Name:Vaishali Ramesh Kale

Email id: kalevaishalir16@gmail.com

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.

Business Scenario

A small bookstore sells books to customers. Each book has a unique ISBN, title, author, and price. The bookstore keeps track of inventory with stock quantity. Customers can place orders, and each order can include multiple books. Each order has a unique order ID, date, and total amount. Customers have a unique customer ID, name, and contact details (address, phone number, and email). Employees process orders, and each employee has a unique employee ID, name, and position.

Steps to Create the ER Diagram

Step 1: Identify Entities

1. Book
2. Customer
3. Order
4. OrderDetails (junction table to handle many-to-many relationship between Order and Book)
5. Employee

Step 2: Identify Attributes

1) Book

- ISBN (Primary Key)
- Title
- Author
- Price
- StockQuantity

2) Customer

- CustomerID (Primary Key)
- Name
- Address
- Phone
- Email

3) **Order**

- OrderID (Primary Key)
- OrderDate
- TotalAmount
- CustomerID (Foreign Key)
- EmployeeID (Foreign Key)

4) **OrderDetails**

- OrderID (Composite Primary Key, Foreign Key)
- ISBN (Composite Primary Key, Foreign Key)
- Quantity

5) **Employee**

- EmployeeID (Primary Key)
- Name
- Position

Step 3: Define Relationships and Cardinality

Customer - Order: One customer can place many orders. (1:M)

Order - OrderDetails: One order can contain multiple books, and one book can appear in multiple orders. (1:M)

Book - OrderDetails: One book can appear in multiple orders, and one order can contain multiple books. (M:N)

Employee - Order: One employee processes many orders, but each order is processed by one employee. (1:M)

Step 4: Ensure Normalization to 3NF

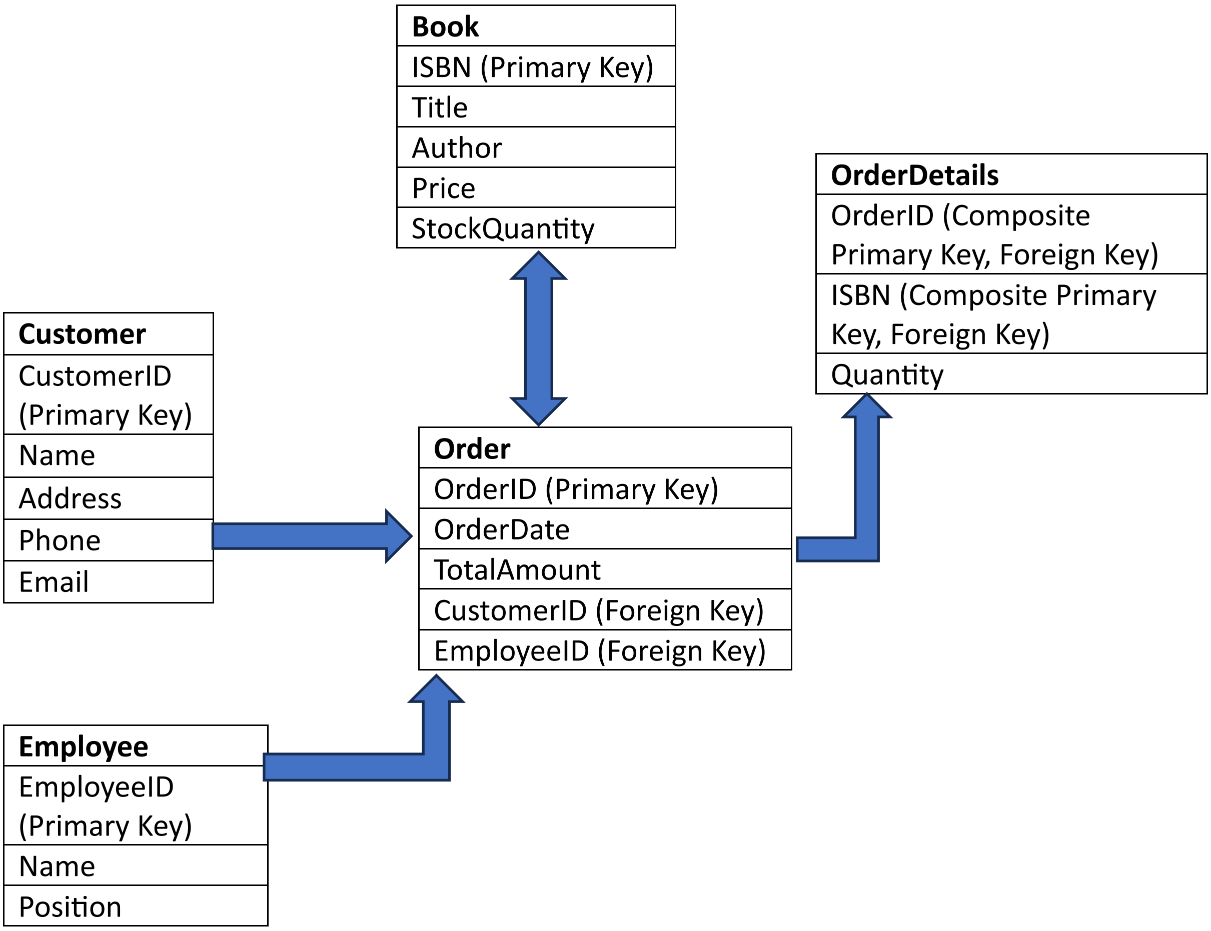
1NF: Each table has a primary key, and each column contains atomic values**.**

