RDBMS Fundamentals:

Assignment 1: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

Business Scenario:

An online shopping store sells various products to customers. Each product has a unique product ID, name, description, category, price, and stock quantity. Customers can browse the product catalog, place orders, and leave reviews for the products they purchase. The store needs to keep track of customer information, including their unique customer ID, name, email, phone number, and address. Each order contains one or more products, and each product can be part of multiple orders. The store also keeps track of the date, total amount, and status of each order. Additionally, customers can leave reviews for products they purchase, with each review having a rating, comment, and timestamp.

Entities and Attributes:

1. **Product**

- ProductID (Primary Key)
- Name
- Description
- Category
- Price
- StockQuantity

2. **Customer**

- CustomerID (Primary Key)
- Name
- Email
- PhoneNumber
- Address

3. **Order**

- OrderID (Primary Key)
- CustomerID (Foreign Key)
- OrderDate
- TotalAmount
- Status

4. **OrderDetails** (Associative entity for many-to-many relationship between Order and Product)

- OrderID (Foreign Key, Composite Key)
- ProductID (Foreign Key, Composite Key)
- Quantity

5. **Review**

- ReviewID (Primary Key)
- ProductID (Foreign Key)
- CustomerID (Foreign Key)

- Rating
- Comment
- Timestamp

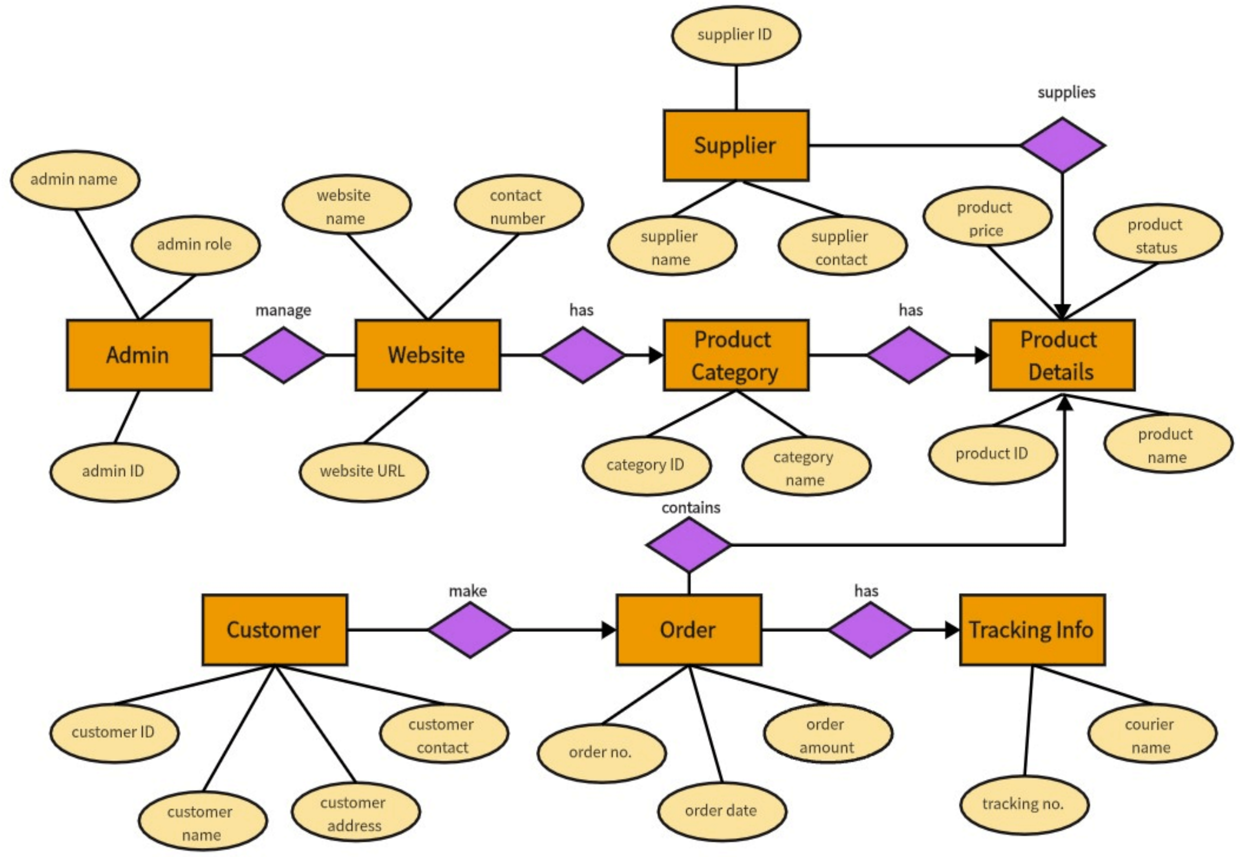
Relationships:

