Name : Tejaswini Anil kamble

Batch Name : CPPE_Java Full Stack

Email : teju000kamble@gmail.com

RDBMS and SQL Assignments

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.

Ans:

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.

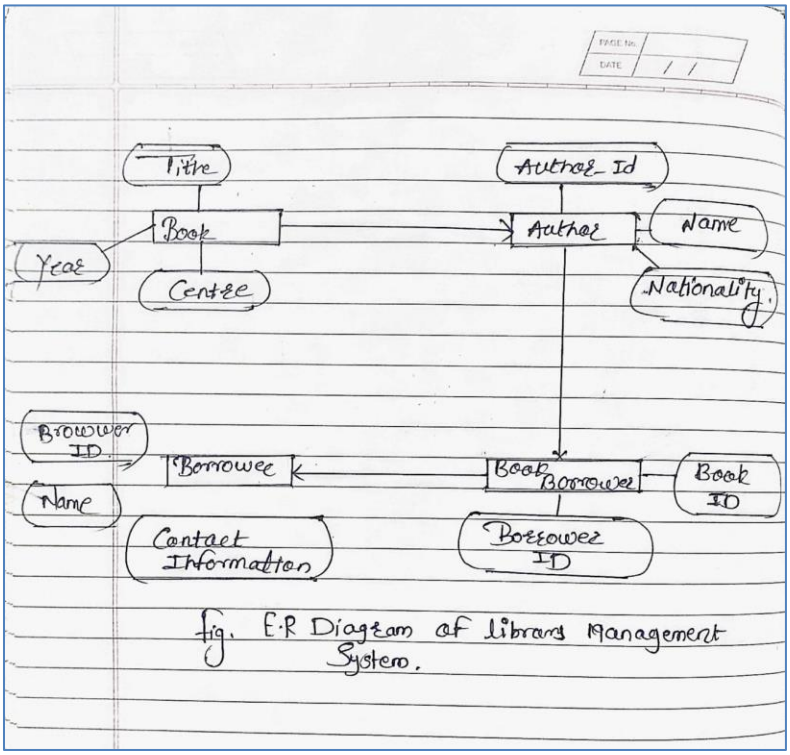
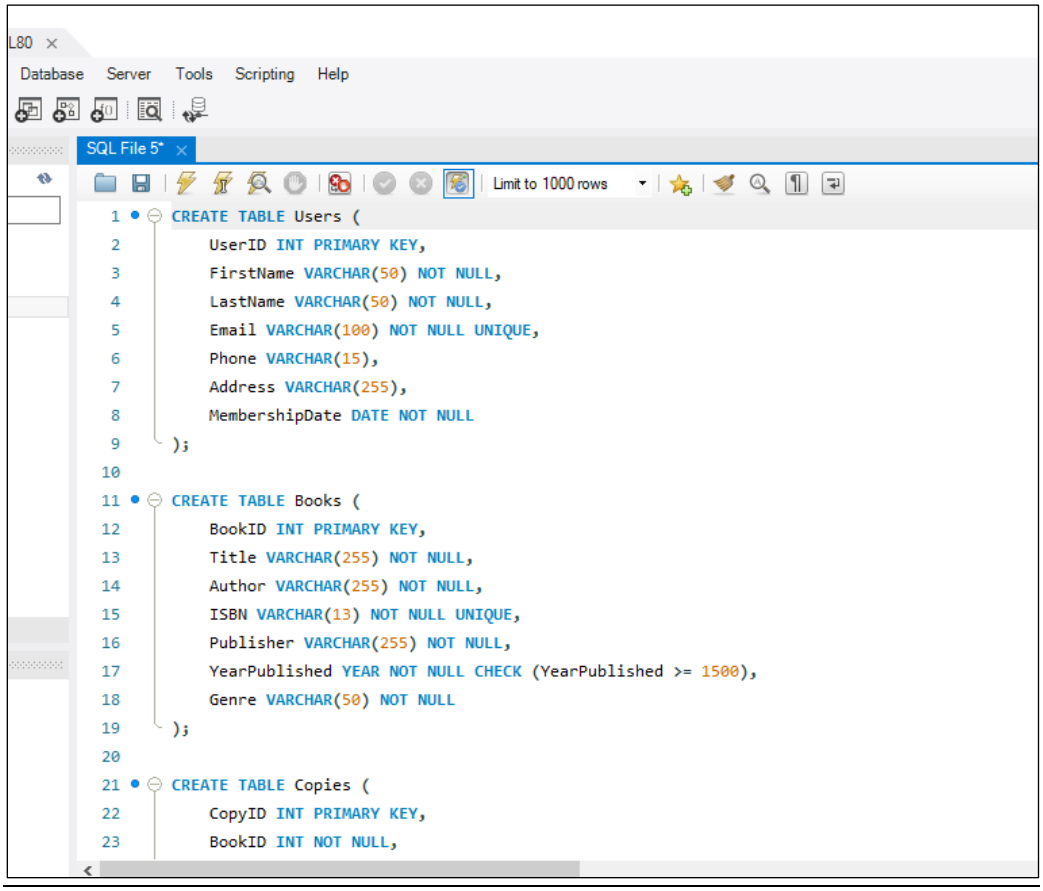
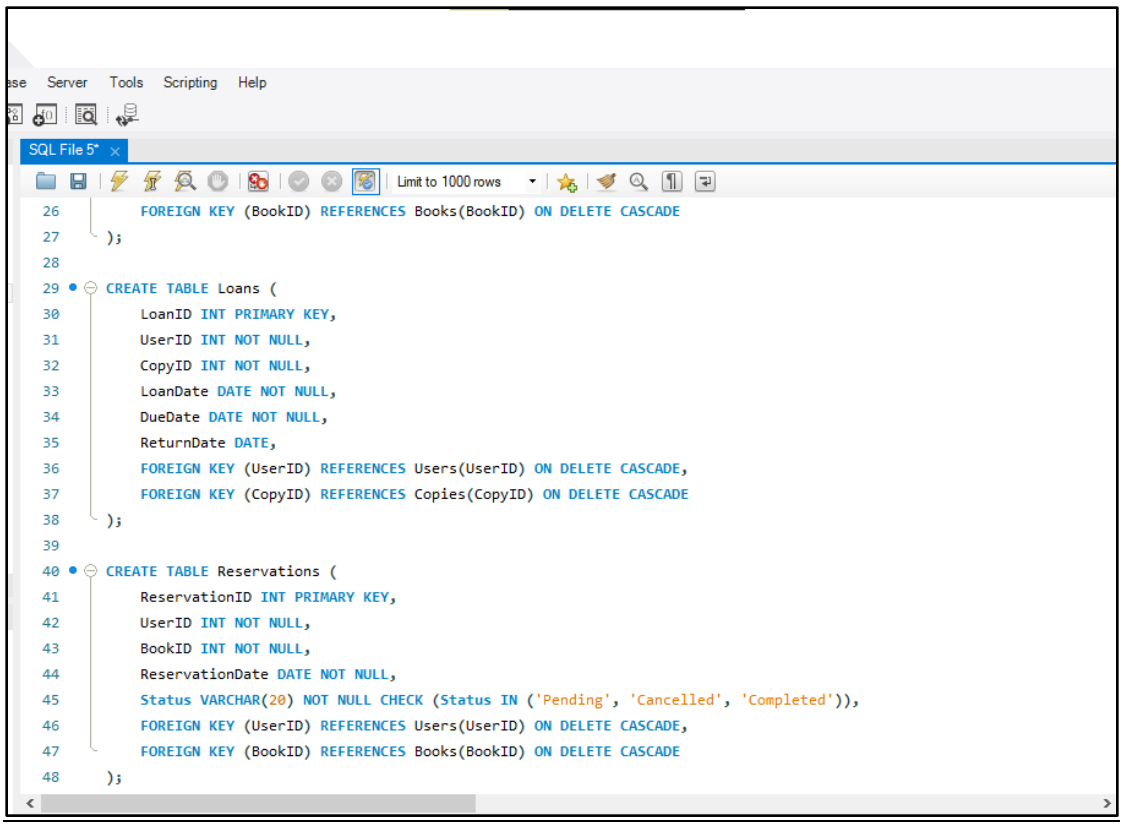


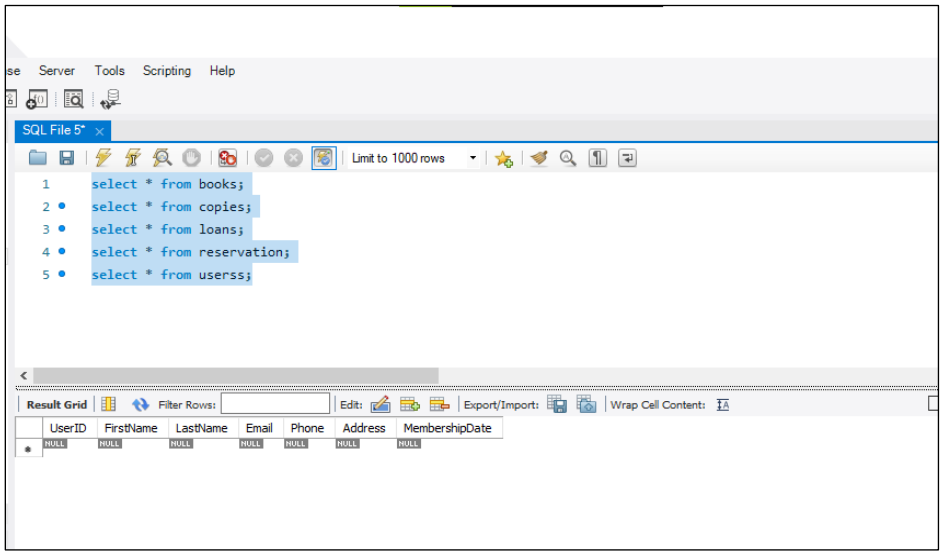
fig. E.R Diagram of library Management System.

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.

Ans:




Server Tools Scripting Help
SQL File 5*
Limit to 1000 rows
FOREIGN KEY (BookID) REFERENCES Books(BookID) ON DELETE CASCADE
);
CREATE TABLE Loans (
LoanID INT PRIMARY KEY,
UserID INT NOT NULL,
CopyID INT NOT NULL,
LoanDate DATE NOT NULL,
DueDate DATE NOT NULL,
ReturnDate DATE,
FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE,
FOREIGN KEY (CopyID) REFERENCES Copies(CopyID) ON DELETE CASCADE
);
CREATE TABLE Reservations (
ReservationID INT PRIMARY KEY,
UserID INT NOT NULL,
BookID INT NOT NULL,
ReservationDate DATE NOT NULL,
Status VARCHAR(20) NOT NULL CHECK (Status IN ('Pending', 'Cancelled', 'Completed')),
FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE,
FOREIGN KEY (BookID) REFERENCES Books(BookID) ON DELETE CASCADE
);


Server Tools Scripting Help
SQL File 5*
Limit to 1000 rows
select * from books;
select * from copies;
select * from loans;
select * from reservation;
select * from userss;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
UserID
FirstName
LastName
Email
Phone
Address
MembershipDate
NULL

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.

