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.

Create an ER diagram for a bookstore. Here's a simplified scenario:

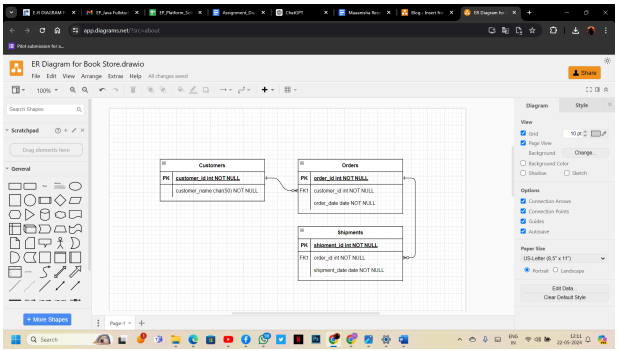
1. Entities:
- **Book**
- **Author**
- **Customer**
- **Order**
- **Publisher**

2. Relationships:
- **A Book is written by one or more Authors.**
- **An Author can write one or more Books.**
- **A Customer can place zero or more Orders.**
- **An Order can contain one or more Books.**
- **A Book is published by one Publisher.**
- **A Publisher can publish multiple Books.**

3. Attributes:
- **Book: ISBN (Primary key), Title, Genre, Price, Publication Year, Publisher ID (Foreign key), etc.**
- **Author: Author ID (Primary key), Name, Biography, etc.**
- **Customer: Customer ID (Primary key), Name, Email, Address, Phone, etc.**
- **Order: Order ID (Primary key), Customer ID (Foreign key), Order Date, Total Amount, etc.**
- **Publisher: Publisher ID (Primary key), Name, Address, Phone, Email, etc.**

4. Cardinality:
- **Book - Author: Many-to-Many (M:N)**
- **Author - Book: Many-to-Many (M:N)**
- **Customer - Order: One-to-Many (1:M)**
- **Order - Book: Many-to-Many (M:N)**
- **Book - Publisher: Many-to-One (M:1)**
- **Publisher - Book: One-to-Many (1:M)**

ER diagram for a bookstore 👍



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.

Creating Tables

1. Database Selection:

- Switch to `db man1` database.

2. Authors Table:

- `author_id`: Auto-increment primary key.
- `first_name` and `last_name`: Required text fields (max 50 characters).

3. Categories Table:

- `category_id`: Auto-increment primary key.
- `category_name`: Required, unique text field (max 50 characters).

4. Publishers Table:

- `publisher_id`: Auto-increment primary key.
- `name`: Required, unique text field (max 100 characters).

Creating More Tables and Triggers

5. Members Table:

- `member_id`: Auto-increment primary key.
- `first_name` and `last_name`: Required text fields (max 50 characters).
- `email`: Required, unique text field (max 100 characters).
- `phone_number`: Optional text field (max 15 characters).
- `address`: Optional text field (max 255 characters).
- `membership_date`: Required date field.

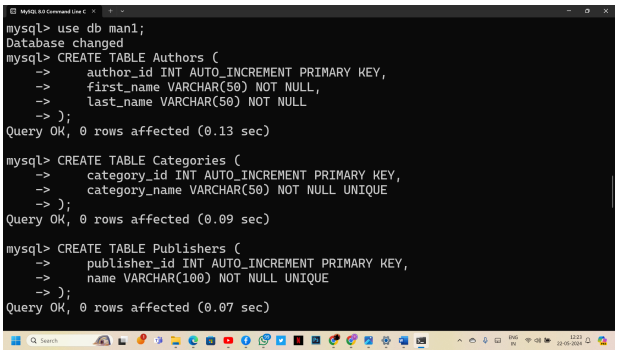
6. Loans Table:

- Attempted but failed due to constraint issues.
- `loan_id`: Auto-increment primary key.
- `book_id` and `member_id`: Required fields.
- `loan_date`, `due_date`, `return_date`: Date fields with constraints.
- Foreign key: `member_id` references `Members`.

