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.

Answer:

1. Identify Entities and Attributes:

Start by brainstorming the main objects or concepts that hold relevant information for your business. These become your entities.

For each entity, list the descriptive characteristics or properties you want to store. These are the attributes.

Example Scenario (Library Management System):

Entities:

Book

Author

Borrower

Attributes:

Book: ISBN, Title, Publication Year, Genre

Author: Author ID (primary key), Name, Nationality

Borrower: Borrower ID (primary key), Name, Contact Information

2. Define Relationships:

Consider how entities interact with each other. A relationship represents an association between two or more entities.

Relationships can be one-to-one (1:1), one-to-many (1:M), or many-to-many (M:N).

Example Scenario Relationships:

A Book can be written by one Author (1:M).

An Author can write many Books (M:1).

A Borrower can borrow many Books (M:N).

A Book can be borrowed by many Borrowers (M:N).

3. Normalize the ER Diagram:

Normalization is a process to minimize data redundancy and improve data integrity in a database. There are three main normal forms (1NF, 2NF, and 3NF) with increasing levels of normalization.

1NF (First Normal Form): Eliminates repeating groups within an entity.

2NF (Second Normal Form): Ensures no partial dependencies on the primary key.

3NF (Third Normal Form): Eliminates transitive dependencies on the primary key.

Normalization Steps for the Library Example:

1NF: We already have 1NF as there are no repeating groups.

2NF: No partial dependencies exist based on primary keys (Author ID and Borrower ID).

3NF: The Borrower entity might have a transitive dependency on Book through the Author entity. To address this, we can create a separate entity

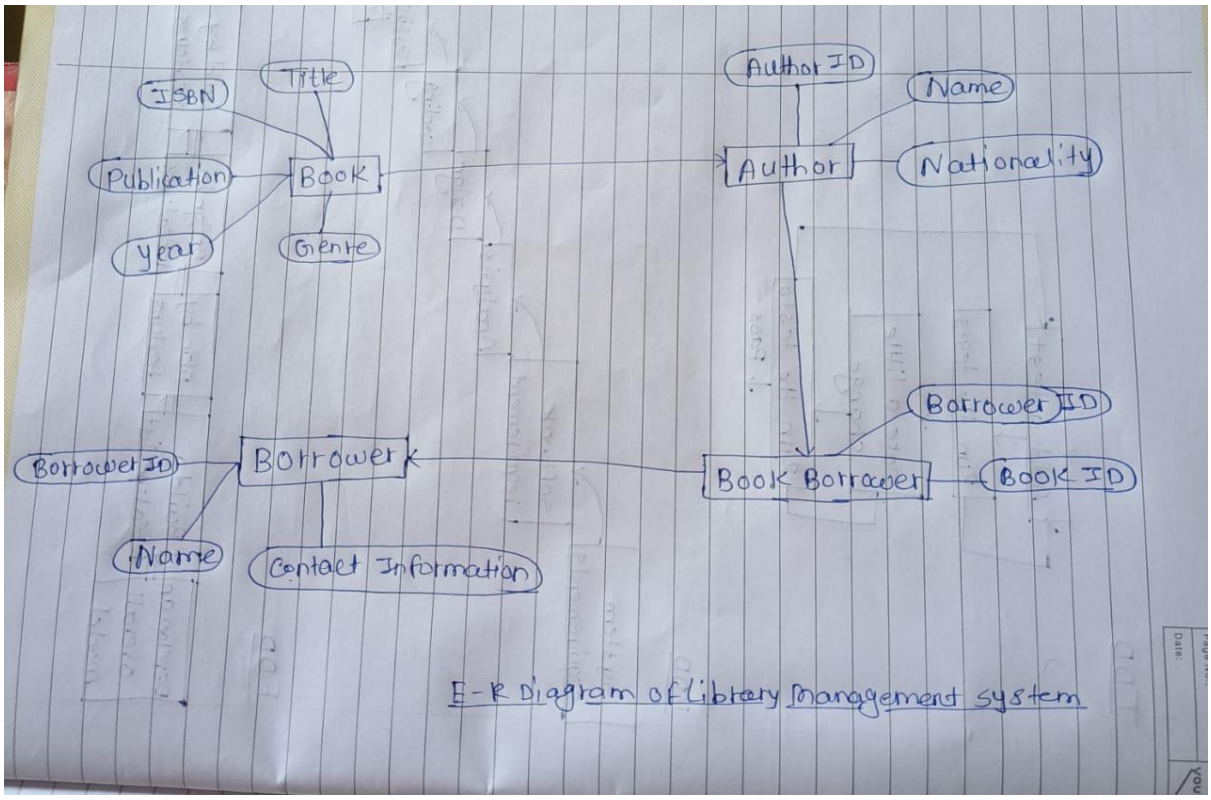
Book_Borrower to link Book and Borrower with their own primary key and eliminate the dependency.

4. Create the ER Diagram:

Use standard ERD symbols: Rectangles for entities, diamonds for relationships, ovals for attributes.

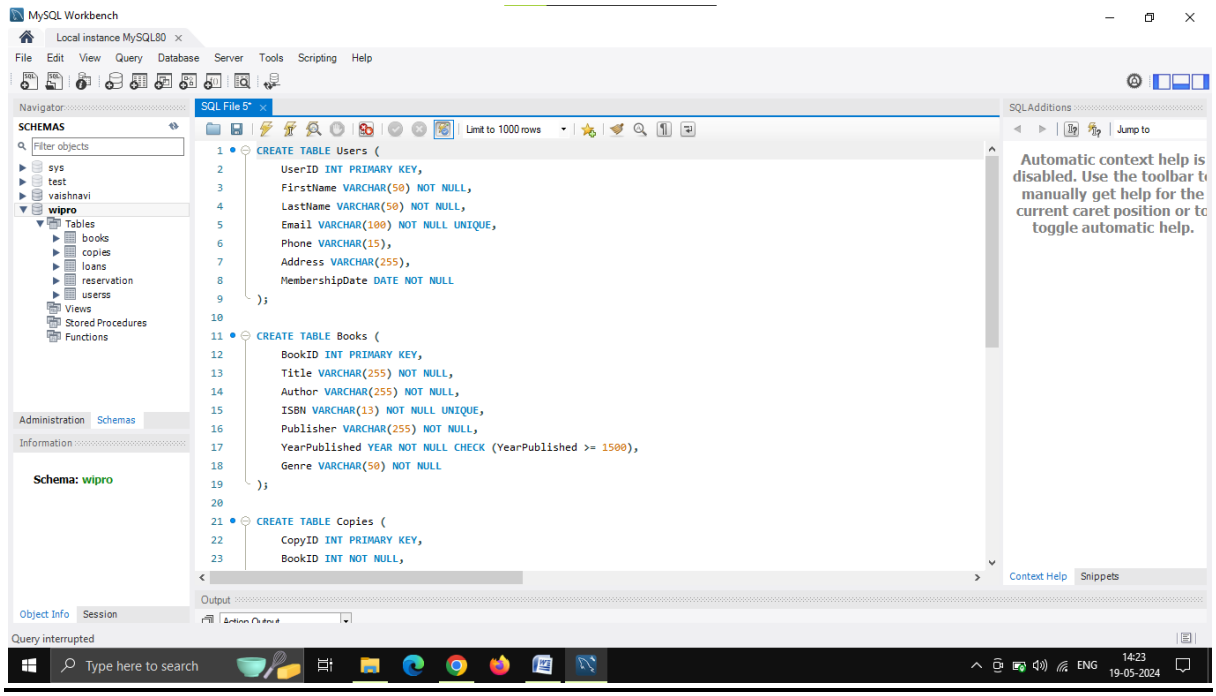
Label entities, attributes, and cardinalities (1:1, 1:M, M:N) on the connecting lines between entities and relationships.