Ans:

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));

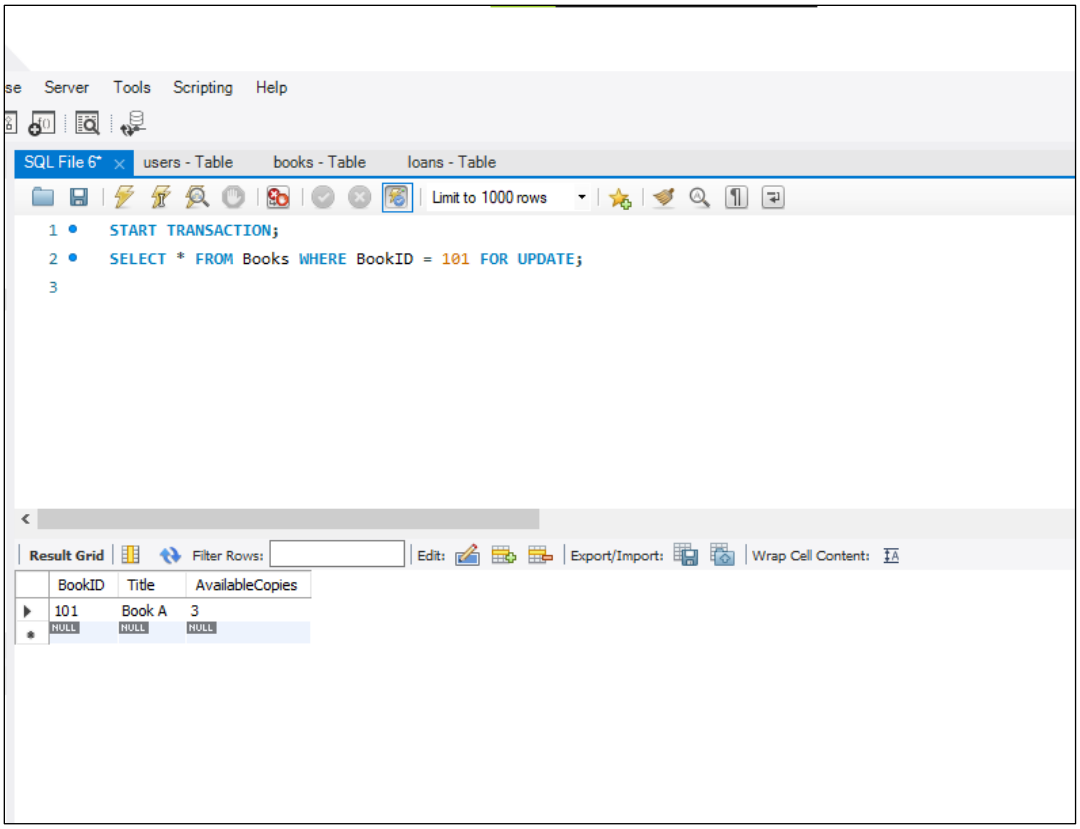
- INSERT INTO Users (UserID, FirstName) VALUES (1, 'John'), (2, 'Jane');

- 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.

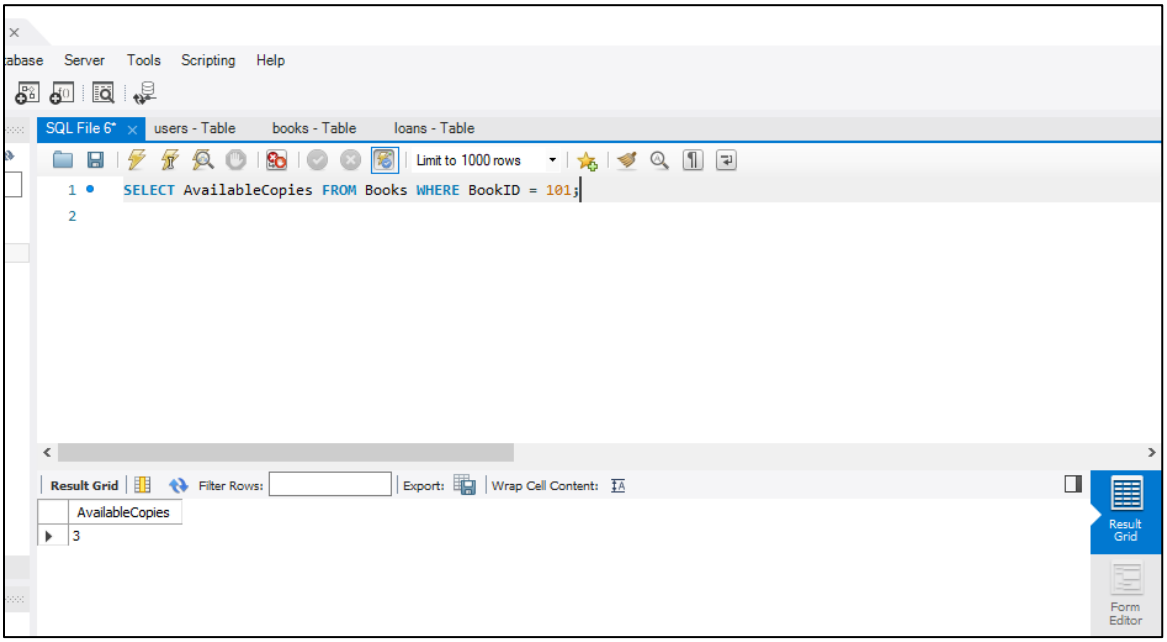
- START TRANSACTION;
- Lock the book row to prevent other transactions from modifying it simultaneously

- 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.

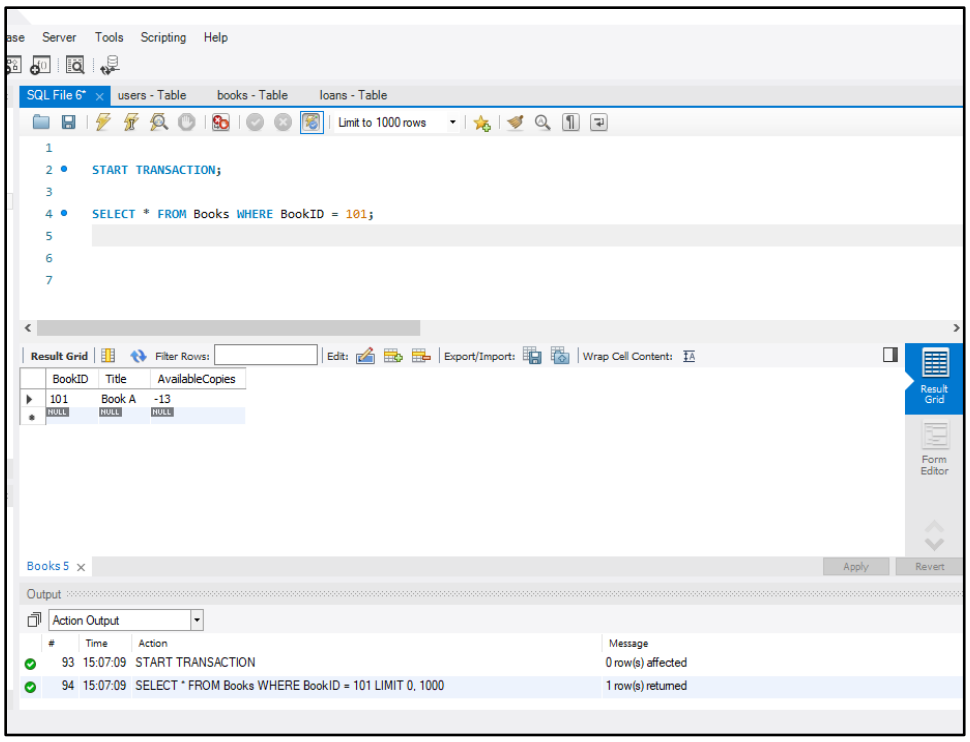
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

SELECT * FROM Books WHERE BookID = 101;

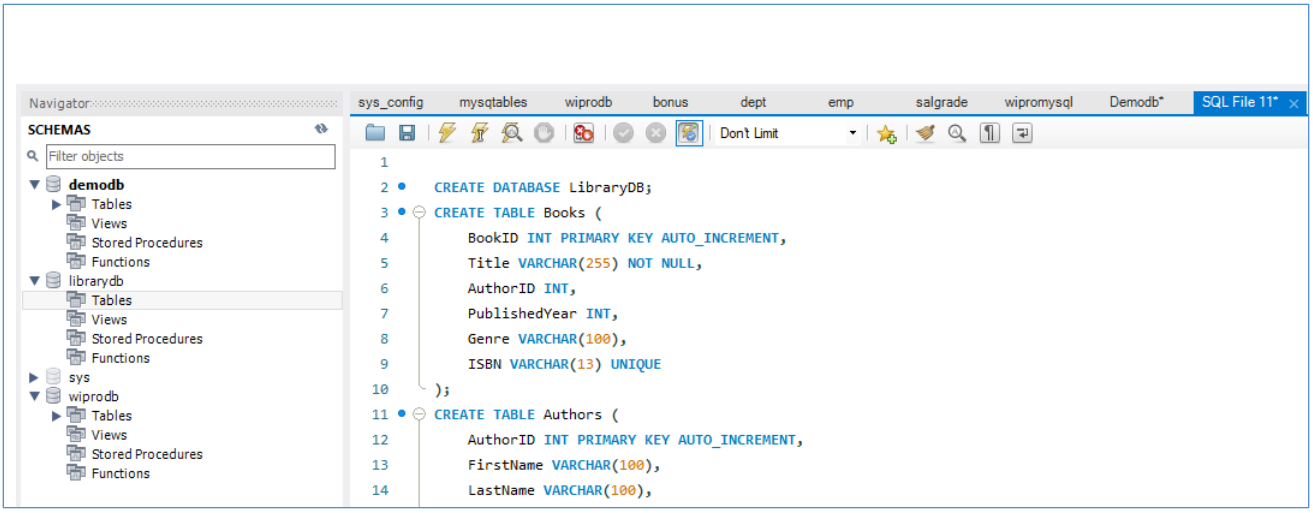
No other transactions can insert, update, or delete rows that would affect the result

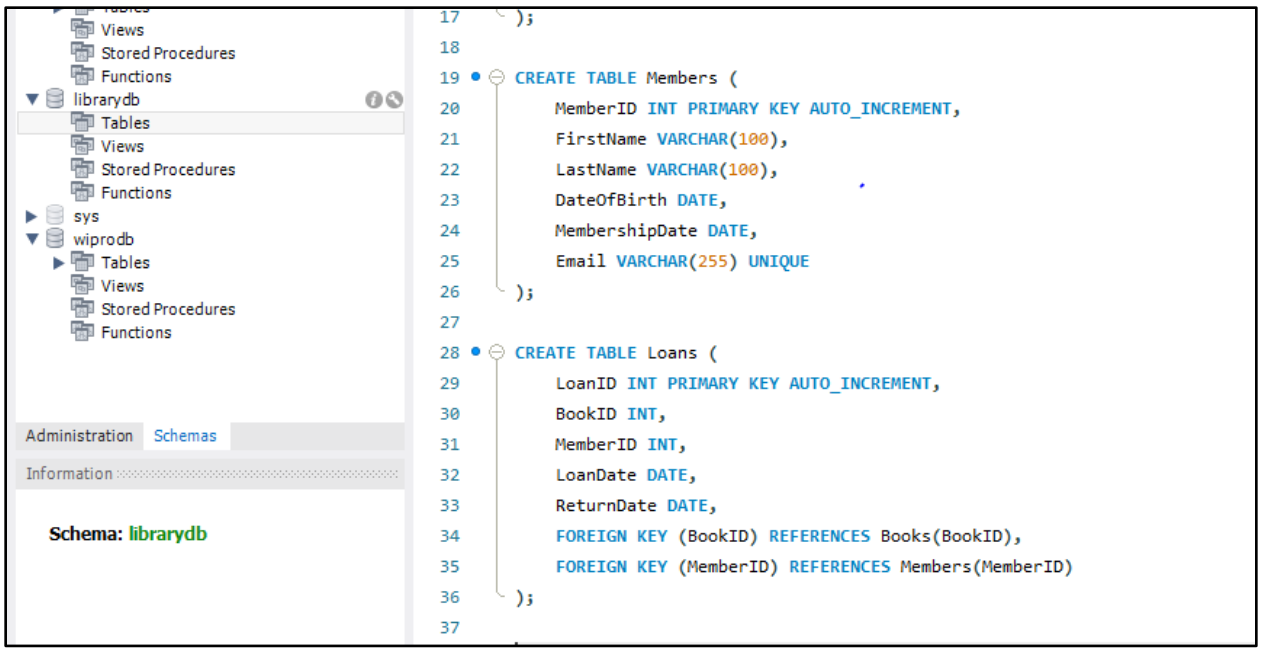
By setting different isolation levels, you can control the level of concurrency and consistency in your transactions. This is crucial for ensuring that your transactions meet the ACID properties and maintain the integrity and reliability of the database.



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.

