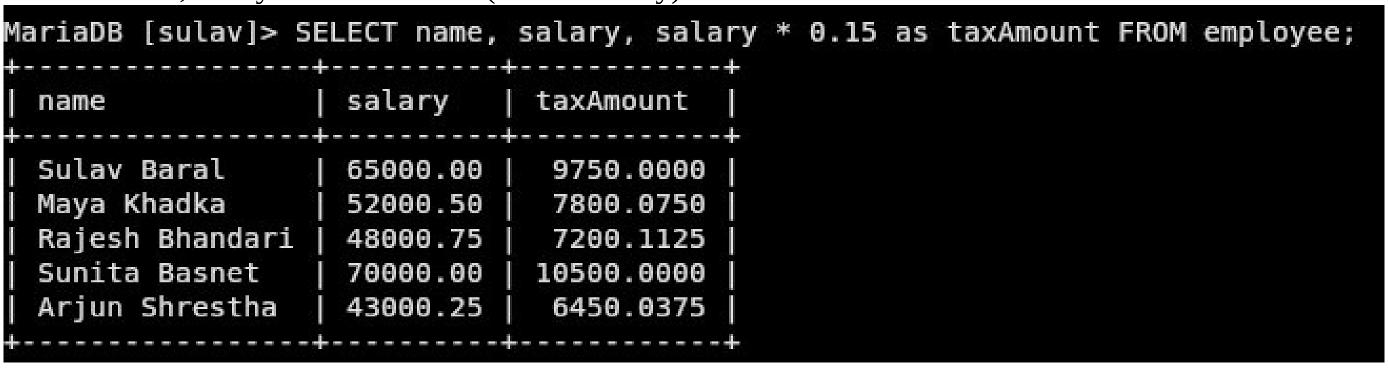
```
MariaDB [sulav]> SELECT e.name as name, e.post as post, e.salary as salary FROM employee e;
+-----------------+---------------+----------+
| name            | post          | salary   |
+-----------------+---------------+----------+
| Sulav Baral     | Engineer      | 65000.00 |
| Maya Khadka     | Consultant    | 52000.50 |
| Rajesh Bhandari | Administrator | 48000.75 |
| Sunita Basnet   | Coordinator   | 70000.00 |
| Arjun Shrestha  | Specialist    | 43000.25 |
+-----------------+---------------+----------+
5 rows in set (0.000 sec)
```

List of different posts:



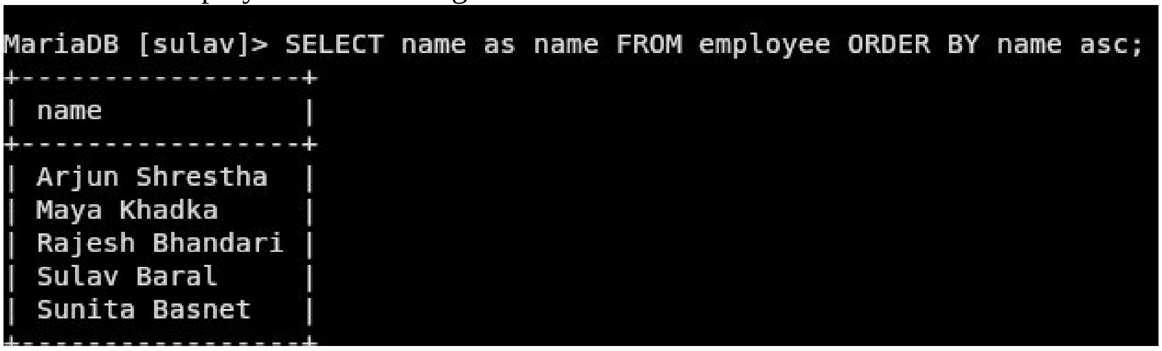
```
MariaDB [sulav]> SELECT distinct post as post FROM employee;
+---------------+
| post          |
+---------------+
| Engineer      |
| Consultant    |
| Administrator |
| Coordinator   |
| Specialist    |
```

List out name, salary and tax amount (15% of salary).



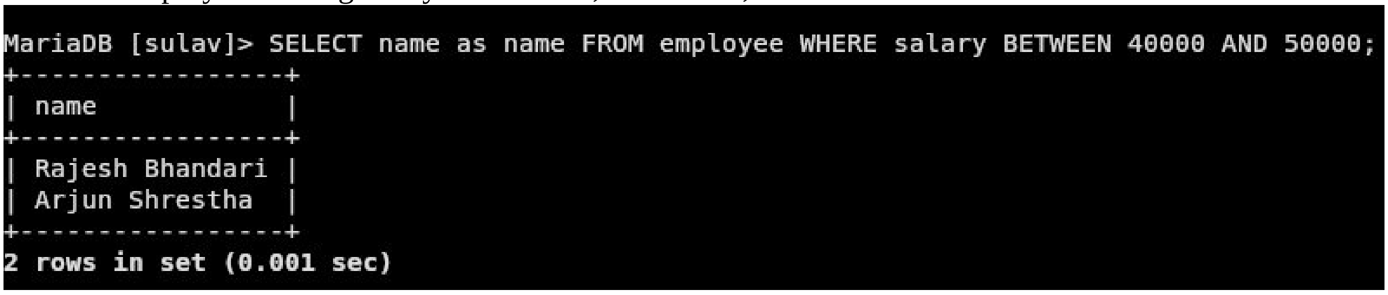
```
MariaDB [sulav]> SELECT name, salary, salary * 0.15 as taxAmount FROM employee;
+-----------------+----------+------------+
| name            | salary   | taxAmount  |
+-----------------+----------+------------+
| Sulav Baral     | 65000.00 |  9750.0000 |
| Maya Khadka     | 52000.50 |  7800.0750 |
| Rajesh Bhandari | 48000.75 |  7200.1125 |
| Sunita Basnet   | 70000.00 | 10500.0000 |
| Arjun Shrestha  | 43000.25 |  6450.0375 |
+-----------------+----------+------------+
```

Name of all employees in ascending order:



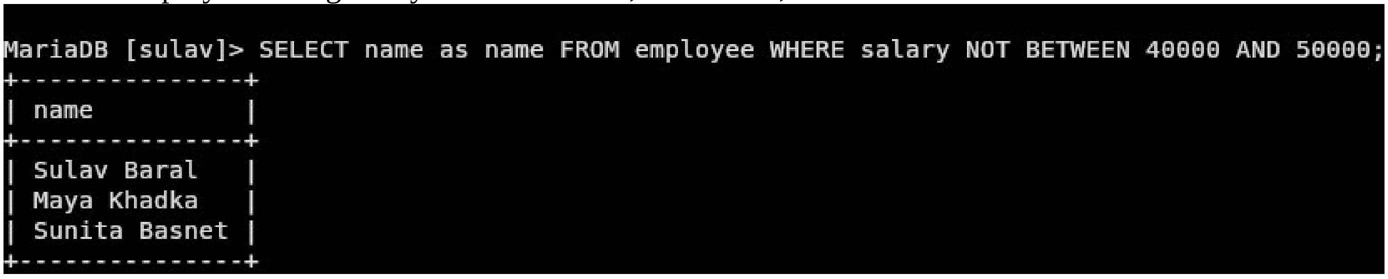
```
MariaDB [sulav]> SELECT name as name FROM employee ORDER BY name asc;
+-----------------+
| name            |
+-----------------+
| Arjun Shrestha  |
| Maya Khadka     |
| Rajesh Bhandari |
| Sulav Baral     |
| Sunita Basnet   |
+-----------------+
```

Name of employees having salary between 40,000 and 50,000:



```
MariaDB [sulav]> SELECT name as name FROM employee WHERE salary BETWEEN 40000 AND 50000;
+-----------------+
| name            |
+-----------------+
| Rajesh Bhandari |
| Arjun Shrestha  |
+-----------------+
2 rows in set (0.001 sec)
```

Name of employee having salary not between 40,000 and 50,000:



```
MariaDB [sulav]> SELECT name as name FROM employee WHERE salary NOT BETWEEN 40000 AND 50000;
+---------------+
| name          |
+---------------+
| Sulav Baral   |
| Maya Khadka   |
| Sunita Basnet |
+---------------+
```

CONCLUSION: This lab report effectively demonstrated the fundamental concepts of relational databases, schema definition, and the use of SQL for data manipulation and retrieval. By creating an employee table and running various queries, we were able to perform essential database operations such as inserting data, filtering records, sorting results, and calculating derived values. The exercises reinforced the importance of SQL as a powerful tool for managing and interacting with structured data, highlighting its role in ensuring data integrity, security, and efficient analysis.