2NF: All non-key attributes are fully functional dependent on the primary key.

3NF: All attributes are functionally dependent only on the primary key, and there are no transitive dependencies.

- **ER Diagram**

Below is the ER diagram based on the described scenario:



Book
ISBN (Primary Key)
Title
Author
Price
StockQuantity

OrderDetails
OrderID (Composite Primary Key, Foreign Key)
ISBN (Composite Primary Key, Foreign Key)
Quantity

Customer
CustomerID (Primary Key)
Name
Address
Phone
Email

Order
OrderID (Primary Key)
OrderDate
TotalAmount
CustomerID (Foreign Key)
EmployeeID (Foreign Key)

Employee
EmployeeID (Primary Key)
Name
Position

Relationships:

- Customer (1) to Order (M)

- Order (1) to OrderDetails (M)

- Book (1) to OrderDetails (M)

- Employee (1) to Order (M)

➢

➢ **ER Diagram Explanation**

- Book table holds information about books in the bookstore.
- Customer table contains customer details.
- Order table records order information and links to Customer and Employee.
- OrderDetails table handles the many-to-many relationship between Order and Book.
- Employee table records the employees who process orders.
- This diagram reflects a normalized design up to 3NF, ensuring that all attributes are dependent on the primary key and eliminating transitive dependencies.

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.

Database Schema

1. Books Table

Stores information about each book in the library.

```
CREATE TABLE Books (

    BookID INT PRIMARY KEY,

    Title VARCHAR(255) NOT NULL,

    Author VARCHAR(255) NOT NULL,

    Publisher VARCHAR(255) NOT NULL,

    Genre VARCHAR(100) NOT NULL,

    ReleaseDate DATE,

    Price DECIMAL(10, 2) CHECK (Price >= 0),

    StockQuantity INT CHECK (StockQuantity >= 0)

);
```

Insert Values:

```
INSERT INTO Books (BookID, Title, Author, Publisher, Genre, ReleaseDate, Price, StockQuantity) VALUES

(1, '1984', 'George Orwell', 'Secker & Warburg', 'Dystopian', '1949-06-08', 15.99, 12),
(2, 'To Kill a Mockingbird', 'Harper Lee', 'J.B. Lippincott & Co.', 'Fiction', '1960-07-11', 10.99, 5),
(3, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Charles Scribner\'s Sons', 'Tragedy', '1925-04-10', 14.99, 8),
(4, 'The Catcher in the Rye', 'J.D. Salinger', 'Little, Brown and Company', 'Fiction', '1951-07-16', 12.99, 7);
```

2. Customers Table

Stores information about each customer.

```
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Address VARCHAR(255) NOT NULL,
   Phone VARCHAR(15) UNIQUE,
   Email VARCHAR(255) UNIQUE,
   MembershipStatus VARCHAR(50) CHECK (MembershipStatus IN ('Active', 'Inactive'))
);
```

