Shubham P. Kumbhar

Wipro Assignments

RDBMS Fundamental

Assignment 1 To 7

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: →

To create an Entity-Relationship (ER) diagram for a Library Management System , we will define the entities, their attributes, relationships, and apply normalization up to the third normal form (3NF). This approach ensures data integrity and minimizes redundancy.

1. Entities and Attributes

1. Publisher:

- Attributes :
  - `Pub_ID` (Primary Key)
  - `Name`
  - `Address`

2. Book:

- Attributes:
  - `Books_Id` (Primary Key)
  - `Title`
  - `Author`
  - `Price`
  - `Available` (Quantity available)
  - `Pub_ID` (Foreign Key)

3. Member:

- Attributes:

- `Memb_id` (Primary Key)

- `Name`

- `Address`

- `Member_type`

- `Memb_date` (Membership start date)

- `Expiry_date`

4. Borrowed_By:

- Attributes

- `Issue` (Issue date)

- `Due_Date`

- `Return_Date`

2. Relationships and Cardinality

1. Publisher - Publishes - Book:

- Relationship: A `Publisher` can publish multiple `Books`, but each `Book` is published by one `Publisher`.

- Cardinality: One-to-Many (`1:N`).

2. Member - Borrows - Book:

- Relationship: A `Member` can borrow multiple `Books`, and each `Book` can be borrowed by multiple `Members` over time. This many-to-many relationship is managed through the `Books_Borrowed_By` entity.

- Cardinality: Many-to-Many (`M:N`), resolved through the `Books_Borrowed_By` entity, resulting in two One-to-Many (`1:N`) relationships:

- One `Member` can have multiple `Books_Borrowed_By` records.

- One `Book` can have multiple `Books_Borrowed_By` records.

3. Normalization

The attributes and relationships are organized to comply with the third normal form (3NF):

- 1NF (First Normal Form): All attributes are atomic, and each entity has a unique primary key.

- 2NF (Second Normal Form: All non-key attributes are fully dependent on the primary key.

- 3NF (Third Normal Form): There are no transitive dependencies, and non-key attributes do not depend on other non-key attributes.

4. ER Diagram Design

Here is a description of how the ER diagram will look:

1. Entities and Attributes:

- Publisher: A rectangle labeled "Publisher" containing `Pub_ID`, `Name`, and `Address`.

- Book: A rectangle labeled "Book" containing `Books_Id`, `Title`, `Author`, `Price`, `Available`, and `Pub_ID` (as a foreign key).

- Member: A rectangle labeled "Member" containing `Memb_id`, `Name`, `Address`, `Member_type`, `Memb_date`, and `Expiry_date`.

- Borrowed_By: A rectangle labeled "Borrowed_By" containing `Borrow_ID`, `Books_Id` (foreign key), `Memb_id` (foreign key), `Issue`, `Due_Date`, and `Return_Date`.

2. Relationships:

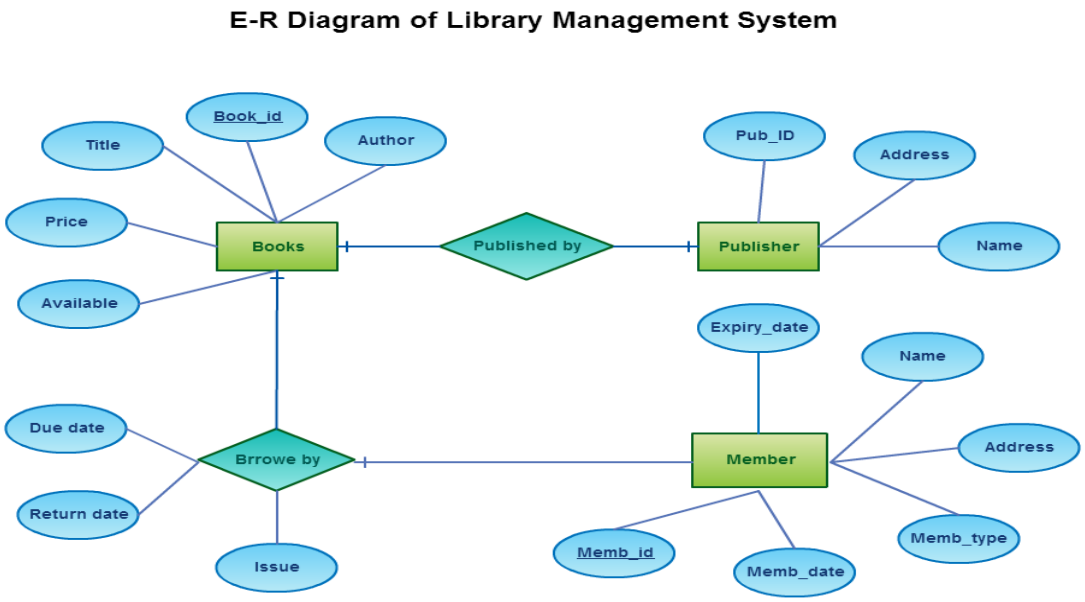
- A line connecting `Publisher` to `Book` with a `1:N` cardinality notation (indicating that one publisher can publish many books).

- A line connecting `Book` to `Books_Borrowed_By` with a `1:N` cardinality notation (indicating that one book can be borrowed multiple times).

- A line connecting `Member` to `Books_Borrowed_By` with a `1:N` cardinality notation (indicating that one member can borrow multiple books).

Summary

This ER diagram effectively models a Library Management System by clearly defining the relationships between books, members, publishers, and borrowing transactions. The entities are normalized to avoid data redundancy and ensure data integrity, up to the third normal form. The use of primary and foreign keys effectively links entities, and the diagram illustrates the cardinality of relationships, providing a clear and concise view of the system's structure.



Assignment 2:

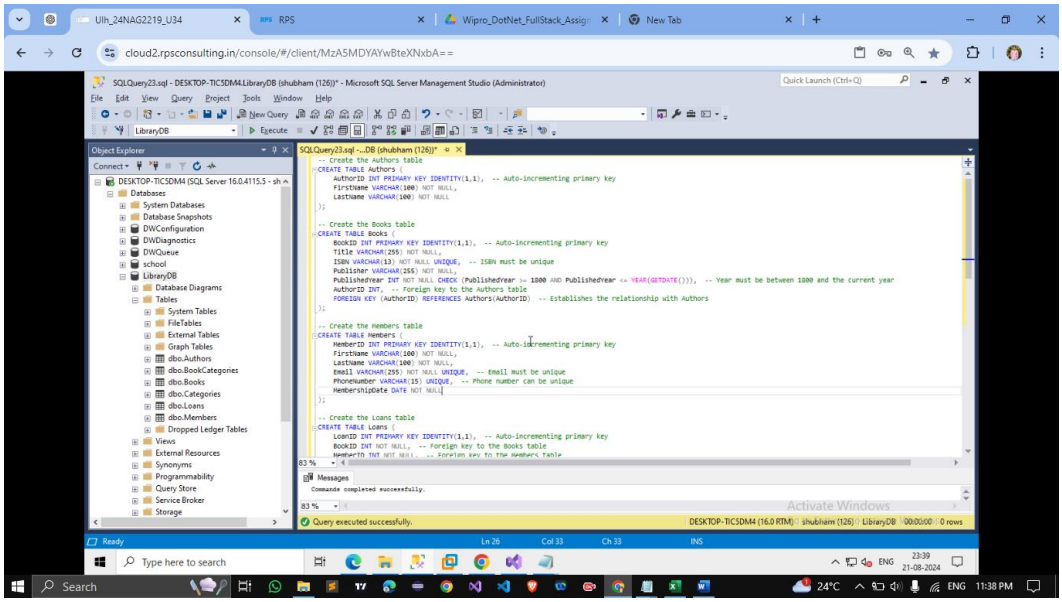
Design a database schema for a library system, including tables, fields,