- **Customer-Order**: One-to-Many (One customer can place multiple orders, each order is placed by one customer)
- **Order-OrderDetails**: One-to-Many (One order can contain multiple products)
- **Product-OrderDetails**: One-to-Many (One product can be in multiple order details)
- **Customer-Review**: One-to-Many (One customer can leave multiple reviews)
- **Product-Review**: One-to-Many (One product can have multiple reviews)

Cardinality:

- **Customer-Order**: One customer can place many orders (1)
- **Order-OrderDetails**: One order can have many order details (1)
- **Product-OrderDetails**: One product can appear in many order details (1)
- **Customer-Review**: One customer can leave many reviews (1)
- **Product-Review**: One product can have many reviews (1)

ER Diagram:



ER Diagram for Online Shopping

Description of the ER Diagram:

1. **Customer**
- Primary Key: CustomerID
- Attributes: Name, Email, PhoneNumber, Address

2. **Order**
- Primary Key: OrderID
- Foreign Key: CustomerID
- Attributes: OrderDate, TotalAmount, Status

3. **OrderDetails**
- Composite Primary Key: OrderID, ProductID
- Foreign Keys: OrderID, ProductID
- Attribute: Quantity

4. **Product**
- Primary Key: ProductID
- Attributes: Name, Description, Category, Price, StockQuantity

5. **Review**
- Primary Key: ReviewID
- Foreign Keys: ProductID, CustomerID
- Attributes: Rating, Comment, Timestamp

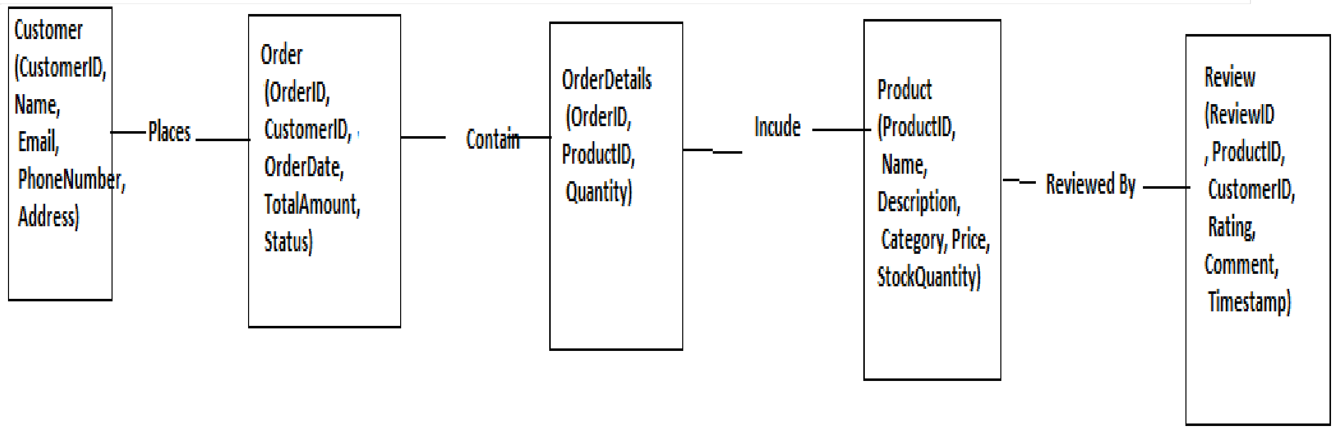
Normalization to Third Normal Form (3NF):

1. **First Normal Form (1NF)**: All attributes are atomic.
2. **Second Normal Form (2NF)**: All non-key attributes are fully functional dependent on the primary key.
3. **Third Normal Form (3NF)**: No transitive dependencies exist between non-key attributes.