Insert Values:

```
INSERT INTO Customers (CustomerID, Name, Address, Phone, Email, MembershipStatus) VALUES
(1, 'John Doe', '123 Main St, Anytown, USA', '555-1234', 'johndoe@example.com', 'Active'),
(2, 'Jane Smith', '456 Oak St, Sometown, USA', '555-5678', 'janesmith@example.com', 'Active'),
(3, 'Alice Johnson', '789 Pine St, Anycity, USA', '555-8765', 'alicejohnson@example.com', 'Inactive'),
(4, 'Bob Brown', '321 Elm St, Othertown, USA', '555-4321', 'bobbrown@example.com', 'Active');
```

3. Employees Table

Stores information about each employee.

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Role VARCHAR(100) NOT NULL
);
```

Insert Values:

```
INSERT INTO Employees (EmployeeID, Name, Role) VALUES
(1, 'Emily Davis', 'Librarian'),
(2, 'Michael Wilson', 'Assistant Librarian'),
(3, 'Sarah Miller', 'Clerk');
```

4. Rentals Table

Stores information about each rental transaction.

```
CREATE TABLE Rentals (
    RentalID INT PRIMARY KEY,
    RentalDate DATE NOT NULL,
    ReturnDate DATE,
    BookID INT,
    CustomerID INT,
    EmployeeID INT,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

Insert Values:

```
INSERT INTO Rentals (RentalID, RentalDate, ReturnDate, BookID, CustomerID, EmployeeID)
VALUES
(1, '2024-05-01', '2024-05-15', 1, 1, 1),
(2, '2024-05-02', '2024-05-16', 2, 2, 2),
(3, '2024-05-03', '2024-05-17', 3, 1, 3),
(4, '2024-05-04', '2024-05-18', 4, 3, 1);
```

select * from books;

select * from customer1;

select * from employees;

select * from rentals;

Additional Constraints

NOT NULL: Ensures that a column cannot have a NULL value.

UNIQUE: Ensures that all values in a column are unique.

CHECK: Ensures that all values in a column satisfy a specific condition.

Primary Key (PK): Uniquely identifies each record in a table.

Foreign Key (FK): Uniquely identifies a record in another table to establish a relationship.

Schema Design with Relationships and Constraints

Explanation of Relationships

Books and Rentals: Each rental involves a specific book. BookID in the Rentals table is a foreign key referencing BookID in the Books table.

Customers and Rentals: Each rental is made by a specific customer. CustomerID in the Rentals table is a foreign key referencing CustomerID in the Customers table.

Employees and Rentals: Each rental is processed by a specific employee. EmployeeID in the Rentals table is a foreign key referencing EmployeeID in the Employees table.

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

Atomicity:

Ensures that a transaction is treated as a single unit, which either completes entirely or does not execute at all. If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged.

Example: If a transaction includes transferring money from one account to another, both the debit from one account and the credit to another must occur. If one fails, neither should occur.

Consistency:

Ensures that a transaction takes the database from one valid state to another valid state. It maintains database rules such as constraints, cascades, triggers, and other rules defined in the schema.

Example: After a transaction, all accounts' balances must remain positive, adhering to the rule that no account should have a negative balance.

Isolation:

Ensures that transactions are executed in isolation from each other. Concurrent transactions do not affect each other, and the results are as if the transactions were executed sequentially.

Example: If two transactions are updating the same account, each transaction will be unaware of the other’s operations until they are both complete, ensuring that the account's final state is correct.

Durability:

Ensures that once a transaction is committed, it is permanently recorded in the database, even in the event of a system failure. Changes made by the transaction are saved and persisted.

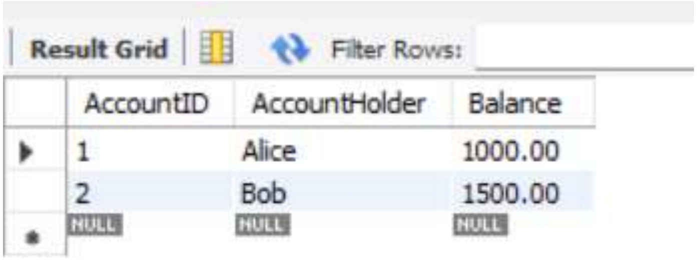
Example: If a transaction to transfer funds is committed, the changes are permanent and will not be undone even if the system crashes immediately afterward.