Lab 8

TITLE: Create the following database schema and run the SQL queries to retrieve the required information.

Schema: employee(<u>eid</u>, name, post, salary, gender, address, contact)

THEORY: A database schema defines the structure of a database, including tables, columns, data types, and constraints. Creating a schema involves designing tables to store related data efficiently and accurately. For example, an employee table may include columns like eid, name, post, salary, gender, address, and contact. This table stores employee details such as ID, name, post, salary, gender, address, and contact.

SQL (Structured Query Language) is used to manage and query data in relational databases. Key operations include:

- Creating tables: Using the CREATE TABLE statement to define columns and constraints (e.g., primary key, not null).
- Inserting data: Using INSERT INTO to add records.
- Retrieving data: Using SELECT statements to query information, apply calculations like COUNT, AVG, and find specific records (e.g., maximum salary using subqueries).
- Aggregating data: Using functions like COUNT(), SUM(), and AVG() to summarize information for analysis.
- Filtering and grouping: Using WHERE and GROUP BY clauses to analyze data by specific conditions, such as gender.

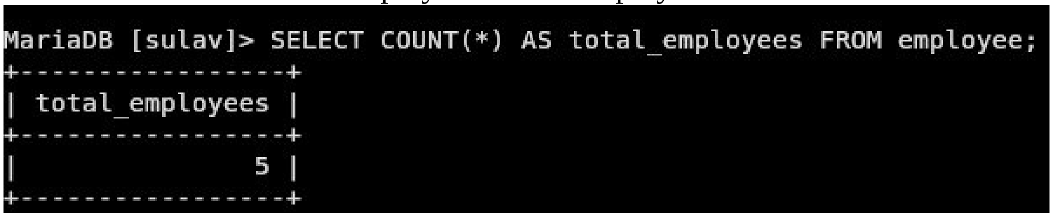
This exercise focuses on aggregate functions and subqueries to provide essential business insights:

- COUNT(): Counts the number of rows (e.g., total employees).
- AVG(): Calculates the average value (e.g., average salary).
- SUM(): Adds values (e.g., total monthly salary).
- GROUP BY: Groups rows based on a column (e.g., gender) to perform aggregate calculations per group.
- Subqueries: Retrieve data based on results of another query

These queries help analyze workforce size, salary distribution by gender, identify top earners, and calculate total salary expenses, supporting informed decision-making and payroll management.

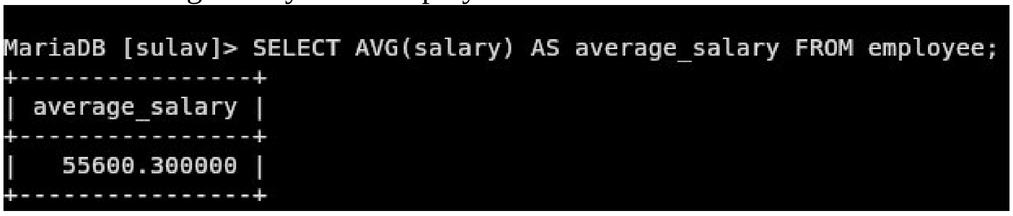
QUERIES:

Count the total number of employees in the ‘employee’ table.



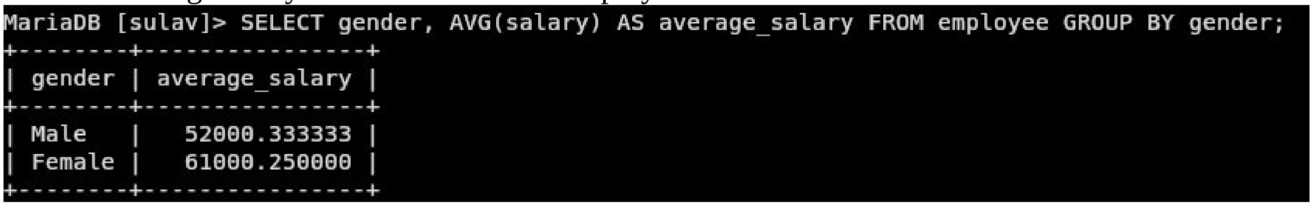
```
MariaDB [sulav]> SELECT COUNT(*) AS total_employees FROM employee;
+-----------------+
| total_employees |
+-----------------+
|               5 |
+-----------------+
```

Find the average salary of all employees.



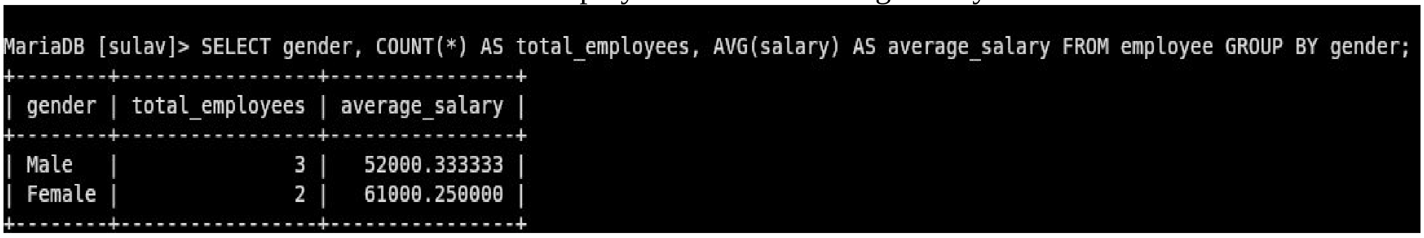
```
MariaDB [sulav]> SELECT AVG(salary) AS average_salary FROM employee;
+----------------+
| average_salary |
+----------------+
|   55600.300000 |
+----------------+
```

Find the average salary of male and female employees.



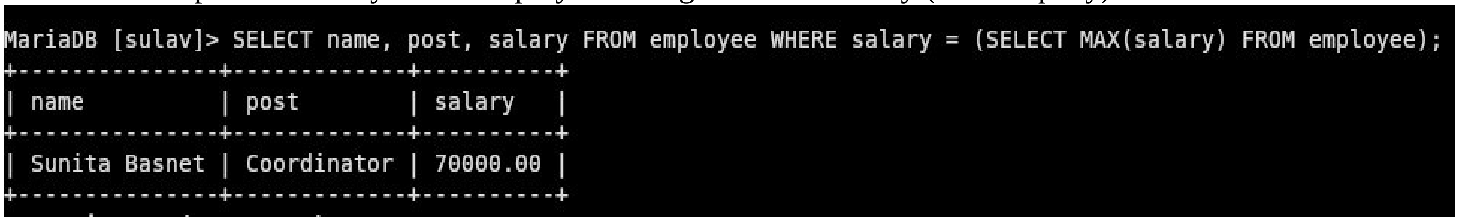
```
MariaDB [sulav]> SELECT gender, AVG(salary) AS average_salary FROM employee GROUP BY gender;
+--------+----------------+
| gender | average_salary |
+--------+----------------+
| Male   |   52000.333333 |
| Female |   61000.250000 |
+--------+----------------+
```

Find the total number of male and female employees and their average salary.