and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

Answer:

```
-- Create the Authors table
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL
);

-- Create the Books table
CREATE TABLE Books (
    BookID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    Title VARCHAR(255) NOT NULL,
    ISBN VARCHAR(13) NOT NULL UNIQUE,  -- ISBN must be unique
    Publisher VARCHAR(255) NOT NULL,
    PublishedYear INT NOT NULL CHECK (PublishedYear >= 1800 AND PublishedYear <= YEAR(GETDATE())),  -- Year must be between 1800 and the current year
    AuthorID INT,  -- Foreign key to the Authors table
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)  -- Establishes the relationship with Authors
);

-- Create the Members table
CREATE TABLE Members (
    MemberID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Email VARCHAR(255) NOT NULL UNIQUE,  -- Email must be unique
    PhoneNumber VARCHAR(15) UNIQUE,  -- Phone number can be unique
    MembershipDate DATE NOT NULL
);

-- Create the Loans table
CREATE TABLE Loans (
```

```
    LoanID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    BookID INT NOT NULL,  -- Foreign key to the Books table
    MemberID INT NOT NULL,  -- Foreign key to the Members table
    LoanDate DATE NOT NULL,
    DueDate DATE NOT NULL ,-- Due date must be after the loan date
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),  -- Establishes the relationship with Books
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)  -- Establishes the relationship with Members
);

-- Create the Categories table
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    CategoryName VARCHAR(100) NOT NULL UNIQUE  -- Category name must be unique
);

-- Create the BookCategories table (many-to-many relationship between Books and Categories)
CREATE TABLE BookCategories (
    BookID INT NOT NULL,  -- Foreign key to the Books table
    CategoryID INT NOT NULL,  -- Foreign key to the Categories table
    PRIMARY KEY (BookID, CategoryID),  -- Composite primary key
    FOREIGN KEY (BookID) REFERENCES Books(BookID),  -- Establishes the relationship with Books
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)  -- Establishes the relationship with Categories
);
```



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 in SQL Server

ACID Properties in SQL Server ensure Data Integrity during a transaction. The ACID is an acronym for Atomicity, Consistency, Isolation, Durability.

- Atomicity: The atomicity property. It means either all the operations (insert, update, delete) inside a transaction take place or none. Or you can say, all the statements (insert, update, delete) inside a transaction are either completed or rolled back.
- Consistency: This ensures database consistency. It means that whatever happens in the middle of the transaction, this will never leave your database in a half-completed state.
    - If the transaction is completed successfully, then it will apply all the changes to the database.
    - If there is an error in a transaction, then all the changes that have already been made will be rolled back automatically. It means the database will restore to its state before the transaction starts.
    - If there is a system failure in the middle of the transaction, all the changes already made will automatically roll back.
- Isolation: Every transaction is individual, and One transaction can't access the result of other transactions until the transaction completes. Or, you can't perform the same operation using multiple transactions at the same time. We will explain this in a separate article.
- Durability: Once the transaction is completed, then the changes it has made to the database will be permanent. Even if there is a system failure or any abnormal changes also, this property will safeguard the committed data.

- ACID Properties in SQL Server Example
- We are going to use Dim products and Sales table to explain the Sql Server ACID properties. The below screenshot will show you the data inside DimProduct table

```
USE [SQLTEST]
GO
SELECT [ProductKey]
      ,[EnglishProductName]
      ,[Color]
      ,[StandardCost]
      ,[ListPrice]
      ,[DealerPrice]
      ,[StockLevel]
  FROM [DimProduct]
```

100 %

Results | Messages

	ProductKey	EnglishProductName	Color	StandardCost	ListPrice	DealerPrice	StockLevel
1	212	Sport-100 Helmet, Red	Red	12.0278	33.6442	20.1865	10000
2	213	Long-Sleeve Logo Jersey, S	Multi	31.7244	48.0673	28.8404	5000
3	214	HL Road Frame - Red, 62	Red	747.9682	1263.4598	758.0759	3000
4	215	LL Road Frame - Black, 60	Black	204.6251	337.22	202.332	4000
5	216	Road-650 Black, 60	Black	486.7066	33.6442	20.1865	5000

and the data inside a sales table is:

```
USE [SQLTEST]
GO
SELECT [ProductKey]
      ,[OrderQuantity]
      ,[UnitPrice]
      ,[SalesAmount]
  FROM [Sales]
```

100 %

Results | Messages

	ProductKey	OrderQuantity	UnitPrice	SalesAmount
1	212	1	20.1865	20.1865
2	212	1	20.1865	20.1865
3	213	200	48.0673	9613.46

For this demonstration, Whenever Sales happen, then we have to update the Stock Level based on the order Quantity. For example, if A orders ten products (product key = 216), then update the stock level to 4990 and insert a new record in the sales table.

Atomicity in SQL ACID

It means all the statements inside a transaction should either succeed or fail as a unit. To demonstrate this SQL Atomicity Acid property, we are using the one UPDATE and an INSERT statement inside a transaction.

```
USE [SQLTEST]
GO
BEGIN TRANSACTION
      UPDATE [DimProduct]
            SET [StockLevel] = 4700
            WHERE [ProductKey] = 213

      INSERT INTO [Sales] ([ProductKey], [OrderQuantity], [UnitPrice], [SalesAmount])
      VALUES (213, 300, 48.0673, 48.0673 * 300)
COMMIT TRANSACTION
```

```
USE [SQLTEST]
GO

BEGIN TRANSACTION
    UPDATE [DimProduct]
        SET [StockLevel] = 4700
        WHERE [ProductKey] = 213

    INSERT INTO [Sales] ([ProductKey], [OrderQuantity], [UnitPrice], [SalesAmount])
    VALUES (213, 300, 48.0673, 48.0673 * 300)
COMMIT TRANSACTION
```

100 %

Messages

```
(1 row(s) affected)

(1 row(s) affected)
```

Let me show you the records in DimProduct, and Sales tables after that transaction.

```
USE [SQLTEST]
GO
SELECT [ProductKey], [EnglishProductName], [Color],
       [StandardCost], [ListPrice], [DealerPrice], [StockLevel]
FROM [DimProduct]

SELECT [ProductKey], [OrderQuantity], [UnitPrice], [SalesAmount]
FROM [Sales]
```

100 %

Results | Messages

	ProductKey	EnglishProductName	Color	StandardCost	ListPrice	DealerPrice	StockLevel
1	212	Sport-100 Helmet, Red	Red	12.0278	33.6442	20.1865	10000
2	213	Long-Sleeve Logo Jersey, S	Multi	31.7244	48.0673	28.8404	4700
3	214	HL Road Frame - Red, 62	Red	747.9682	1263.4598	758.0759	3000
4	215	LL Road Frame - Black, 60	Black	204.6251	337.22	202.332	4000
5	216	Road-650 Black, 60	Black	486.7066	33.6442	20.1865	5000

	ProductKey	OrderQuantity	UnitPrice	SalesAmount
1	212	1	20.1865	20.1865
2	212	1	20.1865	20.1865
3	213	200	48.0673	9613.46
4	213	300	48.0673	14420.19

This time we will insert the wrong information in the Sales table to fail the insertion deliberately.

```
USE [SQLTEST]
GO

BEGIN TRANSACTION
    UPDATE [DimProduct]
        SET [StockLevel] = 4700
        WHERE [ProductKey] = 213

    INSERT INTO [Sales] ([ProductKey], [OrderQuantity], [UnitPrice], [SalesAmount])
    VALUES (213, 300, 48.0673, 'Hey! This is Wrong')
COMMIT TRANSACTION
```

```
USE [SQLTEST]
GO

BEGIN TRANSACTION
    UPDATE [DimProduct]
        SET [StockLevel] = 4700
        WHERE [ProductKey] = 213

    INSERT INTO [Sales] ([ProductKey], [OrderQuantity], [UnitPrice], [SalesAmount])
    VALUES (213, 300, 48.0673, 'Hey! This is Wrong')
COMMIT TRANSACTION
```

100 %

Messages

```
(1 row(s) affected)
Msg 235, Level 16, State 0, Line 9
Cannot convert a char value to money. The char value has incorrect syntax.
```

Let me show you the records in Dim Product and Sales tables after that transaction. As you can see from the above screenshot, a committed row (Update Statement) had rolled back.

```
USE [SQLTEST]
GO
SELECT [ProductKey], [EnglishProductName], [Color],
       [StandardCost], [ListPrice], [DealerPrice], [StockLevel]
FROM [DimProduct]

SELECT [ProductKey], [OrderQuantity], [UnitPrice], [SalesAmount]
FROM [Sales]
```