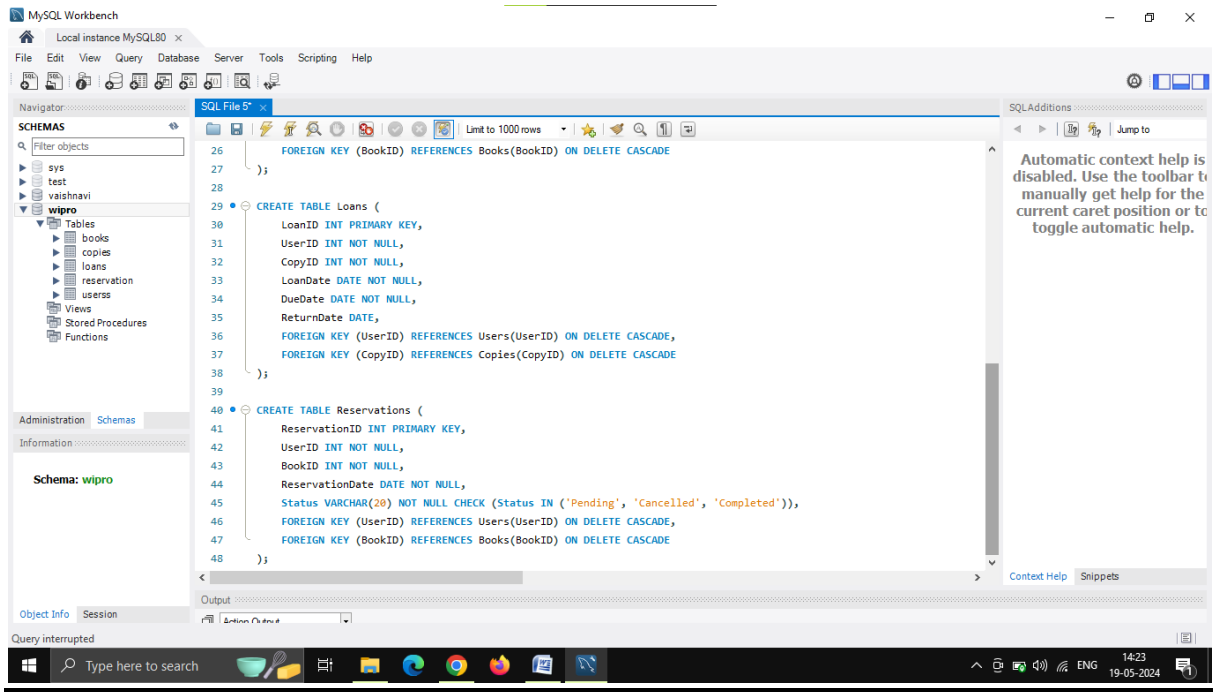
Example ER Diagram (3NF):

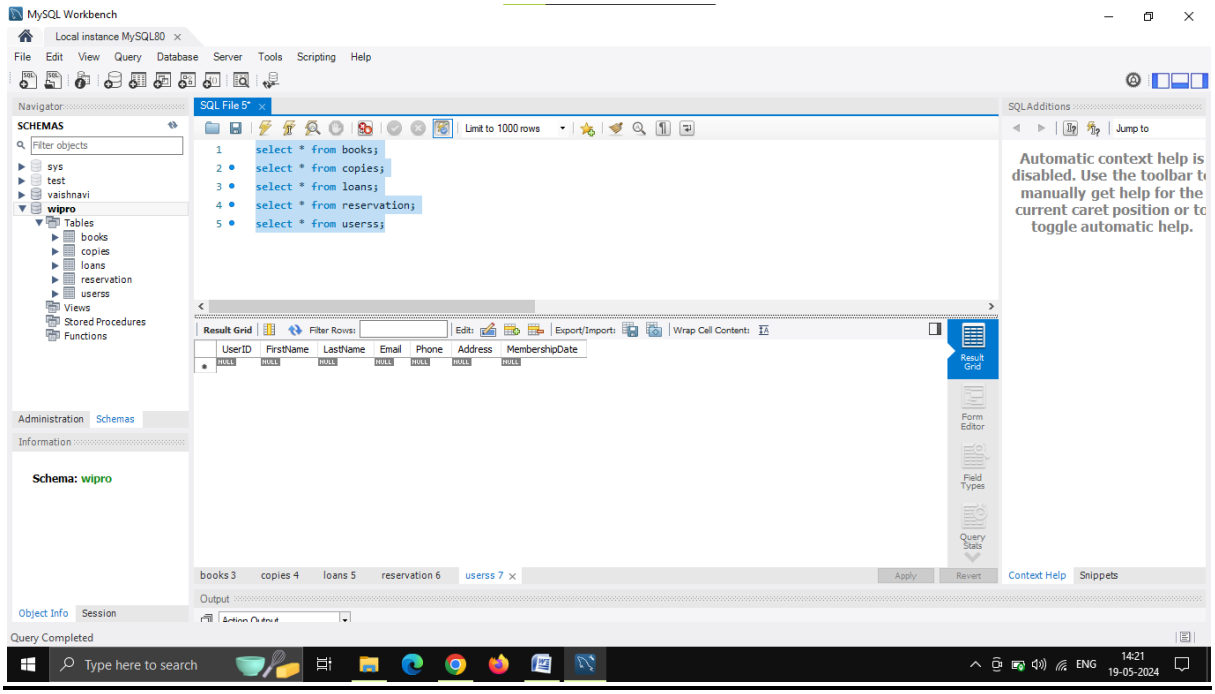


Assignment 2: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

Answer:







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.

Answer:

ACID Properties Explained

Atomicity: This property ensures that a series of database operations within a transaction are treated as a single unit. Either all operations are successfully executed, or none are. If any part of the transaction fails, the entire transaction is rolled back.

Consistency: Consistency ensures that a transaction brings the database from one valid state to another valid state, maintaining the database's predefined rules, such as constraints, cascades, and triggers. After the transaction, all data integrity constraints are still intact.

Isolation: Isolation ensures that transactions are executed independently of each other. Intermediate states of a transaction are invisible to other transactions until the transaction is complete, preventing potential conflicts.

Durability: Durability guarantees that once a transaction has been committed, it will remain in the system permanently, even in the event of a system failure. The changes are recorded in non-volatile memory.

SQL Statements for Transaction with Locking and Isolation Levels

Let's consider a library system where we want to simulate a transaction involving borrowing a book.

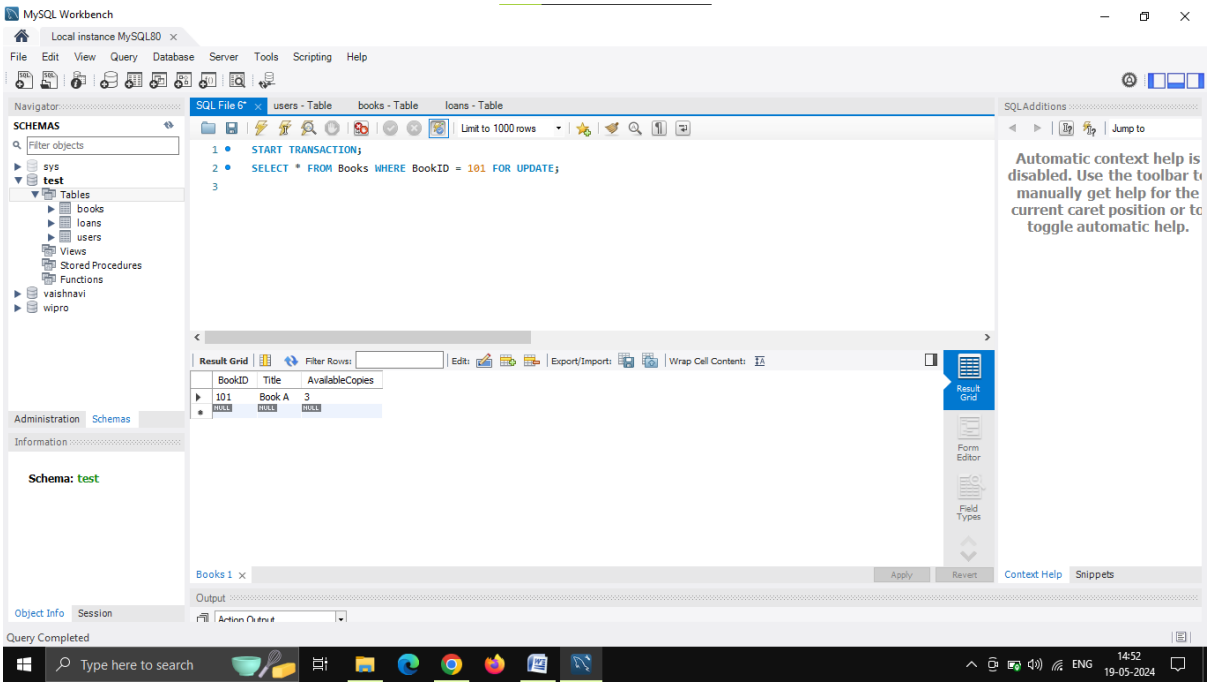
```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL
);

CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    AvailableCopies INT NOT NULL
);

CREATE TABLE Loans (
    LoanID INT PRIMARY KEY,
    UserID INT NOT NULL,
```

```
    BookID INT NOT NULL,
    LoanDate DATE NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID)
);

INSERT INTO Users (UserID, FirstName) VALUES (1, 'John'), (2, 'Jane');
INSERT INTO Books (BookID, Title, AvailableCopies) VALUES (101, 'Book A', 3), (102, 'Book B', 2);
```

