Tribhuvan University

Prithivi Narayan Campus

Pokhara,Kaski

Lab-report of Database Management System

Subject code: CSC

Submitted to:
Gyaneshwor Dhungana
Department of CSIT
Prithivi Narayan Campus

Submitted by:
Sabin Paudel
Roll no: 56
4th Semester

INDEX

LAB : 1
To study and understand the different types of Data Definition Language (DDL) commands in MySQL

Objective:
To study and understand the different types of Data Definition Language (DDL) commands in MySQL by creating a database and two tables: `student` and `book`.

Theory:
Data Definition Language (DDL) commands are used in SQL to define, alter, and manage database structures such as tables, indexes, and constraints. DDL does not manipulate data directly but rather handles the schema of the database.

Key DDL Commands:

1. **CREATE**: This command is used to create new databases, tables, indexes, views, etc.
   - Example:
     `CREATE TABLE table_name (column_name datatype, column_name datatype);`
2. **ALTER**: Used to modify the structure of an existing table, such as adding or deleting columns or constraints.
   - Example:
     `ALTER TABLE table_name ADD column_name datatype;`
3. **DROP**: Removes a database or table from the system.
   - Example:
     `DROP TABLE table_name;`
4. **TRUNCATE**: Deletes all records in a table without deleting the table itself.
   - Example:
     `TRUNCATE TABLE table_name;`
5. **RENAME**: Changes the name of a table.
   - Example:
     `RENAME TABLE old_name TO new_name;`

DDL commands are executed automatically and permanently alter the structure of the database. They do not interact with transactions, so once a DDL command is executed, it is committed and cannot be rolled back.

Lab Work:
1. Creating the Database Schema:
- We will create two tables: `student` and `book`.

```
-- Create a database named 'library_management'
CREATE DATABASE library_management;

-- Select the 'library_management' database
USE library_management;

-- Create the 'student' table
CREATE TABLE student (
    sid INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    address VARCHAR(100),
    contact VARCHAR(15),
    gender CHAR(1),
    age INT
);

-- Create the 'book' table
CREATE TABLE book (
    bid INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(100),
    author VARCHAR(50),
```

```
    publication VARCHAR(50),
    price DECIMAL(5, 2)
);
```

Output:

```
-- Create the 'student' and 'book table
```

```
MariaDB [(none)]> create database library
    -> ;
Query OK, 1 row affected (0.002 sec)

MariaDB [(none)]> use library;
Database changed
MariaDB [library]> CREATE TABLE student (
    ->     sid INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(50),
    ->     address VARCHAR(100),
    ->     contact VARCHAR(15),
    ->     gender CHAR(1),
    ->     age INT
    -> );
Query OK, 0 rows affected (0.024 sec)
```

```
MariaDB [library]> CREATE TABLE book (
    ->     bid INT PRIMARY KEY AUTO_INCREMENT,
    ->     title VARCHAR(100),
    ->     author VARCHAR(50),
    ->     publication VARCHAR(50),
    ->     price DECIMAL(5, 2)
    -> );
Query OK, 0 rows affected (0.020 sec)
```

```
MariaDB [library]> desc book;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| bid         | int(11)      | NO   | PRI | NULL    | auto_increment |
| title       | varchar(100) | YES  |     | NULL    |                |
| author      | varchar(50)  | YES  |     | NULL    |                |
| publication | varchar(50)  | YES  |     | NULL    |                |
| price       | decimal(5,2) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
5 rows in set (0.019 sec)
```

Conclusion:

In this lab, we successfully learned how to use DDL commands in MySQL to create and define the structure of databases and tables. We created a database `library_management` and two tables `student` and `book`. The `CREATE` command was used to create the tables, and the `DESCRIBE` command was used to verify the schema. This exercise reinforced our understanding of defining and managing the structure of a database in SQL.

Lab 2: Study of Different Data Types in MySQL

Objective:

To study and understand the various data types available in MySQL by creating a database schema that utilizes different data types.

Theory:

In MySQL, data types define the kind of data a column can hold. Choosing the right data type is crucial for optimizing storage and ensuring the correctness of the data. MySQL provides several categories of data types, including numeric, string (character), date and time, and boolean data types. Here's a breakdown of commonly used MySQL data types:

1. **Numeric Data Types**:
   - **INT**: Used to store integer values. It can be signed (default) or unsigned, where unsigned means non-negative values only.
     - Example: `INT` can hold values between -2147483648 and 2147483647.
   - **DECIMAL(M, D)**: Used to store fixed-point numbers where `M` is the total number of digits and `D` is the number of digits after the decimal point.
     - Example: `DECIMAL(5, 2)` can store numbers from -999.99 to 999.99.
   - **FLOAT/DOUBLE**: Used to store floating-point numbers, which are approximate representations of real numbers.
2. **String Data Types**:
   - **VARCHAR(n)**: A variable-length string that can hold up to `n` characters. It is more efficient than `CHAR` because it only uses the required space plus one byte.
     - Example: `VARCHAR(100)` can store up to 100 characters.
   - **CHAR(n)**: A fixed-length string. If the value is shorter than `n` characters, it is padded with spaces.
     - Example: `CHAR(10)` will always store 10 characters, padding with spaces if necessary.
   - **TEXT**: Used for large text data. Unlike `VARCHAR`, it does not require a specified length.
3. **Date and Time Data Types**:
   - **DATE**: Stores date values in 'YYYY-MM-DD' format. It is used when only the date (and not the time) is needed.
   - **DATETIME**: Stores date and time in the format 'YYYY-MM-DD HH:MM '. It can represent dates from 1000 to 9999 AD.
   - **TIMESTAMP**: Stores both date and time, but it is timezone-aware. The value is automatically updated to the current time when a record is modified, unless explicitly set otherwise.
4. **Boolean Data Type**:
   - **BOOLEAN**: It stores `TRUE` (1) or `FALSE` (0). MySQL considers it as a synonym for the `TINYINT(1)` type.
5. **Unique Constraints**:
   - **PRIMARY KEY**: This uniquely identifies each record in a table and cannot contain null values.
   - **UNIQUE**: This constraint ensures that all values in a column are different.