Simulating a Transaction with Locking and Isolation Levels

use a simple example involving a bank account table where multiple transactions occur.

Step 1: Create the Schema

```
CREATE DATABASE BankDB;

USE BankDB;

CREATE TABLE Accounts (

    AccountID INT PRIMARY KEY,

    AccountHolder VARCHAR(255) NOT NULL,
    Balance DECIMAL(10, 2) CHECK (Balance >= 0)
);

INSERT INTO Accounts (AccountID, AccountHolder, Balance) VALUES
(1, 'Alice', 1000.00),
(2, 'Bob', 1500.00);
```

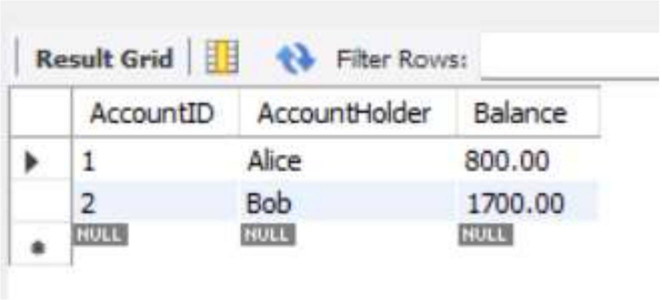


AccountID	AccountHolder	Balance
1	Alice	1000.00
2	Bob	1500.00
NULL	NULL	NULL

Step 2: Demonstrate a Transaction

Let's simulate a transfer of $200 from Alice's account to Bob's account.

```
-- Start the transaction
START TRANSACTION;

-- Lock the accounts involved
SELECT * FROM Accounts WHERE AccountID IN (1, 2) FOR UPDATE;

-- Deduct from Alice's account
UPDATE Accounts SET Balance = Balance - 200 WHERE AccountID = 1;

-- Add to Bob's account
UPDATE Accounts SET Balance = Balance + 200 WHERE AccountID = 2;

-- Commit the transaction
COMMIT;
```



AccountID	AccountHolder	Balance
1	Alice	800.00
2	Bob	1700.00
NULL	NULL	NULL

In this transaction, the FOR UPDATE clause locks the selected rows to prevent other transactions from modifying them until the current transaction is complete.

Step 3: Demonstrate Isolation Levels

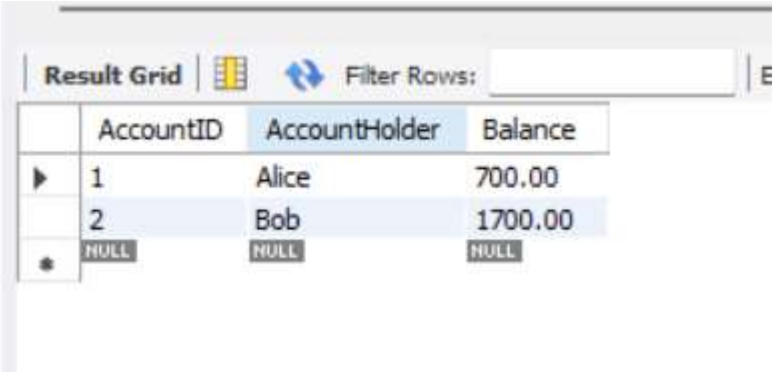
Isolation levels determine how transaction integrity is visible to other users and systems. Here are the common isolation levels:

1. Read Uncommitted: Allows dirty reads, meaning transactions can see uncommitted changes made by other transactions.
2. Read Committed: Prevents dirty reads; transactions cannot see uncommitted changes.
3. Repeatable Read: Ensures that if a transaction reads a value, the same value will be read again within the same transaction.
4. Serializable: The highest isolation level, which ensures complete isolation from other transactions, preventing phantom reads.

Let's demonstrate these isolation levels.