Ans: 1: Create a New Database CREATE DATABASE LibraryDB;

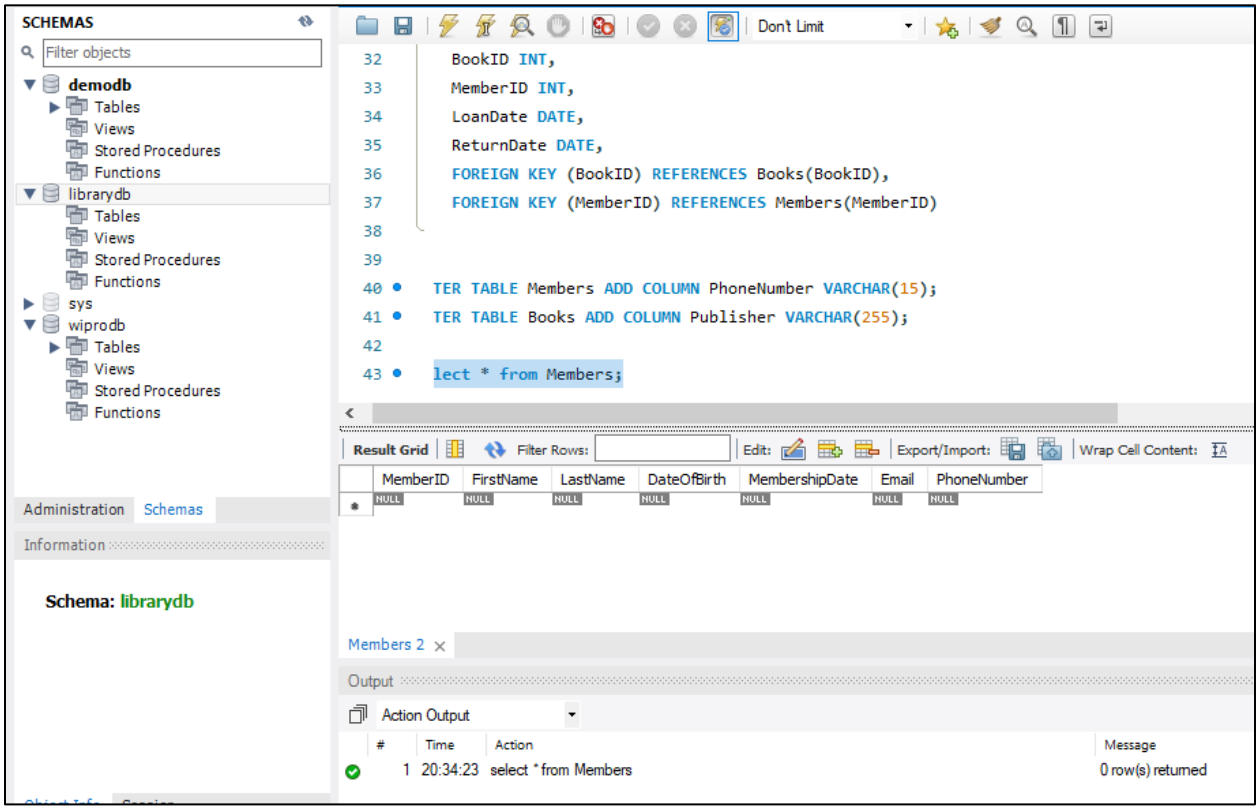


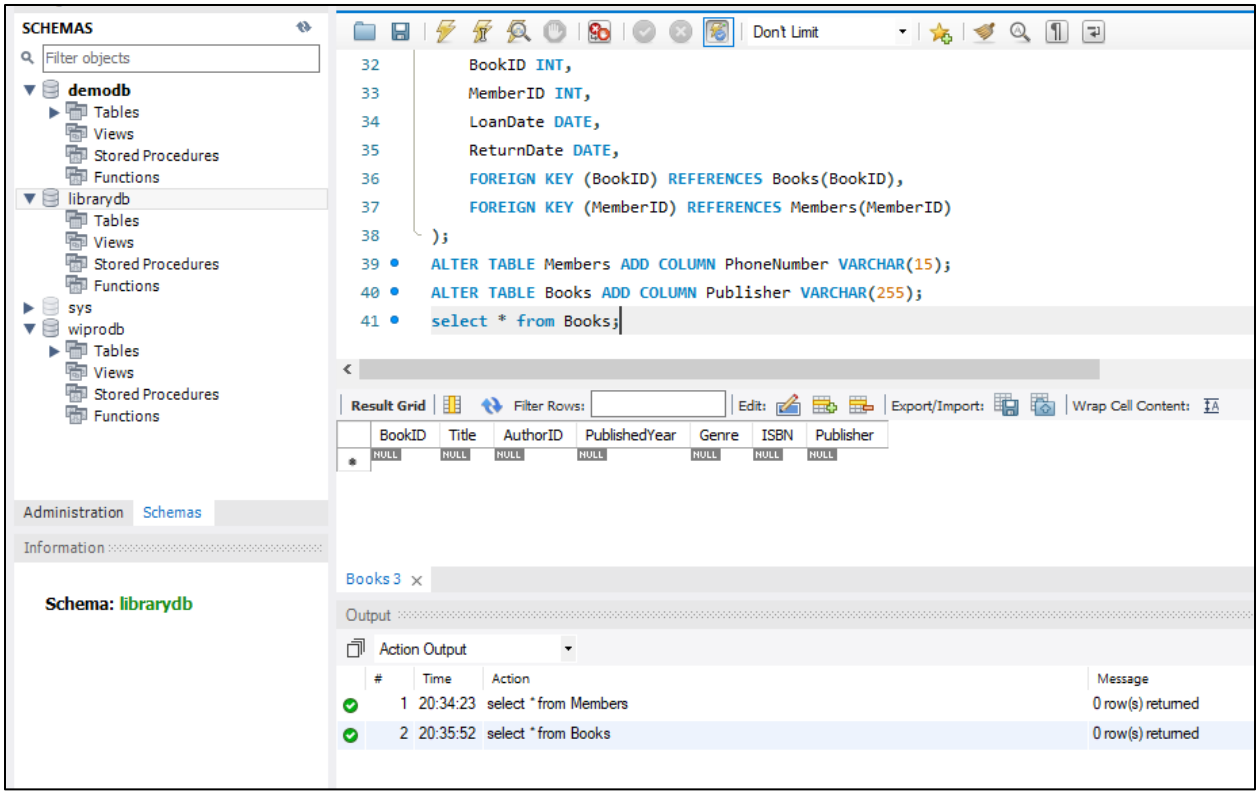


Use ALTER Statements to Modify Table Structures

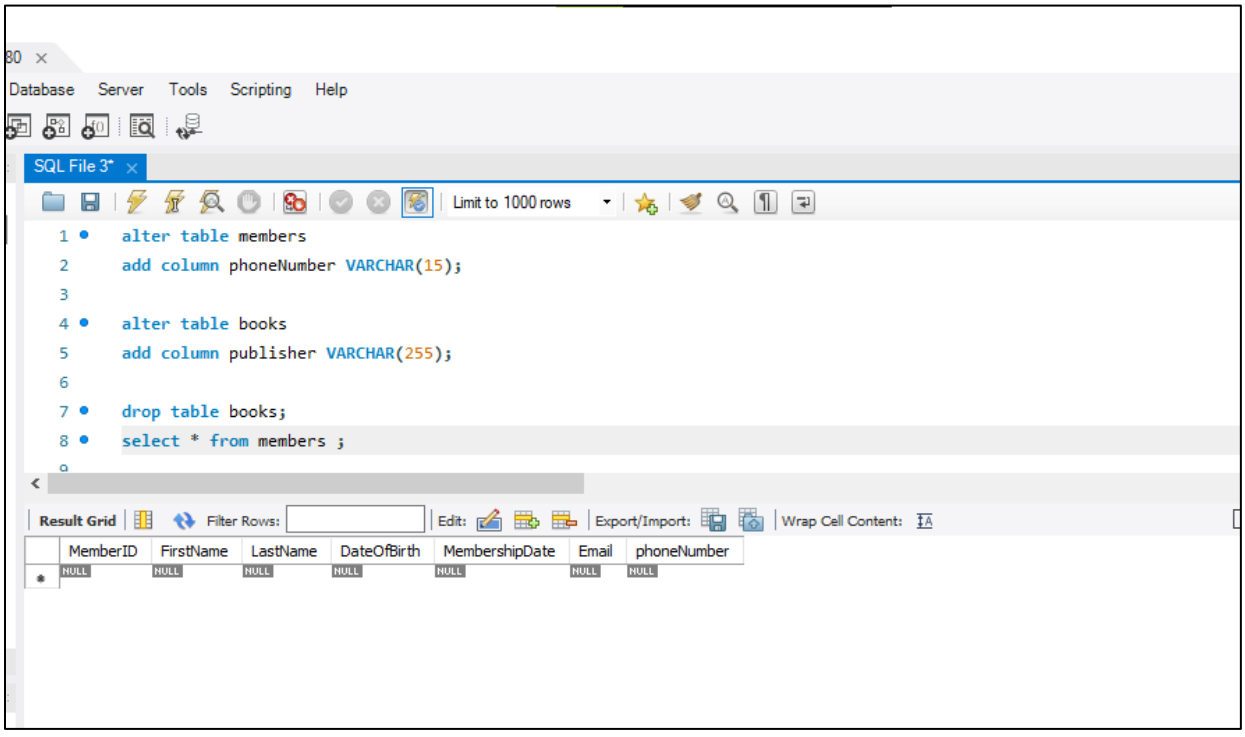
ALTER TABLE Members ADD COLUMN PhoneNumber VARCHAR(15);

ALTER TABLE Books ADD COLUMN Publisher VARCHAR(255);





DROP a Redundant Table DROP TABLE Books;

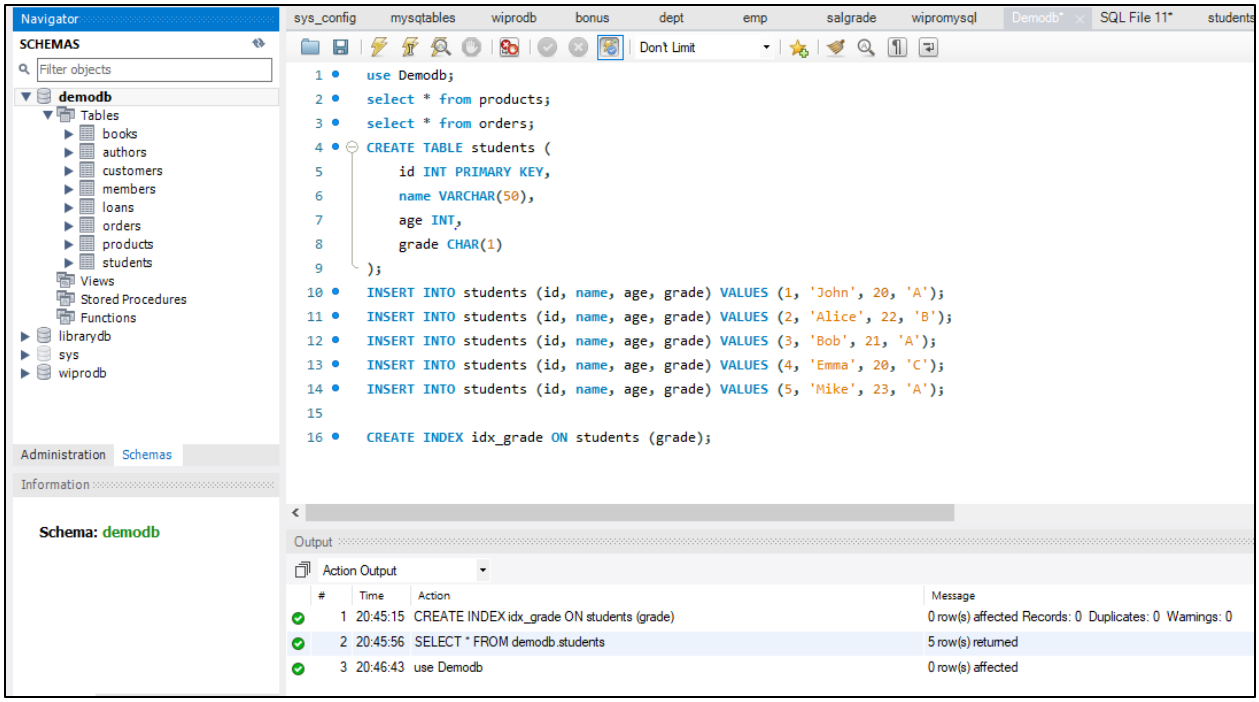


Assignment 5: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

Ans:

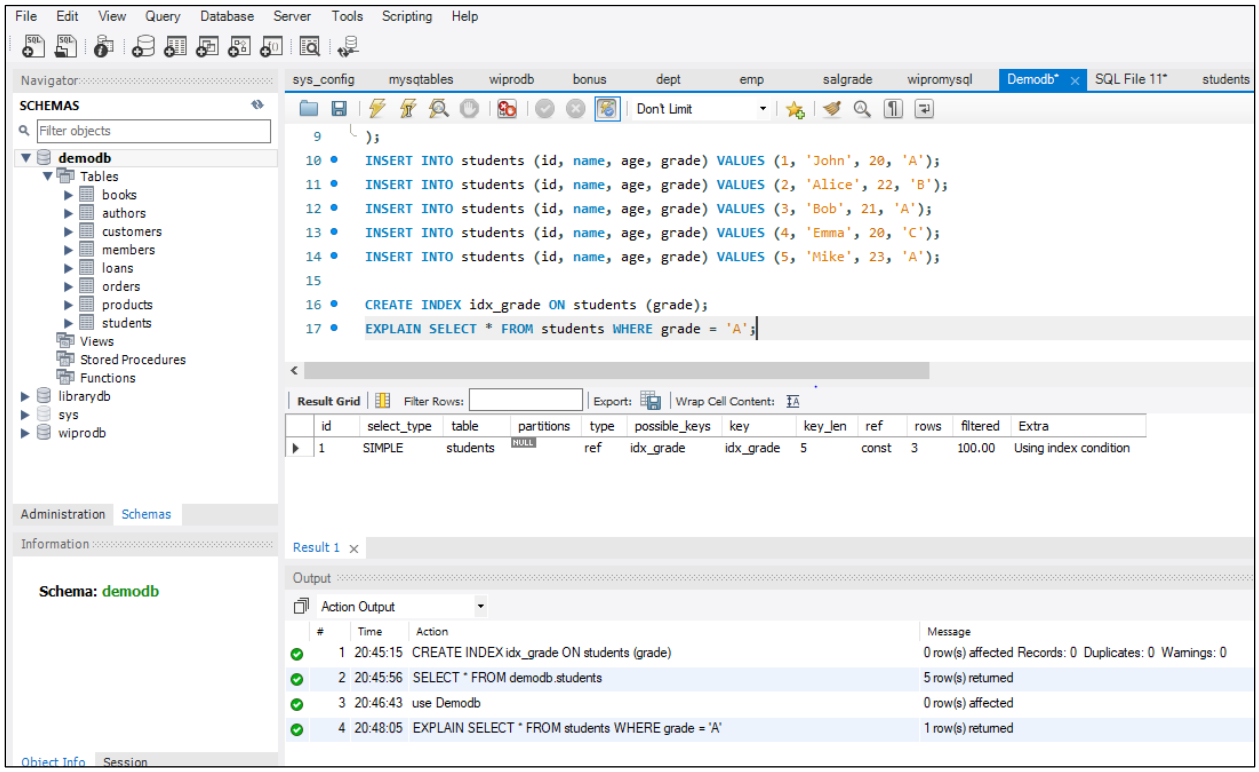
Create an index on the 'grade' column

CREATE INDEX idx_grade ON students (grade);



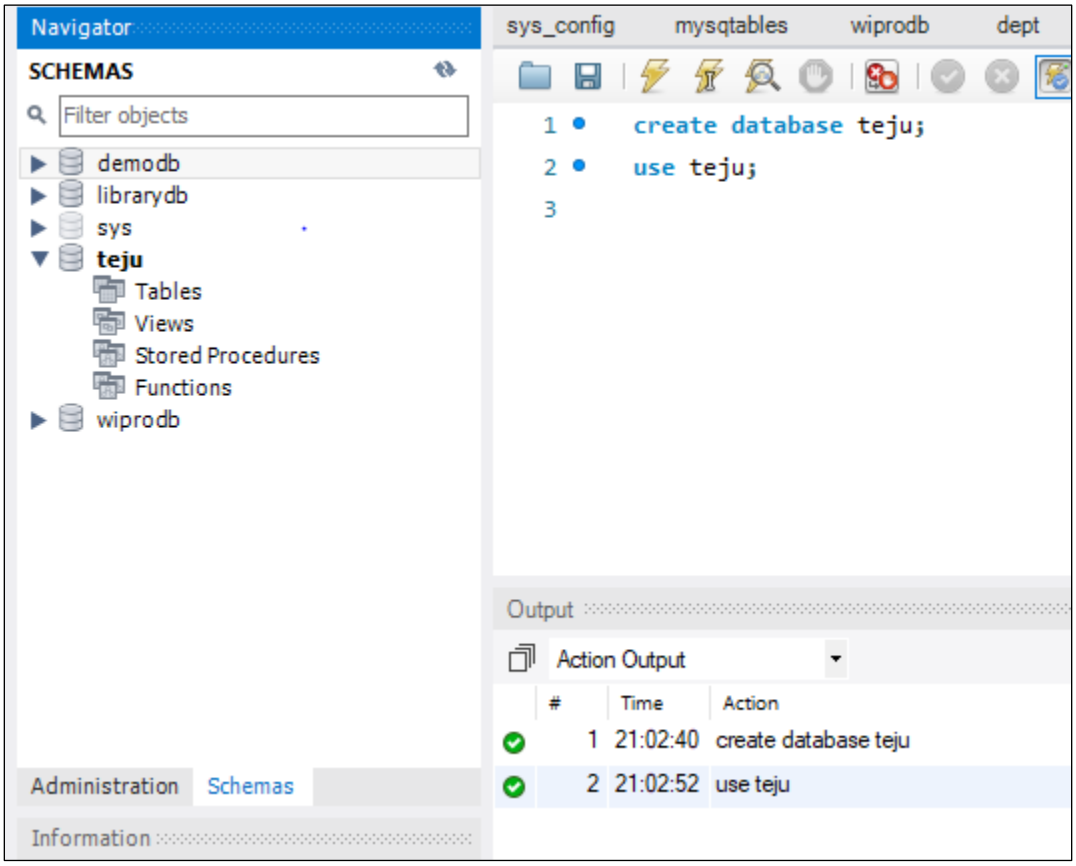
Query without index

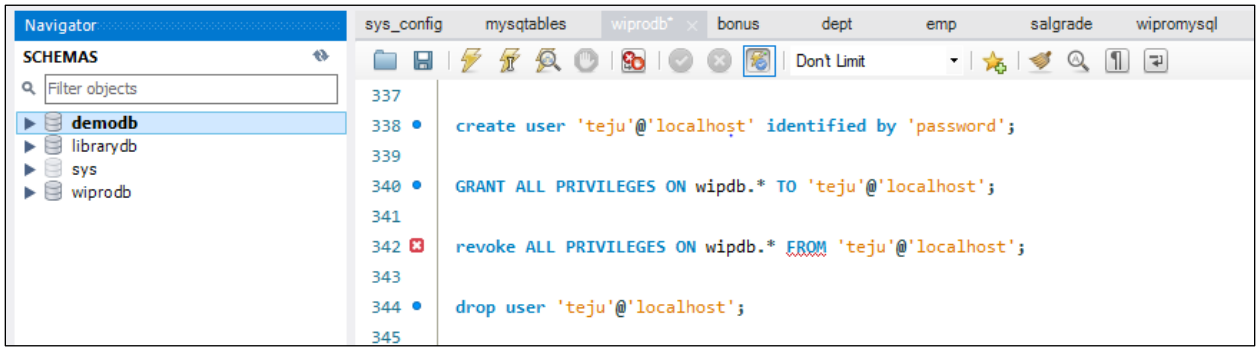
EXPLAIN SELECT * FROM students WHERE grade = 'A';

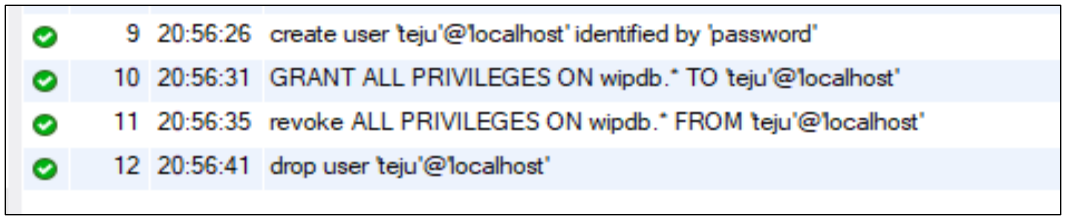


Assignment 6: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

Ans:







✓	9	20:56:26	create user 'teju'@'localhost' identified by 'password'
✓	10	20:56:31	GRANT ALL PRIVILEGES ON wipdb.* TO 'teju'@'localhost'
✓	11	20:56:35	revoke ALL PRIVILEGES ON wipdb.* FROM 'teju'@'localhost'
✓	12	20:56:41	drop user 'teju'@'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.

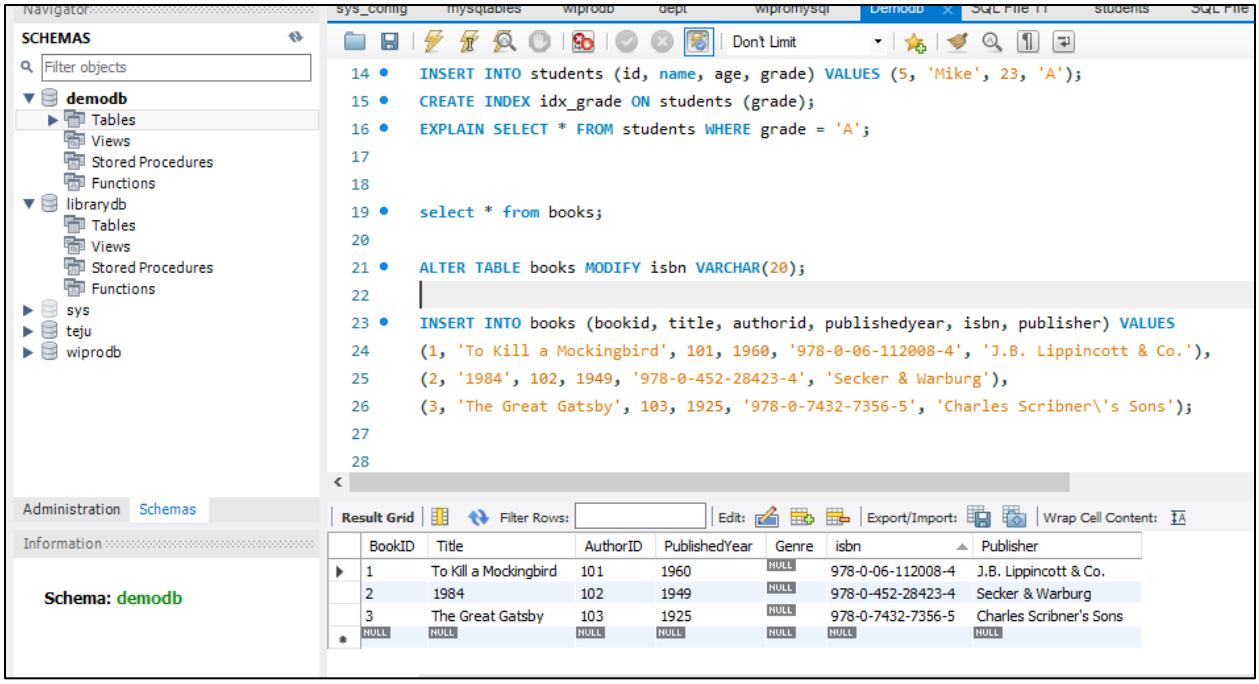
Ans:

INSERT INTO books (bookid, title, authorid, publishedyear, isbn, publisher) VALUES

(1, 'To Kill a Mockingbird', 101, 1960, '978-0-06-112008-4', 'J.B. Lippincott & Co.'),

(2, '1984', 102, 1949, '978-0-452-28423-4', 'Secker & Warburg'),

(3, 'The Great Gatsby', 103, 1925, '978-0-7432-7356-5', 'Charles Scribner\'s Sons');



Update book information

UPDATE books

SET title = 'To Kill a Mockingbird (Updated)', authorid = 104, publishedyear = 1961, isbn = '978-0-06-112008-5', publisher = 'J.B. Lippincott & Co. (Updated)'

WHERE bookid = 1;