Lab Work:

1. Creating the Database and Tables:

- Create the database `school_management` and the following tables: `students`, `courses`, and `teachers`.

```
-- Create the database
CREATE DATABASE school_management;

-- Select the database
```

```
USE school_management;

-- Create the 'students' table
CREATE TABLE students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    email VARCHAR(100) UNIQUE,
    phone_number CHAR(10),
    enrollment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN
);

-- Create the 'courses' table
CREATE TABLE courses (
    course_id INT PRIMARY KEY AUTO_INCREMENT,
    course_name VARCHAR(100),
    credits DECIMAL(3, 1),
    course_start_date DATE,
    course_end_date DATE
);

-- Create the 'teachers' table
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    hire_date DATE,
    salary FLOAT,
    department VARCHAR(50)
);
```

Output:

Student table;

```
MariaDB [(none)]> CREATE DATABASE school_management;
Query OK, 1 row affected (0.002 sec)

MariaDB [(none)]> USE school_management;
Database changed
MariaDB [school_management]> CREATE TABLE students (
    ->     student_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     first_name VARCHAR(50),
    ->     last_name VARCHAR(50),
    ->     date_of_birth DATE,
    ->     email VARCHAR(100) UNIQUE,
    ->     phone_number CHAR(10),
    ->     enrollment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ->     is_active BOOLEAN
    -> );
Query OK, 0 rows affected (0.032 sec)
```

Courses table;

```
MariaDB [school_management]> CREATE TABLE courses (
    ->     course_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     course_name VARCHAR(100),
    ->     credits DECIMAL(3, 1),
    ->     course_start_date DATE,
    ->     course_end_date DATE
    -> );
Query OK, 0 rows affected (0.021 sec)
```

Teachers table;

```
MariaDB [school_management]> CREATE TABLE teachers (
    ->     teacher_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(100),
    ->     hire_date DATE,
    ->     salary FLOAT,
    ->     department VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.020 sec)
```

Table Description:

```
MariaDB [school_management]> desc students;
+-----------------+--------------+------+-----+---------------------+----------------+
| Field           | Type         | Null | Key | Default             | Extra          |
+-----------------+--------------+------+-----+---------------------+----------------+
| student_id      | int(11)      | NO   | PRI | NULL                | auto_increment |
| first_name      | varchar(50)  | YES  |     | NULL                |                |
| last_name       | varchar(50)  | YES  |     | NULL                |                |
| date_of_birth   | date         | YES  |     | NULL                |                |
| email           | varchar(100) | YES  | UNI | NULL                |                |
| phone_number    | char(10)     | YES  |     | NULL                |                |
| enrollment_date | timestamp    | NO   |     | current_timestamp() |                |
| is_active       | tinyint(1)   | YES  |     | NULL                |                |
+-----------------+--------------+------+-----+---------------------+----------------+
8 rows in set (0.018 sec)

MariaDB [school_management]> desc courses;
+-------------------+--------------+------+-----+---------+----------------+
| Field             | Type         | Null | Key | Default | Extra          |
+-------------------+--------------+------+-----+---------+----------------+
| course_id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| course_name       | varchar(100) | YES  |     | NULL    |                |
| credits           | decimal(3,1) | YES  |     | NULL    |                |
| course_start_date | date         | YES  |     | NULL    |                |
| course_end_date   | date         | YES  |     | NULL    |                |
+-------------------+--------------+------+-----+---------+----------------+
MariaDB [school_management]> desc teachers;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| teacher_id | int(11)      | NO   | PRI | NULL    | auto_increment |
| name       | varchar(100) | YES  |     | NULL    |                |
| hire_date  | date         | YES  |     | NULL    |                |
| salary     | float        | YES  |     | NULL    |                |
| department | varchar(50)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
5 rows in set (0.018 sec)
```

Conclusion:

In this lab, we successfully learned how to define and use different data types in MySQL by creating tables with columns using `INT`, `VARCHAR`, `CHAR`, `DATE`, `TIMESTAMP`, `BOOLEAN`, `DECIMAL`, and `FLOAT`. The data types were chosen based on the type of data being stored in each field, optimizing both storage and data retrieval efficiency. The `students`, `courses`, and `teachers` tables were created in the `school_management` database, illustrating the application of various data types.

Lab 3: Implementing Referential Integrity using Primary Key and Foreign Key

Objective:

To implement referential integrity in MySQL using primary keys and foreign keys by creating a new table `enrollments` that references the `students` and `courses` tables in the `school_management` database.

Theory:

Referential integrity ensures the accuracy and consistency of data within a database by enforcing relationships between tables. It is achieved using **primary keys** and **foreign keys**.

- **Primary Key**: A primary key is a unique identifier for a record in a table. It ensures that no two records can have the same primary key value, and it cannot be null. The primary key ensures that each record is unique.
- **Foreign Key**: A foreign key is a field (or collection of fields) in one table that refers to the **primary key** in another table. It enforces a relationship between two tables and ensures that the value in the foreign key column corresponds to a valid record in the referenced table.
    - For example, if `student_id` is a foreign key in the `enrollments` table, it must refer to a valid `student_id` in the `students` table. This ensures that every enrollment entry corresponds to a valid student.

Referential integrity is crucial because it prevents orphaned records, which are records that reference other non-existent records. By using foreign keys, databases can enforce relationships and ensure that changes in one table (e.g., deleting a student) don't leave inconsistent data in related tables (e.g., enrollments with invalid student references).

MySQL allows the use of foreign keys to enforce these relationships. Additionally, constraints such as **ON DELETE** and **ON UPDATE** can be used to define what happens when a referenced record is deleted or updated (e.g., CASCADE, RESTRICT, SET NULL).

Lab Work:

1. Adding the `enrollments` Table with Referential Integrity Constraints:

- We will add the `enrollments` table to the `school_management` database, ensuring that the `student_id` and `course_id` fields reference the `students` and `courses` tables respectively.

```
-- Create the 'enrollments' table with foreign key constraints
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    course_id INT,
    enrollment_status ENUM('enrolled', 'completed', 'dropped'),
    grade CHAR(2),
    FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE
CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- **Explanation**:
    - The `enrollment_id` field is the primary key and uniquely identifies each record in the `enrollments` table.
    - The `student_id` and `course_id` fields are foreign keys that reference the `student_id` field from the `students` table and the `course_id` field from the `courses` table respectively.
    - Referential integrity is enforced with `ON DELETE CASCADE` and `ON UPDATE CASCADE`. This ensures that if a student or course is deleted, the associated enrollment records are also deleted.

Output:

Table and Description of enrollment table;

```
MariaDB [school_management]> CREATE TABLE enrollments (
    ->     enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     student_id INT,
    ->     course_id INT,
    ->     enrollment_status ENUM('enrolled', 'completed', 'dropped'),
    ->     grade CHAR(2),
    ->     FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE ON UPDATE CASCADE,
    ->     FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE ON UPDATE CASCADE
    -> );
Query OK, 0 rows affected (0.048 sec)

MariaDB [school_management]> desc enrollments;
+-------------------+--------------------------------------+------+-----+---------+----------------+
| Field             | Type                                 | Null | Key | Default | Extra          |
+-------------------+--------------------------------------+------+-----+---------+----------------+
| enrollment_id     | int(11)                              | NO   | PRI | NULL    | auto_increment |
| student_id        | int(11)                              | YES  | MUL | NULL    |                |
| course_id         | int(11)                              | YES  | MUL | NULL    |                |
| enrollment_status | enum('enrolled','completed','dropped') | YES  |     | NULL    |                |
| grade             | char(2)                              | YES  |     | NULL    |                |
+-------------------+--------------------------------------+------+-----+---------+----------------+
5 rows in set (0.019 sec)