100 %

Results | Messages

	ProductKey	EnglishProductName	Color	StandardCost	ListPrice	DealerPrice	StockLevel
1	212	Sport-100 Helmet, Red	Red	12.0278	33.6442	20.1865	10000
2	213	Long-Sleeve Logo Jersey, S	Multi	31.7244	48.0673	28.8404	4700
3	214	HL Road Frame - Red, 62	Red	747.9682	1263.4598	758.0759	3000
4	215	LL Road Frame - Black, 60	Black	204.6251	337.22	202.332	4000
5	216	Road-650 Black, 60	Black	486.7066	33.6442	20.1865	5000

	ProductKey	OrderQuantity	UnitPrice	SalesAmount
1	212	1	20.1865	20.1865
2	212	1	20.1865	20.1865
3	213	200	48.0673	9613.46
4	213	300	48.0673	14420.19

Consistency in SQL Server ACID

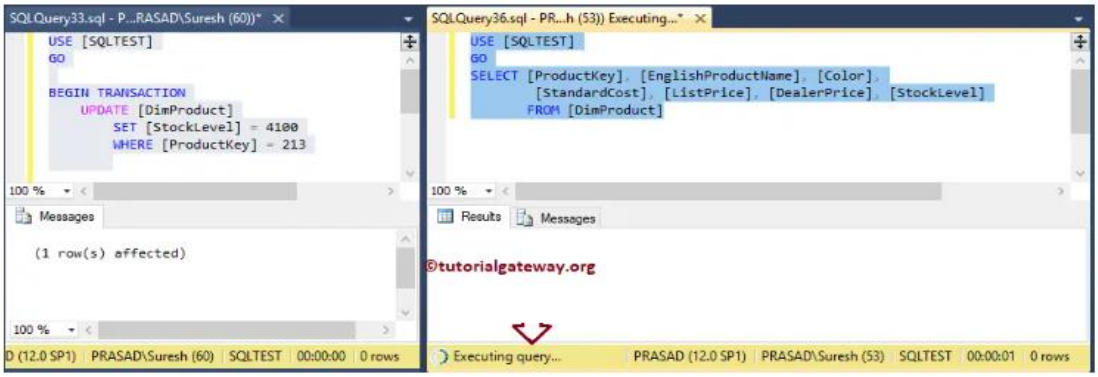
Let me take the above example to explain this SQL ACID property. Say the transaction has updated the stock with new data, and suddenly there is a system failure (right before the insertion into sales or in the middle). In this situation, the system will roll back the updates. Otherwise, you can't trace the stock information.

Isolation in SQL Server ACID

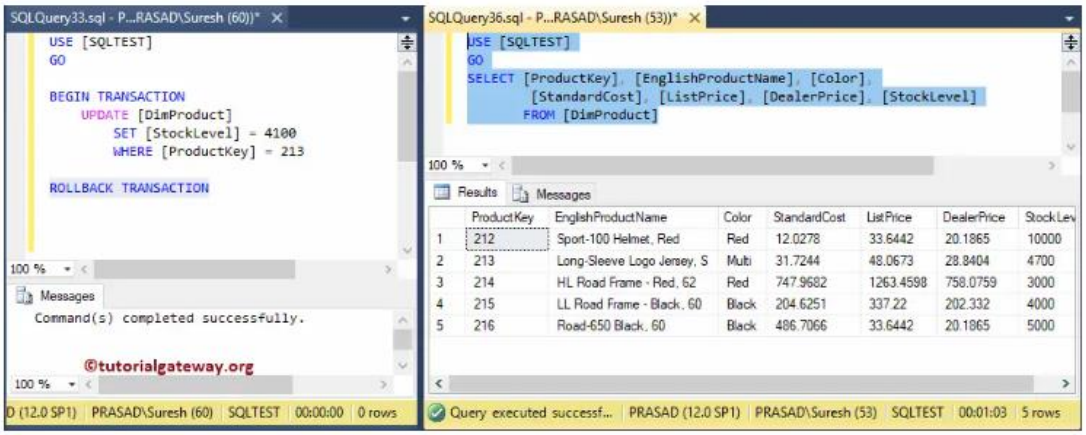
One transaction can't access the result of other transactions until the transaction completes. For this Acid Property in SQL Server, it uses Locks to lock the table. As you can see, we are using two separate instances :

- First Instance: we started the transaction and updated the record, but we haven't committed or rolled back the transaction.
- Second Instance: Use the Select statement to select the records present in the Dim Product table.

As you can see from the below screenshot, the select statement is not returning any information. Because we can't access one transaction result without completing the transaction.



Let me execute the Rollback transaction. It will immediately show the result of the Select statement because the lock released from the Dim Product table.



Summary

The ACID properties (Atomicity, Consistency, Isolation, Durability) are crucial for ensuring reliable database transactions. The SQL examples above show how to implement a transaction with locking mechanisms and demonstrate different isolation levels to control concurrency. By using appropriate isolation levels, you can balance the needs for data integrity and performance in a multi-user database environment.

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:

1.Create a new database: create database LibraryDB;

2. Create Tables Based on the Library Schema:

```
-- Create the Authors table
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL
);

-- Create the Books table
CREATE TABLE Books (
    BookID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    Title VARCHAR(255) NOT NULL,
    ISBN VARCHAR(13) NOT NULL UNIQUE,  -- ISBN must be unique
    Publisher VARCHAR(255) NOT NULL,
    PublishedYear INT NOT NULL CHECK (PublishedYear >= 1800 AND PublishedYear <= YEAR(GETDATE())),  -- Year must be between 1800 and the current year
    AuthorID INT,  -- Foreign key to the Authors table
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)  -- Establishes the relationship with Authors
);

-- Create the Members table
CREATE TABLE Members (
    MemberID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Email VARCHAR(255) NOT NULL UNIQUE,  -- Email must be unique
```

```
    PhoneNumber VARCHAR(15) UNIQUE,  -- Phone number can be unique
    MembershipDate DATE NOT NULL
);

-- Create the Loans table
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    BookID INT NOT NULL,  -- Foreign key to the Books table
    MemberID INT NOT NULL,  -- Foreign key to the Members table
    LoanDate DATE NOT NULL,
    DueDate DATE NOT NULL ,-- Due date must be after the loan date
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),  -- Establishes the relationship with Books
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)  -- Establishes the relationship with Members
);

-- Create the Categories table
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY IDENTITY(1,1),  -- Auto-incrementing primary key
    CategoryName VARCHAR(100) NOT NULL UNIQUE  -- Category name must be unique
);

-- Create the BookCategories table (many-to-many relationship between Books and Categories)
CREATE TABLE BookCategories (
    BookID INT NOT NULL,  -- Foreign key to the Books table
    CategoryID INT NOT NULL,  -- Foreign key to the Categories table
    PRIMARY KEY (BookID, CategoryID),  -- Composite primary key
    FOREIGN KEY (BookID) REFERENCES Books(BookID),  -- Establishes the relationship with Books
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)  -- Establishes the relationship with
Categories
);
```

3. Alter Table Structures:

-- Add a new column to the Books table

ALTER TABLE Books

ADD NumberOfPages INT;