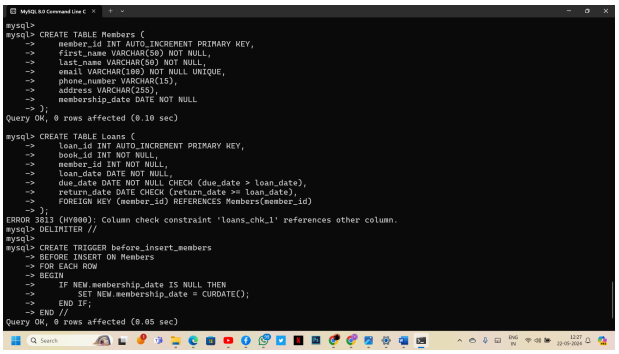
7. Trigger for Members:

- Automatically sets `membership_date` to today's date if not provided.

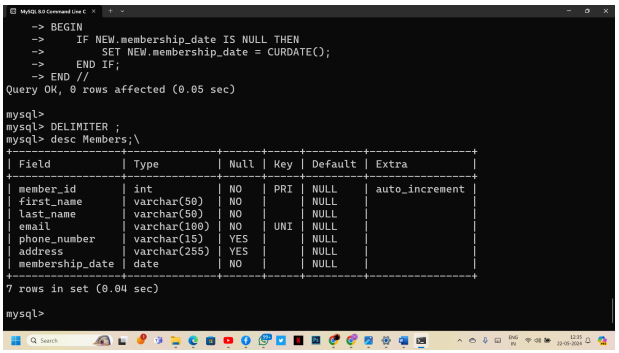
Output for the Library system:



```
mysql> use db man1;
Database changed
mysql> CREATE TABLE Authors (
    ->     author_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     first_name VARCHAR(50) NOT NULL,
    ->     last_name VARCHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE Categories (
    ->     category_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     category_name VARCHAR(50) NOT NULL UNIQUE
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> CREATE TABLE Publishers (
    ->     publisher_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(100) NOT NULL UNIQUE
    -> );
Query OK, 0 rows affected (0.07 sec)
```



```
mysql>
mysql> CREATE TABLE Members (
    ->     member_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     first_name VARCHAR(50) NOT NULL,
    ->     last_name VARCHAR(50) NOT NULL,
    ->     email VARCHAR(100) NOT NULL UNIQUE,
    ->     phone_number VARCHAR(15),
    ->     address VARCHAR(255),
    ->     membership_date DATE NOT NULL
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> CREATE TABLE Loans (
    ->     loan_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     book_id INT NOT NULL,
    ->     member_id INT NOT NULL,
    ->     loan_date DATE NOT NULL,
    ->     due_date DATE NOT NULL CHECK (due_date > loan_date),
    ->     return_date DATE CHECK (return_date >= loan_date),
    ->     FOREIGN KEY (member_id) REFERENCES Members(member_id)
    -> );
ERROR 3813 (HY000): Column check constraint 'loans_chk_1' references other column.
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER before_insert_members
    -> BEFORE INSERT ON Members
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.membership_date IS NULL THEN
    ->         SET NEW.membership_date = CURDATE();
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.05 sec)
```



```
    -> BEGIN
    ->     IF NEW.membership_date IS NULL THEN
    ->         SET NEW.membership_date = CURDATE();
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> DELIMITER ;
mysql> desc Members;\
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| member_id       | int          | NO   | PRI | NULL    | auto_increment |
| first_name      | varchar(50)  | NO   |     | NULL    |                |
| last_name       | varchar(50)  | NO   |     | NULL    |                |
| email           | varchar(100) | NO   | UNI | NULL    |                |
| phone_number    | varchar(15)  | YES  |     | NULL    |                |
| address         | varchar(255) | YES  |     | NULL    |                |
| membership_date | date         | NO   |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
7 rows in set (0.04 sec)

mysql>
```

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 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.