Transaction Example

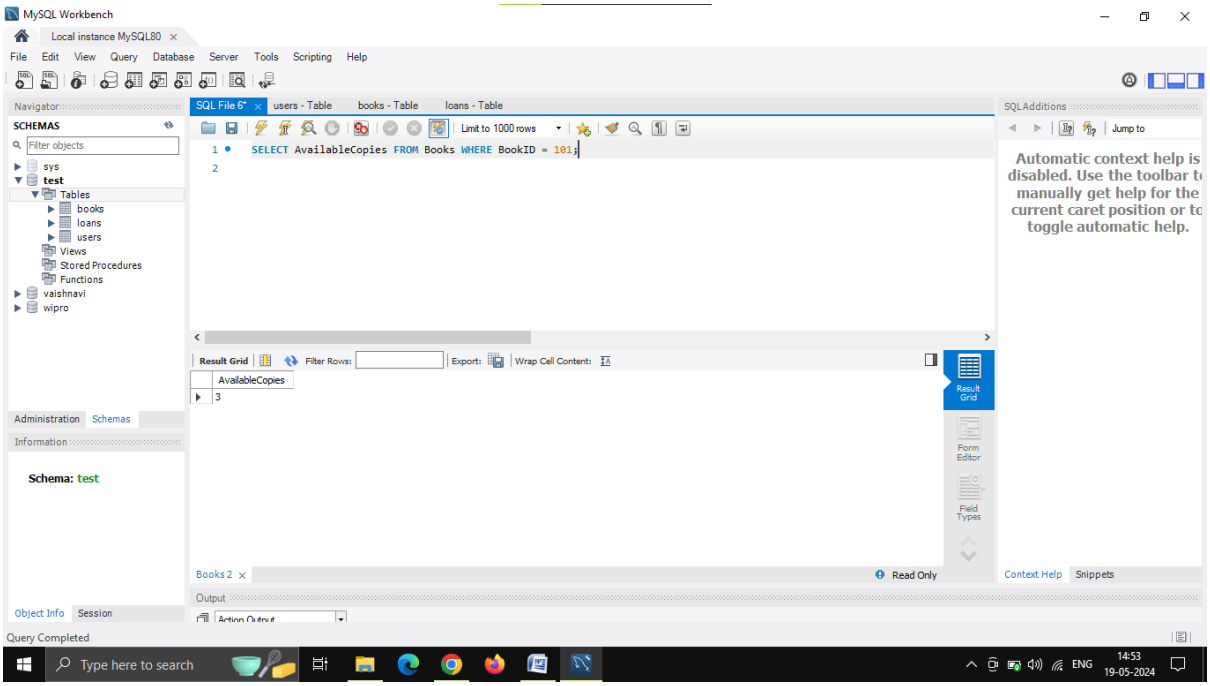
Borrowing a book involves decreasing the AvailableCopies and inserting a new record into the Loans table.

```
START TRANSACTION;
-- Lock the book row to prevent other transactions from modifying it simultaneously
SELECT * FROM Books WHERE BookID = 101 FOR UPDATE;
```



-- Check if the book is available

SELECT AvailableCopies FROM Books WHERE BookID = 101;



-- Decrease the number of available copies

UPDATE Books

SET AvailableCopies = AvailableCopies - 1

```
WHERE BookID = 101;


-- Insert a new loan record
INSERT INTO Loans (LoanID, UserID, BookID, LoanDate)
VALUES (1, 1, 101, CURDATE());
COMMIT;
```

Isolation Levels and Concurrency Control

Different isolation levels can be set to demonstrate concurrency control. Here's how you can set and demonstrate each isolation level:

Read Uncommitted: Allows dirty reads, where one transaction can see uncommitted changes made by another transaction.

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
START TRANSACTION;
SELECT * FROM Books WHERE BookID = 101;

-- Changes from other transactions are visible even if not committed
```

Read Committed: Prevents dirty reads. Only committed changes are visible.

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT * FROM Books WHERE BookID = 101;
```

-- Changes from other transactions are visible only if committed

Repeatable Read: Ensures that if a transaction reads a row, it will see the same data if it reads it again within the same transaction, preventing non-repeatable reads.

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

SELECT * FROM Books WHERE BookID = 101;

-- Subsequent reads will see the same data, even if other transactions modify it

Serializable: The highest isolation level, ensuring complete isolation from other transactions. It prevents phantom reads and guarantees that the transaction operates in a serializable manner.

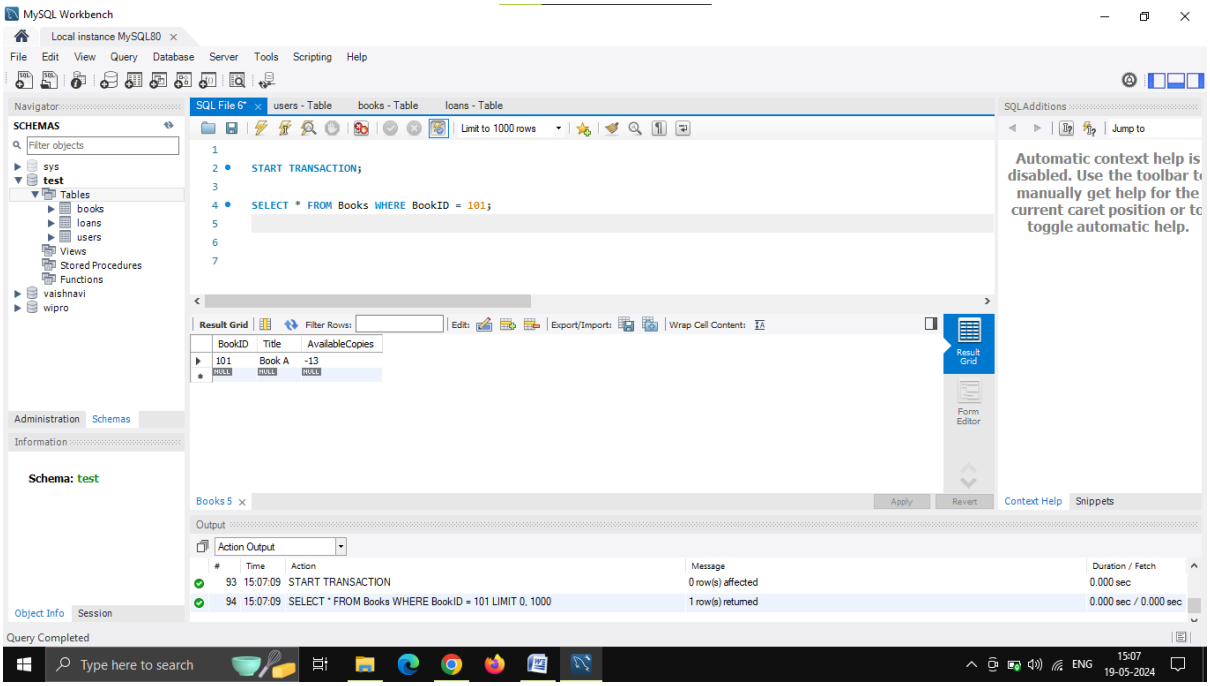
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

SELECT * FROM Books WHERE BookID = 101;

-- No other transactions can insert, update, or delete rows that would affect the result

By setting different isolation levels, you can control the level of concurrency and consistency in your transactions. This is crucial for ensuring that your transactions meet the ACID properties and maintain the integrity and reliability of the database.



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.