-- Modify the Books table to make Publisher optional

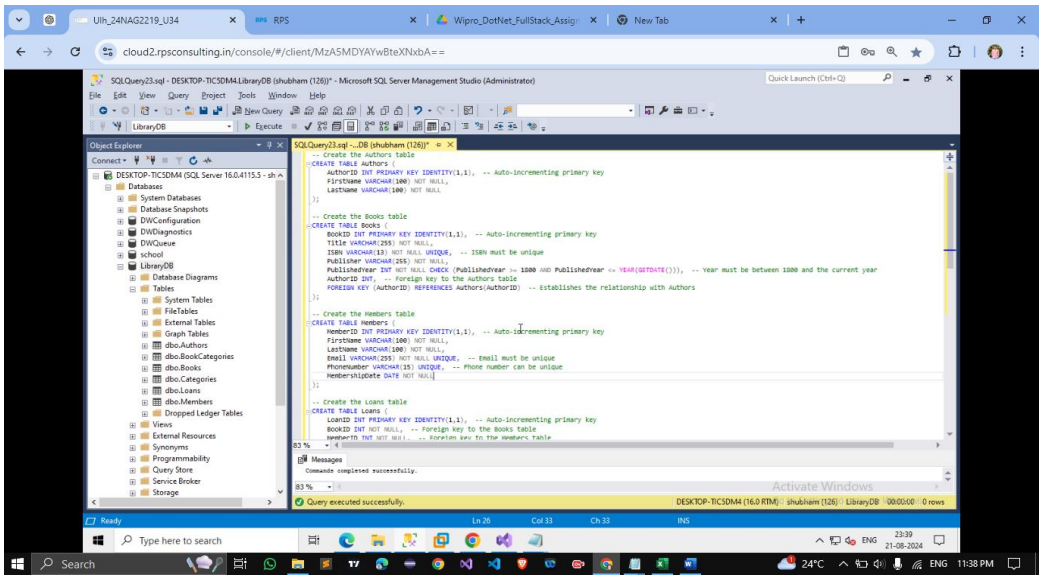
ALTER TABLE Books

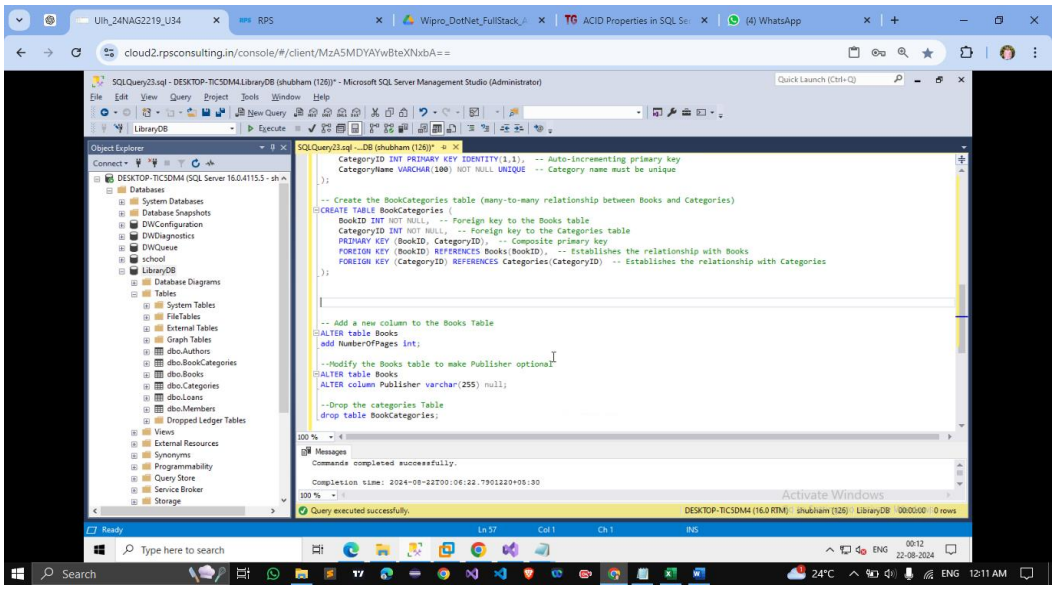
ALTER COLUMN Publisher VARCHAR(255) NULL;

4. Drop a Redundant Table:

-- Drop the Categories table

DROP TABLE BookCategories;





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 : →

Step 1: Create a sample table and populate it with data

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(255),
   Age INT,
   Department VARCHAR(255)
);

INSERT INTO Employees (EmployeeID, Name, Age, Department)
VALUES
(1, 'Shubham Kumbhar', 30, 'Sales'),
(2, 'Sanket Mane', 25, 'Marketing'),
(3, ‘Pankaj Sarnobat’, 40, 'IT'),
(4, 'Sudhansh Bidkar', 35, 'HR'),
(5, 'Sangram Patil', 28, 'Finance');
```

Step 2: Create an index on the Department column

```
CREATE INDEX idx_Department ON Employees (Department);
```

Step 3: Execute a query with and without the index

```
-- Query without index
SELECT * FROM Employees WHERE Department = 'Sales';