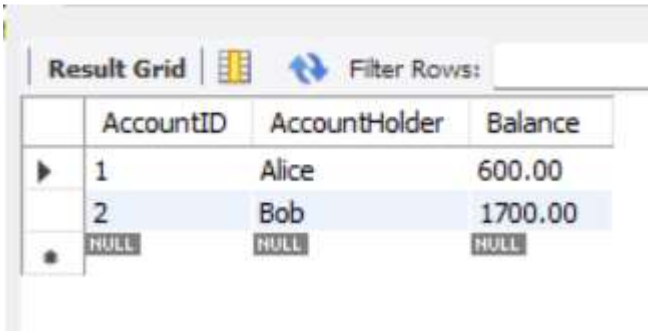
```
-- Set isolation level to Read Uncommitted

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;


-- Start Transaction 1

START TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;


-- In a different session: Transaction 2

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

START TRANSACTION;

SELECT * FROM Accounts WHERE AccountID = 1;
```

```
-- This might show the uncommitted balance of Alice's account


-- Commit or Rollback Transaction 1

COMMIT;
```



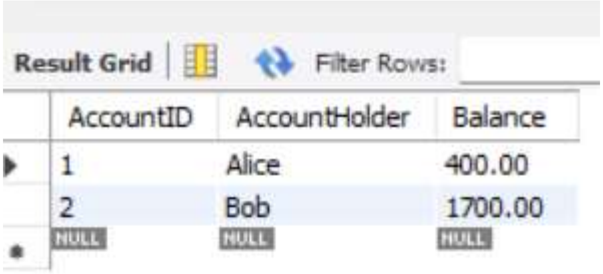
	AccountID	AccountHolder	Balance
▶	1	Alice	700.00
	2	Bob	1700.00
*	NULL	NULL	NULL

```
-------------------------------------------------------------------------------

-- Set isolation level to Read Committed

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;


-- Start Transaction 1

START TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;


-- In a different session: Transaction 2

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;

SELECT * FROM Accounts WHERE AccountID = 1;

-- This will not show the uncommitted balance of Alice's account


-- Commit or Rollback Transaction 1

COMMIT;
```



AccountID	AccountHolder	Balance
1	Alice	600.00
2	Bob	1700.00
NULL	NULL	NULL

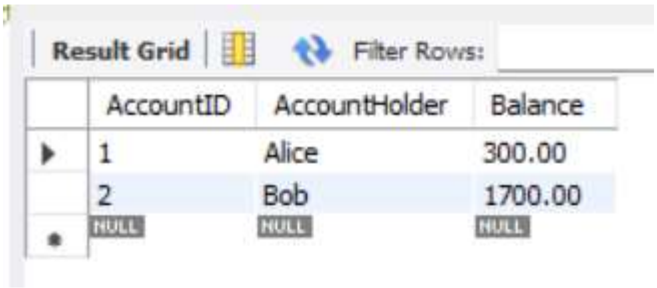
```
----------------------------------------------------------------------

-- Set isolation level to Repeatable Read

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;


-- Start Transaction 1

START TRANSACTION;

SELECT Balance FROM Accounts WHERE AccountID = 1;


-- In a different session: Transaction 2

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;

-- This will be blocked until Transaction 1 is complete


-- Commit or Rollback Transaction 1

COMMIT;
```



AccountID	AccountHolder	Balance
1	Alice	400.00
2	Bob	1700.00
NULL	NULL	NULL

```
---------------------------------------------------------------------------------------------

-- Set isolation level to Serializable

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
-- Start Transaction 1

START TRANSACTION;

SELECT Balance FROM Accounts WHERE AccountID = 1;


-- In a different session: Transaction 2

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;

-- This will be blocked until Transaction 1 is complete


-- Commit or Rollback Transaction 1

COMMIT;
```



	AccountID	AccountHolder	Balance
▶	1	Alice	300.00
	2	Bob	1700.00
●	NULL	NULL	NULL

Summary

Atomicity ensures that all operations within a transaction are completed successfully or none are.

Consistency maintains data integrity by ensuring that any transaction will bring the database from one valid state to another.

Isolation manages concurrent transaction execution to prevent them from interfering with each other.

Durability guarantees that once a transaction is committed, the changes are permanent.

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.