The described ER diagram and schema adhere to 3NF by ensuring:
- Each entity has a unique primary key.
- Attributes depend only on the primary key.
- No transitive dependencies between non-key attributes.

Diagram Representation:

To represent this ER diagram visually, you would typically use a tool like draw.io, Lucidchart, or any ER diagram tool. However, given this text-based format, here's how you might describe the relationships:


Customer (CustomerID, Name, Email, PhoneNumber, Address)
Places
Order (OrderID, CustomerID, OrderDate, TotalAmount, Status)
Contain
OrderDetails (OrderID, ProductID, Quantity)
Incude
Product (ProductID, Name, Description, Category, Price, StockQuantity)
Reviewed By
Review (ReviewID, ProductID, CustomerID, Rating, Comment, Timestamp)

Assignment 2: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

To design a database schema for a library system, we need to identify the main entities, their attributes, and the relationships between them. We will also specify constraints such as NOT NULL, UNIQUE, CHECK, primary keys (PK), and foreign keys (FK).

Entities and Attributes:

1. **Book**

- BookID (Primary Key)
- Title
- Author
- Publisher
- YearPublished
- Genre
- ISBN (Unique)
- CopiesAvailable

2. **Member**

- MemberID (Primary Key)
- Name
- Email (Unique)
- PhoneNumber
- Address
- DateOfMembership

3. **Loan**

- LoanID (Primary Key)
- BookID (Foreign Key)
- MemberID (Foreign Key)
- LoanDate
- DueDate
- ReturnDate

4. **Author**

- AuthorID (Primary Key)
- Name
- Biography

5. **BookAuthor** (Associative entity for many-to-many relationship between Book and Author)

- BookID (Foreign Key)
- AuthorID (Foreign Key)

Tables and Fields with Constraints:

Book:

```
CREATE TABLE Book (
BookID INT PRIMARY KEY,
Title VARCHAR(255) NOT NULL,
Publisher VARCHAR(255),
YearPublished YEAR,
Genre VARCHAR(100),
ISBN VARCHAR(13) UNIQUE NOT NULL,
CopiesAvailable INT CHECK (CopiesAvailable >= 0)
);
```

Member:

```
CREATE TABLE Member (
    MemberID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    PhoneNumber VARCHAR(15),
    Address VARCHAR(255),
    DateOfMembership DATE NOT NULL
);
```

Loan:

```
CREATE TABLE Loan (
    LoanID INT PRIMARY KEY,
    BookID INT NOT NULL,
    MemberID INT NOT NULL,
    LoanDate DATE NOT NULL,
    DueDate DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Book(BookID),
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
);
```

Author:

```
CREATE TABLE Author (
    AuthorID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Biography TEXT
);
```

BookAuthor:

```
CREATE TABLE BookAuthor (
    BookID INT NOT NULL,
    AuthorID INT NOT NULL,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Book(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID)
);
```

Relationships:

1. **Book** and **Author**: Many-to-Many

- An associative table (BookAuthor) is created to handle this relationship.

2. **Book** and **Loan**: One-to-Many

- A book can be loaned out multiple times, but each loan is for one book.

3. **Member** and **Loan**: One-to-Many

- A member can borrow multiple books, but each loan record is associated with one member.

Summary of Constraints:

- **NOT NULL**: Ensures that a column cannot have a NULL value.
- **UNIQUE**: Ensures that all values in a column are unique.
- **CHECK**: Ensures that all values in a column satisfy a specific condition.
- **PRIMARY KEY (PK)**: Uniquely identifies each record in a table.
- **FOREIGN KEY (FK)**: Creates a link between two tables, ensuring referential integrity.