-- Query with index
SELECT * FROM Employees WHERE Department = 'Sales';
```

Step 4: Analyze query performance

With the index, the query executes faster because the database can quickly locate the rows with the specified department using the index.

Step 5: Drop the index and analyze query performance again

DROP INDEX idx_Department ON Employees;

After dropping the index, the query executes slower because the database has to scan the entire table to find the rows with the specified department.

Conclusion :

Creating an index on a column used in WHERE clauses can significantly improve query performance by allowing the database to quickly locate the required data. However, indexes also require maintenance and can slow down write operations, so they should be used judiciously.

Index Types:

- Clustered Index: Reorders the physical rows of the table according to the index.

- Non-Clustered Index: Creates a separate data structure containing the index keys and pointers to the corresponding table rows.

Index Benefits

- Improved query performance

- Faster data retrieval

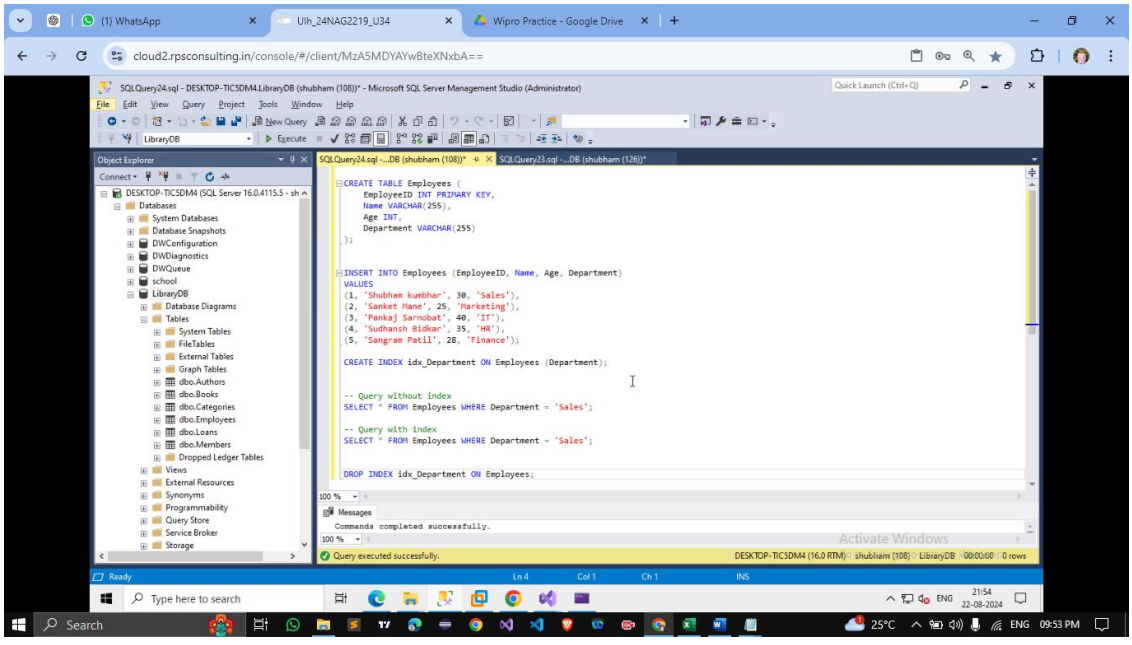
- Reduced disk I/O

Index Drawbacks

- Increased storage space

- Slower write operations (INSERT, UPDATE, DELETE)

- Maintenance overhead



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: →

1. Creating a New Database User

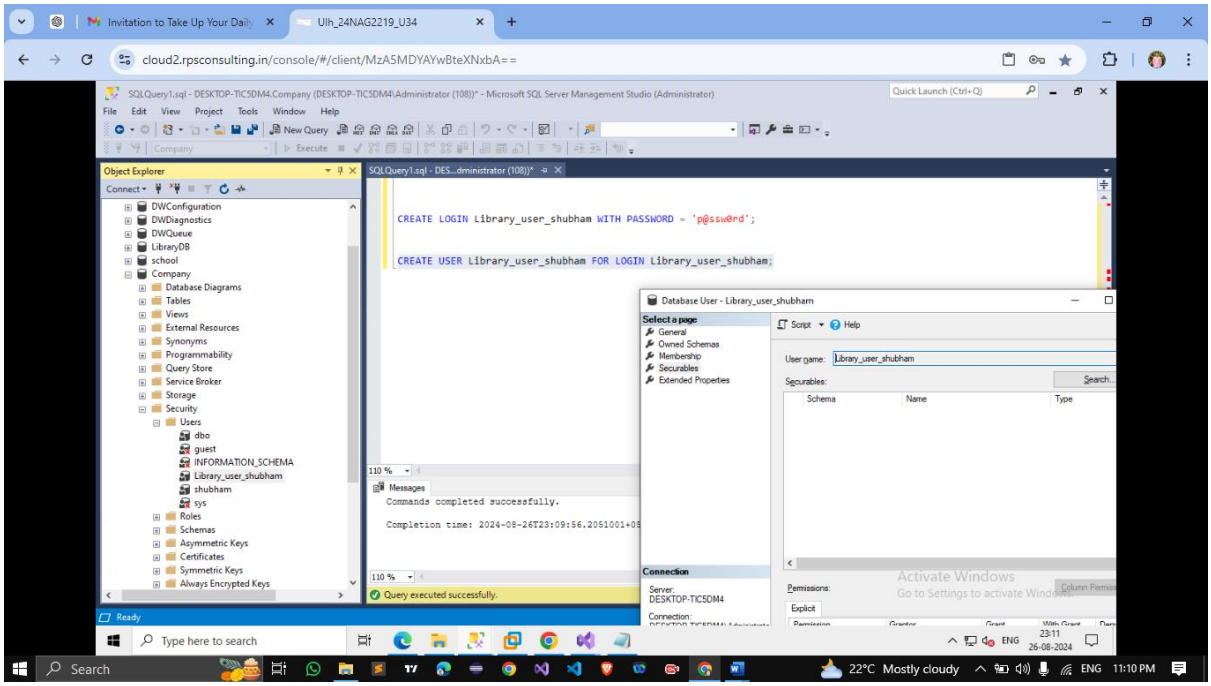
Let's start by creating a new database user named Library_user_shubham with a password.

-- Create a new database user with a password

CREATE LOGIN Library_user_shubham WITH PASSWORD = 'p@ssw0rd';

-- Create a user in the current database for the login

CREATE USER Library_user_shubham FOR LOGIN Library_user_shubham;

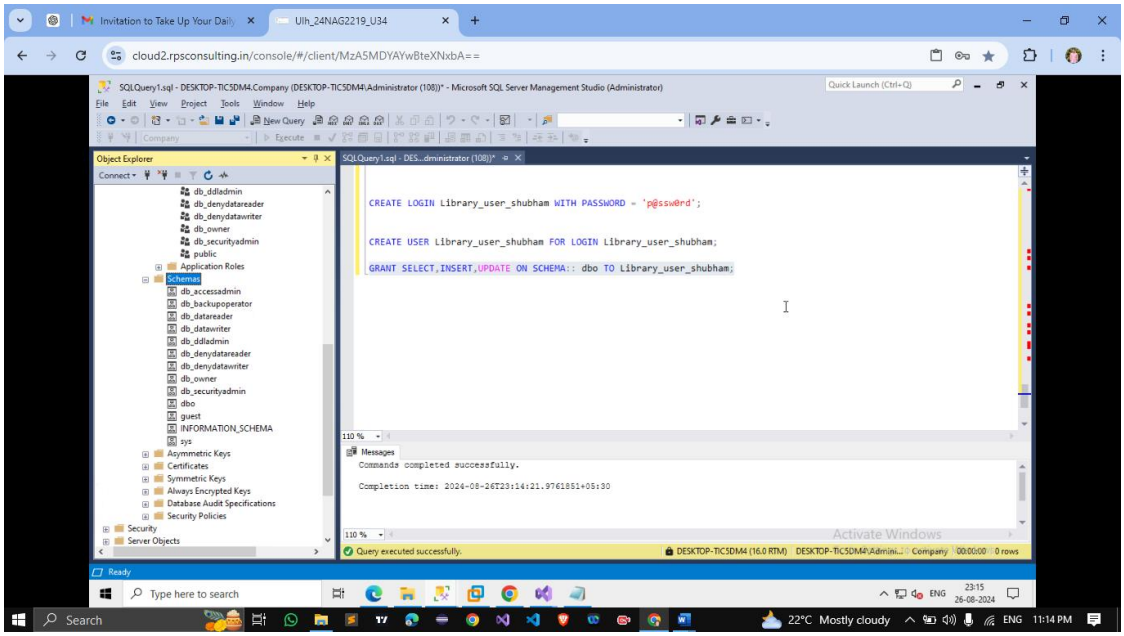


2. Granting Specific Privileges

Now, let's grant specific privileges to the Library_user_shubham . We'll give the user privileges to SELECT, INSERT, and UPDATE on the Books table.

-- Grant SELECT, INSERT, and UPDATE privileges on the SCHEMA to Library_user_shubham

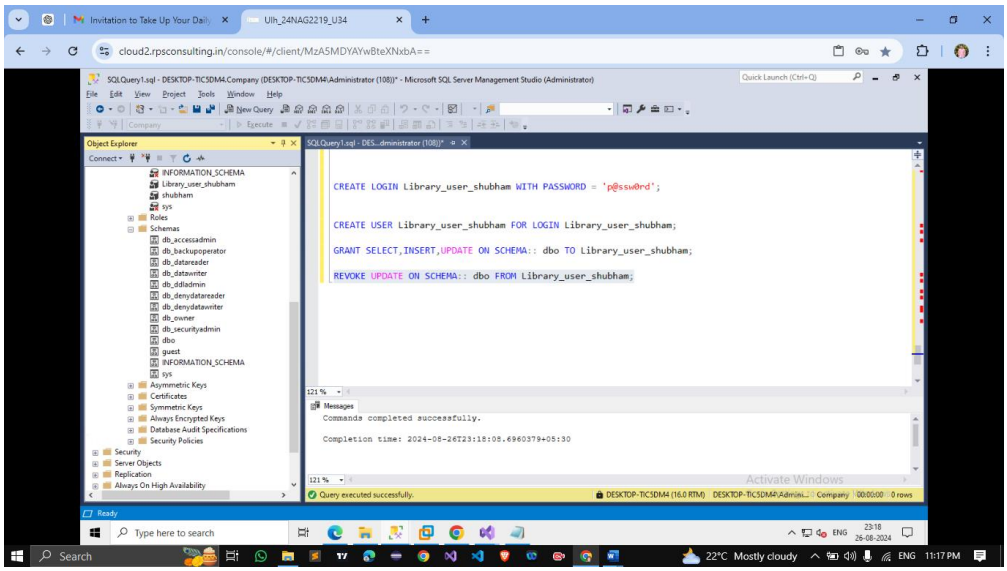
GRANT SELECT, INSERT, UPDATE ON SCHEMA :: dbo TO Library_user_shubham;



3. Revoking Certain Privileges

Suppose you want to revoke the UPDATE privilege from Library_user_shubham later. You can do this with the REVOKE command.

REVOKE UPDATE ON SCHEMA:: dbo FROM Library_user_shubham;



4. Dropping the User

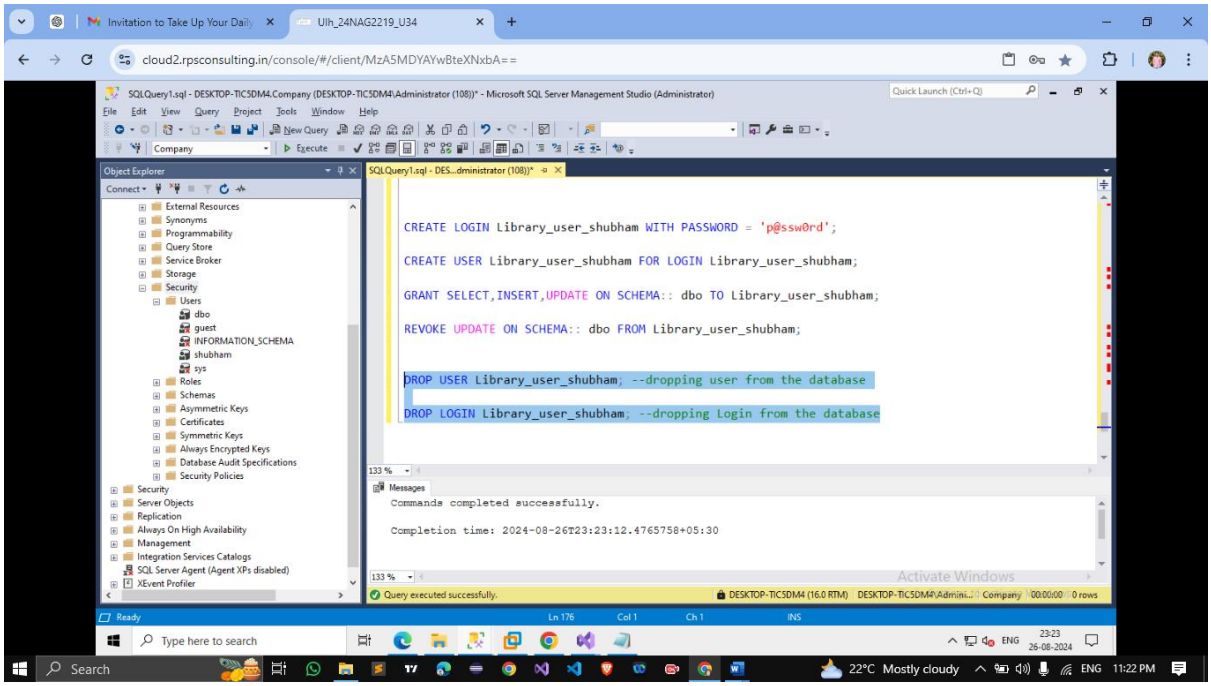
Finally, if the user is no longer needed, you can drop the user and the login.

-- Drop the user from the database

DROP USER Library_user_shubham;

-- Drop the login from the server

DROP LOGIN Library_user_shubham;



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 : →

1. Creating the TablesWe will create three tables: Books, Members, and BorrowedBooks.

Table: Books

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255),
    Price DECIMAL(10, 2),
    Available INT
);
```