UPDATE books

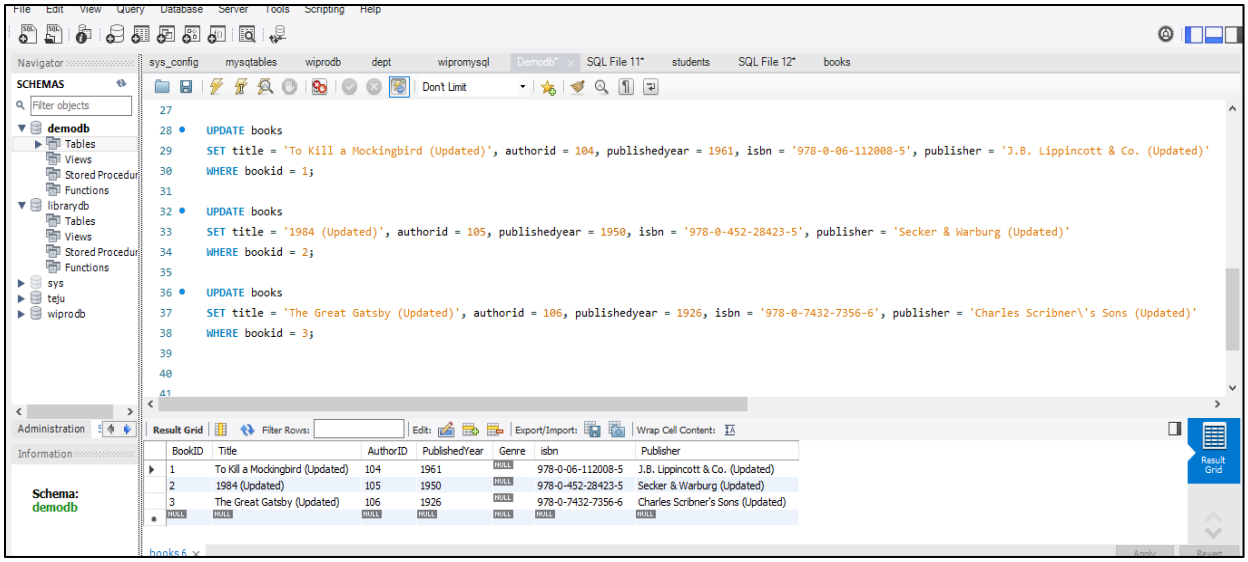
SET title = '1984 (Updated)', authorid = 105, publishedyear = 1950, isbn = '978-0-452-28423-5', publisher = 'Secker & Warburg (Updated)'

WHERE bookid = 2;

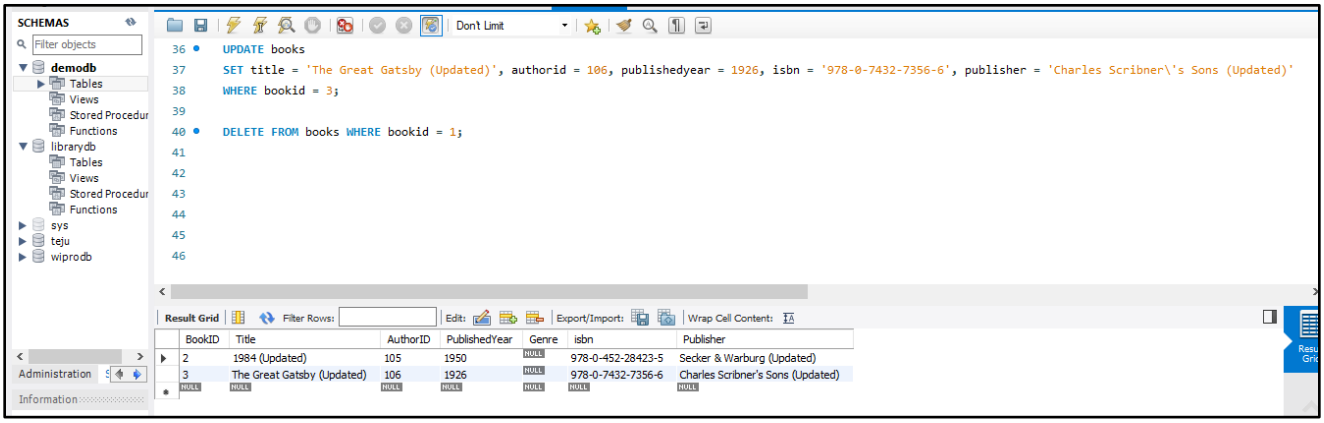
UPDATE books

SET title = 'The Great Gatsby (Updated)', authorid = 106, publishedyear = 1926, isbn = '978-0-7432-7356-6', publisher = 'Charles Scribner\'s Sons (Updated)'

WHERE bookid = 3;



Delete a book record



BULK INSERT Books

```
FROM 'C:\path\to\books.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- If the first row contains headers
);
```

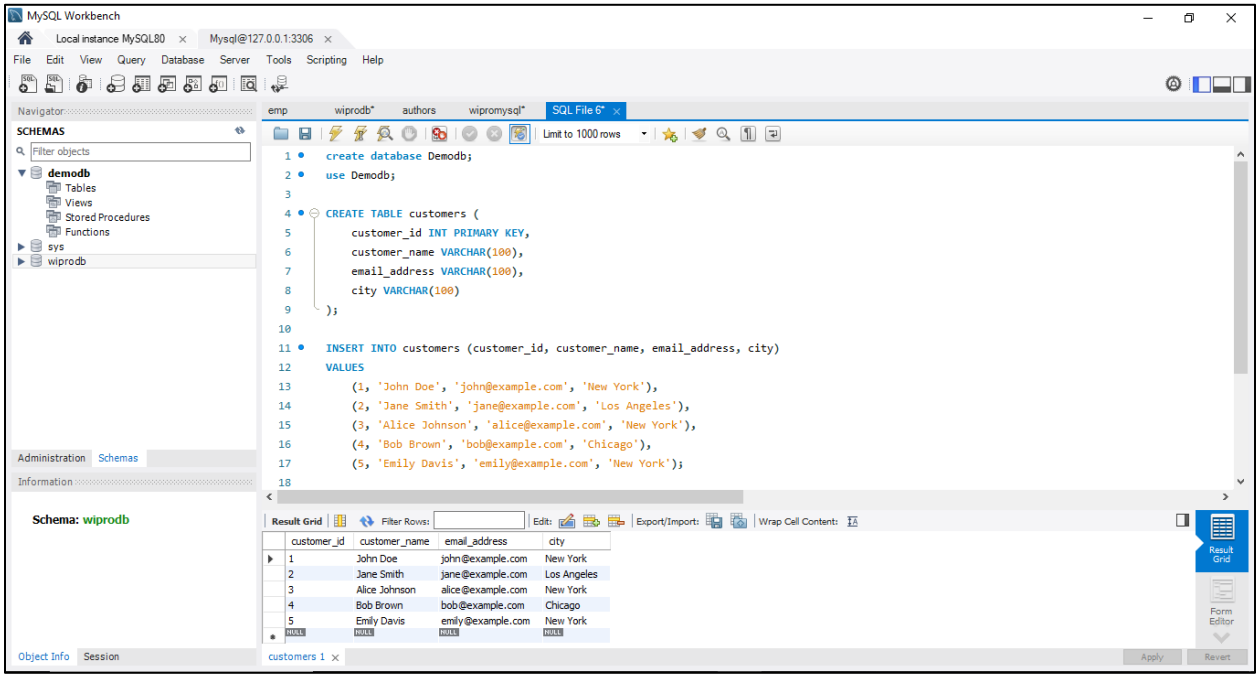
Bulk insert data into the Members table from a CSV file

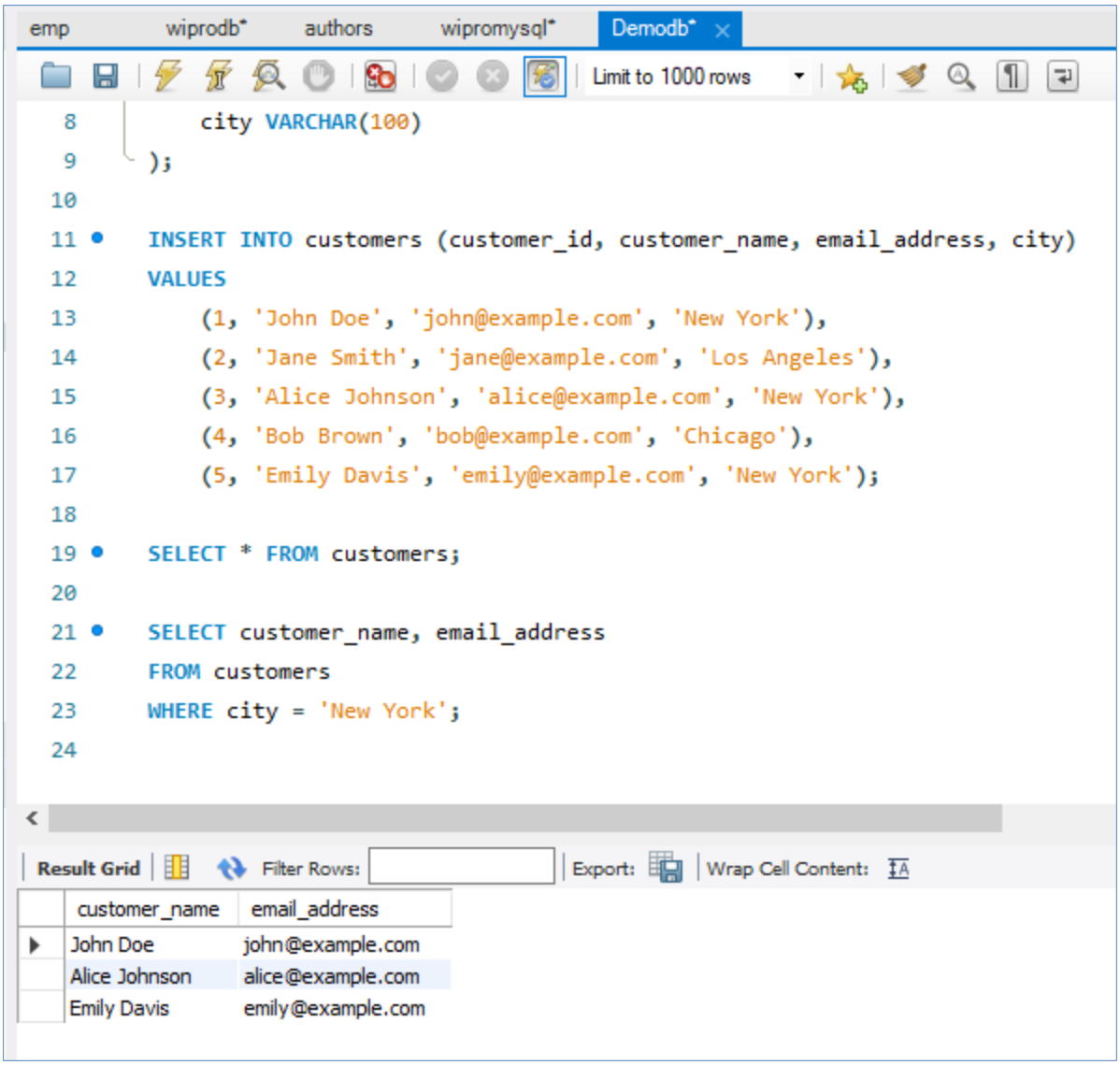
```
BULK INSERT Members
FROM 'C:\path\to\members.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- If the first row contains headers
);
```

