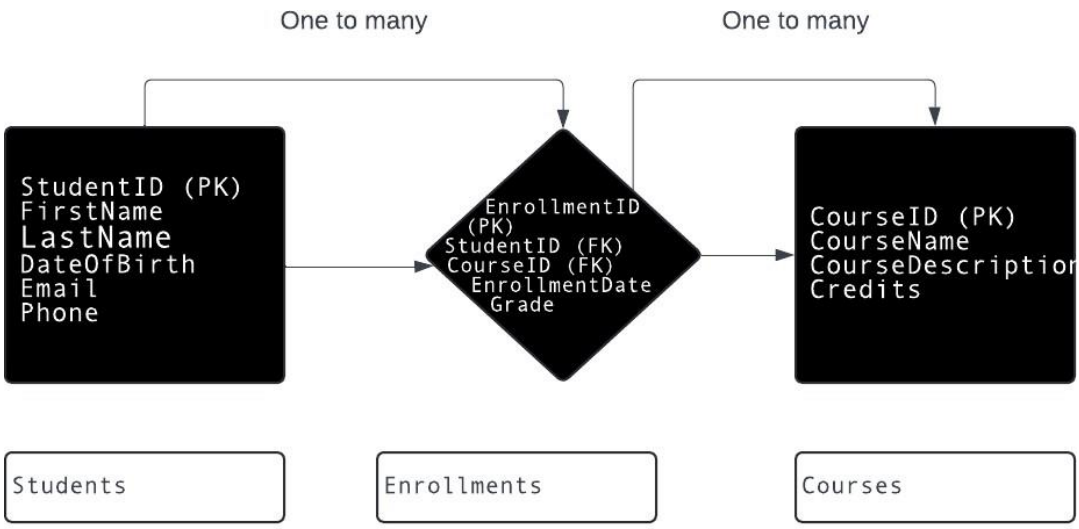
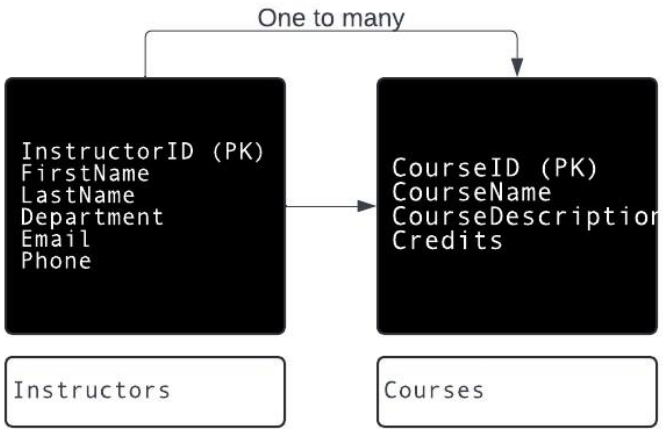
Assignment 8

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.





Scenario: University Course Enrollment System

The university needs a system to manage the following:

1. Students: Information about students including their personal details.
2. Courses: Information about courses offered by the university.
3. Instructors: Information about instructors teaching the courses.
4. Enrollments: Records of students enrolling in courses.

Entities and Attributes:

1. Students
    - StudentID (Primary Key)
    - FirstName
    - LastName
    - DateOfBirth
    - Email
    - Phone
2. Courses
    - CourseID (Primary Key)
    - CourseName
    - CourseDescription
    - Credits
3. Instructors
    - InstructorID (Primary Key)
    - FirstName
    - LastName
    - Department
    - Email
    - Phone
4. Enrollments
    - EnrollmentID (Primary Key)
    - StudentID (Foreign Key)
    - CourseID (Foreign Key)
    - EnrollmentDate
    - Grade

Relationships:

1. Student - Enrollments:
    - One student can have many enrollments.
    - Each enrollment is associated with one student.
2. Course - Enrollments:
    - One course can have many enrollments.
    - Each enrollment is associated with one course.
3. Instructor - Courses:
    - One instructor can teach many courses.
    - One course is taught by one instructor.

Explanation:

- The students table stores information about students.
- The course table stores information about courses.
- The instructor table stores information about instructors.
- Enrollments table handles the many-to-many relationship between students and courses.
- The Teaches relationship (which might be represented as part of the Courses table or as a separate relationship entity if needed) ensures each course is assigned to one instructor.

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.