This design provides a robust structure for a library system, capturing the necessary data and enforcing constraints to maintain data integrity and consistency.

Assignment 3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

ACID Properties of a Transaction:

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable processing of database transactions.

1. **Atomicity**: This property ensures that a transaction is all-or-nothing. It means that either all operations within the transaction are completed successfully, or none are. If any operation fails, the transaction is rolled back, leaving the database unchanged.

2. **Consistency**: This ensures that a transaction brings the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, and triggers.

3. **Isolation**: This property ensures that the operations of a transaction are isolated from those of other transactions. Intermediate states of a transaction are invisible to other transactions until the transaction is committed. This prevents transactions from interfering with each other.

4. **Durability**: Once a transaction is committed, it remains so, even in the event of a system crash. Changes made by the transaction are permanently stored in the database.

1.

Here's how we can do this using SQL:

```
-- Start the transaction
```

```
BEGIN TRANSACTION;

--- Lock the book row for update
SELECT * FROM Book WHERE BookID = 1 FOR UPDATE;
-- Update the CopiesAvailable count
UPDATE Book
SET CopiesAvailable = CopiesAvailable - 1
WHERE BookID = 1;
-- Insert a new loan record
INSERT INTO Loan (LoanID, BookID, MemberID, LoanDate, DueDate)
VALUES (1, 1, 123, '2024-05-27', '2024-06-27');
-- Commit the transaction
COMMIT;
```

Simulating a Transaction with SQL:

Let's consider a simplified library system where we handle book loans. We'll demonstrate locking and different isolation levels using SQL statements.

Transaction with Locking

1. **Start a transaction**
2. **Update a book's CopiesAvailable** when a book is loaned out
3. **Insert a new loan record**
4. **Commit the transaction**

Demonstrating Isolation Levels:

Isolation levels control the visibility of data changes made by one transaction to other concurrent transactions. Here are the four standard isolation levels with examples of concurrency control:

1. **Read Uncommitted**
2. **Read Committed**
3. **Repeatable Read**
4. **Serializable**
1. **Repeatable Read**
2. **Serializable**

1. Read Uncommitted
Allows transactions to read uncommitted changes made by other transactions. This can lead to dirty reads.

2. Read Committed
Prevents dirty reads by ensuring that a transaction can only read committed changes. This is the default isolation level in many databases.

3. Repeatable Read
Ensures that if a transaction reads a value, subsequent reads within the same transaction will see the same value. This prevents non-repeatable reads but can allow phantom reads.

4. Serializable
The highest isolation level that ensures complete isolation. Transactions are executed sequentially, effectively serializing access to the data.

Summary

- **Atomicity**: Ensures all or nothing execution of transactions.
- **Consistency**: Maintains database integrity.
- **Isolation**: Prevents transactions from interfering with each other.
- **Durability**: Ensures permanence of committed transactions.

By setting different isolation levels, you can control the concurrency behavior of transactions to balance between performance and consistency.

Assignment 4: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

write SQL statements to create a new database and tables that reflect the library schema. We will include constraints such as NOT NULL, UNIQUE, CHECK, and primary and foreign keys. After creating the initial schema, we will use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

Step 1: Create the Database
Step 2: Create Tables
Book Table
Member Table
Loan Table
Author Table
BookAuthor Table (Associative Table for Many-to-Many Relationship)

Step 3: Use ALTER Statements to Modify the Table Structures

Add a New Column to the Book Table

Add a Constraint to the Member Table
Add a New Column to the Loan Table
Step 4: Use DROP Statements to Remove a Redundant Table

Drop the BookAuthor Table (Assuming it's redundant for this example)