Answer:

Step 1: Create a New Database

```
CREATE DATABASE LibraryDB;
```

Step 2: Create Tables

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    AuthorID INT,
    PublishedYear INT,
```

```
    Genre VARCHAR(100),
    ISBN VARCHAR(13) UNIQUE
);

CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(100),
    LastName VARCHAR(100),
    DateOfBirth DATE,
    Nationality VARCHAR(100)
);

CREATE TABLE Members (
    MemberID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(100),
    LastName VARCHAR(100),
    DateOfBirth DATE,
    MembershipDate DATE,
    Email VARCHAR(255) UNIQUE
);

CREATE TABLE Loans (
    LoanID INT PRIMARY KEY AUTO_INCREMENT,
```

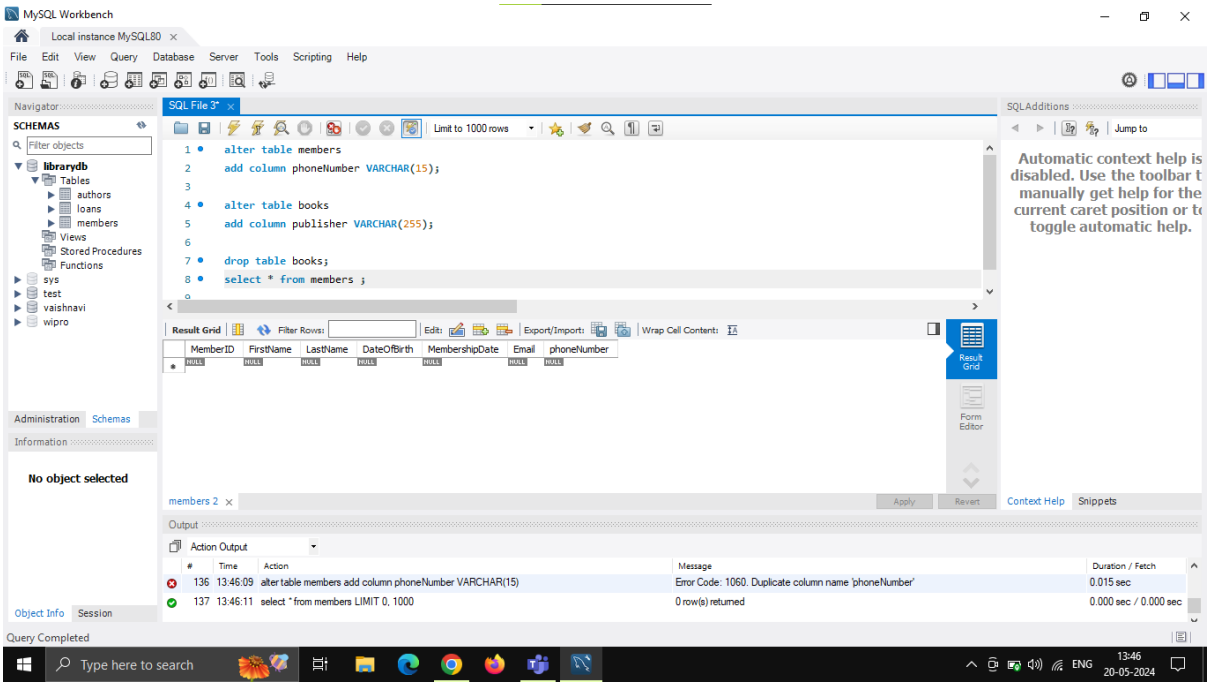
```
    BookID INT,
    MemberID INT,
    LoanDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

Step 3: Use ALTER Statements to Modify Table Structures

```
ALTER TABLE Members
ADD COLUMN PhoneNumber VARCHAR(15);

ALTER TABLE Books
ADD COLUMN Publisher VARCHAR(255);
```

Step 4: DROP a Redundant Table

```
DROP TABLE Books;
```



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.

Answer:

Create a table

```
CREATE TABLE students (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    grade CHAR(1)
);
```

Insert some sample data

```
INSERT INTO students (id, name, age, grade) VALUES (1, 'John', 20, 'A');
```

INSERT INTO students (id, name, age, grade) VALUES (2, 'Alice', 22, 'B');

INSERT INTO students (id, name, age, grade) VALUES (3, 'Bob', 21, 'A');

INSERT INTO students (id, name, age, grade) VALUES (4, 'Emma', 20, 'C');

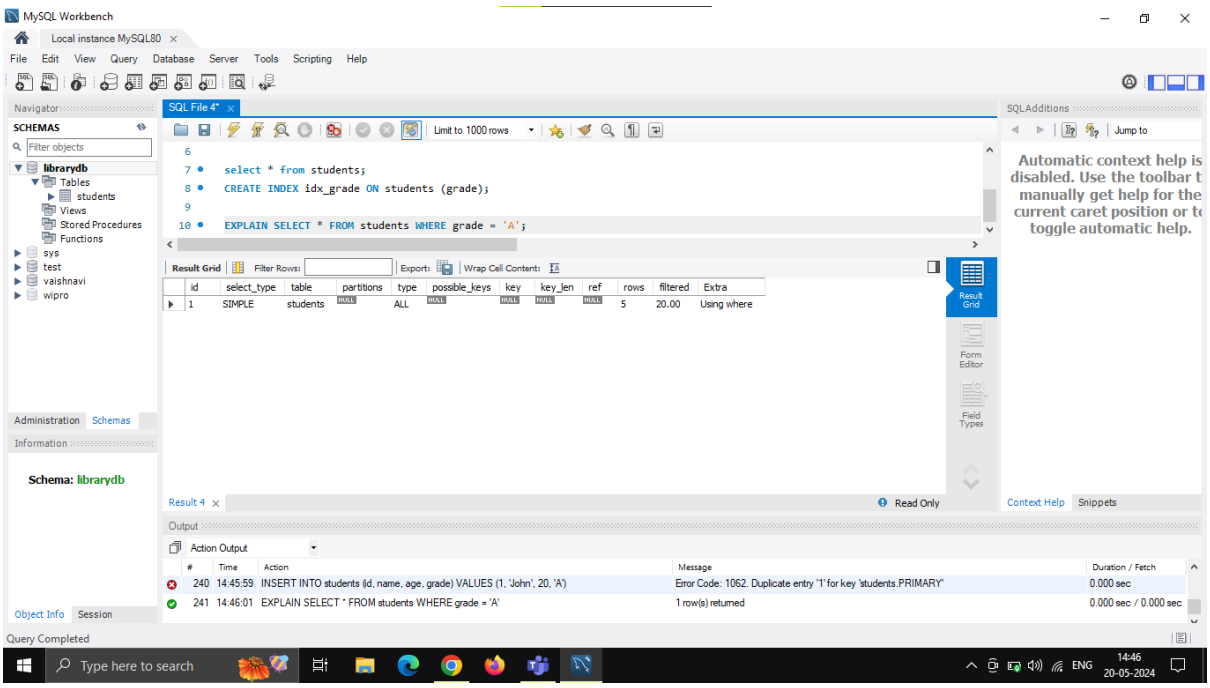
INSERT INTO students (id, name, age, grade) VALUES (5, 'Mike', 23, 'A');

Create an index on the 'grade' column

CREATE INDEX idx_grade ON students (grade);

Query without index

EXPLAIN SELECT * FROM students WHERE grade = 'A';