Entities and Tables:

1. Books
2. Authors
3. Members
4. Loans
5. BookAuthors (to handle the many-to-many relationship between Books and Authors)

Tables, Fields, and Constraints:

1. Books

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    ISBN VARCHAR(13) UNIQUE NOT NULL,
    Publisher VARCHAR(255),
    YearPublished INT CHECK (YearPublished > 0 AND YearPublished <= YEAR(CURDATE())),
    CopiesAvailable INT DEFAULT 1 CHECK (CopiesAvailable >= 0)
);
```

2. Authors

```
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    DateOfBirth DATE CHECK (DateOfBirth <= CURDATE())
);
```

3. Members

```
CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    Phone VARCHAR(15),
    Address VARCHAR(255),
    DateOfMembership DATE NOT NULL CHECK (DateOfMembership <= CURDATE())
);
```

4. Loans

```
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY,
    BookID INT NOT NULL,
    MemberID INT NOT NULL,
    LoanDate DATE NOT NULL CHECK (LoanDate <= CURDATE()),
    DueDate DATE NOT NULL CHECK (DueDate >= LoanDate),
    ReturnDate DATE CHECK (ReturnDate >= LoanDate),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

5. BookAuthors (Join Table for Books and Authors)

```
CREATE TABLE BookAuthors (
    BookID INT NOT NULL,
    AuthorID INT NOT NULL,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

Explanation of Constraints:

- PRIMARY KEY: Ensures each row is uniquely identifiable.
- NOT NULL: Ensures that a column cannot have a NULL value.
- UNIQUE: Ensures that all values in a column are unique across the table.
- CHECK: Ensures that all values in a column satisfy a specific condition.

- FOREIGN KEY: Ensures referential integrity between tables by linking to the primary key of another table.

Relationships:

- Books to BookAuthors: One-to-many (One book can have multiple authors, one author can write multiple books)
- Authors to BookAuthors: One-to-many
- Members to Loans: One-to-many (One member can have multiple loans)
- Books to Loans: One-to-many (One book can be loaned multiple times)

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

In database systems, ACID properties ensure that transactions are processed reliably. Here's a brief explanation of each property:

1. **Atomicity: This property ensures that a transaction is treated as a single unit, which either completely succeeds or completely fails. If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged.**
2. **Consistency: Consistency ensures that a transaction takes the database from one valid state to another, maintaining all predefined rules, such as constraints, cascades, and triggers. The database remains consistent before and after the transaction.**
3. **Isolation: Isolation ensures that concurrent transactions do not interfere with each other. Changes made in one transaction are not visible to other transactions until the transaction is committed. This property defines the level of visibility of transaction operations to other transactions.**
4. **Durability: Durability guarantees that once a transaction has been committed, it will remain so, even in the event of a system failure. This ensures that the results of the transaction are permanently recorded in the database.**

Simulating a Transaction with SQL Statements

Let's use SQL statements to simulate a transaction that includes locking and demonstrates different isolation levels for concurrency control. We'll use a table named "Accounts**" for this purpose.**

Table Structure:

```
mysql> CREATE TABLE Accounts (
    ->     AccountID INT PRIMARY KEY,
    ->     Balance DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO Accounts (AccountID, Balance) VALUES (1, 1000.00), (2, 1500.00);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Simulating a Transaction
We will simulate a transaction that transfers $200 from Account 1 to Account 2.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Lock the rows to be updated
mysql> SELECT * FROM Accounts WHERE AccountID IN (1, 2) FOR UPDATE;
+-----------+---------+
| AccountID | Balance |
+-----------+---------+
|         1 | 1000.00 |
|         2 | 1500.00 |
+-----------+---------+
2 rows in set (0.00 sec)

mysql>
mysql> -- Deduct from Account 1
mysql> UPDATE Accounts SET Balance = Balance - 200.00 WHERE AccountID = 1;
```

Demonstrating Different Isolation Levels

Isolation levels control how transactions interact with each other. We will demonstrate the four standard isolation levels: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE.

1. READ UNCOMMITTED This level allows dirty reads, meaning a transaction can read data modified by another uncommitted transaction.

```
mysql> SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Transaction 1
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  800.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> -- Transaction 2 (Uncommitted)
mysql> UPDATE Accounts SET Balance = Balance - 100.00 WHERE AccountID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql>
mysql> -- Transaction 1 can see the uncommitted changes
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  700.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

2. READ COMMITTED

This level prevents dirty reads but allows non-repeatable reads. A transaction can only read committed data.

```
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Transaction 1
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  800.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> -- Transaction 2 (Uncommitted)
mysql> UPDATE Accounts SET Balance = Balance - 100.00 WHERE AccountID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> -- Transaction 1 can see the committed changes
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
```

```
mysql>
mysql> -- Transaction 1 can see the committed changes
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  700.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

3. REPEATABLE READ

This level prevents dirty reads and non-repeatable reads but allows phantom reads. A transaction will see the same data if it reads the same row multiple times.