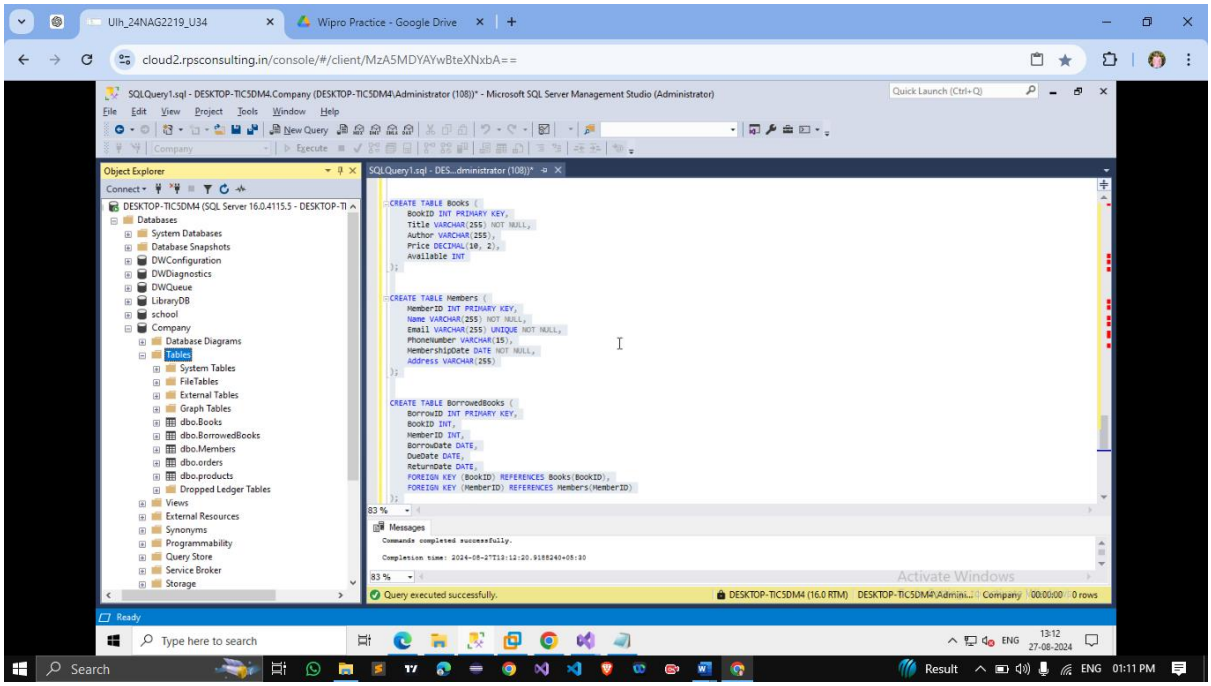
Table: Members

```
CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
```

```
    PhoneNumber VARCHAR(15),
    MembershipDate DATE NOT NULL,
    Address VARCHAR(255)
);
```

Table: BorrowedBooks

```
CREATE TABLE BorrowedBooks (
    BorrowID INT PRIMARY KEY,
    BookID INT,
    MemberID INT,
    BorrowDate DATE,
    DueDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```



2. Inserting New Records

Insert Records into Books Table

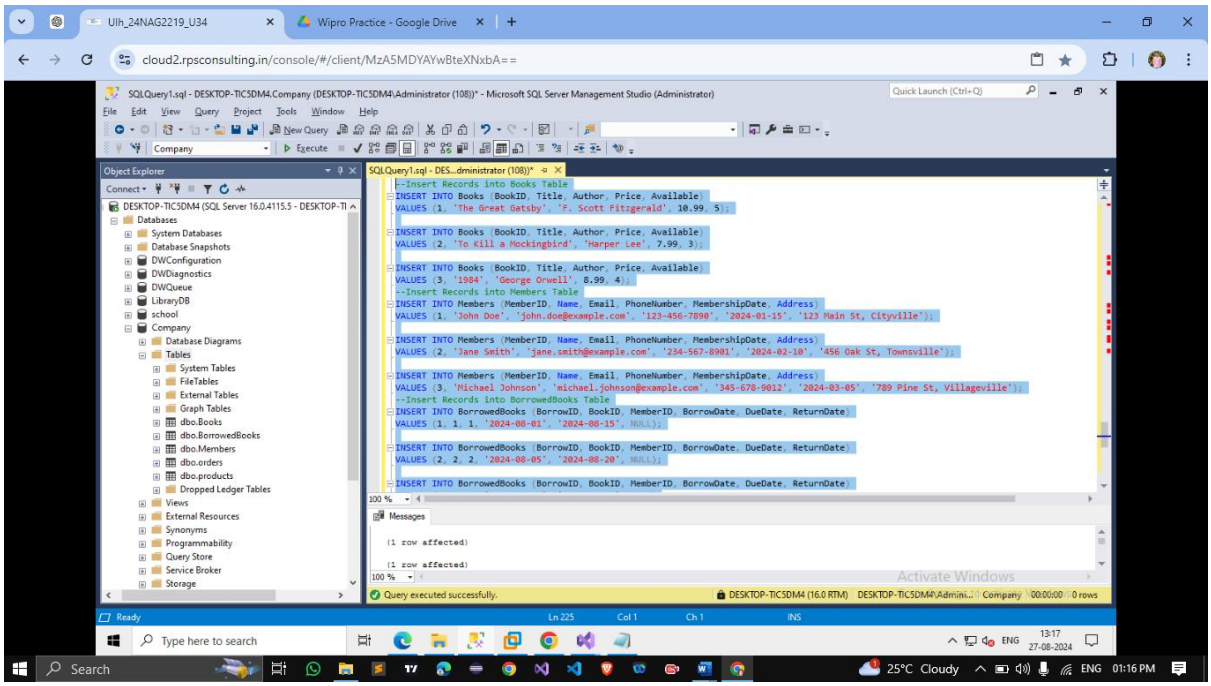
```
INSERT INTO Books (BookID, Title, Author, Price, Available)
VALUES
(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 10.99, 5),
(2, 'To Kill a Mockingbird', 'Harper Lee', 7.99, 3),
(3, '1984', 'George Orwell', 8.99, 4);
```

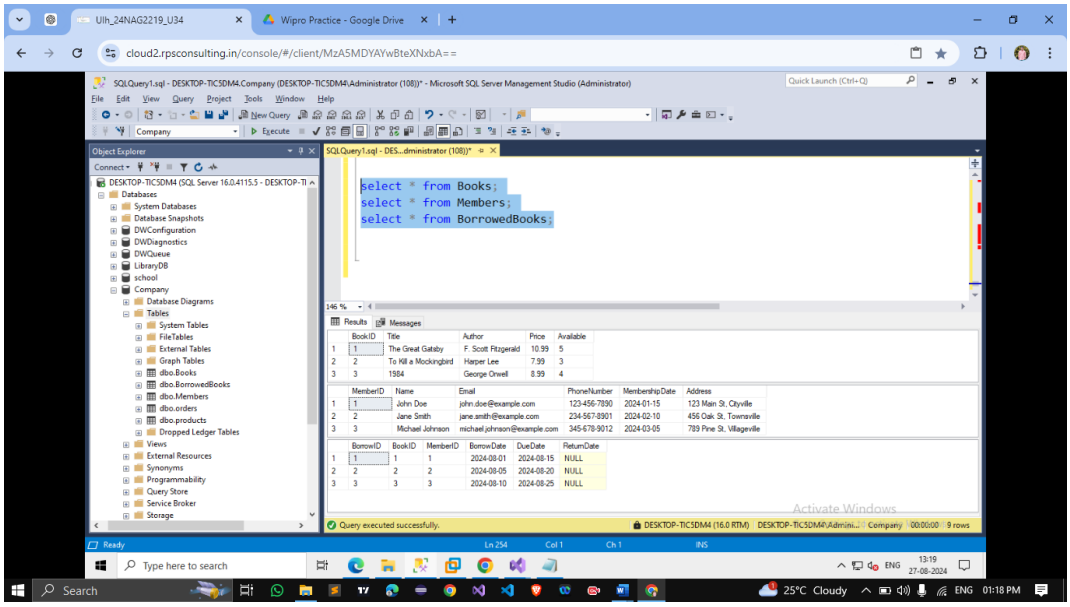
Insert Records into Members Table