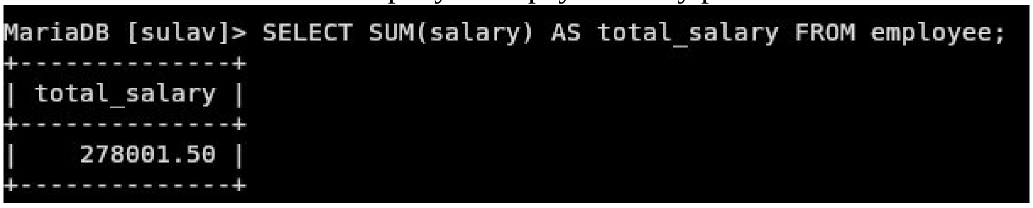
```
MariaDB [sulav]> SELECT gender, COUNT(*) AS total_employees, AVG(salary) AS average_salary FROM employee GROUP BY gender;
+--------+-----------------+----------------+
| gender | total_employees | average_salary |
+--------+-----------------+----------------+
| Male   |               3 |   52000.333333 |
| Female |               2 |   61000.250000 |
+--------+-----------------+----------------+
```

Find the name post and salary of the employee having maximum salary (use subquery)



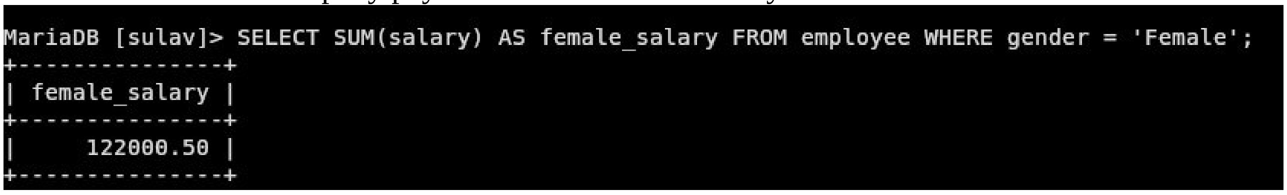
```
MariaDB [sulav]> SELECT name, post, salary FROM employee WHERE salary = (SELECT MAX(salary) FROM employee);
+---------------+-------------+----------+
| name          | post        | salary   |
+---------------+-------------+----------+
| Sunita Basnet | Coordinator | 70000.00 |
+---------------+-------------+----------+
```

Find the total amount the company must pay as salary per month.



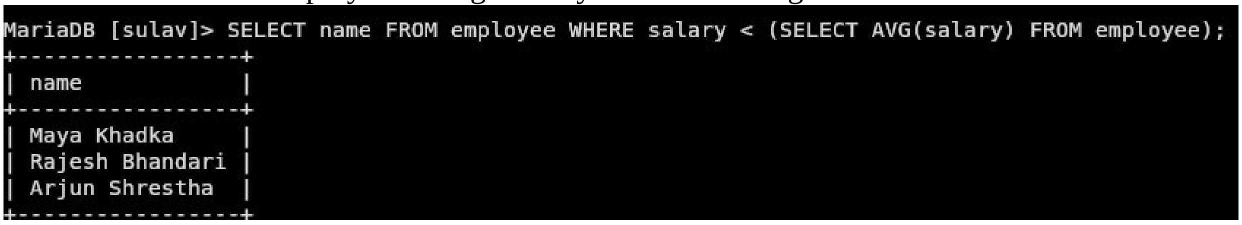
```
MariaDB [sulav]> SELECT SUM(salary) AS total_salary FROM employee;
+--------------+
| total_salary |
+--------------+
|    278001.50 |
+--------------+
```

How much does the company pay to all the females as salary?



```
MariaDB [sulav]> SELECT SUM(salary) AS female_salary FROM employee WHERE gender = 'Female';
+---------------+
| female_salary |
+---------------+
|     122000.50 |
+---------------+
```

Find the name of an employee having a salary less than average.



```
MariaDB [sulav]> SELECT name FROM employee WHERE salary < (SELECT AVG(salary) FROM employee);
+-----------------+
| name            |
+-----------------+
| Maya Khadka     |
| Rajesh Bhandari |
| Arjun Shrestha  |
+-----------------+
```

CONCLUSION:

In this lab, we created and manipulated a database schema using MySQL. We practiced essential operations such as ALTER (to modify the structure of a table), UPDATE (to modify existing records), and DELETE (to remove records). We also learned to: Add new columns to existing tables, Update records based on conditions and subqueries, use aggregate functions (AVG, SUM, COUNT) for data analysis, apply string manipulation (CONCAT) in updates, and Change table and column names for better schema management. This lab reinforced the practical use of SQL in managing databases effectively, ensuring flexibility in modifying structures, maintaining data accuracy, and supporting organizational decision-making.

Lab 9

TITLE: Create the following database schema and run the different UPDATE, DELETE, and ALTER queries using MySQL.

THEORY: In relational database management systems (RDBMS) like MySQL, it is often necessary not only to create and retrieve data but also to modify, update, or restructure existing tables. SQL provides powerful commands such as UPDATE, DELETE, and ALTER to perform these tasks efficiently. These operations ensure that the database remains accurate, flexible, and adaptable to new requirements.

UPDATE Command

The UPDATE command is used to modify existing records in a table. It allows users to change one or more column values based on specific conditions.

General Syntax:

- UPDATE table_name
- SET column1 = value1, column2 = value2
- WHERE condition;
- **Example:** Increasing all employees' salaries by 10%:
- UPDATE employee
- SET salary = salary * 1.10;

The WHERE clause is optional, but without it, the update applies to **all rows** in the table.

DELETE Command

The DELETE command removes rows from a table based on a specified condition. It helps maintain data accuracy by eliminating unnecessary or invalid records.

General Syntax:

- DELETE FROM table_name
- WHERE condition;
- **Example:** Deleting all male employees from Pokhara:
- DELETE FROM employee
- WHERE city = 'Pokhara' AND gender = 'Male';

If the WHERE clause is omitted, all rows will be deleted, leaving the table structure intact.

ALTER Command

The ALTER command modifies the structure of an existing table without deleting its data. It can be used to add, delete, or modify columns, as well as rename columns or tables.

- **Adding a column:**
- ALTER TABLE employee
- ADD COLUMN postal_code VARCHAR(10) AFTER city;
- **Renaming a column:**
- ALTER TABLE employee
- CHANGE COLUMN name full_name VARCHAR(30);
- **Renaming a table:**
- RENAME TABLE employee TO emp;

Thus, ALTER provides schema flexibility as organizational needs evolve.