Complete SQL Script :

```
-- Create the database
CREATE DATABASE LibrarySystem;
USE LibrarySystem;

-- Create tables
CREATE TABLE Book (
   BookID INT PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   Publisher VARCHAR(255),
   YearPublished YEAR,
   Genre VARCHAR(100),
   ISBN VARCHAR(13) UNIQUE NOT NULL,
   CopiesAvailable INT CHECK (CopiesAvailable >= 0)
);

CREATE TABLE Member (
   MemberID INT PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Email VARCHAR(255) UNIQUE NOT NULL,
   PhoneNumber VARCHAR(15),
   Address VARCHAR(255),
   DateOfMembership DATE NOT NULL
);

CREATE TABLE Loan (
   LoanID INT PRIMARY KEY,
   BookID INT NOT NULL,
   MemberID INT NOT NULL,
   LoanDate DATE NOT NULL,
   DueDate DATE NOT NULL,
   ReturnDate DATE,
   FOREIGN KEY (BookID) REFERENCES Book(BookID),
   FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
);

CREATE TABLE Author (
   AuthorID INT PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Biography TEXT
);

CREATE TABLE BookAuthor (
   BookID INT NOT NULL,
   AuthorID INT NOT NULL,
   PRIMARY KEY (BookID, AuthorID),
   FOREIGN KEY (BookID) REFERENCES Book(BookID),
   FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID)
);

-- Alter tables
ALTER TABLE Book ADD COLUMN Edition VARCHAR(50);
ALTER TABLE Member ADD CONSTRAINT CHK_Member_Email CHECK (Email LIKE '%_@__%.__%');
ALTER TABLE Loan ADD COLUMN Fine DECIMAL(5, 2) DEFAULT 0.00;
-- Drop redundant table
DROP TABLE BookAuthor;
```

Conclusion:

This SQL script creates a library system database with the necessary tables, applies some modifications using ALTER statements, and removes a redundant table using a DROP statement.

Assignment 5: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

Creating and Using an Index in SQL:

Indexes are used to speed up the retrieval of data from a database table. They improve query performance by allowing the database engine to find rows more quickly and efficiently. However, they also come with some overhead in terms of storage and maintenance during data modifications (inserts, updates, and deletes).

Step 1: Create the Index

Let's create an index on the **ISBN** column of the **Book** table, which is commonly used for searching.

```
-- Create an index on the ISBN column
CREATE INDEX idx_isbn ON Book(ISBN);
```

Step 2: Analyze Query Performance with the Index

Consider the following query that searches for a book by its ISBN:

```
-- Query to find a book by its ISBN
SELECT * FROM Book WHERE ISBN = '978-3-16-148410-0';
```

Step 3: Drop the Index

To understand the impact of the index on query performance, let's drop the index and run the same query again.

```
-- Drop the index on the ISBN column
DROP INDEX idx_isbn ON Book;
```

Step 4: Analyze Query Performance without the Index

Run the same query again:

```
-- Query to find a book by its ISBN (without the index)
SELECT * FROM Book WHERE ISBN = '978-3-16-148410-0';
```

Performance Analysis

1. **With Index**: The query optimizer uses the index **idx_isbn** to quickly locate the record. The query execution time is significantly reduced because it doesn't have to perform a full table scan.

2. **Without Index**: The database engine must perform a full table scan to locate the record, checking each row's **ISBN** value. This increases the query execution time, especially if the **Book** table contains a large number of rows.

3. Example SQL Script

Here is a complete SQL script demonstrating the creation, usage, and removal of an index, along with performance analysis comments:

```
-- Create the database and use it
CREATE DATABASE LibrarySystem;
USE LibrarySystem;

-- Create the Book table
CREATE TABLE Book (
   BookID INT PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   Publisher VARCHAR(255),
   YearPublished YEAR,
   Genre VARCHAR(100),
   ISBN VARCHAR(13) UNIQUE NOT NULL,
   CopiesAvailable INT CHECK (CopiesAvailable >= 0)
);

-- Insert sample data
INSERT INTO Book (BookID, Title, Publisher, YearPublished, Genre, ISBN, CopiesAvailable)
VALUES
(1, 'Book One', 'Publisher A', 2020, 'Fiction', '978-3-16-148410-0', 5),
(2, 'Book Two', 'Publisher B', 2019, 'Non-Fiction', '978-1-4028-9462-6', 3);

-- Create an index on the ISBN column
CREATE INDEX idx_isbn ON Book(ISBN);

-- Analyze query performance with the index
-- Execution time should be fast
SELECT * FROM Book WHERE ISBN = '978-3-16-148410-0';

-- Drop the index
DROP INDEX idx_isbn ON Book;

-- Analyze query performance without the index
-- Execution time should be slower due to full table scan
SELECT * FROM Book WHERE ISBN = '978-3-16-148410-0';
```