Bulk insert data into the Loans table from a CSV file

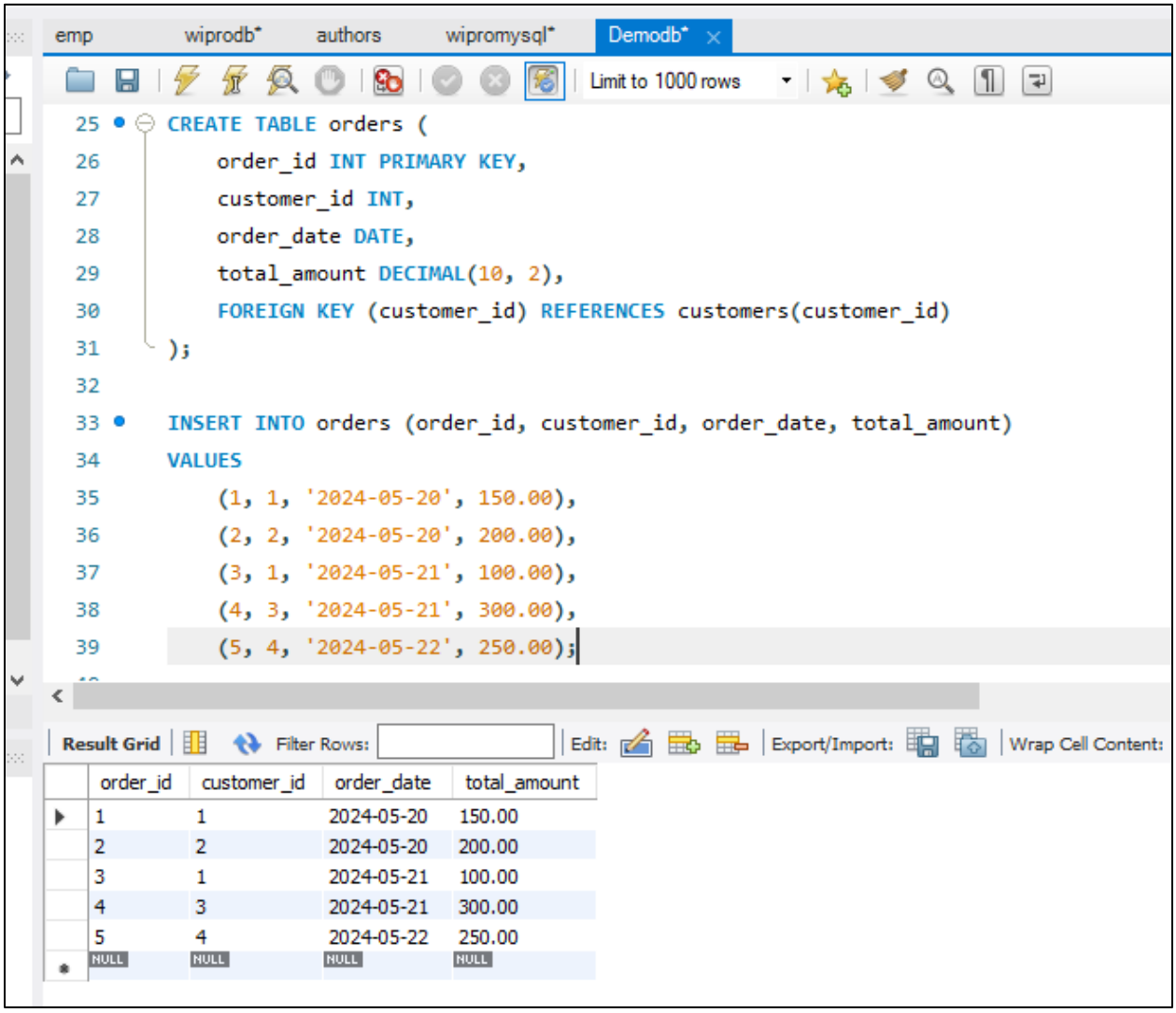
```
BULK INSERT Loans
FROM 'C:\path\to\loans.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- If the first row contains headers
);
```

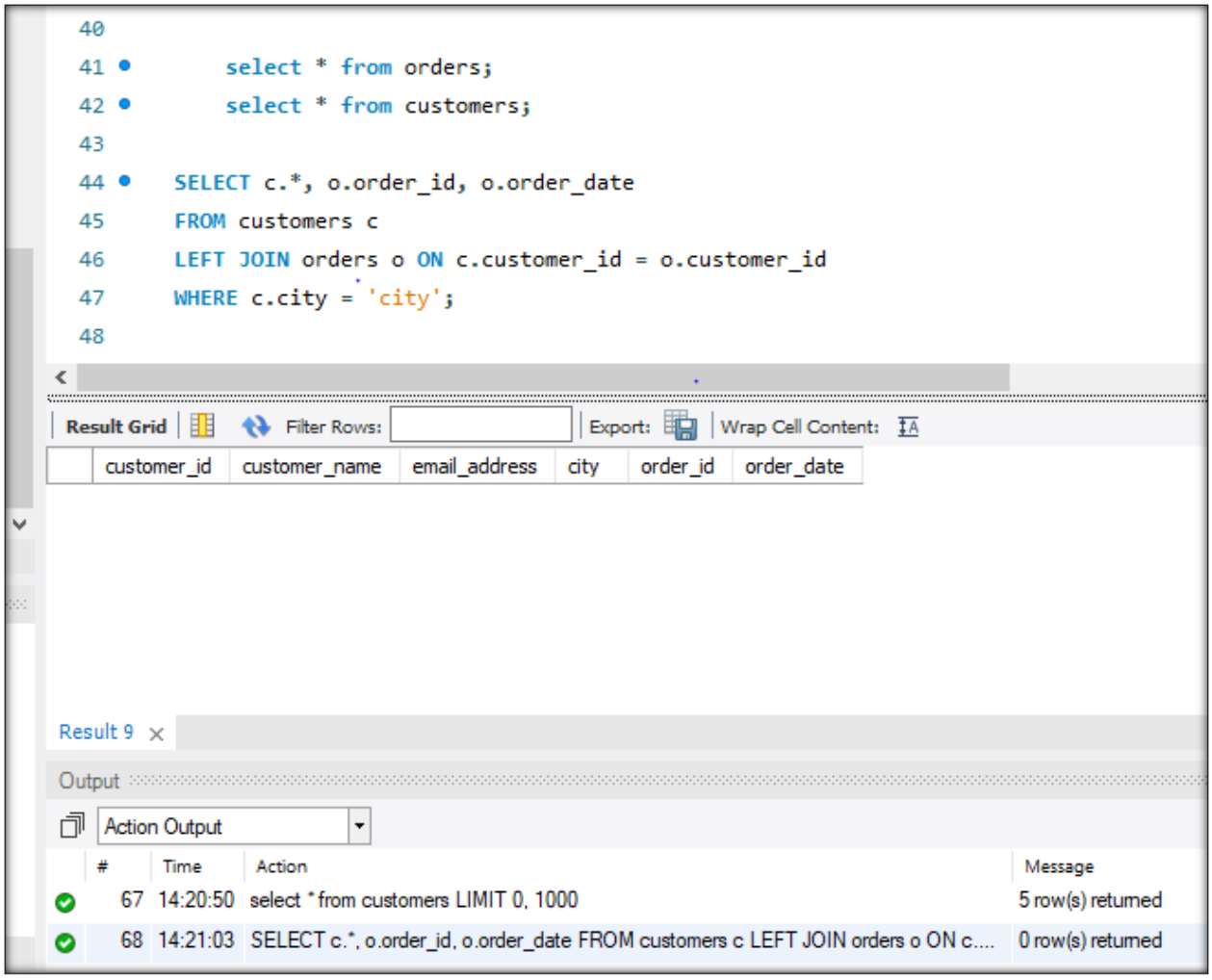
Assignment 8: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.



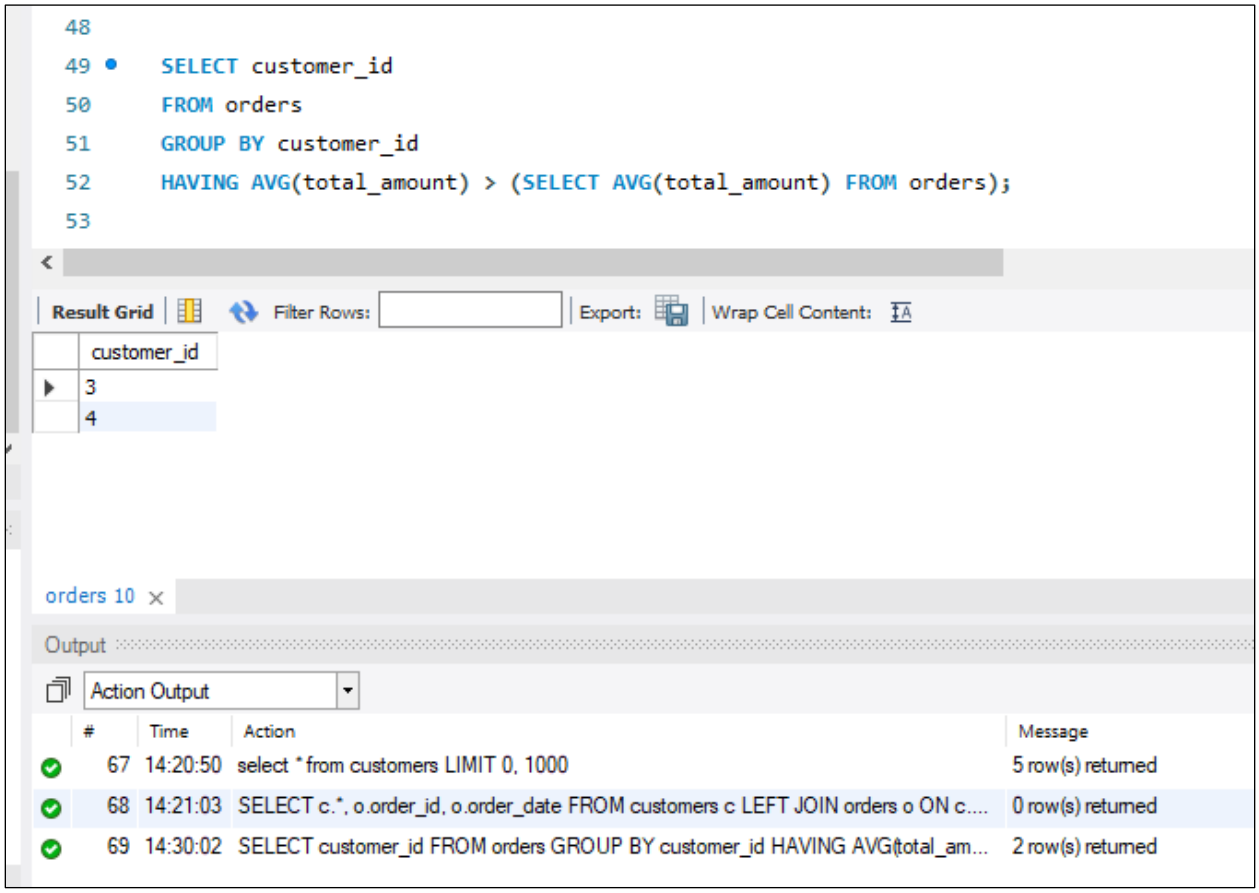

emp
wiprodb*
authors
wipromysql*
Demodb*
Limit to 1000 rows
city VARCHAR(100)
);
INSERT INTO customers (customer_id, customer_name, email_address, city)
VALUES
(1, 'John Doe', 'john@example.com', 'New York'),
(2, 'Jane Smith', 'jane@example.com', 'Los Angeles'),
(3, 'Alice Johnson', 'alice@example.com', 'New York'),
(4, 'Bob Brown', 'bob@example.com', 'Chicago'),
(5, 'Emily Davis', 'emily@example.com', 'New York');
SELECT * FROM customers;
SELECT customer_name, email_address
FROM customers
WHERE city = 'New York';
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
customer_name
email_address
John Doe
john@example.com
Alice Johnson
alice@example.com
Emily Davis
emily@example.com

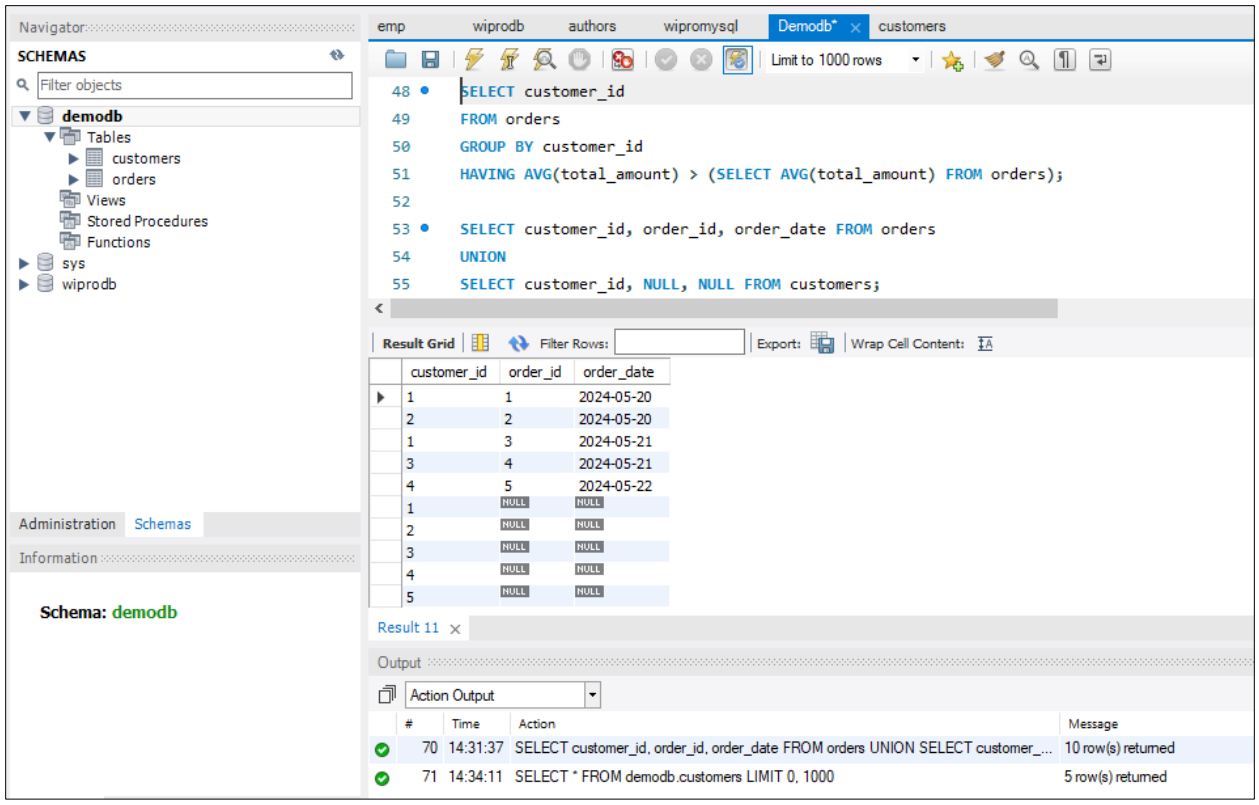
Assignment 9: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.