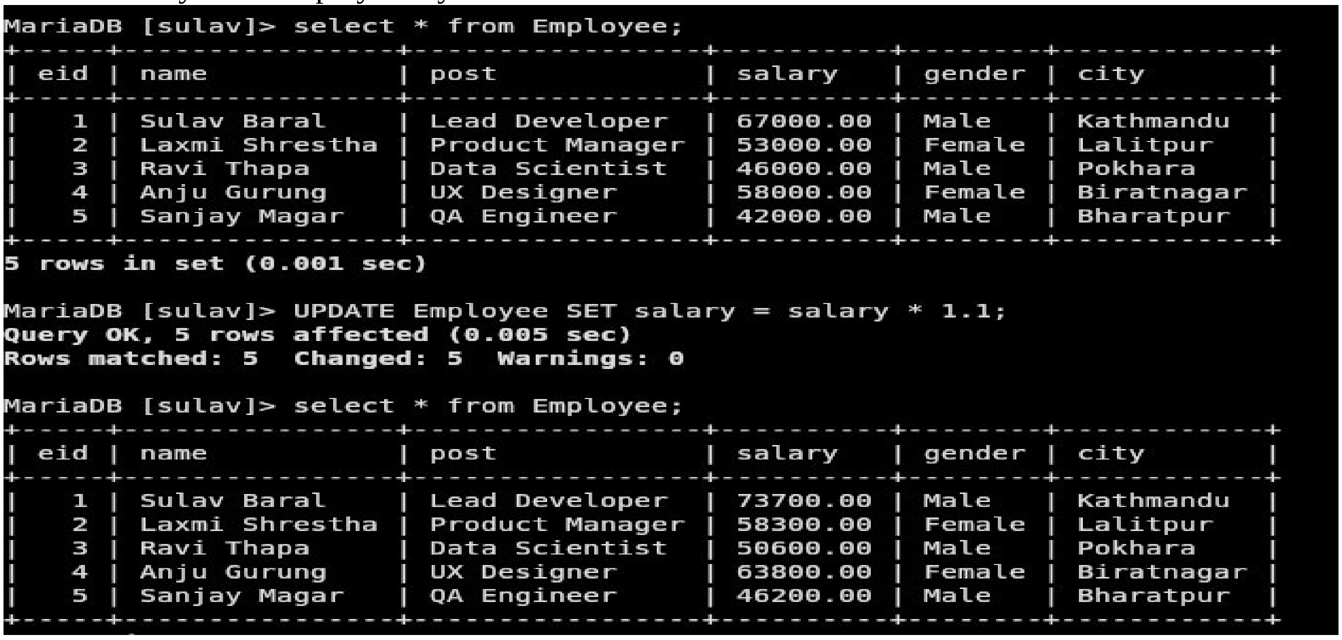
Practical Applications of These Queries

- **Updating Salaries:** Companies frequently need to revise salaries, bonuses, or designations. Using UPDATE, we can apply global increments (e.g., all employees get a 10% raise) or targeted changes (e.g., only managers get a raise).
- **Maintaining Accuracy:** With DELETE, invalid or outdated records can be removed. For example, removing all male employees from Pokhara ensures the database reflects current staff data.

- **Adapting Schema:** Businesses often need to modify their data model. Adding a postal_code column allows more precise employee location tracking, while renaming name to full_name improves clarity. Similarly, renaming tables helps align with new naming standards.

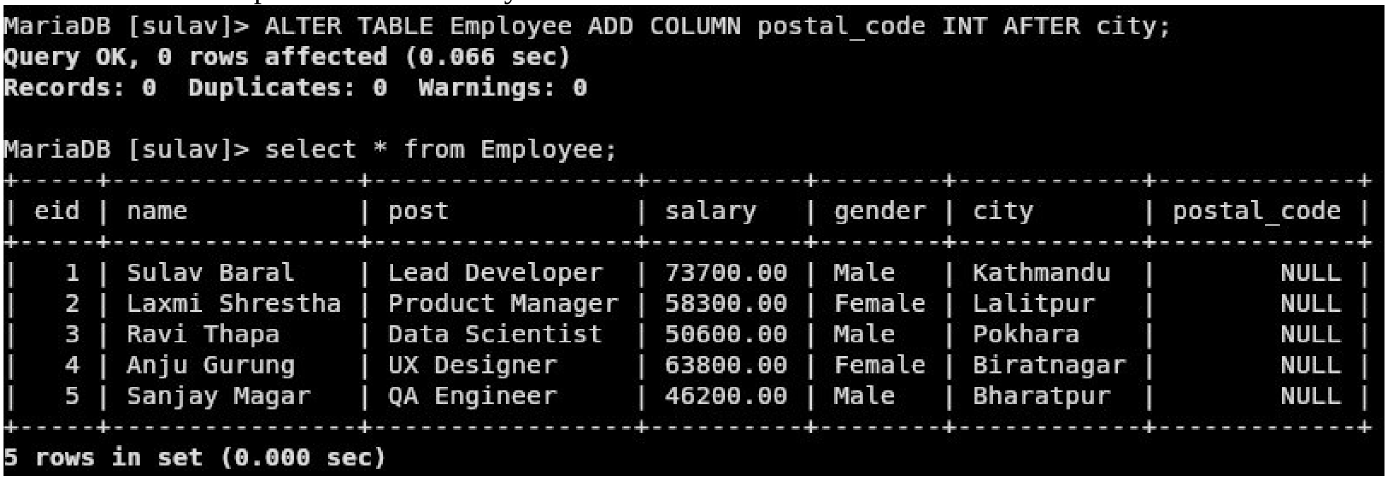
QUERIES:

Increase salary of all employees by 10%:



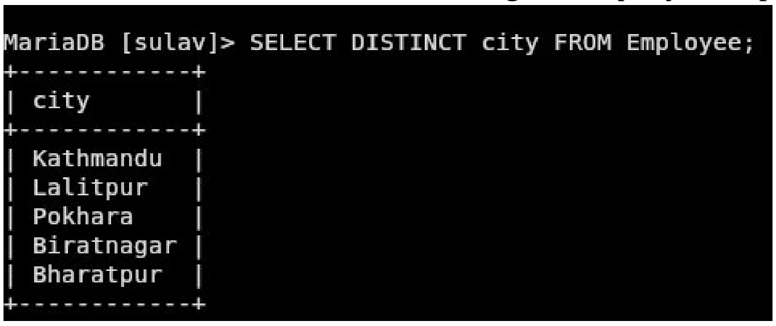
```
MariaDB [sulav]> select * from Employee;
+-----+-----------------+-----------------+----------+--------+------------+
| eid | name            | post            | salary   | gender | city       |
+-----+-----------------+-----------------+----------+--------+------------+
|   1 | Sulav Baral     | Lead Developer  | 67000.00 | Male   | Kathmandu  |
|   2 | Laxmi Shrestha  | Product Manager | 53000.00 | Female | Lalitpur   |
|   3 | Ravi Thapa      | Data Scientist  | 46000.00 | Male   | Pokhara    |
|   4 | Anju Gurung     | UX Designer     | 58000.00 | Female | Biratnagar |
|   5 | Sanjay Magar    | QA Engineer     | 42000.00 | Male   | Bharatpur  |
+-----+-----------------+-----------------+----------+--------+------------+
5 rows in set (0.001 sec)

MariaDB [sulav]> UPDATE Employee SET salary = salary * 1.1;
Query OK, 5 rows affected (0.005 sec)
Rows matched: 5  Changed: 5  Warnings: 0

MariaDB [sulav]> select * from Employee;
+-----+-----------------+-----------------+----------+--------+------------+
| eid | name            | post            | salary   | gender | city       |
+-----+-----------------+-----------------+----------+--------+------------+
|   1 | Sulav Baral     | Lead Developer  | 73700.00 | Male   | Kathmandu  |
|   2 | Laxmi Shrestha  | Product Manager | 58300.00 | Female | Lalitpur   |
|   3 | Ravi Thapa      | Data Scientist  | 50600.00 | Male   | Pokhara    |
|   4 | Anju Gurung     | UX Designer     | 63800.00 | Female | Biratnagar |
|   5 | Sanjay Magar    | QA Engineer     | 46200.00 | Male   | Bharatpur  |
+-----+-----------------+-----------------+----------+--------+------------+
```