Conclusion:

Indexes significantly improve query performance by reducing the amount of data the database engine needs to scan. However, they also require additional storage and can slow down write operations. It's essential to balance the use of indexes based on the specific read and write patterns of your database workload.

Assignment 6: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

Step 1: Create the Database User

We'll create a user named **library_user** with a password.

```
-- Create a new user
CREATE USER 'library_user'@'localhost' IDENTIFIED BY 'securepassword';
```

Step 2: Grant Privileges

We will grant **library_user** the following privileges:

- SELECT: To read data from the tables.
- INSERT: To insert new records into the tables.
- UPDATE: To update existing records in the tables.
- DELETE: To delete records from the tables.

```
-- Grant specific privileges to the new user
GRANT SELECT, INSERT, UPDATE, DELETE ON LibrarySystem.* TO 'library_user'@'localhost';
```

Step 3: Revoke Privileges

Assume we want to revoke the **UPDATE** and **DELETE** privileges from **library_user**.

```
-- Revoke specific privileges from the user
REVOKE UPDATE, DELETE ON LibrarySystem.* FROM 'library_user'@'localhost';
```

Step 4: Drop the User

Finally, we drop the user from the database.

```
-- Drop the user
DROP USER 'library_user'@'localhost';
```

Complete SQL Script

Here is the complete SQL script for creating a user, granting privileges, revoking privileges, and dropping the user:

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

-- Use the database
USE LibrarySystem;

-- Create the Book table for demonstration purposes
CREATE TABLE Book (
   BookID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Publisher VARCHAR(255),
    YearPublished YEAR,
    Genre VARCHAR(100),
    ISBN VARCHAR(13) UNIQUE NOT NULL,
    CopiesAvailable INT CHECK (CopiesAvailable >= 0)
);

-- Create a new user
CREATE USER 'library_user'@'localhost' IDENTIFIED BY 'securepassword';

-- Grant specific privileges to the new user
GRANT SELECT, INSERT, UPDATE, DELETE ON LibrarySystem.* TO 'library_user'@'localhost';

-- Revoke specific privileges from the user
REVOKE UPDATE, DELETE ON LibrarySystem.* FROM 'library_user'@'localhost';

-- Drop the user
DROP USER 'library_user'@'localhost';
```

Explanation of Commands:

CREATE USER: This command creates a new user in the MySQL database server with a specified username and password.
CREATE USER 'library_user'@'localhost' IDENTIFIED BY 'securepassword';

2 . GRANT: This command grants specified privileges to the user for the database.
GRANT SELECT, INSERT, UPDATE, DELETE ON LibrarySystem.* TO 'library_user'@'localhost';

REVOKE: This command revokes specified privileges from the user.
REVOKE UPDATE, DELETE ON LibrarySystem.* FROM 'library_user'@'localhost';

DROP USER: This command removes the user from the database server.
DROP USER 'library_user'@'localhost';

Assignment 7: Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.

Step 1: Insert New Records

Insert New Books

```
-- Insert new records into the Book table

INSERT INTO Book (BookID, Title, Publisher, YearPublished, Genre, ISBN, CopiesAvailable)

VALUES

(1, 'The Great Gatsby', 'Scribner', 1925, 'Fiction', '978-0743273565', 3),

(2, '1984', 'Secker & Warburg', 1949, 'Dystopian', '978-0451524935', 5),

(3, 'To Kill a Mockingbird', 'J.B. Lippincott & Co.', 1960, 'Fiction', '978-0061120084', 4);
```

Insert New Members