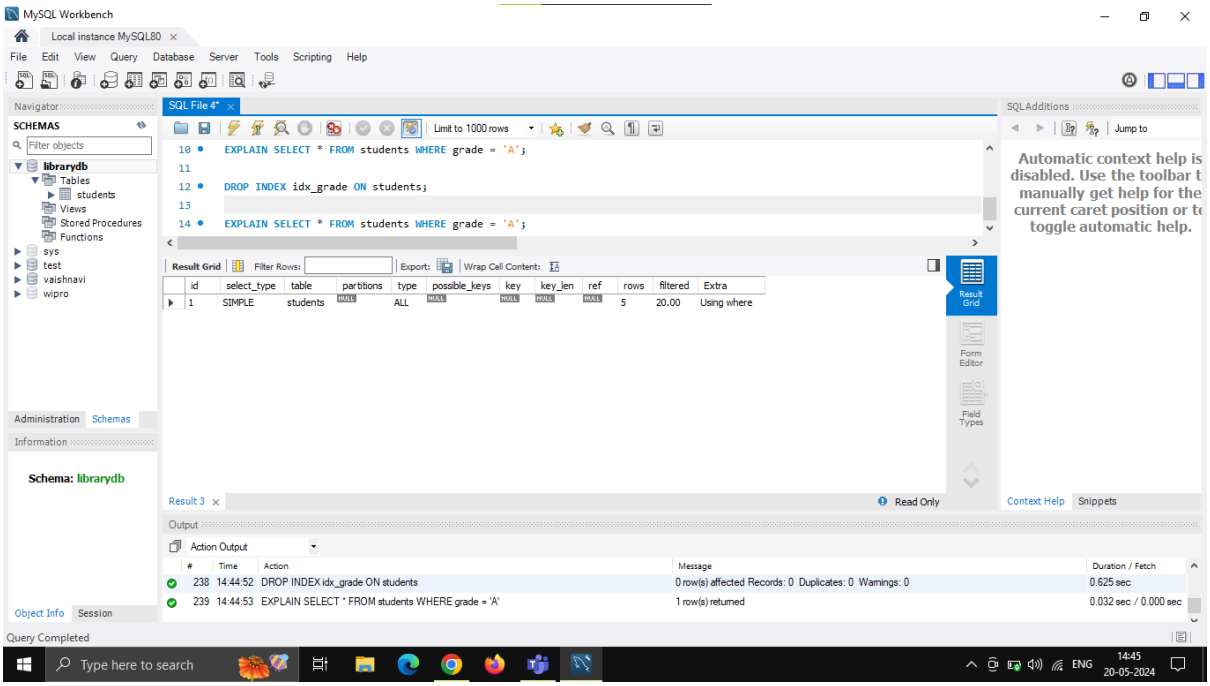


Remove the index

DROP INDEX idx_grade ON students;

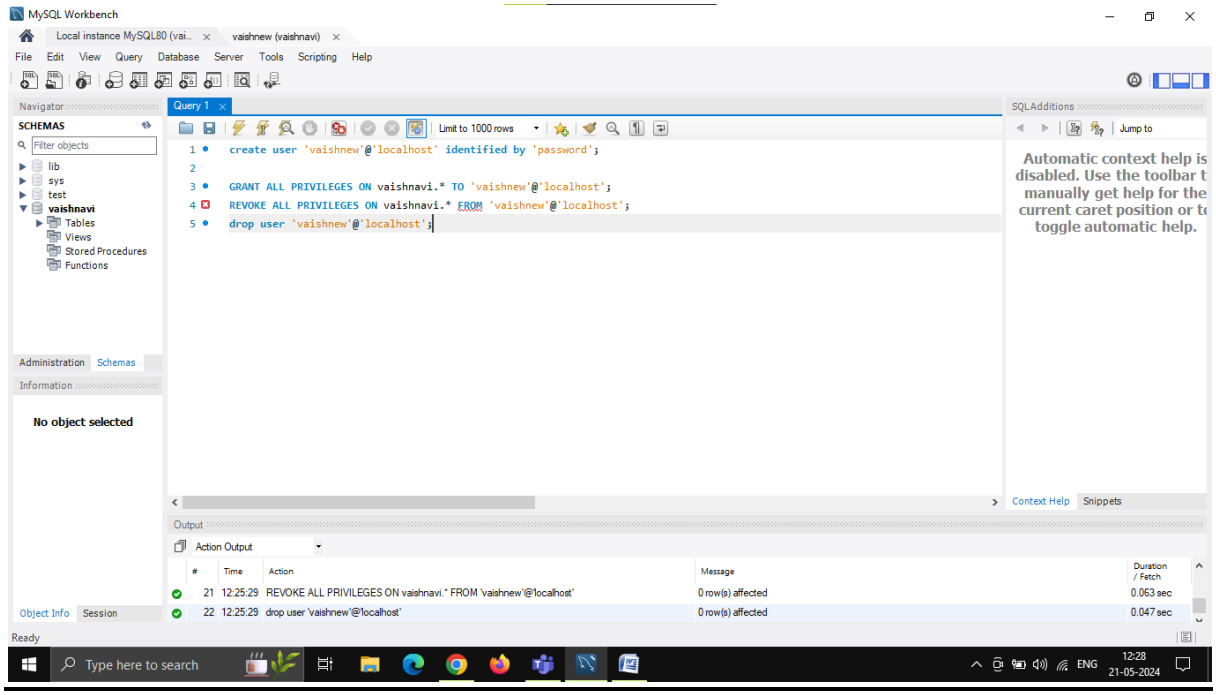
Query after removing the index

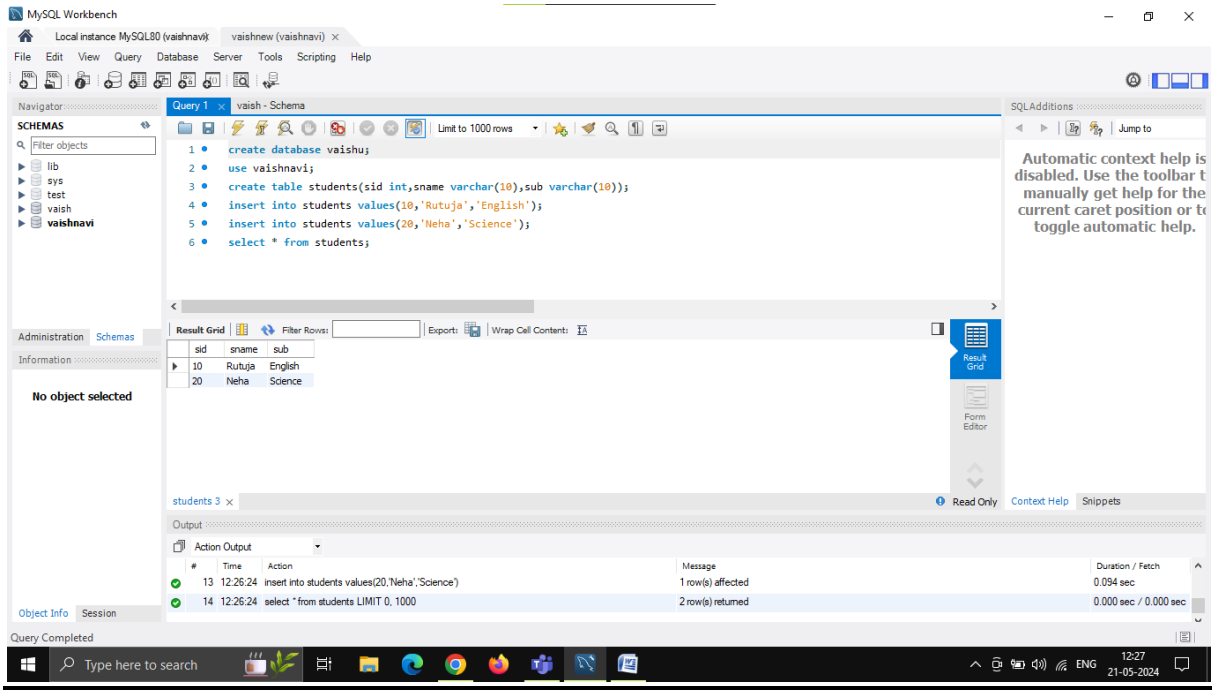
EXPLAIN SELECT * FROM students WHERE grade = 'A'



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.

Answer:





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.

Answer:

Insert a new book record

INSERT INTO Books (title, author, publication_year, isbn)

VALUES ('Title', 'Author', 2024, 'ISBN');

INSERT INTO Members (name, email, phone)