Add a new column postal code after city:

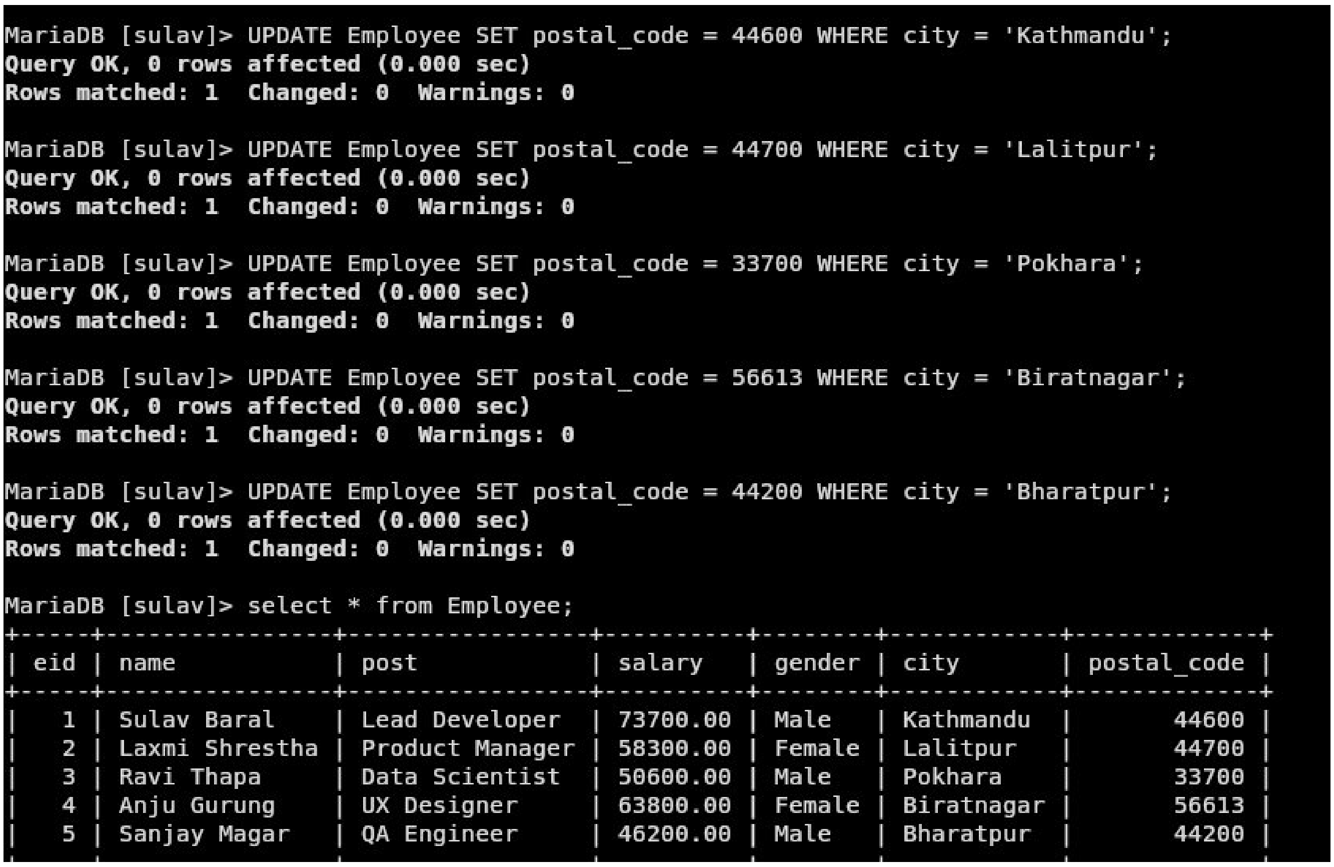


```
MariaDB [sulav]> ALTER TABLE Employee ADD COLUMN postal_code INT AFTER city;
Query OK, 0 rows affected (0.066 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> select * from Employee;
+-----+----------------+-----------------+----------+--------+------------+-------------+
| eid | name           | post            | salary   | gender | city       | postal_code |
+-----+----------------+-----------------+----------+--------+------------+-------------+
|   1 | Sulav Baral    | Lead Developer  | 73700.00 | Male   | Kathmandu  |        NULL |
|   2 | Laxmi Shrestha | Product Manager | 58300.00 | Female | Lalitpur   |        NULL |
|   3 | Ravi Thapa     | Data Scientist  | 50600.00 | Male   | Pokhara    |        NULL |
|   4 | Anju Gurung    | UX Designer     | 63800.00 | Female | Biratnagar |        NULL |
|   5 | Sanjay Magar   | QA Engineer     | 46200.00 | Male   | Bharatpur  |        NULL |
+-----+----------------+-----------------+----------+--------+------------+-------------+
5 rows in set (0.000 sec)
```

List out different cities in the table using select query and update the postal code for each city.

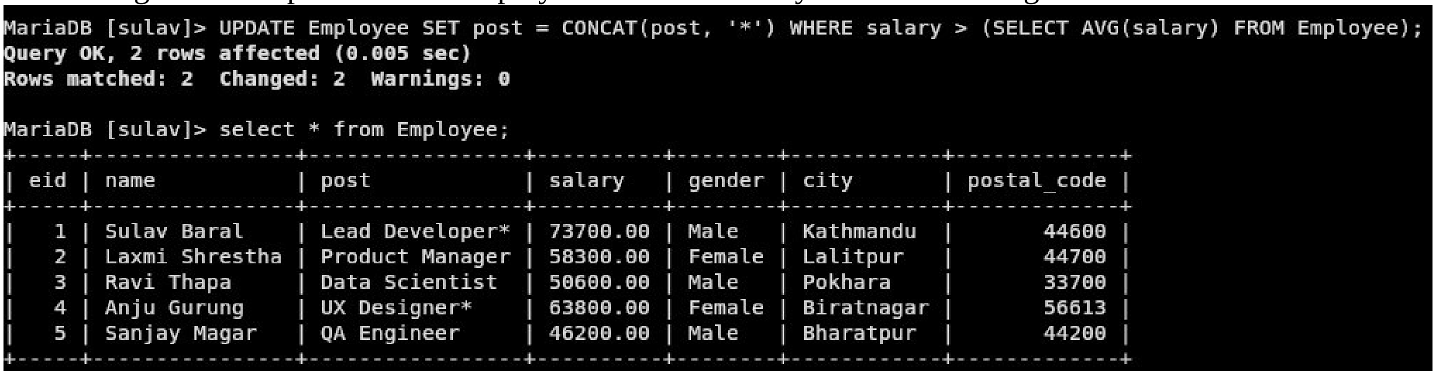


```
MariaDB [sulav]> SELECT DISTINCT city FROM Employee;
+------------+
| city       |
+------------+
| Kathmandu  |
| Lalitpur   |
| Pokhara    |
| Biratnagar |
| Bharatpur  |
+------------+
```