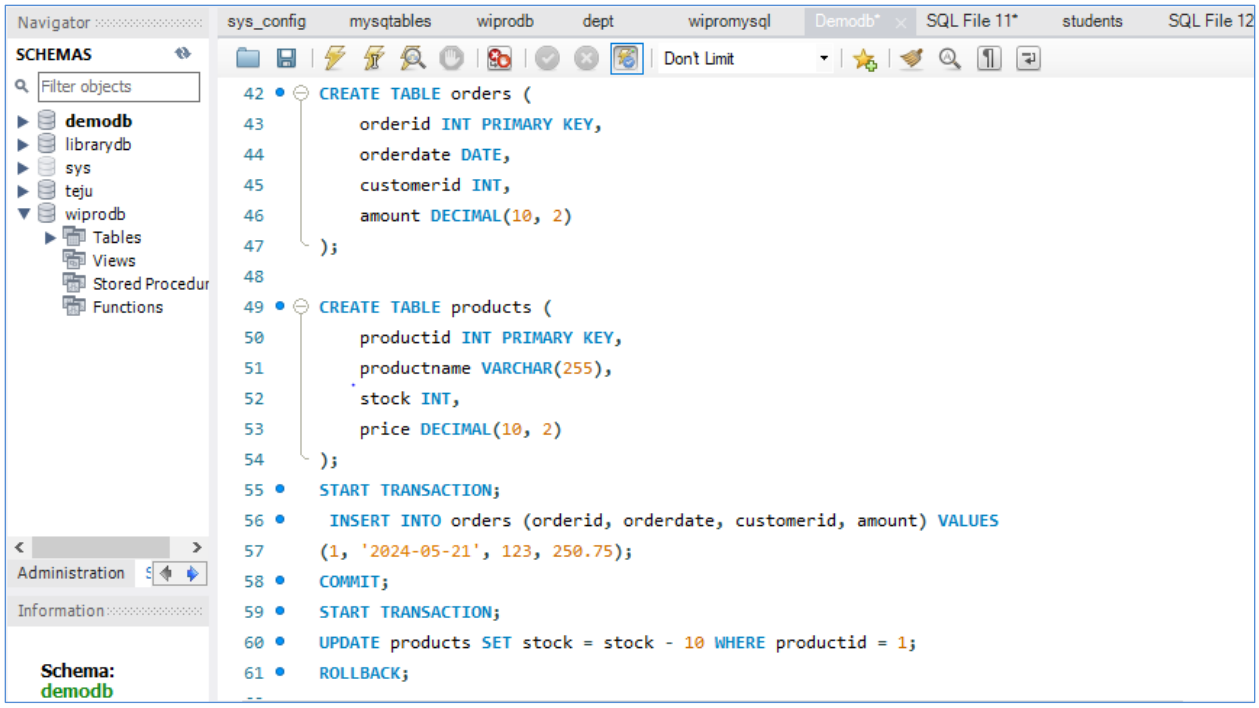

40
41 select * from orders;
42 select * from customers;
43
44 SELECT c.*, o.order_id, o.order_date
45 FROM customers c
46 LEFT JOIN orders o ON c.customer_id = o.customer_id
47 WHERE c.city = 'city';
48
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
customer_id
customer_name
email_address
city
order_id
order_date
Result 9
Output
Action Output
Time Action Message
67 14:20:50 select * from customers LIMIT 0, 1000 5 row(s) returned
68 14:21:03 SELECT c.*, o.order_id, o.order_date FROM customers c LEFT JOIN orders o ON c.... 0 row(s) returned

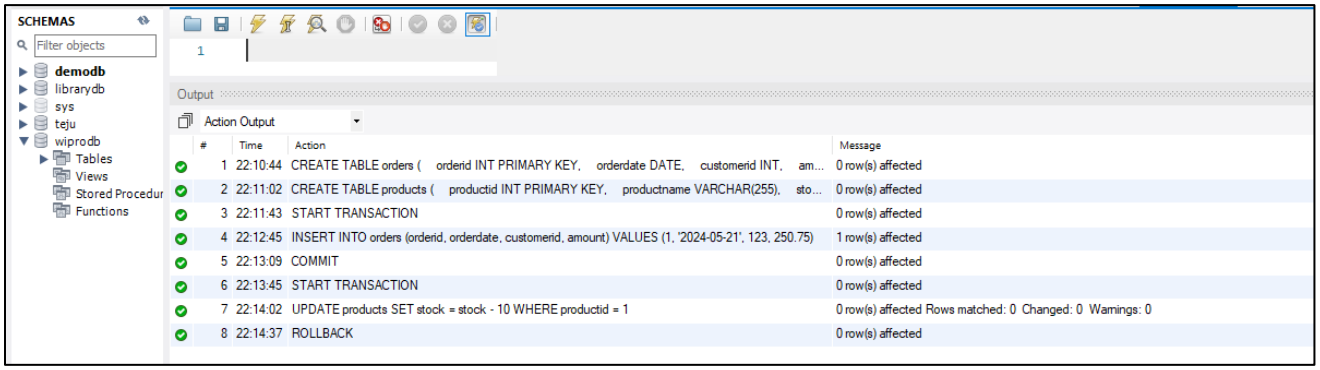
Assignment 10: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

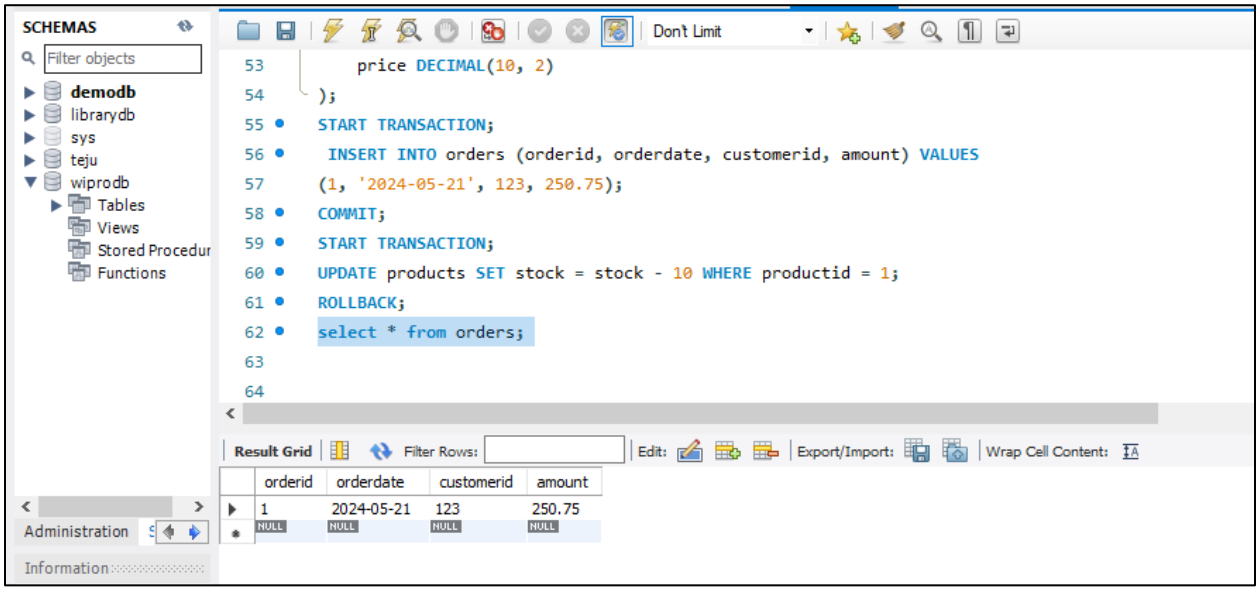




Assignment 11: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

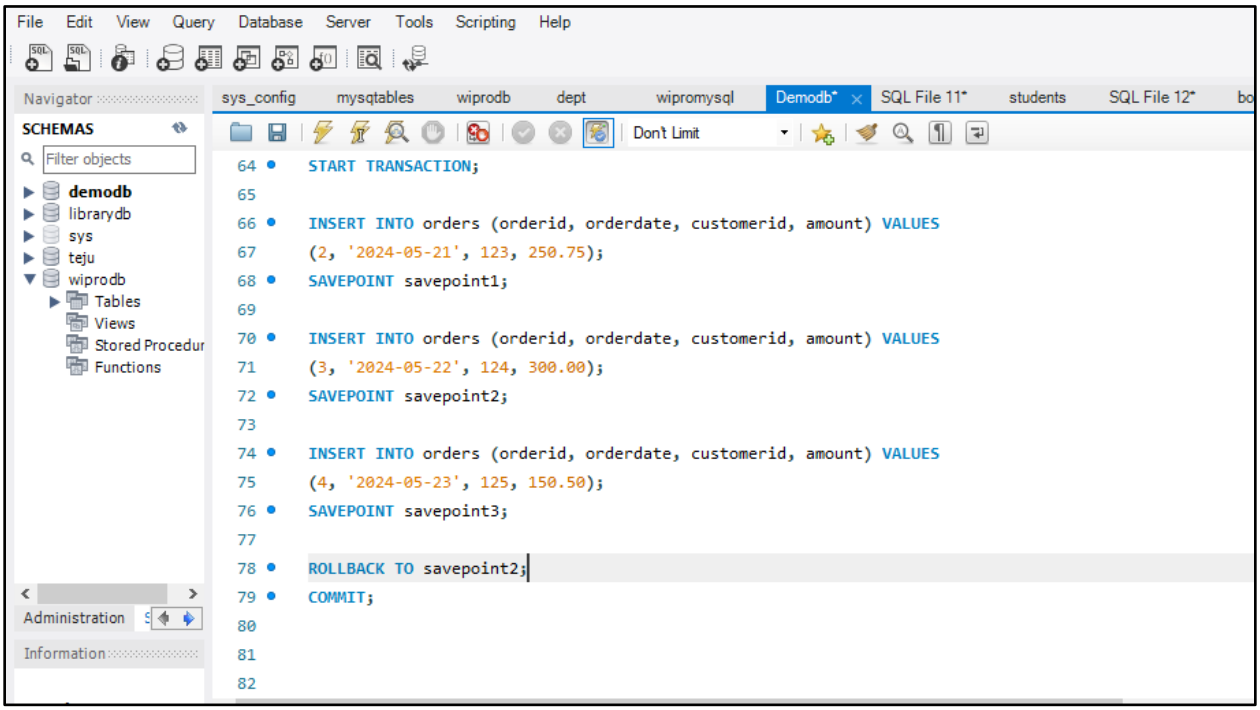

Navigator
SCHEMAS
Filter objects
demodb
librarydb
sys
teju
wiprodb
Tables
Views
Stored Procedur
Functions
Administration
Information
Schema:
demodb
sys_config
mysqtables
wiprodb
dept
wipromysql
Demodb*
SQL File 11*
students
SQL File 12
Don't Limit
42 CREATE TABLE orders (
43 orderid INT PRIMARY KEY,
44 orderdate DATE,
45 customerid INT,
46 amount DECIMAL(10, 2)
47);
48
49 CREATE TABLE products (
50 productid INT PRIMARY KEY,
51 productname VARCHAR(255),
52 stock INT,
53 price DECIMAL(10, 2)
54);
55 START TRANSACTION;
56 INSERT INTO orders (orderid, orderdate, customerid, amount) VALUES
57 (1, '2024-05-21', 123, 250.75);
58 COMMIT;
59 START TRANSACTION;
60 UPDATE products SET stock = stock - 10 WHERE productid = 1;
61 ROLLBACK;