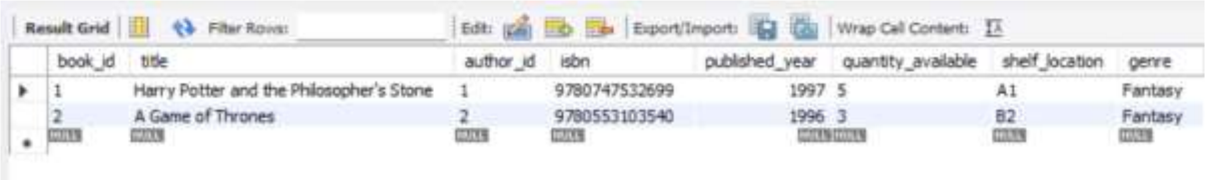
```
C-- Create the database
CREATE DATABASE librarysystem;


-- Use the database
USE librarysystem;
show databases ;
show tables;
select * from Authors;
-- Create Authors table
CREATE TABLE Authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    author_name VARCHAR(255) NOT NULL
);
INSERT INTO Authors (author_name) VALUES ('J.K. Rowling'), ('George R.R. Martin');


-- Create Books table
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author_id INT,
    isbn VARCHAR(13) NOT NULL UNIQUE,
    published_year YEAR NOT NULL,
    quantity_available INT NOT NULL CHECK (quantity_available >= 0),
    shelf_location VARCHAR(100),
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
INSERT INTO Books (title, author_id, isbn, published_year, quantity_available, shelf_location,
genre)
```

```sql
VALUES
('Harry Potter and the Philosopher\'s Stone', 1, '9780747532699', 1997, 5, 'A1', 'Fantasy'),
('A Game of Thrones', 2, '9780553103540', 1996, 3, 'B2', 'Fantasy');

-- Create Members table
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    phone VARCHAR(20) NOT NULL
);
INSERT INTO Members (name, email, address, phone, date_of_birth)
VALUES
('John Doe', 'john.doe@example.com', '123 Elm St', '555-1234', '1980-05-15'),
('Jane Smith', 'jane.smith@example.com', '456 Oak St', '555-5678', '1990-10-25');



-- Create Transactions table
CREATE TABLE Transactions (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    transaction_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    CHECK (transaction_date <= due_date),
    CHECK (return_date IS NULL OR transaction_date <= return_date)
);
- Insert sample data into Transactions table
INSERT INTO Transactions (book_id, member_id, transaction_date, due_date, return_date)
VALUES
(1, 1, '2023-01-01', '2023-01-15', '2023-01-10'),
(2, 2, '2023-01-05', '2023-01-20', NULL);

create table OldBooks(old_book_id int auto_increment primary key,
                                      title VARCHAR(255) NOT NULL,
                                      author VARCHAR(255),    published_year YEAR);
select * FROM Authors;
SELECT * FROM Books;
SELECT * FROM Members;
SELECT * FROM Transactions;

-- Modify the Books table to add a new column for genre
ALTER TABLE Books
ADD COLUMN genre VARCHAR(100);
SELECT * FROM Books;

-- Modify the Members table to add a new column for date of birth
ALTER TABLE Members
ADD COLUMN date_of_birth DATE;
SELECT * FROM Members;

-- Assume there is a redundant table OldBooks that we need to drop
DROP TABLE IF EXISTS OldBooks;
```

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.

```
-- Create the database and use it

CREATE DATABASE LibrarySystem;

USE LibrarySystem;


-- Create the Books table

CREATE TABLE Books (

    book_id INT AUTO_INCREMENT PRIMARY KEY,

    title VARCHAR(255) NOT NULL,

    author_id INT,

    isbn VARCHAR(13) NOT NULL UNIQUE,

    published_year YEAR NOT NULL,

    quantity_available INT NOT NULL CHECK (quantity_available >= 0),

    shelf_location VARCHAR(100),

    FOREIGN KEY (author_id) REFERENCES Authors(author_id)

);


-- Insert sample data into Books table

INSERT INTO Books (title, author_id, isbn, published_year, quantity_available,
shelf_location)

VALUES

('Harry Potter and the Philosopher\'s Stone', 1, '9780747532699', 1997, 5, 'A1'),

('A Game of Thrones', 2, '9780553103540', 1996, 3, 'B2');
```