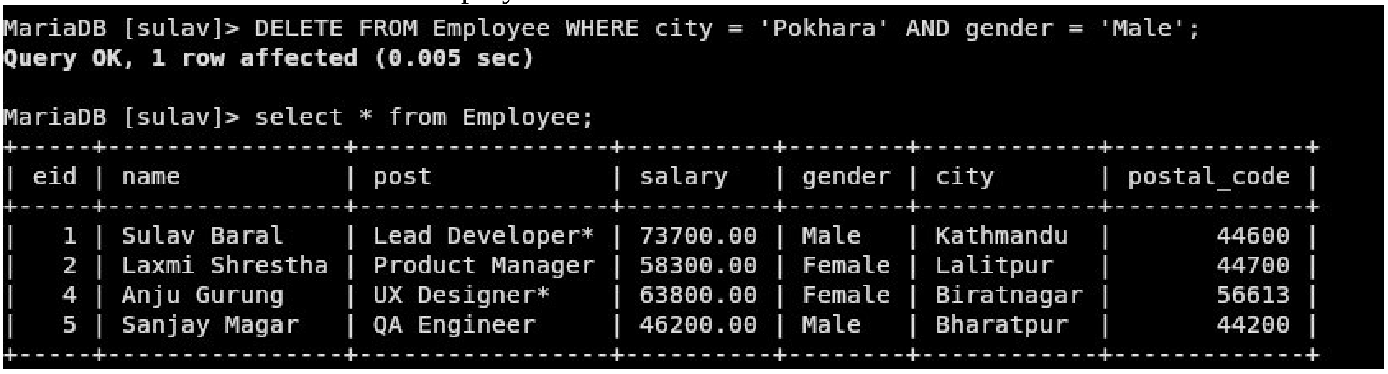
```
MariaDB [sulav]> UPDATE Employee SET postal_code = 44600 WHERE city = 'Kathmandu';
Query OK, 0 rows affected (0.000 sec)
Rows matched: 1  Changed: 0  Warnings: 0

MariaDB [sulav]> UPDATE Employee SET postal_code = 44700 WHERE city = 'Lalitpur';
Query OK, 0 rows affected (0.000 sec)
Rows matched: 1  Changed: 0  Warnings: 0

MariaDB [sulav]> UPDATE Employee SET postal_code = 33700 WHERE city = 'Pokhara';
Query OK, 0 rows affected (0.000 sec)
Rows matched: 1  Changed: 0  Warnings: 0

MariaDB [sulav]> UPDATE Employee SET postal_code = 56613 WHERE city = 'Biratnagar';
Query OK, 0 rows affected (0.000 sec)
Rows matched: 1  Changed: 0  Warnings: 0

MariaDB [sulav]> UPDATE Employee SET postal_code = 44200 WHERE city = 'Bharatpur';
Query OK, 0 rows affected (0.000 sec)
Rows matched: 1  Changed: 0  Warnings: 0

MariaDB [sulav]> select * from Employee;
+-----+----------------+-----------------+----------+--------+------------+-------------+
| eid | name           | post            | salary   | gender | city       | postal_code |
+-----+----------------+-----------------+----------+--------+------------+-------------+
|   1 | Sulav Baral    | Lead Developer  | 73700.00 | Male   | Kathmandu  |       44600 |
|   2 | Laxmi Shrestha | Product Manager | 58300.00 | Female | Lalitpur   |       44700 |
|   3 | Ravi Thapa     | Data Scientist  | 50600.00 | Male   | Pokhara    |       33700 |
|   4 | Anju Gurung    | UX Designer     | 63800.00 | Female | Biratnagar |       56613 |
|   5 | Sanjay Magar   | QA Engineer     | 46200.00 | Male   | Bharatpur  |       44200 |
```

Add a * sign after the post of those employees who have salary more than average.



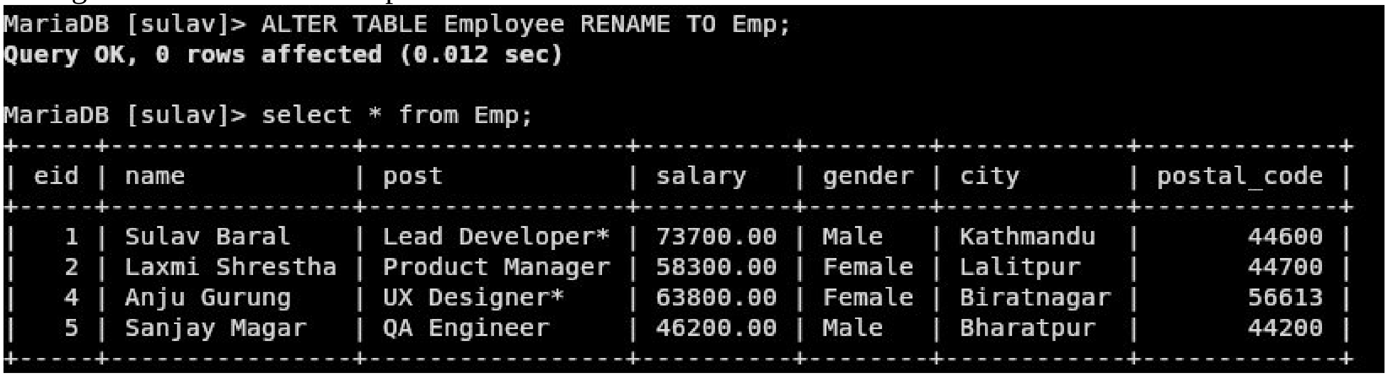
```
MariaDB [sulav]> UPDATE Employee SET post = CONCAT(post, '*') WHERE salary > (SELECT AVG(salary) FROM Employee);
Query OK, 2 rows affected (0.005 sec)
Rows matched: 2  Changed: 2  Warnings: 0

MariaDB [sulav]> select * from Employee;
+-----+----------------+-----------------+----------+--------+------------+-------------+
| eid | name           | post            | salary   | gender | city       | postal_code |
+-----+----------------+-----------------+----------+--------+------------+-------------+
|   1 | Sulav Baral    | Lead Developer* | 73700.00 | Male   | Kathmandu  |       44600 |
|   2 | Laxmi Shrestha | Product Manager | 58300.00 | Female | Lalitpur   |       44700 |
|   3 | Ravi Thapa     | Data Scientist  | 50600.00 | Male   | Pokhara    |       33700 |
|   4 | Anju Gurung    | UX Designer*    | 63800.00 | Female | Biratnagar |       56613 |
|   5 | Sanjay Magar   | QA Engineer     | 46200.00 | Male   | Bharatpur  |       44200 |
+-----+----------------+-----------------+----------+--------+------------+-------------+
```

Delete all those records of those employees who are from Pokhara and are male.