MariaDB [school_management]>
```

Conclusion:

In this lab, we successfully implemented referential integrity in the `school_management` database by adding the `enrollments` table, which references the `students` and `courses` tables through foreign keys. This ensures that every enrollment entry has valid references to both a student and a course. The use of foreign keys with `ON DELETE CASCADE` and `ON UPDATE CASCADE` options helps maintain data consistency, preventing orphaned records. This lab reinforced the importance of referential integrity in relational database management systems.

Lab 5: Study of Different Types of Joins in MySQL

Objective:

To study and explore different types of JOINs in MySQL by creating a database schema for an `online_store` and performing various JOIN operations on the tables.

Theory: \

A **JOIN** clause in MySQL is used to combine rows from two or more tables based on a related column between them. JOINs are essential for retrieving meaningful data from multiple tables in a relational database.

Types of JOINs:

1. **INNER JOIN**:
   - Returns records that have matching values in both tables.
   - Query Example:
     ```
     SELECT orders.order_id, customers.first_name, customers.last_name
     FROM orders
     INNER JOIN customers ON orders.customer_id =
     customers.customer_id;
     ```
2. **LEFT JOIN (LEFT OUTER JOIN)**:
   - Returns all records from the left table, and the matched records from the right table. If no match is found, NULL values are returned for the right table.
   - Query Example:
     ```
     SELECT products.product_name, categories.category_name
     FROM products
     LEFT JOIN categories ON products.category_id =
     categories.category_id;
     ```
3. **RIGHT JOIN (RIGHT OUTER JOIN)**:
   - Returns all records from the right table, and the matched records from the left table. If no match is found, NULL values are returned for the left table.
   - Query Example:
     ```
     SELECT orders.order_id, payments.amount_paid
     FROM payments
     RIGHT JOIN orders ON payments.order_id = orders.order_id;
     ```
4. **FULL OUTER JOIN**:
   - Returns all records when there is a match in either the left or right table. If no match is found, NULL values are returned from both sides. MySQL does not directly support FULL OUTER JOIN, but it can be simulated using a combination of `LEFT JOIN` and `RIGHT JOIN`.
   - Query Example:
     ```
     SELECT customers.first_name, orders.order_id
     FROM customers
     LEFT JOIN orders ON customers.customer_id = orders.customer_id
     UNION
     SELECT customers.first_name, orders.order_id
     FROM customers
     RIGHT JOIN orders ON customers.customer_id = orders.customer_id;
     ```
5. **CROSS JOIN**:
   - Returns the Cartesian product of the two tables, meaning each row from the first table is combined with every row from the second table.
   - Query Example:
     ```
     SELECT customers.first_name, products.product_name
     FROM customers
     CROSS JOIN products;
     ```
6. **SELF JOIN**:
   - A self join is a regular join where a table is joined with itself.
   - Query Example:
     ```
     SELECT c1.category_name AS Parent, c2.category_name AS
     Subcategory
     FROM categories c1
     ```

```
            LEFT JOIN categories c2 ON c1.category_id = c2.parent_category;
```

Lab Work:

1. Creating the Database Schema:

```
-- Create the database
CREATE DATABASE online_store;

-- Select the database
USE online_store;

-- Create the 'categories' table
CREATE TABLE categories (
    category_id INT PRIMARY KEY AUTO_INCREMENT,
    category_name VARCHAR(100),
    parent_category INT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (parent_category) REFERENCES categories(category_id)
);

-- Create the 'products' table
CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100),
    brand VARCHAR(50),
    price DECIMAL(10, 2),
    stock INT,
    category_id INT,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);

-- Create the 'customers' table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(15),
    address TEXT,
    city VARCHAR(50),
    postal_code VARCHAR(10),
    country VARCHAR(50),
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create the 'orders' table
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    shipping_address TEXT,
    total_amount DECIMAL(10, 2),
    order_status ENUM('pending', 'shipped', 'delivered', 'canceled'),
    payment_status ENUM('paid', 'unpaid', 'refunded'),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- Create the 'order_items' table
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT,
    price DECIMAL(10, 2),
```

```
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- Create the 'payments' table
CREATE TABLE payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    amount_paid DECIMAL(10, 2),
    payment_method ENUM('credit_card', 'debit_card', 'paypal',
'bank_transfer'),
    payment_status ENUM('completed', 'failed', 'pending'),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

-- Create the 'reviews' table
CREATE TABLE reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    customer_id INT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- Create the 'cart' table
CREATE TABLE cart (
    cart_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    product_id INT,
    quantity INT,
    added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

2. Exploring JOIN Queries:

- **INNER JOIN**: Fetch all orders along with customer names.
```
SELECT orders.order_id, customers.first_name, customers.last_name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```
- **LEFT JOIN**: List all products and their categories.
```
SELECT products.product_name, categories.category_name
FROM products
LEFT JOIN categories ON products.category_id = categories.category_id;
```
- **RIGHT JOIN**: List all orders and payments.
```
SELECT orders.order_id, payments.amount_paid
FROM payments
RIGHT JOIN orders ON payments.order_id = orders.order_id;
```
- **SELF JOIN**: Display categories with subcategories.
```
SELECT parent.category_name AS ParentCategory, child.category_name AS
SubCategory
FROM categories parent
LEFT JOIN categories child ON parent.category_id =
child.parent_category;
```

Output:
Tables

```
MariaDB [online_store]> show tables;
+------------------------+
| Tables_in_online_store |
+------------------------+
| cart                   |
| categories             |
| customers              |
| order_items            |
| orders                 |
| payments               |
| products               |
| reviews                |
+------------------------+
8 rows in set (0.001 sec)
```

Schema of each table:

- **Cart**

```
MariaDB [online_store]> desc cart;
+-------------+-----------+------+-----+---------------------+----------------+
| Field       | Type      | Null | Key | Default             | Extra          |
+-------------+-----------+------+-----+---------------------+----------------+
| cart_id     | int(11)   | NO   | PRI | NULL                | auto_increment |
| customer_id | int(11)   | YES  | MUL | NULL                |                |
| product_id  | int(11)   | YES  | MUL | NULL                |                |
| quantity    | int(11)   | YES  |     | NULL                |                |
| added_at    | timestamp | NO   |     | current_timestamp() |                |
+-------------+-----------+------+-----+---------------------+----------------+
```

- **Categories**

```
MariaDB [online_store]> desc categories;
+-----------------+--------------+------+-----+---------------------+----------------+
| Field           | Type         | Null | Key | Default             | Extra          |
+-----------------+--------------+------+-----+---------------------+----------------+
| category_id     | int(11)      | NO   | PRI | NULL                | auto_increment |
| category_name   | varchar(100) | YES  |     | NULL                |                |
| parent_category | int(11)      | YES  | MUL | NULL                |                |
| created_at      | timestamp    | NO   |     | current_timestamp() |                |
+-----------------+--------------+------+-----+---------------------+----------------+
```

- **Customers**