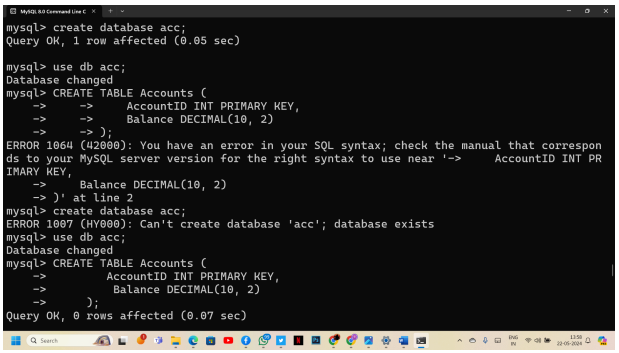
2. Consistency: Consistency ensures that a transaction takes the database from one valid state to another, maintaining all predefined rules, such as constraints, cascades, and triggers. The database remains consistent before and after the transaction.

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.

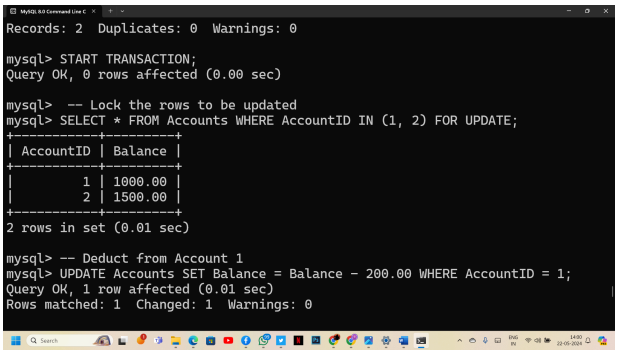
4. Durability: Durability guarantees that once a transaction has been committed, it will remain so, even in the event of a system failure. This ensures that the results of the transaction are permanently recorded in the database

Output 👍

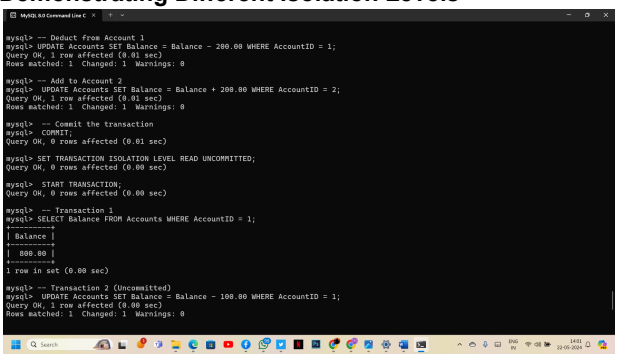
Simulating a Transaction with SQL Statements



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



Demonstrating Different Isolation Levels



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.

MySQL command-line interface with operations related to creating a database and tables for a library system. Here's a summary of the actions taken:

1. **Creating the Library Database:**
   - The user successfully creates a database named "Library."

2. **Creating the Books Table:**
   - The user attempts to create a "Books" table with columns for book ID, title, author ID, genre, publication year, and ISBN.
   - A foreign key constraint is included to reference the "Authors" table, but it fails because the "Authors" table does not exist yet.

3. **Creating the Authors Table:**
   - The user successfully creates an "Authors" table with columns for author ID, author name, nationality, and birth year.

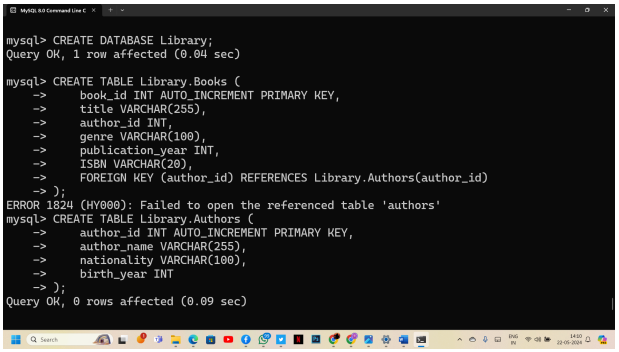
Creating and modifying tables. Here's a summary of the actions taken:

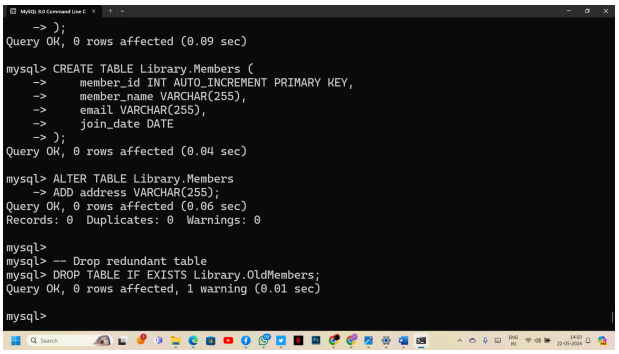
1. Creating the Members Table:
   - The user successfully creates a "Members" table with columns for member ID, member name, email, and join date.