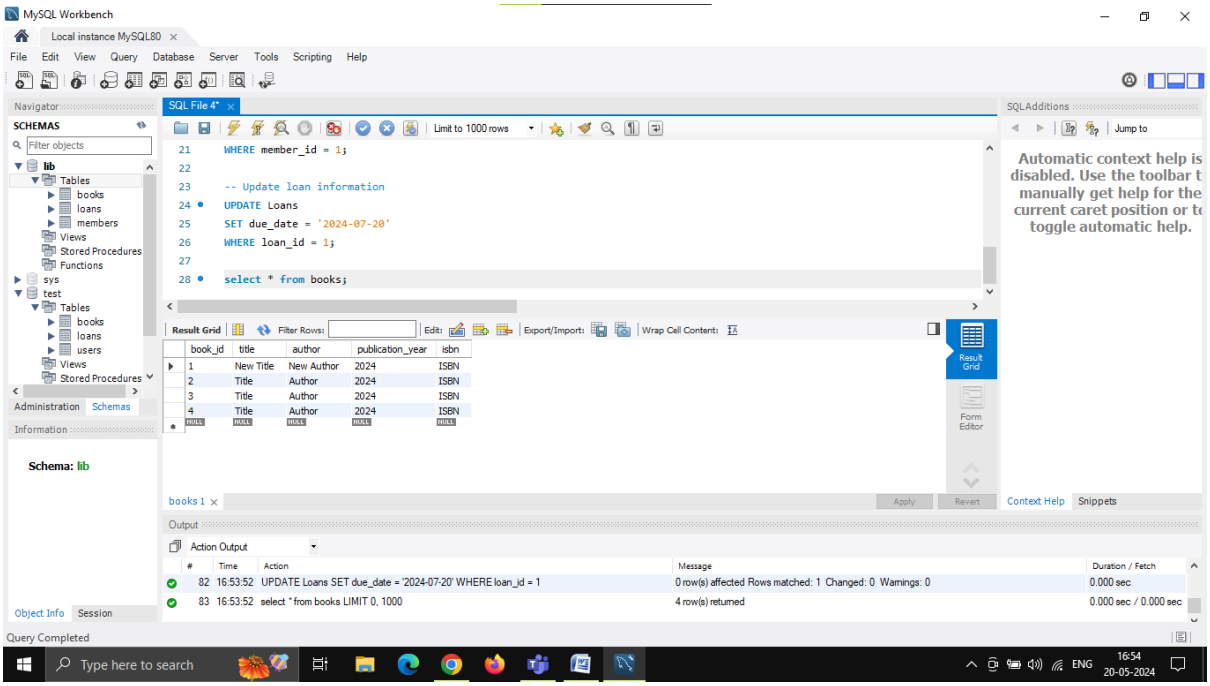
VALUES ('Name', 'email@example.com', '1234567890');

INSERT INTO Loans (book_id, member_id, loan_date, due_date)

VALUES (1, 1, '2024-05-20', '2024-06-20');

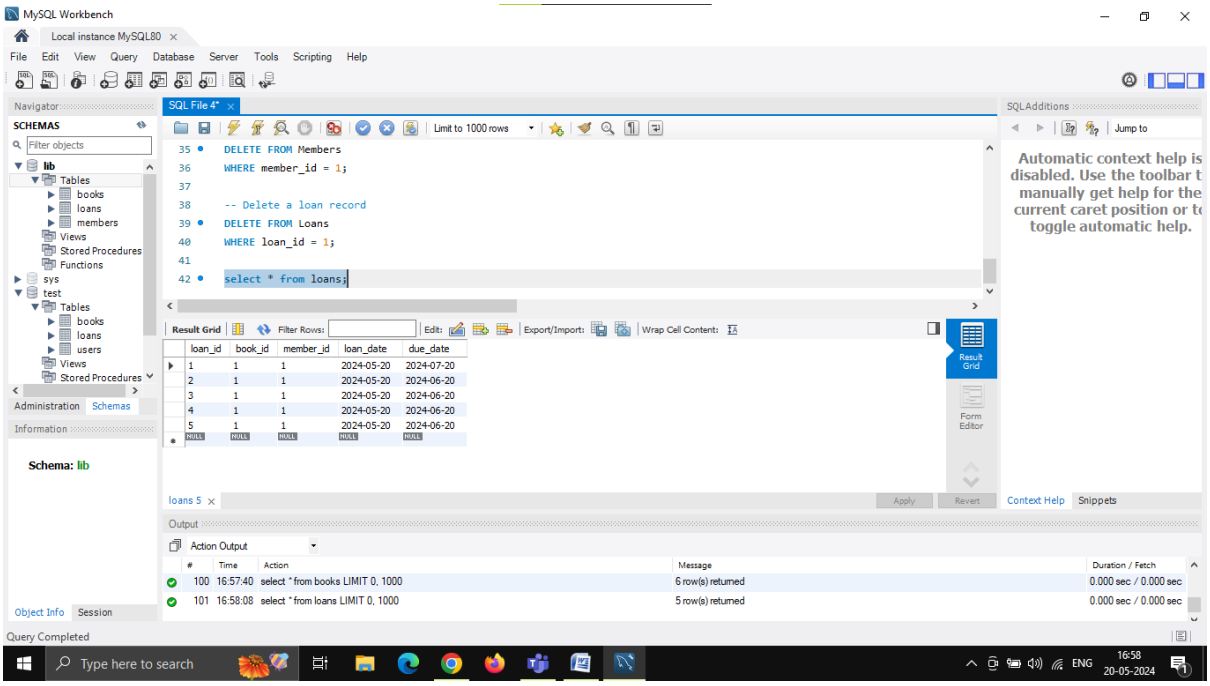
Update book information

```
UPDATE Books

SET title = 'New Title', author = 'New Author'

WHERE book_id = 1;


UPDATE Members

SET email = 'new_email@example.com'

WHERE member_id = 1;


UPDATE Loans

SET due_date = '2024-07-20'

WHERE loan_id = 1;
```



Delete a book record

```
DELETE FROM Books

WHERE book_id = 1;


DELETE FROM Members

WHERE member_id = 1;


DELETE FROM Loans

WHERE loan_id = 1;
```



Bulk insert data into the Books table from a CSV file

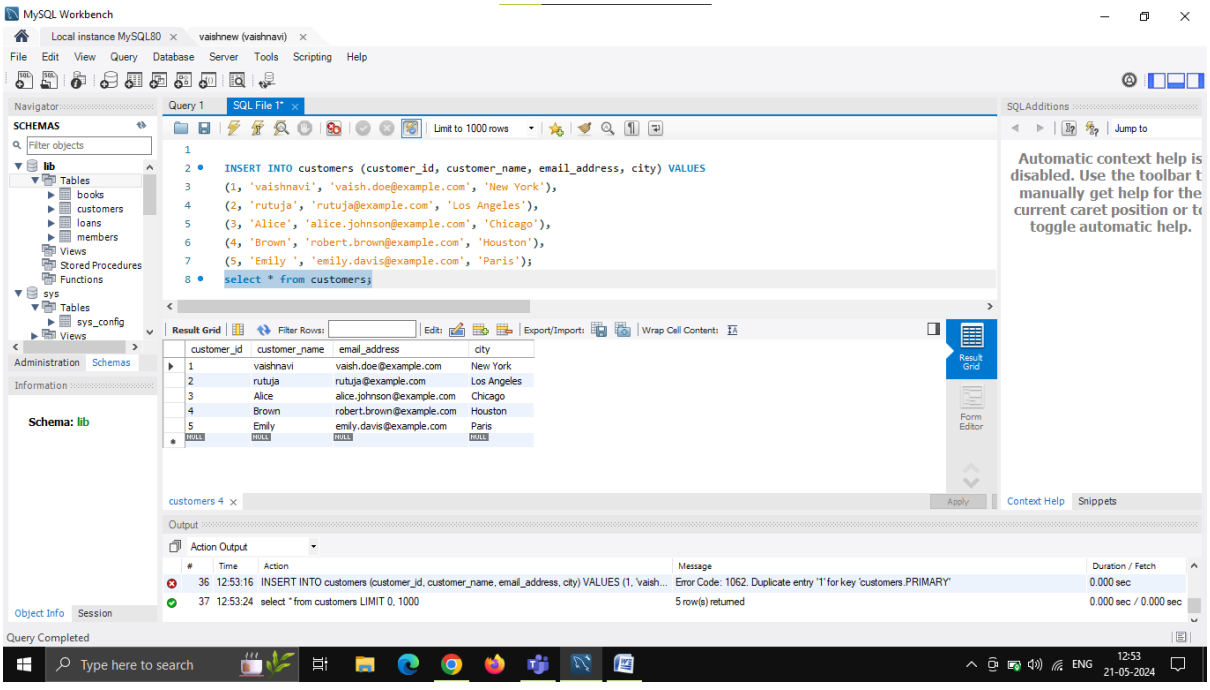
```
BULK INSERT Books

FROM 'C:\path\to\books.csv'

WITH (

    FIELDTERMINATOR = ',',
```