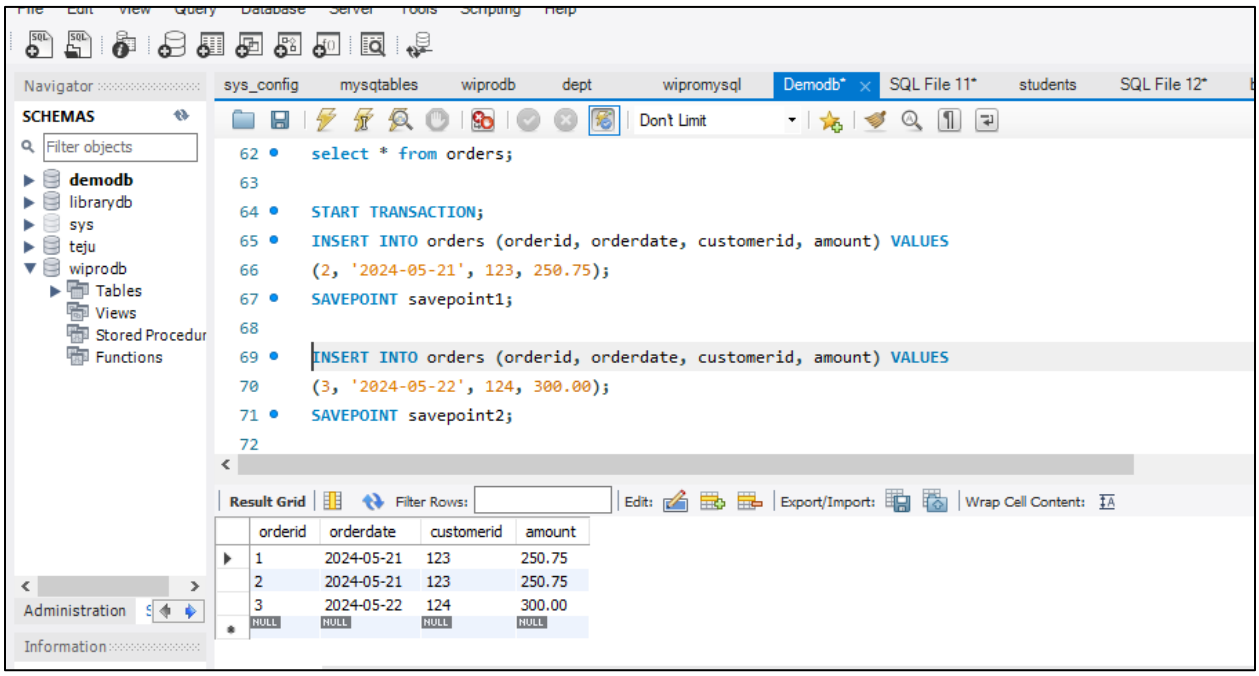

SCHEMAS
Filter objects
demodb
librarydb
sys
teju
wiprodb
Tables
Views
Stored Procedur
Functions
1
Output
Action Output
Time Action Message
1 22:10:44 CREATE TABLE orders (orderid INT PRIMARY KEY, orderdate DATE, customerid INT, am... 0 row(s) affected
2 22:11:02 CREATE TABLE products (productid INT PRIMARY KEY, productname VARCHAR(255), sto... 0 row(s) affected
3 22:11:43 START TRANSACTION 0 row(s) affected
4 22:12:45 INSERT INTO orders (orderid, orderdate, customerid, amount) VALUES (1, '2024-05-21', 123, 250.75) 1 row(s) affected
5 22:13:09 COMMIT 0 row(s) affected
6 22:13:45 START TRANSACTION 0 row(s) affected
7 22:14:02 UPDATE products SET stock = stock - 10 WHERE productid = 1 0 row(s) affected Rows matched: 0 Changed: 0 Warnings: 0
8 22:14:37 ROLLBACK 0 row(s) affected

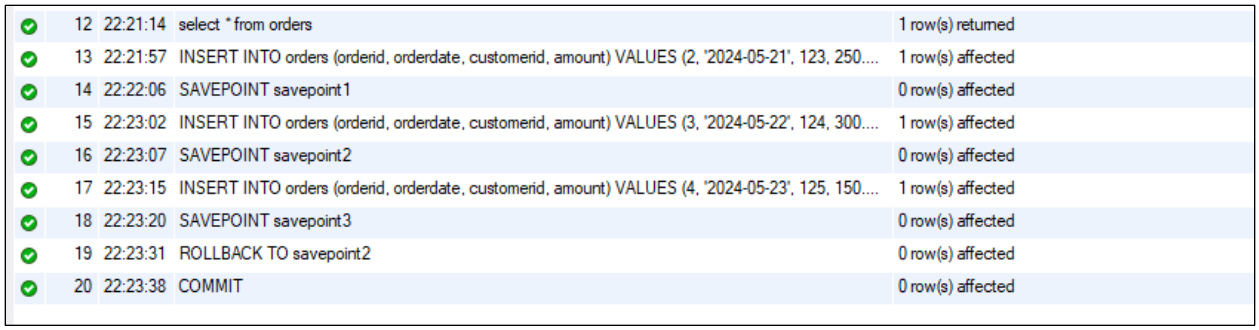

SCHEMAS
Filter objects
demodb
librarydb
sys
teju
wiprodb
Tables
Views
Stored Procedur
Functions
Administration
Information
Don't Limit
53 price DECIMAL(10, 2)
54);
55 START TRANSACTION;
56 INSERT INTO orders (orderid, orderdate, customerid, amount) VALUES
57 (1, '2024-05-21', 123, 250.75);
58 COMMIT;
59 START TRANSACTION;
60 UPDATE products SET stock = stock - 10 WHERE productid = 1;
61 ROLLBACK;
62 select * from orders;
63
64
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
orderid orderdate customerid amount
1 2024-05-21 123 250.75
NULL NULL NULL NULL

Assignment 12: 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.

Ans:







✓	12	22:21:14	select * from orders	1 row(s) returned
✓	13	22:21:57	INSERT INTO orders (orderid, orderdate, customerid, amount) VALUES (2, '2024-05-21', 123, 250....	1 row(s) affected
✓	14	22:22:06	SAVEPOINT savepoint1	0 row(s) affected
✓	15	22:23:02	INSERT INTO orders (orderid, orderdate, customerid, amount) VALUES (3, '2024-05-22', 124, 300....	1 row(s) affected
✓	16	22:23:07	SAVEPOINT savepoint2	0 row(s) affected
✓	17	22:23:15	INSERT INTO orders (orderid, orderdate, customerid, amount) VALUES (4, '2024-05-23', 125, 150....	1 row(s) affected
✓	18	22:23:20	SAVEPOINT savepoint3	0 row(s) affected
✓	19	22:23:31	ROLLBACK TO savepoint2	0 row(s) affected
✓	20	22:23:38	COMMIT	0 row(s) affected

Assignment 13 : 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.

Ans:

Report: Leveraging Transaction Logs for Data Recovery in MySQL

Introduction: Transaction logs are vital components of MySQL database management, playing a crucial role in ensuring data integrity and facilitating recovery in the event of system failures or unexpected shutdowns. These logs record every change made to a database, providing a detailed trail of transactions. This report explores the significance of transaction logs for data recovery in MySQL and illustrates their importance through a hypothetical scenario.

The Importance of Transaction Logs for Data Recovery: Transaction logs, specifically the binary logs in MySQL, serve as reliable sources of information for recovering data after a system failure. They maintain a chronological record of all events that modify the database, including inserts, updates, and deletes. By capturing changes before they are permanently written to the database, transaction logs enable the reconstruction of the database to a consistent state, even in the face of unforeseen disruptions.

Key Functions of Transaction Logs in MySQL:

1. **Redo Logging:** MySQL's binary logs capture the changes made to the database, allowing for the replay of transactions that were committed but not yet written to disk at the time of the failure. This process, known as redo logging, ensures that committed transactions are not lost during recovery.

2. **Undo Logging:** InnoDB, the default storage engine for MySQL, uses undo logs to store information necessary to reverse or undo transactions that were in progress but not yet committed at the time of the failure. This capability enables the restoration of the database to its pre-transaction state, maintaining data consistency.

3. **Point-in-Time Recovery:** MySQL's binary logs enable point-in-time recovery, allowing database administrators to restore the database to a specific moment before the failure occurred. By replaying transactions up to the desired timestamp, organizations can minimize data loss and maintain business continuity.

Hypothetical Scenario: Consider a scenario where an e-commerce platform experiences an unexpected shutdown of its MySQL database server due to a hardware failure. As a result, critical customer order data becomes inaccessible, posing a significant operational risk. However, due to the diligent use of MySQL binary logs, the organization can recover the data swiftly and minimize the impact on its operations.

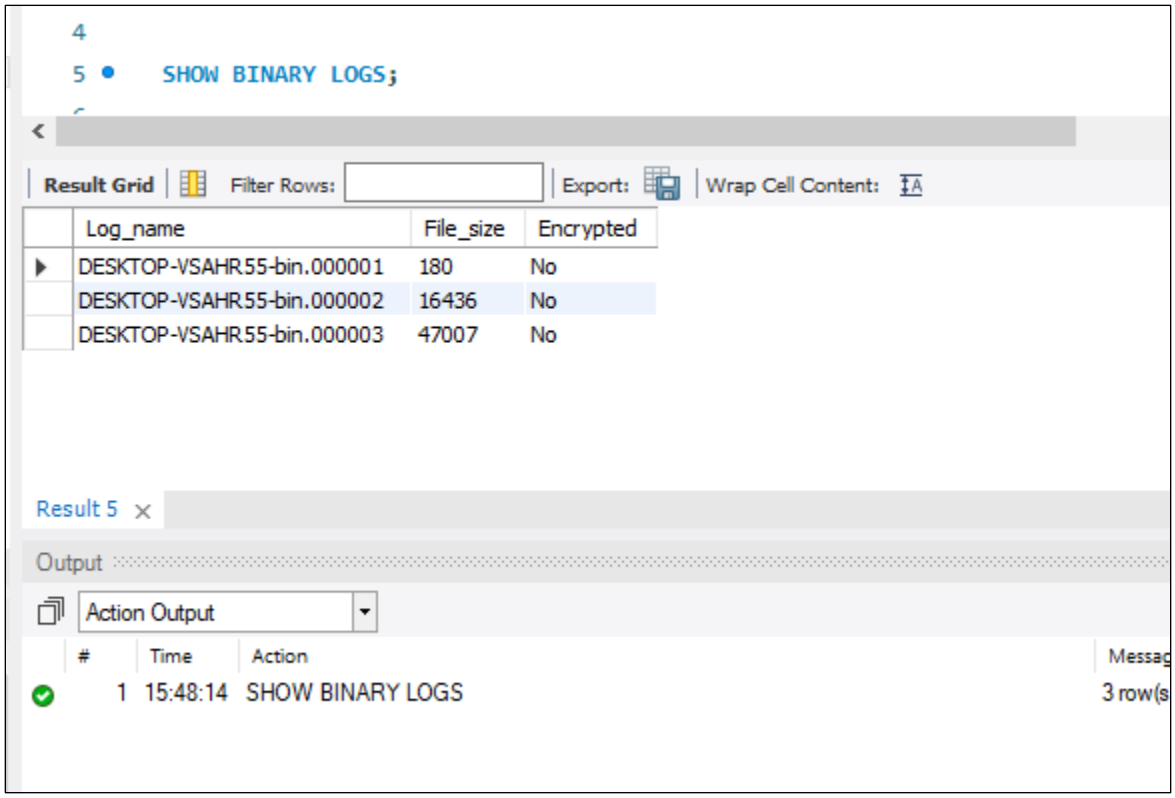
- **Event:** The database server abruptly shuts down, leading to the loss of unsaved changes and potentially jeopardizing the integrity of customer order records.

- **Response:** Upon restarting the database server, database administrators immediately initiate the recovery process using MySQL binary logs. By applying the binary logs, the system reconstructs the database to a consistent state, ensuring that all committed transactions are preserved.

Outcome: Despite the unexpected shutdown, the e-commerce platform successfully restores access to critical customer order data with minimal data loss. The MySQL binary logs prove instrumental in facilitating rapid recovery, demonstrating their indispensable role in ensuring data resilience and business continuity.

Step-by-Step Recovery Process:

1. **Identify the Binary Log Files:** Locate the binary log files generated by MySQL before the crash.


4
5 SHOW BINARY LOGS;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Log_name
File_size
Encrypted
DESKTOP-VSAHR55-bin.000001
180
No
DESKTOP-VSAHR55-bin.000002
16436
No
DESKTOP-VSAHR55-bin.000003
47007
No
Result 5
Output
Action Output
#
Time
Action
Messag
1
15:48:14
SHOW BINARY LOGS
3 row(s