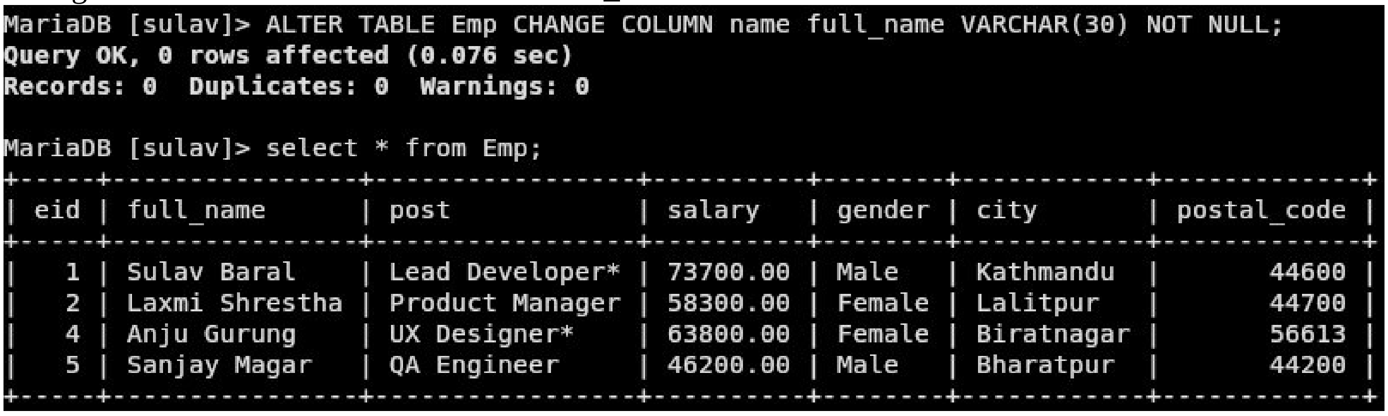
```
MariaDB [sulav]> DELETE FROM Employee WHERE city = 'Pokhara' AND gender = 'Male';
Query OK, 1 row affected (0.005 sec)

MariaDB [sulav]> select * from Employee;
+-----+----------------+-----------------+----------+--------+------------+-------------+
| eid | name           | post            | salary   | gender | city       | postal_code |
+-----+----------------+-----------------+----------+--------+------------+-------------+
|   1 | Sulav Baral    | Lead Developer* | 73700.00 | Male   | Kathmandu  |       44600 |
|   2 | Laxmi Shrestha | Product Manager | 58300.00 | Female | Lalitpur   |       44700 |
|   4 | Anju Gurung    | UX Designer*    | 63800.00 | Female | Biratnagar |       56613 |
|   5 | Sanjay Magar   | QA Engineer     | 46200.00 | Male   | Bharatpur  |       44200 |
+-----+----------------+-----------------+----------+--------+------------+-------------+
```

Change the table name to ‘emp’



```
MariaDB [sulav]> ALTER TABLE Employee RENAME TO Emp;
Query OK, 0 rows affected (0.012 sec)

MariaDB [sulav]> select * from Emp;
+-----+----------------+-----------------+----------+--------+------------+-------------+
| eid | name           | post            | salary   | gender | city       | postal_code |
+-----+----------------+-----------------+----------+--------+------------+-------------+
|   1 | Sulav Baral    | Lead Developer* | 73700.00 | Male   | Kathmandu  |       44600 |
|   2 | Laxmi Shrestha | Product Manager | 58300.00 | Female | Lalitpur   |       44700 |
|   4 | Anju Gurung    | UX Designer*    | 63800.00 | Female | Biratnagar |       56613 |
|   5 | Sanjay Magar   | QA Engineer     | 46200.00 | Male   | Bharatpur  |       44200 |
+-----+----------------+-----------------+----------+--------+------------+-------------+
```

Change the name of the column ‘name’ to ‘full_name’



```
MariaDB [sulav]> ALTER TABLE Emp CHANGE COLUMN name full_name VARCHAR(30) NOT NULL;
Query OK, 0 rows affected (0.076 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [sulav]> select * from Emp;
+-----+----------------+-----------------+----------+--------+------------+-------------+
| eid | full_name      | post            | salary   | gender | city       | postal_code |
+-----+----------------+-----------------+----------+--------+------------+-------------+
|   1 | Sulav Baral    | Lead Developer* | 73700.00 | Male   | Kathmandu  |       44600 |
|   2 | Laxmi Shrestha | Product Manager | 58300.00 | Female | Lalitpur   |       44700 |
|   4 | Anju Gurung    | UX Designer*    | 63800.00 | Female | Biratnagar |       56613 |
|   5 | Sanjay Magar   | QA Engineer     | 46200.00 | Male   | Bharatpur  |       44200 |
+-----+----------------+-----------------+----------+--------+------------+-------------+
```

CONCLUSION:

In real-world database management, simply creating tables is not enough — ongoing updates, deletions, and alterations are essential for accuracy, relevance, and adaptability.UPDATE ensures that existing records remain current. DELETE keeps the database clean and consistent. ALTER allows structural changes without losing data. Together, these commands empower administrators to maintain both the data and the structure of the database dynamically, ensuring it remains aligned with evolving organizational requirements.

Lab 10

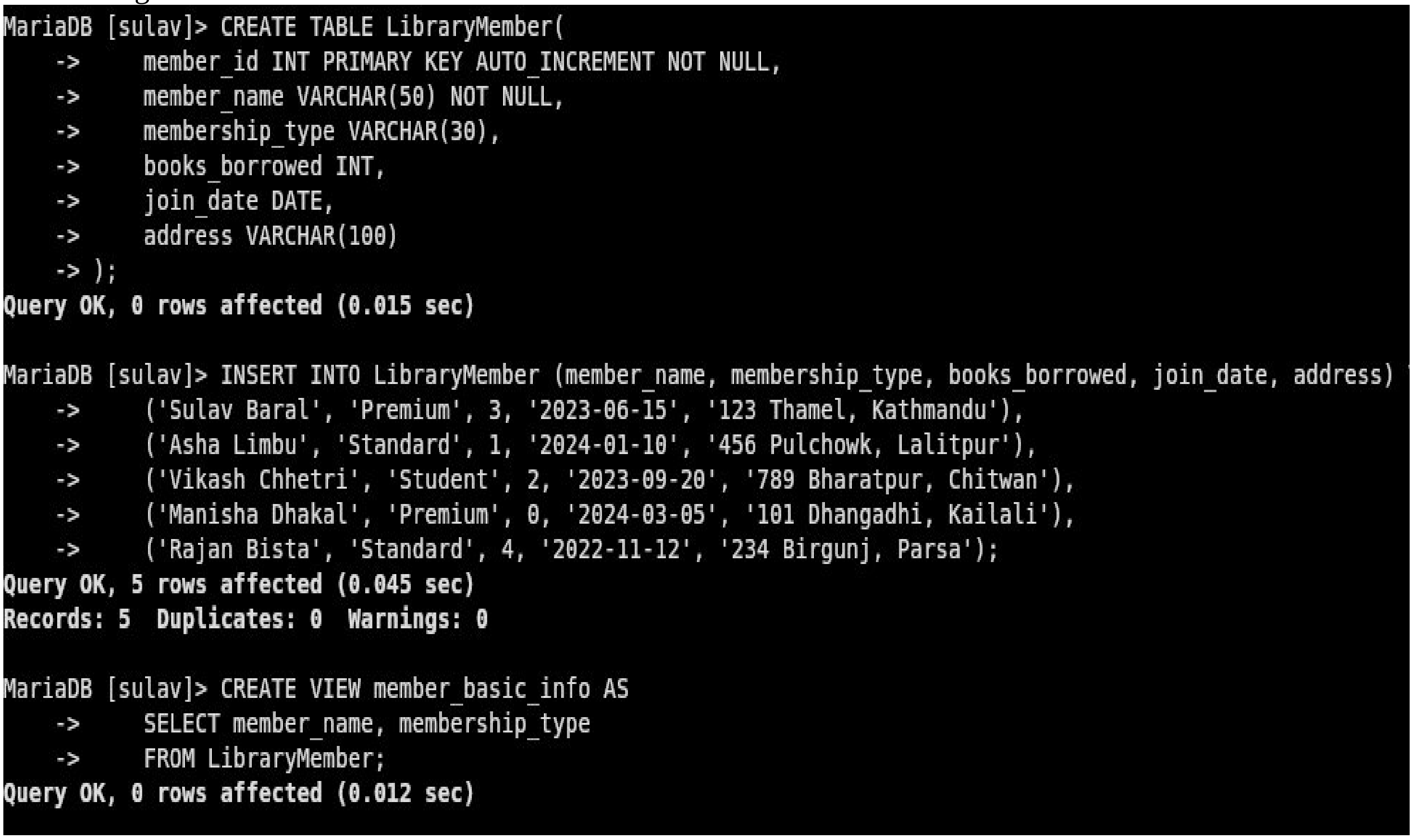
TITLE: To define views in MYSQL

THEORY: In MySQL, a view is a virtual table that is based on the result of a SQL query. Unlike a physical table, a view does not store data itself but provides a way to represent data from one or more tables. Views are useful for simplifying complex queries, providing data security, and customizing the way users interact with data.