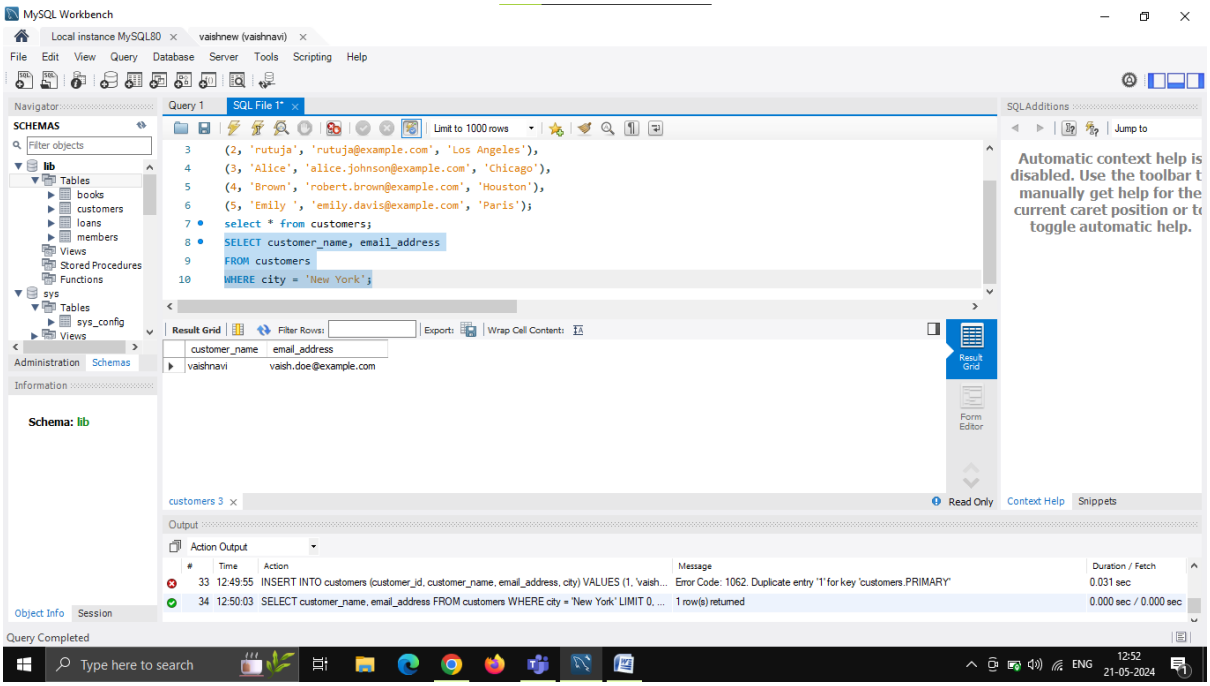
```
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- If the first row contains headers
);

-- Bulk insert data into the Members table from a CSV file
BULK INSERT Members
FROM 'C:\path\to\members.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- If the first row contains headers
);

-- Bulk insert data into the Loans table from a CSV file
BULK INSERT Loans
FROM 'C:\path\to\loans.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- If the first row contains headers
);
```

Assignment 1: Write a SELECT query to retrieve all columns from a customers' table, and modify it to return only the customer name and email address for customers in a specific city.

Answer:

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100) NOT NULL,
    email_address VARCHAR(100) NOT NULL,
    city VARCHAR(50) NOT NULL
);
```



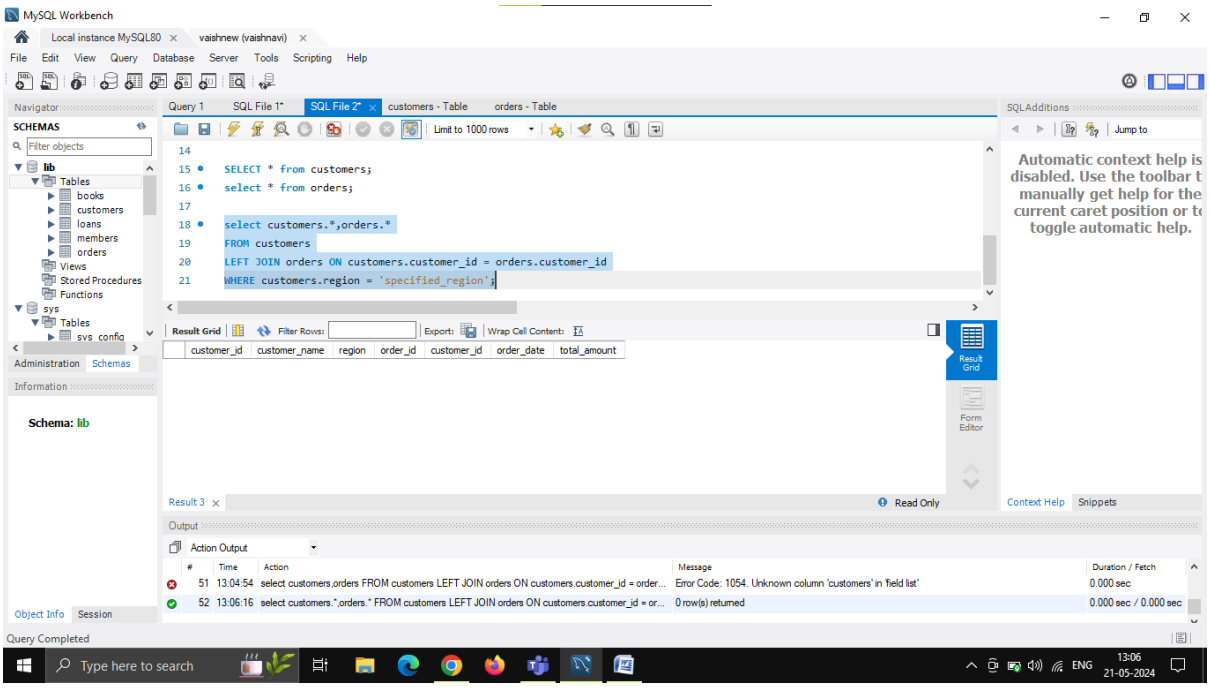


Assignment 2: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

Answer:

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    region VARCHAR(50)
);


CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
```

```
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```



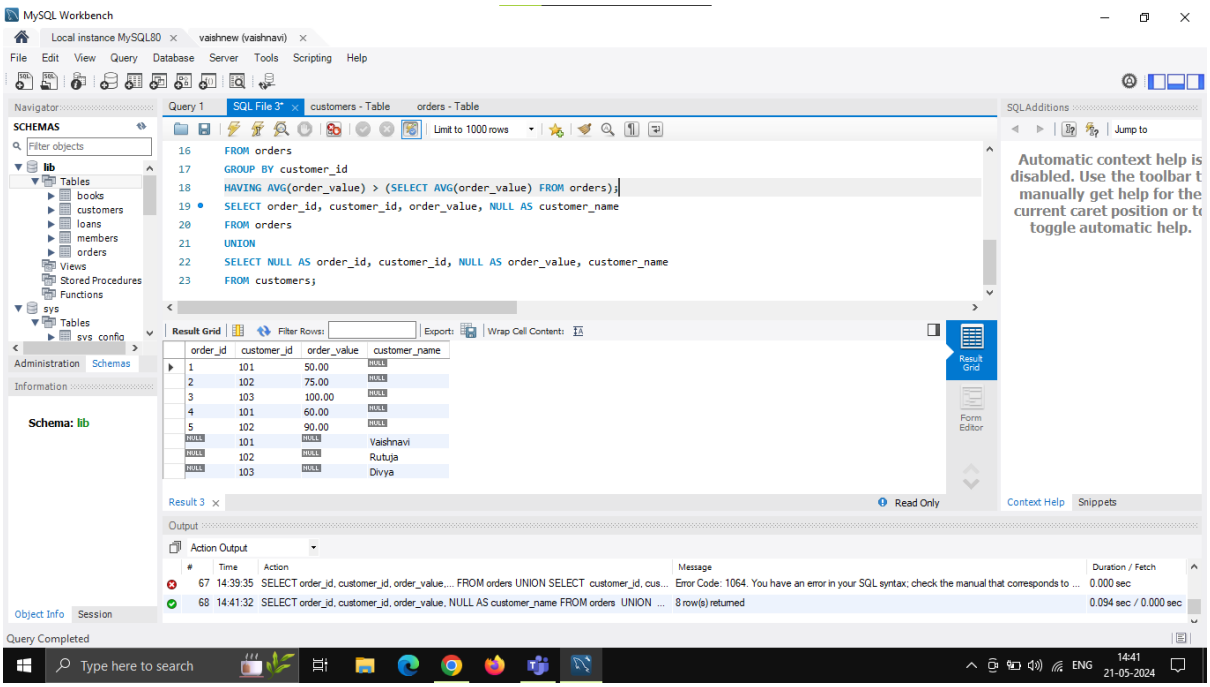
Assignment 3: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

Answer:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_value DECIMAL(10,2)
);
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
```

customer_name VARCHAR(100)