```
MariaDB [online_store]> desc customers;
+-------------------+--------------+------+-----+---------------------+----------------+
| Field             | Type         | Null | Key | Default             | Extra          |
+-------------------+--------------+------+-----+---------------------+----------------+
| customer_id       | int(11)      | NO   | PRI | NULL                | auto_increment |
| first_name        | varchar(50)  | YES  |     | NULL                |                |
| last_name         | varchar(50)  | YES  |     | NULL                |                |
| email             | varchar(100) | YES  | UNI | NULL                |                |
| phone_number      | varchar(15)  | YES  |     | NULL                |                |
| address           | text         | YES  |     | NULL                |                |
| city              | varchar(50)  | YES  |     | NULL                |                |
| postal_code       | varchar(10)  | YES  |     | NULL                |                |
| country           | varchar(50)  | YES  |     | NULL                |                |
| registration_date | timestamp    | NO   |     | current_timestamp() |                |
+-------------------+--------------+------+-----+---------------------+----------------+
```

- **Order_items**

```
MariaDB [online_store]> desc order_items;
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| order_item_id | int(11)       | NO   | PRI | NULL    | auto_increment |
| order_id      | int(11)       | YES  | MUL | NULL    |                |
| product_id    | int(11)       | YES  | MUL | NULL    |                |
| quantity      | int(11)       | YES  |     | NULL    |                |
| price         | decimal(10,2) | YES  |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
5 rows in set (0.019 sec)
```

- **Orders**

```
+------------------+--------------------------------------------------------+------+-----+---------------------+----------------+
| Field            | Type                                                   | Null | Key | Default             | Extra          |
+------------------+--------------------------------------------------------+------+-----+---------------------+----------------+
| order_id         | int(11)                                                | NO   | PRI | NULL                | auto_increment |
| customer_id      | int(11)                                                | YES  | MUL | NULL                |                |
| order_date       | timestamp                                              | NO   |     | current_timestamp() |                |
| shipping_address | text                                                   | YES  |     | NULL                |                |
| total_amount     | decimal(10,2)                                          | YES  |     | NULL                |                |
| order_status     | enum('pending','shipped','delivered','canceled')       | YES  |     | NULL                |                |
| payment_status   | enum('paid','unpaid','refunded')                       | YES  |     | NULL                |                |
+------------------+--------------------------------------------------------+------+-----+---------------------+----------------+
7 rows in set (0.018 sec)
```

- **Payments**

```
MariaDB [online_store]> desc payments;
+----------------+------------------------------------------------------------+------+-----+---------------------+----------------+
| Field          | Type                                                       | Null | Key | Default             | Extra          |
+----------------+------------------------------------------------------------+------+-----+---------------------+----------------+
| payment_id     | int(11)                                                    | NO   | PRI | NULL                | auto_increment |
| order_id       | int(11)                                                    | YES  | MUL | NULL                |                |
| payment_date   | timestamp                                                  | NO   |     | current_timestamp() |                |
| amount_paid    | decimal(10,2)                                              | YES  |     | NULL                |                |
| payment_method | enum('credit_card','debit_card','paypal','bank_transfer')  | YES  |     | NULL                |                |
| payment_status | enum('completed','failed','pending')                       | YES  |     | NULL                |                |
+----------------+------------------------------------------------------------+------+-----+---------------------+----------------+
6 rows in set (0.019 sec)
```

- **Products**

```
MariaDB [online_store]> desc products;
+--------------+---------------+------+-----+---------------------+-------------------------------+
| Field        | Type          | Null | Key | Default             | Extra                         |
+--------------+---------------+------+-----+---------------------+-------------------------------+
| product_id   | int(11)       | NO   | PRI | NULL                | auto_increment                |
| product_name | varchar(100)  | YES  |     | NULL                |                               |
| brand        | varchar(50)   | YES  |     | NULL                |                               |
| price        | decimal(10,2) | YES  |     | NULL                |                               |
| stock        | int(11)       | YES  |     | NULL                |                               |
| category_id  | int(11)       | YES  | MUL | NULL                |                               |
| description  | text          | YES  |     | NULL                |                               |
| created_at   | timestamp     | NO   |     | current_timestamp() |                               |
| updated_at   | timestamp     | NO   |     | current_timestamp() | on update current_timestamp() |
+--------------+---------------+------+-----+---------------------+-------------------------------+
```

- **Reviews**

```
MariaDB [online_store]> desc reviews;
+-------------+-----------+------+-----+---------------------+----------------+
| Field       | Type      | Null | Key | Default             | Extra          |
+-------------+-----------+------+-----+---------------------+----------------+
| review_id   | int(11)   | NO   | PRI | NULL                | auto_increment |
| product_id  | int(11)   | YES  | MUL | NULL                |                |
| customer_id | int(11)   | YES  | MUL | NULL                |                |
| rating      | int(11)   | YES  |     | NULL                |                |
| review_text | text      | YES  |     | NULL                |                |
| review_date | timestamp | NO   |     | current_timestamp() |                |
+-------------+-----------+------+-----+---------------------+----------------+
6 rows in set (0.016 sec)
```

Join Output:

- **Inner Join :**

```
MariaDB [online_store]> SELECT orders.order_id, customers.first_name, customers.last_name
    -> FROM orders
    -> INNER JOIN customers ON orders.customer_id = customers.customer_id;
+----------+------------+-----------+
| order_id | first_name | last_name |
+----------+------------+-----------+
|        1 | John       | Doe       |
|        2 | Jane       | Smith     |
+----------+------------+-----------+
2 rows in set (0.000 sec)
```

- **Left Join:**

```
MariaDB [online_store]> SELECT products.product_name, categories.category_name
    -> FROM products
    -> LEFT JOIN categories ON products.category_id = categories.category_id;
+--------------------+---------------+
| product_name       | category_name |
+--------------------+---------------+
| iPhone 12          | Mobile Phones |
| MacBook Pro        | Laptops       |
| Samsung Galaxy S21 | Mobile Phones |
+--------------------+---------------+
```

- **Right Join:**

```
MariaDB [online_store]> SELECT orders.order_id, payments.amount_paid
    -> FROM payments
    -> RIGHT JOIN orders ON payments.order_id = orders.order_id;
+----------+-------------+
| order_id | amount_paid |
+----------+-------------+
|        1 |        NULL |
|        2 |        NULL |
+----------+-------------+
2 rows in set (0.000 sec)
```

- **Self Join:**