```
-- Insert new records into the Member table
INSERT INTO Member (MemberID, Name, Email, PhoneNumber, Address, DateOfMembership)
VALUES
(1, 'John Doe', 'johndoe@example.com', '555-1234', '123 Elm St', '2024-01-15'),
(2, 'Jane Smith', 'janesmith@example.com', '555-5678', '456 Oak St', '2024-02-20');
```

Step 2: Update Existing Records

Update Book Information

```
-- Update the CopiesAvailable for a specific book
UPDATE Book
SET CopiesAvailable = 2
WHERE BookID = 1;
```

Update Member Information

```
-- Update the email address of a member
UPDATE Member
SET Email = 'john.d.newemail@example.com'
WHERE MemberID = 1;
```

Step 3: Delete Records

Delete a Book Record

```
-- Delete a book record based on specific criteria
DELETE FROM Book
WHERE YearPublished < 1950;
```

Delete a Member Record

```
-- Delete a member record based on specific criteria
DELETE FROM Member
WHERE DateOfMembership < '2024-01-01';
```

Step 4: Bulk Insert Operation

Assuming we have an external CSV file named **books.csv** with the following content:

```
-- Bulk insert operation to load data from an external CSV file
LOAD DATA INFILE '/path/to/books.csv'
INTO TABLE Book
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(BookID, Title, Publisher, YearPublished, Genre, ISBN, CopiesAvailable);
```

Complete SQL Script

Here is the complete SQL script for inserting, updating, deleting, and performing a bulk insert:

```
-- Insert new records into the Book table
INSERT INTO Book (BookID, Title, Publisher, YearPublished, Genre, ISBN, CopiesAvailable)
VALUES
(1, 'The Great Gatsby', 'Scribner', 1925, 'Fiction', '978-0743273565', 3),
(2, '1984', 'Secker & Warburg', 1949, 'Dystopian', '978-0451524935', 5),
(3, 'To Kill a Mockingbird', 'J.B. Lippincott & Co.', 1960, 'Fiction', '978-0061120084', 4);

-- Insert new records into the Member table
INSERT INTO Member (MemberID, Name, Email, PhoneNumber, Address, DateOfMembership)
VALUES
(1, 'John Doe', 'johndoe@example.com', '555-1234', '123 Elm St', '2024-01-15'),
(2, 'Jane Smith', 'janesmith@example.com', '555-5678', '456 Oak St', '2024-02-20');

-- Insert new records into the Loan table
INSERT INTO Loan (LoanID, BookID, MemberID, LoanDate, DueDate, ReturnDate)
VALUES
(1, 1, 1, '2024-05-01', '2024-06-01', NULL),
(2, 2, 2, '2024-05-10', '2024-06-10', NULL);

-- Update the CopiesAvailable for a specific book
UPDATE Book
SET CopiesAvailable = 2
WHERE BookID = 1;

-- Update the email address of a member
UPDATE Member
SET Email = 'john.d.newemail@example.com'
WHERE MemberID = 1;

-- Delete a book record based on specific criteria
```

```
DELETE FROM Book
WHERE YearPublished < 1950;

-- Delete a member record based on specific criteria
DELETE FROM Member
WHERE DateOfMembership < '2024-01-01';

-- Bulk insert operation to load data from an external CSV file
LOAD DATA INFILE '/path/to/books.csv'
INTO TABLE Book
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(BookID, Title, Publisher, YearPublished, Genre, ISBN, CopiesAvailable);
```

Explanation of the Bulk Insert Operation

- **LOAD DATA INFILE**: This command loads data from a file into a table.
- **FIELDS TERMINATED BY ','**: Specifies that fields in the CSV file are separated by commas.
- **ENCLOSED BY '"'**: Indicates that fields are enclosed in double quotes.
- **LINES TERMINATED BY '\n'**: Specifies that each line in the file corresponds to a row in the table.
- **IGNORE 1 ROWS**: Skips the header row in the CSV file.