```
INSERT INTO Members (MemberID, Name, Email, PhoneNumber, MembershipDate, Address)
VALUES
(1, 'John Doe', 'john.doe@example.com', '123-456-7890', '2024-01-15', '123 Main St, Cityville'),
(2, 'Jane Smith', 'jane.smith@example.com', '234-567-8901', '2024-02-10', '456 Oak St, Townsville'),
(3, 'Michael Johnson', 'michael.johnson@example.com', '345-678-9012', '2024-03-05', '789 Pine St, Villageville');
```

Insert Records into BorrowedBooks Table

```
INSERT INTO BorrowedBooks (BorrowID, BookID, MemberID, BorrowDate, DueDate, ReturnDate)
VALUES
(1, 1, 1, '2024-08-01', '2024-08-15', NULL),
(2, 2, 2, '2024-08-05', '2024-08-20', NULL),
(3, 3, 3, '2024-08-10', '2024-08-25', NULL);
```





3. Updating Records

-- Update the price of a book

UPDATE Books

SET Price = 12.99

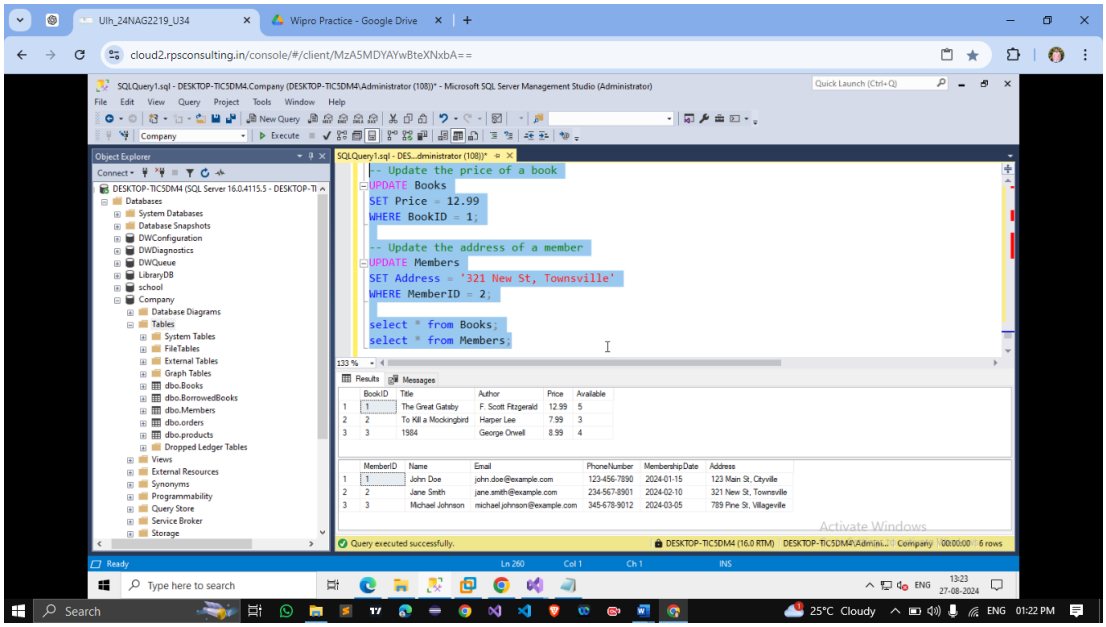
WHERE BookID = 1;

-- Update the address of a member

UPDATE Members

SET Address = '321 New St, Townsville'

WHERE MemberID = 2;



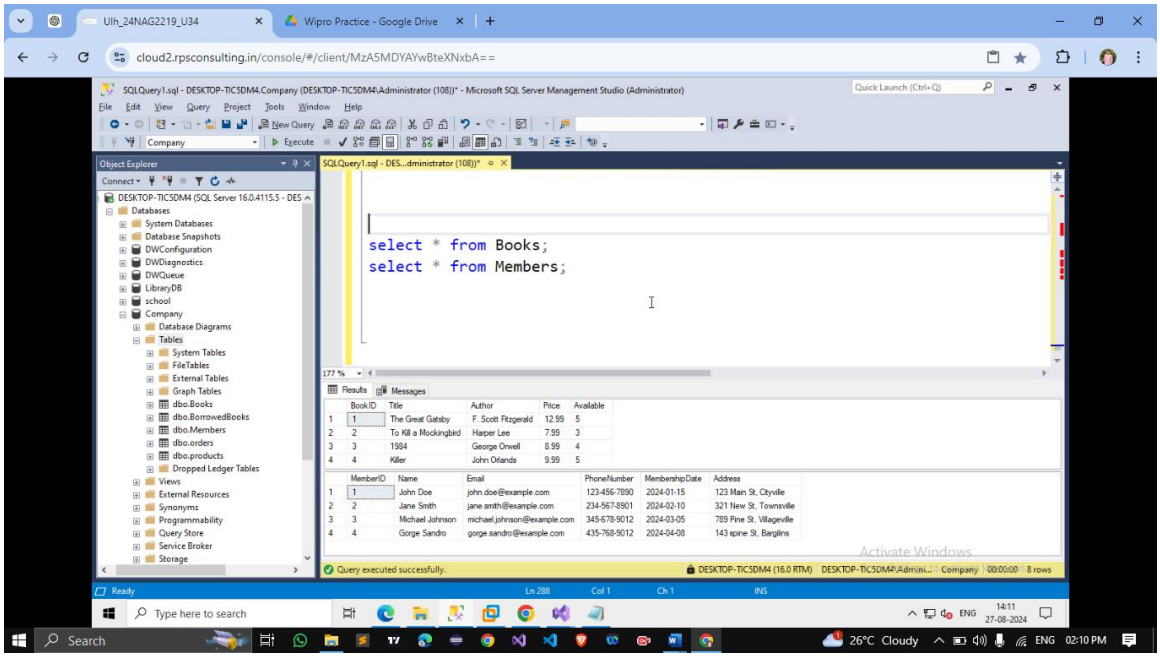
4. Deleting Records

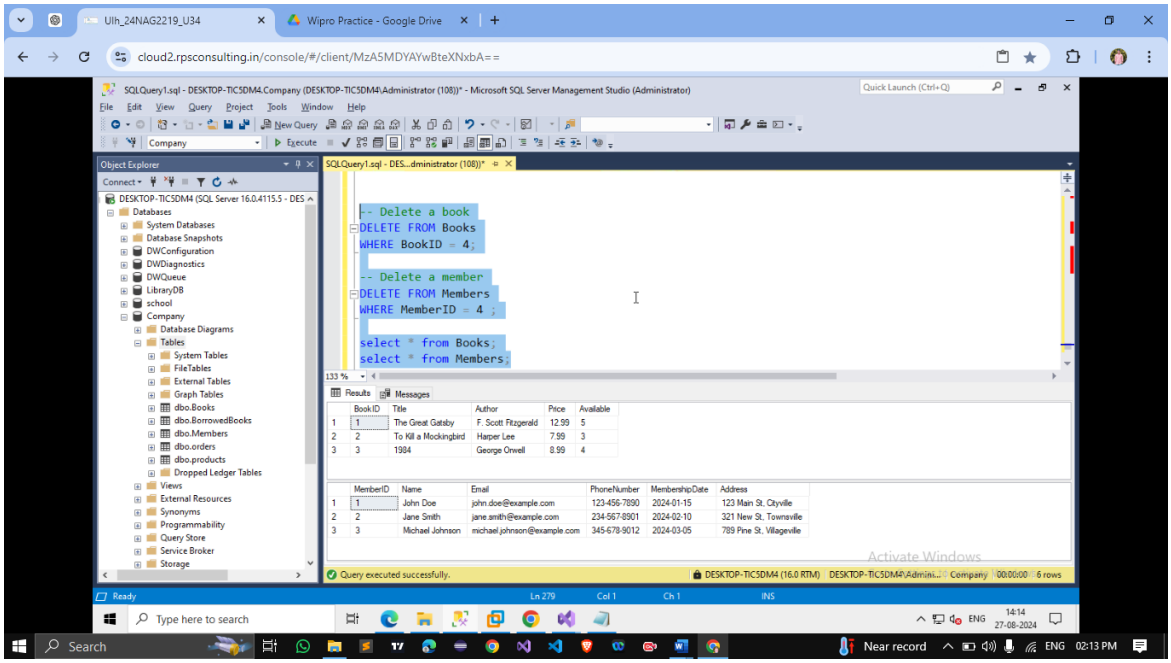
-- Delete a book that is no longer available