```
MariaDB [online_store]> SELECT parent.category_name AS ParentCategory, child.category_name AS SubCategory
    -> FROM categories parent
    -> LEFT JOIN categories child ON parent.category_id = child.parent_category;
+----------------+---------------+
| ParentCategory | SubCategory   |
+----------------+---------------+
| Electronics    | Mobile Phones |
| Electronics    | Laptops       |
| Mobile Phones  | NULL          |
| Laptops        | NULL          |
| Appliances     | NULL          |
+----------------+---------------+
5 rows in set (0.001 sec)
```

Conclusion:

In this lab, we successfully created the database schema for an `online_store`, which included multiple tables such as `products`, `categories`, `customers`, `orders`, etc. We then explored different types of JOINs, including INNER JOIN, LEFT JOIN, RIGHT JOIN, and SELF JOIN, demonstrating how to combine data from multiple related tables effectively. This lab reinforced the concept of relational database management and the importance of JOINs in querying meaningful data from multiple tables.

Lab 4: Designing a Database using ER Model and Implementing it in MySQL

OBJECTIVE:

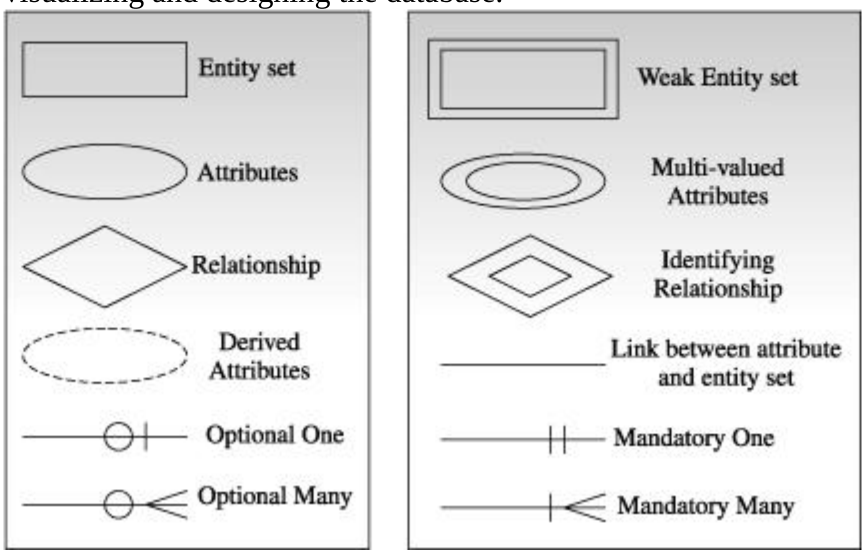
To design a database using an Entity-Relationship (ER) model and implement it in MySQL, covering the process from conceptual design to database creation.

THEORY:

An **Entity-Relationship (ER) model** is a high-level conceptual data model that defines the structure of the database by identifying entities, their attributes, and the relationships between them.

- **Entity**: Represents a real-world object or concept, like `Customer`, `Order`, or `Product`.
- **Attributes**: Characteristics of entities, like `name`, `email`, `price`.
- **Relationships**: Describe how entities are related to one another, like `Customer` places an `Order`.

An ER diagram (ERD) graphically shows the entities, attributes, and relationships, helping in visualizing and designing the database.



IMPLEMENTATION

1. Scenario:

We will design a database for a **Futsal Booking System**. The system has the following entities:

- **Customer**: Registers to book a futsal field.
- **Booking**: Customers place bookings.
- **Futsal Field**: Fields available for booking.

2. Entities and Attributes:

- **Customer**:
  - `customer_id` (Primary Key)
  - `first_name`
  - `last_name`
  - `email`
  - `phone`
  - `city`
- **Booking**:
  - `booking_id` (Primary Key)
  - `customer_id` (Foreign Key to Customer)
  - `field_id` (Foreign Key to Futsal Field)
  - `booking_date`
  - `total_price`
- **Futsal Field**:
  - `field_id` (Primary Key)

- field_name
- location
- price_per_hour
- availability

3. Relationships:

- A **Customer** can make many **Bookings**.
- A **Futsal Field** can be booked multiple times by different **Customers**.
- Each **Booking** is linked to one **Customer** and one **Futsal Field**.

4. ER Diagram:

You can represent the ER diagram with:

- **Entities**: Customer, Booking, Futsal Field.
- **Relationships**: One-to-many relationship between `Customer` and `Booking`, and one-to-many between `Futsal Field` and `Booking`.

(Note: Use any ER diagramming tool like MySQL Workbench, Lucidchart, or pen-and-paper for visualizing the diagram.)

5. Schema Design:

Translate the ER diagram into the MySQL schema.

MySQL Implementation:

Step 1: Create the Database

```
CREATE DATABASE futsal_booking;
USE futsal_booking;
```

Step 2: Create Tables

Customer Table:

```
CREATE TABLE Customer (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    city VARCHAR(50)
);
```

Futsal Field Table:

```
CREATE TABLE Futsal_Field (
    field_id INT AUTO_INCREMENT PRIMARY KEY,
    field_name VARCHAR(50),
    location VARCHAR(100),
    price_per_hour DECIMAL(10,2),
    availability BOOLEAN
);
```

Booking Table:

```
CREATE TABLE Booking (
    booking_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    field_id INT,
    booking_date DATE,
    total_price DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (field_id) REFERENCES Futsal_Field(field_id)
);
```

Step 3: Insert Sample Data

Insert into Customer Table:

```
INSERT INTO Customer (first_name, last_name, email, phone, city)
VALUES
('John', 'Doe', 'john@example.com', '9876543210', 'Pokhara'),
('Jane', 'Smith', 'jane@example.com', '9845123456', 'Syangja');
```

Insert into Futsal Field Table:

```
INSERT INTO Futsal_Field (field_name, location, price_per_hour, availability)
VALUES
('City Arena', 'Pokhara', 500.00, TRUE),
('Syangja Turf', 'Syangja', 450.00, TRUE);
```

Insert into Booking Table:

```
INSERT INTO Booking (customer_id, field_id, booking_date, total_price)
VALUES
(1, 1, '2024-09-25', 1000.00),
(2, 2, '2024-09-26', 900.00);
```

Step 4: Run Queries to Retrieve Data

a. Retrieve all customers:

```
SELECT * FROM Customer;
```

b. Retrieve all bookings:

```
SELECT * FROM Booking;
```

c. Retrieve customer bookings with futsal field details:

```
SELECT
    Customer.first_name,
    Customer.last_name,
    Futsal_Field.field_name,
    Booking.booking_date,
    Booking.total_price
FROM
    Booking
JOIN
    Customer ON Booking.customer_id = Customer.customer_id
JOIN
    Futsal_Field ON Booking.field_id = Futsal_Field.field_id;
```

OUTPUT:

a)

```
MariaDB [futsal_booking]> SELECT * FROM Customer;
+-------------+------------+-----------+------------------+------------+---------+
| customer_id | first_name | last_name | email            | phone      | city    |
+-------------+------------+-----------+------------------+------------+---------+
|           1 | John       | Doe       | john@example.com | 9876543210 | Pokhara |
|           2 | Jane       | Smith     | jane@example.com | 9845123456 | Syangja |
+-------------+------------+-----------+------------------+------------+---------+
```

b)

```
MariaDB [futsal_booking]> SELECT * FROM Booking;
+------------+-------------+----------+--------------+-------------+
| booking_id | customer_id | field_id | booking_date | total_price |
+------------+-------------+----------+--------------+-------------+
|          1 |           1 |        1 | 2024-09-25   |     1000.00 |
|          2 |           2 |        2 | 2024-09-26   |      900.00 |
+------------+-------------+----------+--------------+-------------+
2 rows in set (0.001 sec)
```

c)