- Definition:
- CREATE VIEW view_name AS
- SELECT columns
- FROM table_name
- WHERE condition;
- Key Features of Views:
  - Simplification: Complex joins or queries can be stored as views and accessed like a table.
  - Security: Views can restrict access to certain columns or rows, showing only relevant data.
  - Data Independence: Users can work with views without needing to know the underlying schema design.
  - Reusability: Views can be reused multiple times in queries without rewriting SQL code.

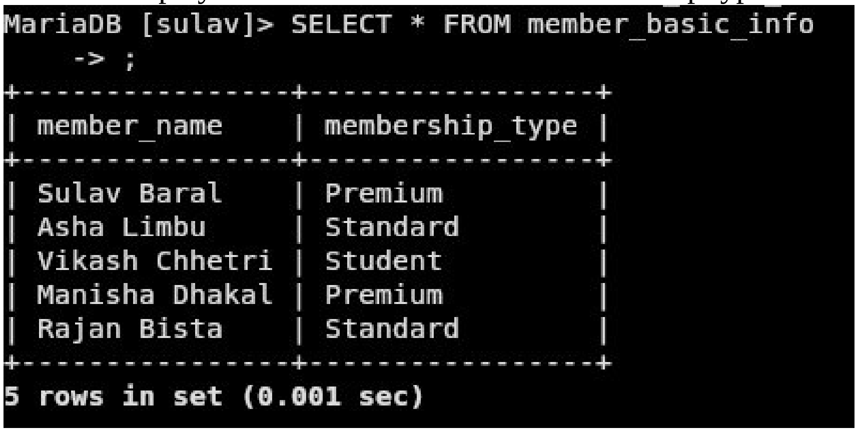
QUERIES:

creating database:



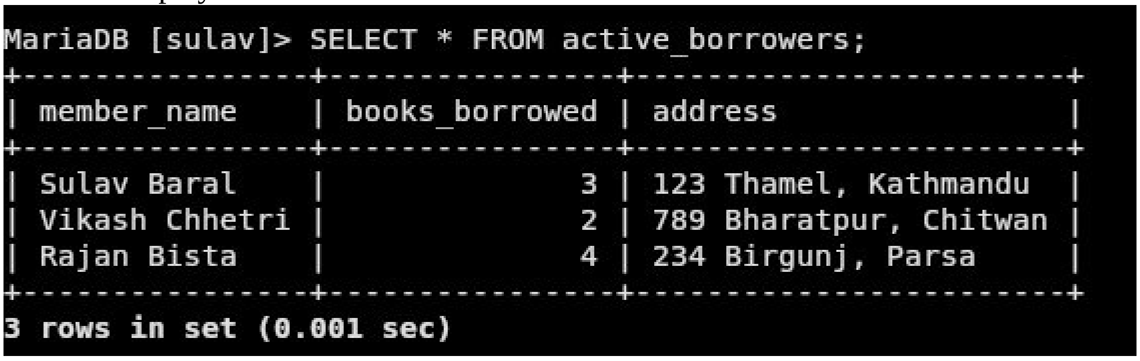
```
MariaDB [sulav]> CREATE TABLE LibraryMember(
    ->     member_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    ->     member_name VARCHAR(50) NOT NULL,
    ->     membership_type VARCHAR(30),
    ->     books_borrowed INT,
    ->     join_date DATE,
    ->     address VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.015 sec)

MariaDB [sulav]> INSERT INTO LibraryMember (member_name, membership_type, books_borrowed, join_date, address)
    ->     ('Sulav Baral', 'Premium', 3, '2023-06-15', '123 Thamel, Kathmandu'),
    ->     ('Asha Limbu', 'Standard', 1, '2024-01-10', '456 Pulchowk, Lalitpur'),
    ->     ('Vikash Chhetri', 'Student', 2, '2023-09-20', '789 Bharatpur, Chitwan'),
    ->     ('Manisha Dhakal', 'Premium', 0, '2024-03-05', '101 Dhangadhi, Kailali'),
    ->     ('Rajan Bista', 'Standard', 4, '2022-11-12', '234 Birgunj, Parsa');
Query OK, 5 rows affected (0.045 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [sulav]> CREATE VIEW member_basic_info AS
    ->     SELECT member_name, membership_type
    ->     FROM LibraryMember;
Query OK, 0 rows affected (0.012 sec)
```

View 1: Display member names and their membership types



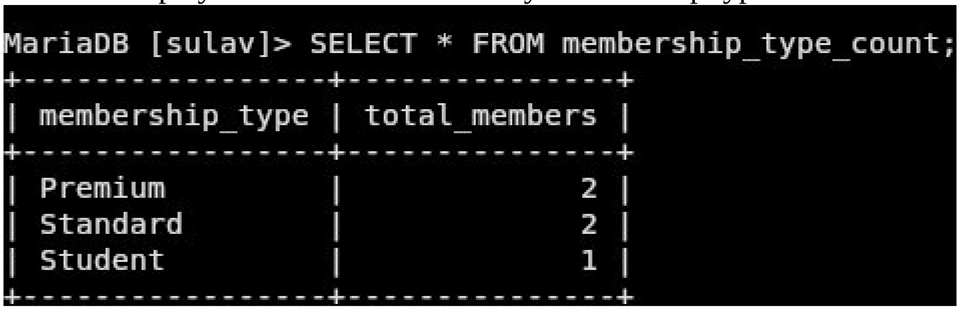
```
MariaDB [sulav]> SELECT * FROM member_basic_info
    -> ;
+----------------+-----------------+
| member_name    | membership_type |
+----------------+-----------------+
| Sulav Baral    | Premium         |
| Asha Limbu     | Standard        |
| Vikash Chhetri | Student         |
| Manisha Dhakal | Premium         |
| Rajan Bista    | Standard        |
+----------------+-----------------+
5 rows in set (0.001 sec)
```

View 2: Display members who have borrowed more than 1 book



```
MariaDB [sulav]> SELECT * FROM active_borrowers;
+----------------+----------------+-----------------------+
| member_name    | books_borrowed | address               |
+----------------+----------------+-----------------------+
| Sulav Baral    |              3 | 123 Thamel, Kathmandu |
| Vikash Chhetri |              2 | 789 Bharatpur, Chitwan |
| Rajan Bista    |              4 | 234 Birgunj, Parsa    |
+----------------+----------------+-----------------------+
3 rows in set (0.001 sec)
```

View 3: Display the count of members by membership type



```
MariaDB [sulav]> SELECT * FROM membership_type_count;
+-----------------+---------------+
| membership_type | total_members |
+-----------------+---------------+
| Premium         |             2 |
| Standard        |             2 |
| Student         |             1 |
+-----------------+---------------+
```

CONCLUSION:

In this lab, we learned how to create and use views in MySQL. Views simplify complex queries, provide controlled access to data, and improve data management. By creating different views (basic details, high salary employees, and city-wise distribution), we saw how views can be applied for reporting and security purposes. This demonstrates the importance of views in making databases more efficient and user-friendly.

Index