2. Altering the Members Table:
   - The user successfully adds an "address" column to the "Members" table.

3. Dropping a Redundant Table:
   - The user successfully drops an "OldMembers" table if it exists, indicating it is no longer needed.

Outputs are as Follows 👍





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 an Index on a Table and Its Impact on Query Performance

Step 1: Creating the `employees` Table
First, a table named `employees` is created with columns for `id`, `name`, `department`, and `salary`.

Step 2: Inserting Data into the `employees` Table
Several records are inserted into the `employees` table, including employees from different departments with varying salaries.

Step 3: Creating an Index on the `department` Column
An index named `idx_department` is created on the `department` column. This index improves query performance by allowing the database to quickly find all rows where the `department` column matches a specific value, without scanning the entire table.

Step 4: Querying with the Index
With the index in place, queries that filter by the `department` column (e.g., finding all employees in the 'IT' department) are executed much faster because the database uses the index to locate the relevant rows quickly.

Step 5: Dropping the Index
The index on the `department` column is then removed using a `DROP INDEX` statement. Without the index, the database engine must perform a full table scan for queries involving the `department` column, which means examining every row.

Summary

Indexes are powerful tools for improving query performance by allowing the database to find rows more efficiently. However, they should be used judiciously, as they also require additional storage space and can impact the performance of write operations like inserts, updates, and deletes.

Here Output as Follows 👍

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)
```

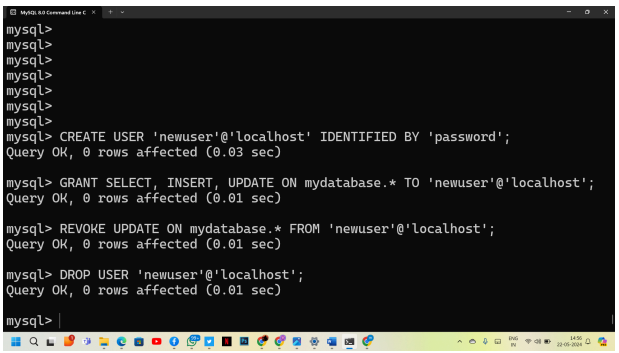
With the index in place, the database can quickly find all rows where the department column equals 'IT' without having to scan the entire table.

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.

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.

Managing Database Users and Privileges **1. Create a New User:** - Use the `CREATE USER` command to define a new user with a username and password. - Example: Create a user `new user` with password `password`. **2. Grant Privileges:** - Use the `GRANT` command to assign specific permissions to the user. - Example: Grant `SELECT`, `INSERT`, and `UPDATE` privileges on all tables in `database` to `new user`. **3. Revoke Privileges:** - Use the `REVOKE` command to remove specific permissions from the user. - Example: Revoke the `UPDATE` privilege on all tables in `database` from `newuser`. **4. Drop the User:** - Use the `DROP USER` command to completely remove the user from the database. - Example: Delete the `new user` account from the database. These steps allow you to create, manage, and delete database users and their privileges efficiently.	**Output as follows;** 

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.

1. **INSERT** new records:
   - Use `INSERT INTO` to add new data to the library tables.
   - For example, to add a new book, specify its ISBN, title, author, and publication year.
   - Similarly, for adding a member, provide their ID, name, email, and address.
   - To record a loan, include the loan ID, book ID, member ID, loan date, and due date.

2. **UPDATE** existing records:
   - Employ `UPDATE` to modify existing information in the tables.
   - For instance, update a member's email address or extend the due date of a loan.

3. **DELETE** records based on specific criteria:
   - Use `DELETE FROM` to remove data from tables based on conditions.
   - For example, delete a book by its ISBN, remove a member by their ID, or cancel a loan by its ID.

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).

These operations cover the basics of adding, updating, deleting, and importing data in the library database.

Here Output as Follows 👍