```
+------------+-----------+--------------+--------------+-------------+
| first_name | last_name | field_name   | booking_date | total_price |
+------------+-----------+--------------+--------------+-------------+
| John       | Doe       | City Arena   | 2024-09-25   |     1000.00 |
| Jane       | Smith     | Syangja Turf | 2024-09-26   |      900.00 |
+------------+-----------+--------------+--------------+-------------+
2 rows in set (0.001 sec)
```

CONCLUSION:

This lab taught the process of designing a database using the ER model and implementing it in MySQL. We learned how to define entities, attributes, and relationships, followed by creating tables and inserting data into them. Querying the database based on relationships between entities also provides insights into real-world scenarios of data interaction.

Lab 6: SELECT Queries in MySQL

OBJECTIVE

To design and execute complex SELECT queries in MySQL to manipulate and retrieve data from a database.

THEORY

The `SELECT` statement is used in SQL to query data from one or more tables in a database. It allows users to retrieve specific columns, filter results, sort data, and perform calculations on the data.

Basic Structure of a SELECT Query:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Key Clauses:

- **SELECT:** Specifies the columns to retrieve.
- **FROM:** Indicates the table(s) from which to select data.
- **WHERE:** Filters the results based on specified conditions.
- **ORDER BY:** Sorts the results by one or more columns.
- **GROUP BY:** Groups rows with the same values in specified columns for aggregate functions.
- **JOIN:** Combines rows from two or more tables based on related columns.

Common Aggregate Functions:

- `COUNT()`: Returns the number of rows.
- `SUM()`: Calculates the total sum of a numeric column.
- `AVG()`: Computes the average of a numeric column.
- `MAX()`: Finds the maximum value in a column.
- `MIN()`: Finds the minimum value in a column.

LAB WORK: Execute the Following SELECT Operations

a. Retrieve all products from the products table.

```
SELECT * FROM products;
```

b. Find all products where the price is greater than 50.

```
SELECT * FROM products WHERE price > 50;
```

c. Retrieve customers from the customers table who live in the city 'Pokhara'.

```
SELECT * FROM customers WHERE city = 'Pokhara';
```

d. Fetch orders where the order status is 'pending'.

```
SELECT * FROM orders WHERE order_status = 'pending';
```

e. Find all products that belong to the category 'Electronics'.

```
SELECT * FROM products WHERE category = 'Electronics';
```

f. Retrieve all customers who have placed an order with a total amount greater than 500.

```
SELECT DISTINCT customers.*
FROM customers
JOIN orders ON customers.id = orders.customer_id
WHERE orders.total_amount > 500;
```

g. Find the total number of products in the store.

```
SELECT COUNT(*) AS total_products FROM products;
```

h. Retrieve the maximum, minimum, and average price of all products.

```
SELECT MAX(price) AS max_price, MIN(price) AS min_price, AVG(price) AS
average_price FROM products;
```

i. Find the total amount of all orders placed by a specific customer (e.g., customer_id = 3).

```
SELECT SUM(total_amount) AS total_spent
FROM orders
WHERE customer_id = 3;
```

j. Count the number of orders that have the status 'completed'.

```
SELECT COUNT(*) AS completed_orders
FROM orders
WHERE order_status = 'completed';
```

k. Display all products ordered by price in ascending order.

```
SELECT * FROM products ORDER BY price ASC;
```

l. Retrieve the latest 10 orders placed by customers, ordered by order_date in descending order.

```
SELECT * FROM orders ORDER BY order_date DESC LIMIT 10;
```

m. Retrieve a list of all orders along with the customer's first and last name.

```
SELECT orders.*, customers.first_name, customers.last_name
FROM orders
JOIN customers ON orders.customer_id = customers.id;
```

n. List all order items along with the corresponding product name and price.

```
SELECT order_items.*, products.name, products.price
FROM order_items
JOIN products ON order_items.product_id = products.id;
```

o. Retrieve the list of products and their category names.

```
SELECT products.*, categories.name AS category_name
FROM products
JOIN categories ON products.category_id = categories.id;
```

p. Fetch all orders along with the total price for each order (using SUM in the join).

```
SELECT orders.id, SUM(order_items.price * order_items.quantity) AS
total_price
FROM orders
JOIN order_items ON orders.id = order_items.order_id
GROUP BY orders.id;
```

q. Retrieve all products that are low in stock (stock less than 5) and have been ordered by any customer.

```
SELECT DISTINCT products.*
FROM products
JOIN order_items ON products.id = order_items.product_id
WHERE products.stock < 5;
```

r. List the top 5 products that have been ordered the most (use COUNT and GROUP BY).

```
SELECT products.name, COUNT(order_items.id) AS order_count
FROM order_items
JOIN products ON order_items.product_id = products.id
GROUP BY products.id
ORDER BY order_count DESC
LIMIT 5;
```

s. Fetch the total amount of money each customer has spent, along with their name.

```
SELECT customers.first_name, customers.last_name, SUM(orders.total_amount) AS
total_spent
FROM customers
JOIN orders ON customers.id = orders.customer_id
GROUP BY customers.id;
```

t. Find customers who have never placed an order.

```
SELECT * FROM customers
WHERE id NOT IN (SELECT DISTINCT customer_id FROM orders);
```

u. List all orders where the customer's total payment is less than the total amount of the order (use payments and orders tables).

```
SELECT orders.*
FROM orders
JOIN payments ON orders.id = payments.order_id
WHERE payments.amount < orders.total_amount;
```

OUTPUT

a. Retrieve all products from the products table.

```
mysql> SELECT * FROM products;
+------------+--------------+-------------------+----------+-------+-------------+-------------------------------+---------------------+---------------------+
| product_id | product_name | brand             | price    | stock | category_id | description                   | created_at          | updated_at          |
+------------+--------------+-------------------+----------+-------+-------------+-------------------------------+---------------------+---------------------+
|          1 | Smartphone   | Samsung           | 25000.00 |    10 |           1 | High quality smartphone       | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          2 | Shirt        | Vipasha           |  1200.00 |    25 |           2 | Trendy and comfortable shirt  | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          3 | Mixer        | Mitsubishi        |  3500.00 |    15 |           3 | Mixer for various food items  | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          4 | Book         | Nepali Book Press |   500.00 |    30 |           4 | Educational book              | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          5 | Laptop       | Dell              | 70000.00 |     5 |           1 | High-performance laptop       | 2024-09-22 16:50:00 | 2024-09-22 16:50:00 |
+------------+--------------+-------------------+----------+-------+-------------+-------------------------------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

b. Find all products where the price is greater than 500.

```
mysql> SELECT * FROM products WHERE price > 500;
+------------+--------------+------------+----------+-------+-------------+-------------------------------+---------------------+---------------------+
| product_id | product_name | brand      | price    | stock | category_id | description                   | created_at          | updated_at          |
+------------+--------------+------------+----------+-------+-------------+-------------------------------+---------------------+---------------------+
|          1 | Smartphone   | Samsung    | 25000.00 |    10 |           1 | High quality smartphone       | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          2 | Shirt        | Vipasha    |  1200.00 |    25 |           2 | Trendy and comfortable shirt  | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          3 | Mixer        | Mitsubishi |  3500.00 |    15 |           3 | Mixer for various food items  | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          5 | Laptop       | Dell       | 70000.00 |     5 |           1 | High-performance laptop       | 2024-09-22 16:50:00 | 2024-09-22 16:50:00 |
+------------+--------------+------------+----------+-------+-------------+-------------------------------+---------------------+---------------------+
4 rows in set (0.00 sec)
```