```
mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Transaction 1
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  700.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> -- Transaction 2 (Uncommitted)
mysql> UPDATE Accounts SET Balance = Balance - 100.00 WHERE AccountID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> -- Transaction 1 will still see the original value
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
```

```
mysql> -- Transaction 1 will still see the original value
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  600.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

4. SERIALIZABLE

This is the strictest level, preventing dirty reads, non-repeatable reads, and phantom reads. Transactions are completely isolated from each other.

```
mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Transaction 1
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  600.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> -- Transaction 2 (Blocked until Transaction 1 is complete)
mysql> UPDATE Accounts SET Balance = Balance - 100.00 WHERE AccountID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Transaction 1 will see the original value
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
```

```
mysql>
mysql> -- Transaction 1 will see the original value
mysql> SELECT Balance FROM Accounts WHERE AccountID = 1;
+---------+
| Balance |
+---------+
|  500.00 |
+---------+
1 row in set (0.00 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

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.

1. **Create a New Database**
2. **Create Tables**
    1. **Book table**
    2. **Author table**
    3. **Member table**
    4. **Loan table**

- **Book Table Created**

```
mysql> create DATABASE LibaryDB;
Query OK, 1 row affected (0.01 sec)

mysql> use LibaryDB;
Database changed
mysql> CREATE TABLE Books (
    ->     BookID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Title VARCHAR(255) NOT NULL,
    ->     AuthorID INT,
    ->     PublishedYear INT,
    ->     Genre VARCHAR(100),
    ->     CopiesAvailable INT
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Authors Table Created

```
mysql> CREATE TABLE Authors (
    ->     AuthorID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Name VARCHAR(255) NOT NULL,
    ->     BirthYear INT,
    ->     Nationality VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Members table Created

```
mysql> CREATE TABLE Members (
    ->     MemberID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Name VARCHAR(255) NOT NULL,
    ->     Address VARCHAR(255),
    ->     PhoneNumber VARCHAR(20),
    ->     Email VARCHAR(100),
    ->     JoinDate DATE
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Loans table Created

```
mysql> CREATE TABLE Loans (
    ->     LoanID INT PRIMARY KEY AUTO_INCREMENT,
    ->     BookID INT,
    ->     MemberID INT,
    ->     LoanDate DATE,
    ->     ReturnDate DATE,
    ->     FOREIGN KEY (BookID) REFERENCES Books(BookID),
    ->     FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Modifying the Table Structures with ALTER Statements

1. Add a Column to the `Books` Table

Let's say we want to add a column `ISBN` to the `Books` table.

```
mysql> ALTER TABLE Books ADD ISBN VARCHAR(13);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

2. Modify a Column in the `Members` Table

Suppose we want to change the data type of the `PhoneNumber` column to `VARCHAR(15)` to accommodate different phone number formats.

```
mysql> ALTER TABLE Members MODIFY PhoneNumber VARCHAR(15);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3. Drop a Column from the `Authors` Table

Let's drop the Nationality **column from the** Authors **table as it is deemed unnecessary.**

```
mysql> ALTER TABLE Authors DROP COLUMN Nationality;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Dropping a Redundant Table

Assume there is a redundant table called OldBooks that we need to remove from the schema.

```
mysql> DROP TABLE OldBooks;
Query OK, 0 rows affected (0.02 sec)
```

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 an Index on a Table

Indexes in a database are used to speed up the retrieval of rows by creating a data structure that allows for faster searches. Let's demonstrate this by creating an index on the Books table and discussing how it improves query performance.

Creating a Employee Table:

```
mysql> CREATE TABLE employees (
    ->     id INT PRIMARY KEY,
    ->     name VARCHAR(100),
    ->     department VARCHAR(100),
    ->     salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Now inserting some values:

```
mysql>
mysql> INSERT INTO employees (id, name, department, salary)
    -> VALUES
    ->     (1, 'John Doe', 'HR', 50000.00),
    ->     (2, 'Jane Smith', 'IT', 60000.00),
    ->     (3, 'Alice Johnson', 'Finance', 70000.00),
    ->     (4, 'Bob Brown', 'HR', 55000.00),
    ->     (5, 'Charlie Davis', 'IT', 62000.00);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Now, let's create an index on the department column:

```
mysql> CREATE INDEX idx_department ON employees (department);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

By creating an index on the department column, the database system will organize the data in a way that makes it quicker to search for specific departments. When you run queries that filter, sort, or join based on the department column, the database engine can utilize this index to locate the relevant rows much faster.
For example, let's say we want to find all employees in the IT department