);



Assignment 4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

Answer:

INSERT INTO orders (order_id, customer_id, order_date, total_amount)

VALUES (28,101, '2024-05-21', 250.00);

COMMIT;

UPDATE products

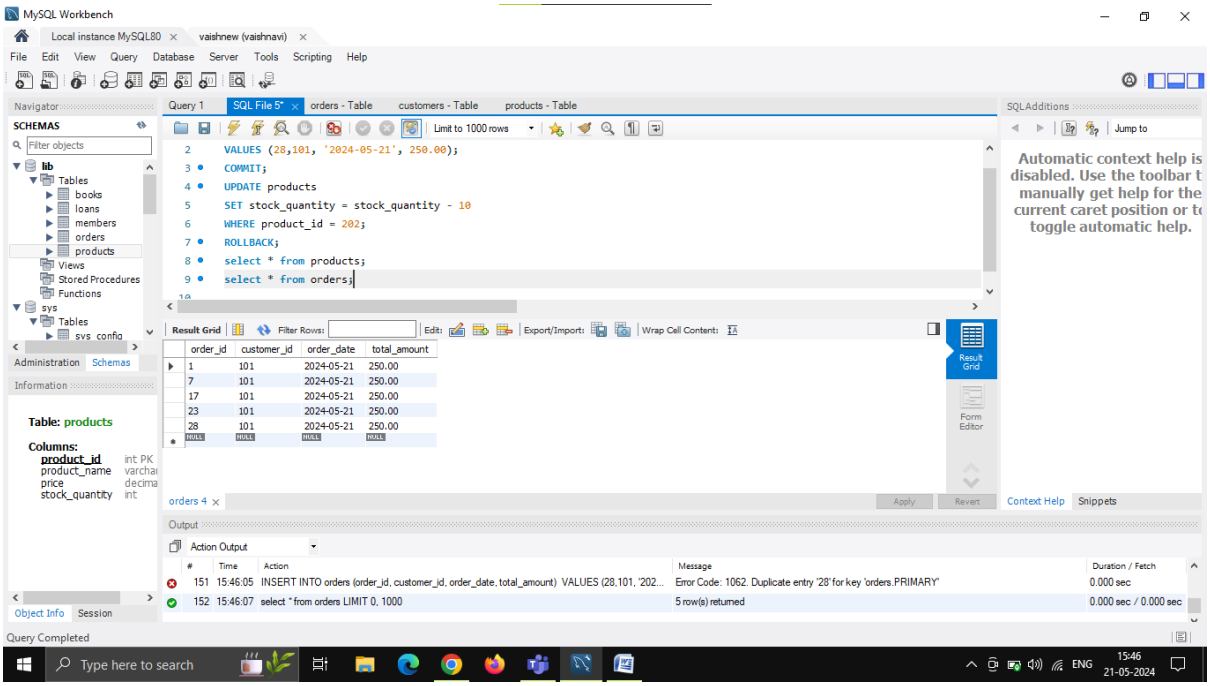
SET stock_quantity = stock_quantity - 10

WHERE product_id = 202;

ROLLBACK;

select * from products;

select * from orders;



Assignment 5: Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

Answer:

BEGIN;

INSERT INTO orders (order_id, customer_id, order_value) VALUES (19, 101, '16');

SAVEPOINT sp1;

INSERT INTO orders (order_id, customer_id, order_value) VALUES (10, 102, '21');

SAVEPOINT sp2;

INSERT INTO orders (order_id, customer_id, order_value) VALUES (13, 103, '43');

SAVEPOINT sp3;

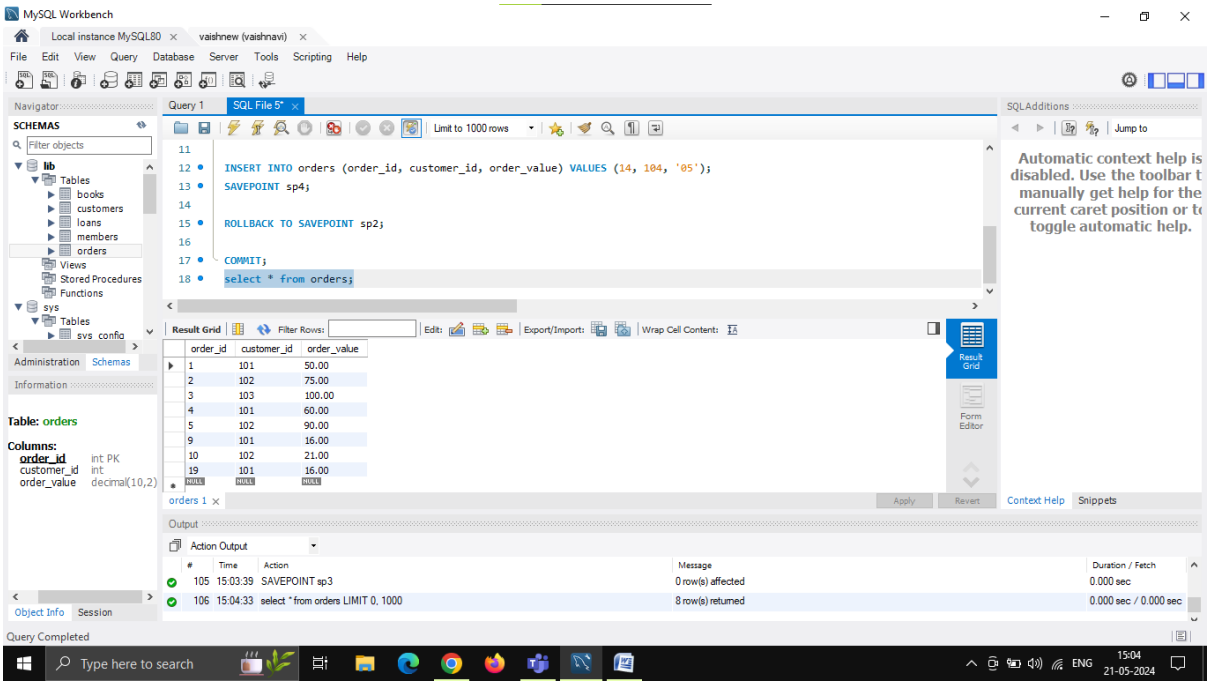
INSERT INTO orders (order_id, customer_id, order_value) VALUES (14, 104, '05');

SAVEPOINT sp4;

ROLLBACK TO SAVEPOINT sp2;

COMMIT;

select * from orders;



Assignment 6: Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Answer:

Transaction Logs: Guardians of Data Integrity

Transaction logs are the unsung heroes of data management. These chronological records of all database modifications play a critical role in ensuring data integrity and facilitating recovery in the event of system failures.

This report explores how transaction logs function and their significance in data restoration.

The Power of Logs

A transaction log acts as a detailed journal, capturing every database operation – inserts, updates, and deletes. Each entry includes crucial information, such as:

Transaction ID: A unique identifier for tracking individual operations.

Timestamp: Records the exact time of the modification.

Before Image: Stores the state of the data prior to the change.

After Image: Captures the data's state after the modification.

This comprehensive record allows database systems to perform two key functions:

Rollback: If a transaction encounters an error or is interrupted, the log enables the system to revert the changes, essentially undoing the entire operation.

Recovery: In case of a system crash or unexpected shutdown, transaction logs serve as a roadmap to restore the database to a consistent state.

Scenario: Data Rescue with Transaction Logs

Imagine a bustling online store. During peak buying hours, the database server suffers a power outage. Panic sets in – countless customer orders might be lost! Thankfully, the system utilizes transaction logs.

Here's how transaction logs save the day:

Identifying Incomplete Transactions: Upon restarting, the database system scans the transaction log. It identifies unfinished transactions abruptly halted by the power outage.

Rollback of Incomplete Transactions: Using the "Before Image" information, the system meticulously reverses any changes attempted during these

incomplete transactions. This ensures no corrupted or inconsistent data enters the database.

Recovery to Consistent State: The log then guides the system in applying all committed transactions (those completed before the outage). The "After Image" information ensures data is restored to its most recent accurate state.