c. Retrieve customers from the customers table who live in the city ‘Kaski’.

```
mysql> SELECT * FROM customers WHERE city = 'Kaski';
+-------------+------------+-----------+-----------------------------+--------------+---------+-------+-------------+---------+---------------------+
| customer_id | first_name | last_name | email                       | phone_number | address | city  | postal_code | country | registration_date   |
+-------------+------------+-----------+-----------------------------+--------------+---------+-------+-------------+---------+---------------------+
|           5 | Kriti      | Chaudhary | kriti.chaudhary@example.com | 9800000005   | Pokhara | Kaski | 33700       | Nepal   | 2024-09-22 16:50:00 |
+-------------+------------+-----------+-----------------------------+--------------+---------+-------+-------------+---------+---------------------+
1 row in set (0.00 sec)
```

d. Fetch orders where the order status is 'pending'.

```
mysql> SELECT * FROM orders WHERE order_status = 'pending';
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+
| order_id | customer_id | order_date          | shipping_address | total_amount | order_status | payment_status |
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+
|        1 |           1 | 2024-09-20 12:00:00 | Gokarneshwar     |     25000.00 | pending      | unpaid         |
|        5 |           6 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         |
|       10 |          10 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         |
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+
3 rows in set (0.01 sec)
```

Find all products that belong to the category 'Electronics'.

```
mysql> SELECT products.product_name , products.brand , products.description,categories.* FROM products JOIN categories ON prod
ucts.category_id = categories.category_id WHERE categories.categor
y_name = 'Electronics';

+--------------+---------+-------------------------+-------------+---------------+-----------------+---------------------+
| product_name | brand   | description             | category_id | category_name | parent_category | created_at          |
+--------------+---------+-------------------------+-------------+---------------+-----------------+---------------------+
| Smartphone   | Samsung | High quality smartphone |           1 | Electronics   |            NULL | 2024-09-22 16:43:46 |
| Laptop       | Dell    | High-performance laptop |           1 | Electronics   |            NULL | 2024-09-22 16:43:46 |
+--------------+---------+-------------------------+-------------+---------------+-----------------+---------------------+
2 rows in set (0.00 sec)
```

e. Retrieve all customers who have placed an order with a total amount greater than 500.

```
mysql> SELECT DISTINCT customers.* FROM customers JOIN orders ON customers.customer_id = orders.customer_id where orders.total_amount > 500;
+-------------+------------+-----------+---------------------------+--------------+--------------+-----------+-------------+---------+---------------------+
| customer_id | first_name | last_name | email                     | phone_number | address      | city      | postal_code | country | registration_date   |
+-------------+------------+-----------+---------------------------+--------------+--------------+-----------+-------------+---------+---------------------+
|           1 | Sonu       | Sharma    | sonu.sharma@example.com   | 9800000001   | Gokarneshwar | Kathmandu | 44600       | Nepal   | 2024-09-22 16:43:57 |
|           2 | Maya       | Devi      | maya.devi@example.com     | 9800000002   | Patan        | Lalitpur  | 44700       | Nepal   | 2024-09-22 16:43:57 |
|           3 | Ramu       | Giri      | ramu.giri@example.com     | 9800000003   | Birgunj      | Parsa     | 45400       | Nepal   | 2024-09-22 16:43:57 |
|           6 | Hari       | Shrestha  | hari.shrestha@example.com | 9800000006   | Dharan       | Sunsari   | 56700       | Nepal   | 2024-09-22 16:50:00 |
+-------------+------------+-----------+---------------------------+--------------+--------------+-----------+-------------+---------+---------------------+
4 rows in set (0.00 sec)
```

f. Find the total number of products in the store.

```
mysql> SELECT COUNT(*) AS total_products FROM products;
+----------------+
| total_products |
+----------------+
|              5 |
+----------------+
1 row in set (0.00 sec)
```

g. Retrieve the maximum, minimum, and average price of all products.

```
mysql> SELECT MAX(price) AS max_price, MIN(price) AS min_price, AVG(price) AS avg_price FROM products;
+-----------+-----------+--------------+
| max_price | min_price | avg_price    |
+-----------+-----------+--------------+
|  70000.00 |    500.00 | 20040.000000 |
+-----------+-----------+--------------+
1 row in set (0.00 sec)
```

h. Find the total amount of all orders placed by a specific customer (e.g., customer_id = 3).

```
mysql> SELECT SUM(total_amount) AS total_amount_spent FROM orders WHERE customer_id = 3;
+--------------------+
| total_amount_spent |
+--------------------+
|           73500.00 |
+--------------------+
1 row in set (0.00 sec)
```

i. Count the number of orders that have the status 'shipped'.

```
mysql> SELECT COUNT(*) AS completed_orders FROM orders WHERE order_status = 'shipped';
+------------------+
| completed_orders |
+------------------+
|                2 |
+------------------+
1 row in set (0.00 sec)
```

j. Display all products ordered by price in ascending order.

```
mysql> SELECT * FROM products ORDER BY price ASC;
+------------+--------------+-------------------+----------+-------+-------------+------------------------------+---------------------+---------------------+
| product_id | product_name | brand             | price    | stock | category_id | description                  | created_at          | updated_at          |
+------------+--------------+-------------------+----------+-------+-------------+------------------------------+---------------------+---------------------+
|          4 | Book         | Nepali Book Press |   500.00 |    30 |           4 | Educational book             | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          2 | Shirt        | Vipasha           |  1200.00 |    25 |           2 | Trendy and comfortable shirt | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          3 | Mixer        | Mitsubishi        |  3500.00 |    15 |           3 | Mixer for various food items | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          1 | Smartphone   | Samsung           | 25000.00 |    10 |           1 | High quality smartphone      | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
|          5 | Laptop       | Dell              | 70000.00 |     5 |           1 | High-performance laptop      | 2024-09-22 16:50:00 | 2024-09-22 16:50:00 |
+------------+--------------+-------------------+----------+-------+-------------+------------------------------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

k. Retrieve the latest 10 orders placed by customers, ordered by order_date in descending order.

```
mysql> SELECT * FROM orders ORDER BY order_date DESC LIMIT 10;
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+
| order_id | customer_id | order_date          | shipping_address | total_amount | order_status | payment_status |
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+
|        4 |           4 | 2024-09-23 18:00:00 | Dharan           |       500.00 | shipped      | paid           |
|        5 |           6 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         |
|       10 |          10 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         |
|       11 |           3 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         |
|        3 |           3 | 2024-09-22 16:45:00 | Birgunj          |      3500.00 | delivered    | paid           |
|        2 |           2 | 2024-09-21 14:30:00 | Patan            |      1200.00 | shipped      | paid           |
|        1 |           1 | 2024-09-20 12:00:00 | Gokarneshwar     |     25000.00 | pending      | unpaid         |
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+
7 rows in set (0.00 sec)
```

l. Retrieve a list of all orders along with the customer's first and last name.