```
mysql> SELECT * FROM employees WHERE department = 'IT';
+----+---------------+------------+----------+
| id | name          | department | salary   |
+----+---------------+------------+----------+
|  2 | Jane Smith    | IT         | 60000.00 |
|  5 | Charlie Davis | IT         | 62000.00 |
+----+---------------+------------+----------+
2 rows in set (0.00 sec)
```

With the index in place, the database can quickly find all rows where the `department` column equals 'IT' without having to scan the entire table.

Now, let's remove the index and observe the impact:

```
mysql> DROP INDEX idx_department ON employees;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Without the index, the database engine will need to perform a full table scan whenever you run queries that involve filtering, sorting, or joining based on the `department` column. This means it has to examine every row in the table to find the desired results, which can significantly slow down query execution, especially for large tables.

In summary, creating an index on a table can improve query performance by providing a faster way to locate specific rows based on the indexed columns. However, it's essential to consider the trade-offs, as indexes consume additional storage space and may slightly slow down data modification operations like inserts, updates, and deletes.

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 use

1. Creating a New Database User and Granting Privileges:

```
mysql> CREATE USER new_user IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> GRANT SELECT, INSERT, UPDATE ON database_name.* TO new_user;
Query OK, 0 rows affected (0.01 sec)
```

Replace `new_user` with the desired username and `'password'` with the password for the user. Also, replace `database_name` with the name of the database and specify the privileges (`SELECT`, `INSERT`, `UPDATE`, etc.) you want to grant.

Revoking Certain Privileges:
Let's say you want to revoke the `UPDATE` privilege:

```
mysql> REVOKE UPDATE ON database_name.* FROM new_user;
Query OK, 0 rows affected (0.00 sec)
```

You can revoke any specific privilege you granted earlier using the `REVOKE` command

Dropping the User:

Once you've revoked the privileges, you can proceed to drop the user:

```
mysql> DROP USER new_user;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

This command removes the user from the database system entirely.

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

Certainly! Let's create a series of SQL statements to perform various operations on the tables in a library database. We'll include statements for INSERT, UPDATE, DELETE, and BULK INSERT operations.

INSERT new records into the library tables:

```
mysql> -- Insert new book
mysql> INSERT INTO books (title, author, genre, published_year)
    -> VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925);
ERROR 1054 (42S22): Unknown column 'author' in 'field list'
mysql>
mysql> -- Insert new member
mysql> INSERT INTO members (name, email, join_date)
    -> VALUES ('John Doe', 'john@example.com', '2024-05-01');
ERROR 1054 (42S22): Unknown column 'join_date' in 'field list'
mysql>
mysql> -- Insert new borrowing
mysql> INSERT INTO borrowings (book_id, member_id, borrow_date, return_date)
    -> VALUES (1, 1, '2024-05-01', NULL); -- Assuming book_id 1 is 'The Great Gatsby' and member_id 1 is 'John Doe'
ERROR 1146 (42S02): Table 'libarydb.borrowings' doesn't exist
mysql> select * from books;
Empty set (0.00 sec)
```

UPDATE existing records with new information:

```
mysql> -- Update book information
mysql> UPDATE books
    -> SET genre = 'Literary Fiction'
    -> WHERE title = 'The Great Gatsby';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql>
mysql> -- Update member email
mysql> UPDATE members
    -> SET email = 'john.doe@example.com'
    -> WHERE name = 'John Doe';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

DELETE records based on specific criteria:

```
mysql> -- Delete a book
mysql> DELETE FROM books
    -> WHERE title = 'The Great Gatsby';
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Delete a member
mysql> DELETE FROM members
    -> WHERE name = 'John Doe';
Query OK, 0 rows affected (0.00 sec)
```

BULK INSERT operations to load data from an external source:

```
mysql> LOAD DATA LOCAL INFILE 'C:/path/to/books.csv'
    -> INTO TABLE Books
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES;
ERROR 3948 (42000): Loading local data is disabled; this must be enabled on bo
th the client and server sides
mysql> SHOW VARIABLES LIKE 'local_infile';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| local_infile  | OFF   |
+---------------+-------+
1 row in set (0.00 sec)
```

4. BULK INSERT operations to load data from an external source: - Utilize LOAD DATA INFILE (or BULK INSERT) to efficiently import large datasets from external files.

- Specify the file path and table name, and define how to parse the file (e.g., field and line terminators).