```
DELETE FROM Books
WHERE BookID = 4;
```

-- Delete a member who has not borrowed any books

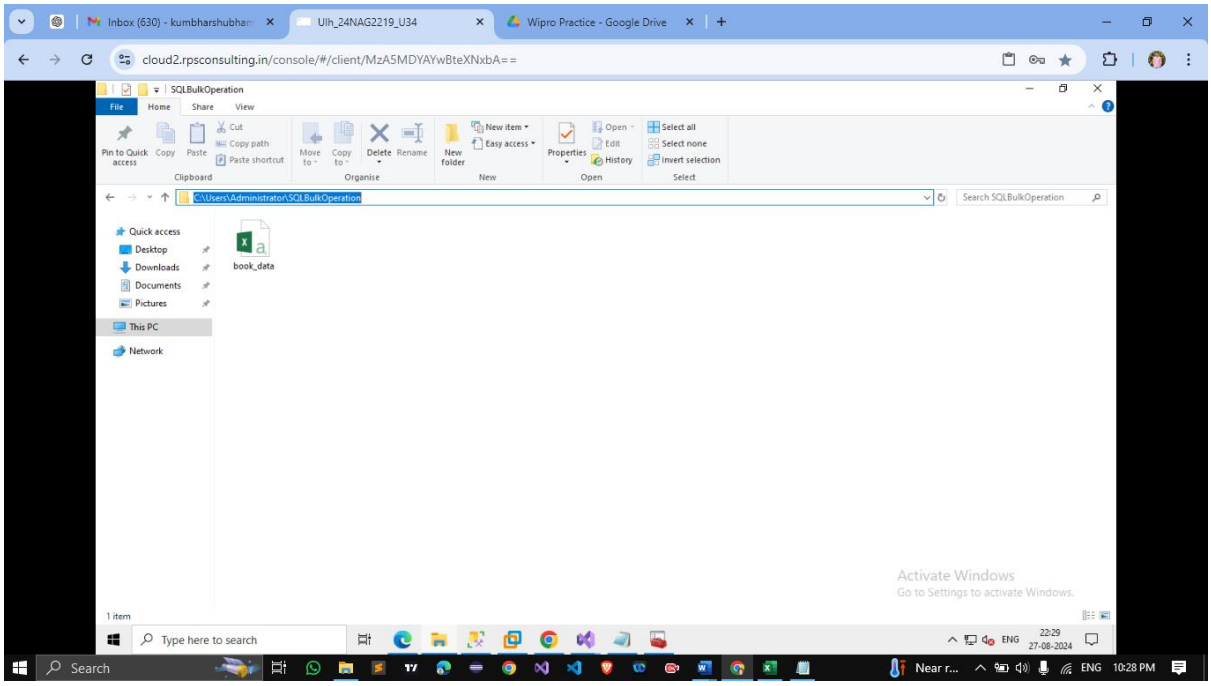
```
DELETE FROM Members
WHERE MemberID = 4;
```

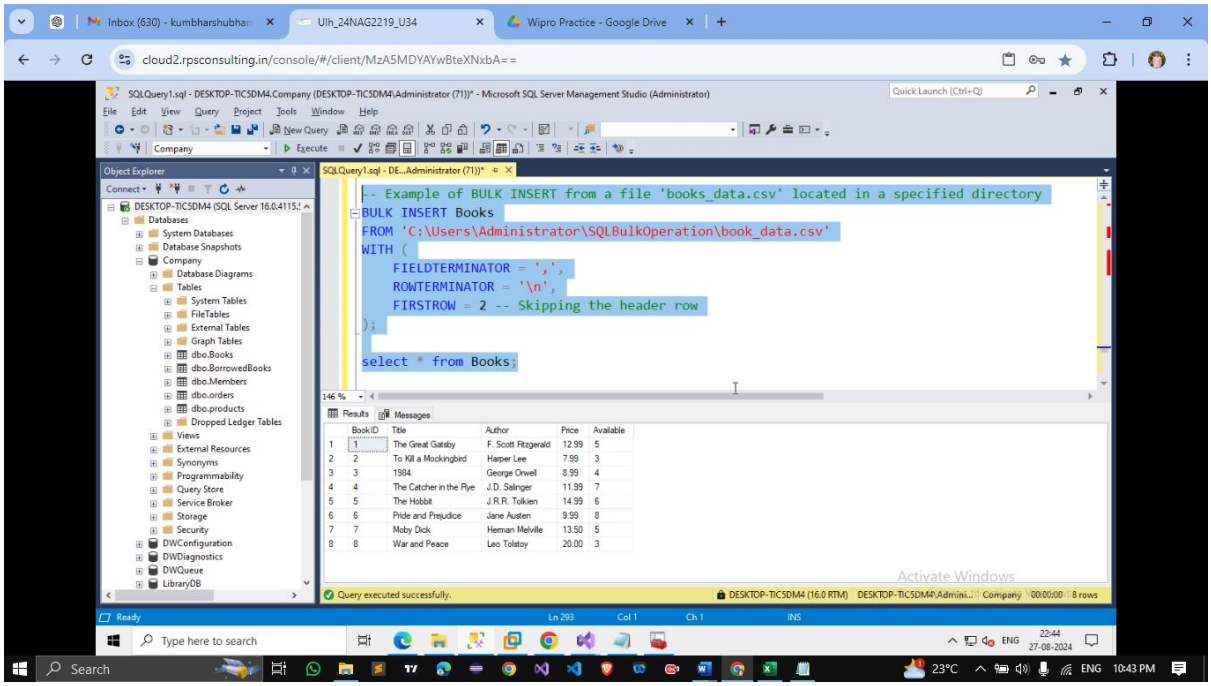




5. BULK INSERT Operations

```
-- Example of BULK INSERT from a file 'books_data.csv' located in a specified directory
BULK INSERT Books
FROM C:\Users\Administrator\SQLBilkOperation\books_data.csv'
WITH (
   FIELDTERMINATOR = ',',
   ROWTERMINATOR = '\n',
   FIRSTROW = 2 -- Skipping the header row
);
```





Explanation:

- Creating Tables: The Books, Members, and BorrowedBooks tables are created to hold book information, member details, and records of borrowed books, respectively.
- Inserting Data: SQL INSERT statements are used to add records to these tables.
- Updating Records: SQL UPDATE statements are used to modify existing records.
- Deleting Records: SQL DELETE statements are used to remove records that meet specific criteria.
- BULK INSERT: This command is used to load large amounts of data from external files efficiently. The path to the file and the delimiter (e.g., comma for CSV) must be specified.