```
mysql> SELECT orders.* , customers.first_name,customers.last_name FROM orders JOIN customers ON orders.customer_id = customers.customer_id;
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+------------+-----------+
| order_id | customer_id | order_date          | shipping_address | total_amount | order_status | payment_status | first_name | last_name |
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+------------+-----------+
|        1 |           1 | 2024-09-20 12:00:00 | Gokarneshwar     |     25000.00 | pending      | unpaid         | Sonu       | Sharma    |
|        2 |           2 | 2024-09-21 14:30:00 | Patan            |      1200.00 | shipped      | paid           | Maya       | Devi      |
|        3 |           3 | 2024-09-22 16:45:00 | Birgunj          |      3500.00 | delivered    | paid           | Ramu       | Giri      |
|       11 |           3 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         | Ramu       | Giri      |
|        4 |           4 | 2024-09-23 18:00:00 | Dharan           |       500.00 | shipped      | paid           | Raj        | Kumar     |
|        5 |           6 | 2024-09-22 16:50:00 | Pokhara          |     70000.00 | pending      | unpaid         | Hari       | Shrestha  |
+----------+-------------+---------------------+------------------+--------------+--------------+----------------+------------+-----------+
6 rows in set (0.00 sec)
```

m. List all order items along with the corresponding product name and price.

```
mysql> SELECT order_items.* , products.product_name , products.price FROM order_items JOIN products ON order_items.product_id = products.product_id;
+---------------+----------+------------+----------+----------+--------------+----------+
| order_item_id | order_id | product_id | quantity | price    | product_name | price    |
+---------------+----------+------------+----------+----------+--------------+----------+
|             1 |        1 |          1 |        1 | 25000.00 | Smartphone   | 25000.00 |
|             2 |        2 |          2 |        1 |  1200.00 | Shirt        |  1200.00 |
|             3 |        3 |          3 |        1 |  3500.00 | Mixer        |  3500.00 |
|             4 |        4 |          4 |        1 |   500.00 | Book         |   500.00 |
+---------------+----------+------------+----------+----------+--------------+----------+
4 rows in set (0.00 sec)
```

n. Retrieve the list of products and their category names.

```
mysql> SELECT p.product_name, c.category_name
    -> FROM products p
    -> JOIN categories c ON p.category_id = c.category_id;
+--------------+-----------------+
| product_name | category_name   |
+--------------+-----------------+
| Smartphone   | Electronics     |
| Laptop       | Electronics     |
| Shirt        | Fashion         |
| Mixer        | Home Appliances |
| Book         | Books           |
+--------------+-----------------+
5 rows in set (0.01 sec)
```

o. Fetch all orders along with the total price for each order (using SUM in the join).

```
mysql> SELECT o.order_id, SUM(oi.price * oi.quantity) AS total_price
    -> FROM orders o
    -> JOIN order_items oi ON o.order_id = oi.order_id
    -> GROUP BY o.order_id;
+----------+-------------+
| order_id | total_price |
+----------+-------------+
|        1 |    25000.00 |
|        2 |     1200.00 |
|        3 |     3500.00 |
|        4 |      500.00 |
+----------+-------------+
4 rows in set (0.00 sec)
```

p. Retrieve all products that are low in stock (stock less than 15) and have been ordered by any customer.

```
mysql> SELECT DISTINCT p.* FROM products p JOIN order_items oi ON p.product_id = oi.product_id WHERE p.stock < 15;
+------------+--------------+---------+----------+-------+-------------+--------------------------+---------------------+---------------------+
| product_id | product_name | brand   | price    | stock | category_id | description              | created_at          | updated_at          |
+------------+--------------+---------+----------+-------+-------------+--------------------------+---------------------+---------------------+
|          1 | Smartphone   | Samsung | 25000.00 |    10 |           1 | High quality smartphone  | 2024-09-22 16:43:53 | 2024-09-22 16:43:53 |
+------------+--------------+---------+----------+-------+-------------+--------------------------+---------------------+---------------------+
1 row in set (0.00 sec)
```

q. List the top 5 products that have been ordered the most (use COUNT and GROUP BY).

```
mysql> SELECT p.product_name, COUNT(oi.product_id) AS total_orders
    -> FROM order_items oi
    -> JOIN products p ON oi.product_id = p.product_id
    -> GROUP BY p.product_id
    -> ORDER BY total_orders DESC
    -> LIMIT 5;
+--------------+--------------+
| product_name | total_orders |
+--------------+--------------+
| Smartphone   |            1 |
| Shirt        |            1 |
| Mixer        |            1 |
| Book         |            1 |
+--------------+--------------+
4 rows in set (0.00 sec)
```

r. Fetch the total amount of money each customer has spent, along with their name.

```
mysql> SELECT c.first_name, c.last_name, SUM(o.total_amount) AS total_spent
    -> FROM customers c
    -> JOIN orders o ON c.customer_id = o.customer_id
    -> GROUP BY c.customer_id;
+------------+-----------+-------------+
| first_name | last_name | total_spent |
+------------+-----------+-------------+
| Sonu       | Sharma    |    25000.00 |
| Maya       | Devi      |     1200.00 |
| Ramu       | Giri      |    73500.00 |
| Raj        | Kumar     |      500.00 |
| Hari       | Shrestha  |    70000.00 |
+------------+-----------+-------------+
5 rows in set (0.00 sec)
```

s. Find customers who have never placed an order.

```
mysql> SELECT c.*
    -> FROM customers c
    -> LEFT JOIN orders o ON c.customer_id = o.customer_id
    -> WHERE o.order_id IS NULL;
+-------------+------------+-----------+-----------------------------+--------------+---------+-------+-------------+---------+---------------------+
| customer_id | first_name | last_name | email                       | phone_number | address | city  | postal_code | country | registration_date   |
+-------------+------------+-----------+-----------------------------+--------------+---------+-------+-------------+---------+---------------------+
|           5 | Kriti      | Chaudhary | kriti.chaudhary@example.com | 9800000005   | Pokhara | Kaski | 33700       | Nepal   | 2024-09-22 16:50:00 |
+-------------+------------+-----------+-----------------------------+--------------+---------+-------+-------------+---------+---------------------+
1 row in set (0.00 sec)
```

t. List all orders where the customer's total payment is less than the total amount of the order.

```
mysql> SELECT o.order_id, o.total_amount, SUM(p.amount_paid) AS total_paid
    -> FROM orders o
    -> JOIN payments p ON o.order_id = p.order_id
    -> GROUP BY o.order_id
    -> HAVING total_paid < o.total_amount;
+----------+--------------+------------+
| order_id | total_amount | total_paid |
+----------+--------------+------------+
|        1 |     25000.00 |       0.00 |
+----------+--------------+------------+
1 row in set (0.00 sec)
```

CONCLUSION

This lab provided hands-on experience with complex SELECT queries in MySQL, highlighting the importance of data retrieval and manipulation in database management. The ability to write effective SQL queries is crucial for developers and analysts to extract meaningful insights from data, enabling informed decision-making.