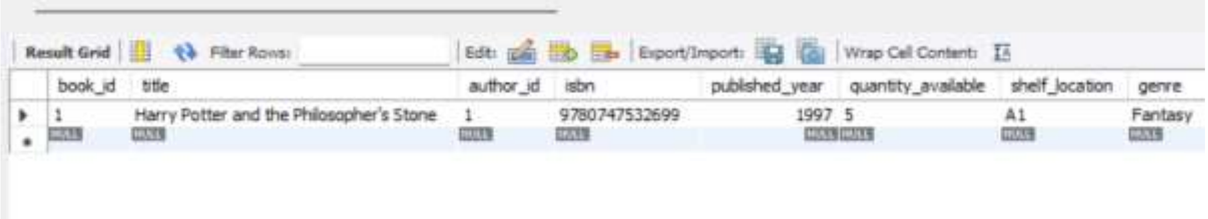
book_id	title	author_id	isbn	published_year	quantity_available	shelf_location	genre
1	Harry Potter and the Philosopher's Stone	1	9780747532699	1997	5	A1	Fantasy
2	A Game of Thrones	2	9780553103540	1996	3	B2	Fantasy
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
-- Create an index on the isbn column
```

CREATE INDEX idx_isbn ON Books(isbn);

-- Query to find a book by its ISBN (with index)

SELECT * FROM Books WHERE isbn = '9780747532699';



book_id	title	author_id	isbn	published_year	quantity_available	shelf_location	genre
1	Harry Potter and the Philosopher's Stone	1	9780747532699	1997	5	A1	Fantasy
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- Drop the index on the isbn column

DROP INDEX idx_isbn ON Books;

-- Query to find a book by its ISBN (without index)

SELECT * FROM Books WHERE isbn = '9780747532699';

Explanation and Impact Analysis

1. Creating the Index: The CREATE INDEX idx_isbn ON Books(isbn); statement creates an index on the isbn column of the Books table.

2. Query Performance with Index: When the index is present, the query SELECT * FROM Books WHERE isbn = '9780747532699'; can quickly find the relevant row using the index, leading to faster query execution.

3. Dropping the Index: The DROP INDEX idx_isbn ON Books; statement removes the index on the isbn column.

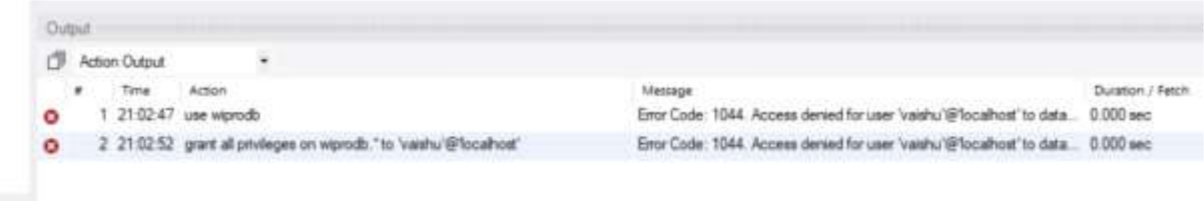
4. Query Performance without Index: After dropping the index, the same query must perform a full table scan, checking each row's isbn until it finds a match. This process is slower, particularly for large datasets, illustrating the performance benefits of using indexes.

Indexes are essential for optimizing database queries, especially in read-heavy applications. However, they also come with a trade-off in terms of storage space and write performance, as indexes need to be updated whenever the underlying data changes. Therefore, it’s important to create indexes on columns that are frequently used in search conditions and to periodically review and optimize indexes as part of database maintenance.

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.

```
-- Create a new database user

CREATE USER 'vaishu'@'localhost' IDENTIFIED BY 'password';


-- Grant specific privileges to the new user

GRANT SELECT, INSERT, UPDATE, DELETE ON LibrarySystem.* TO 'vaishu'@'localhost';


-- Revoke the DELETE privilege from the user

REVOKE DELETE ON LibrarySystem.* FROM ''vaishu' @'localhost';


-- Drop the user

DROP USER ''vaishu'user'@'localhost';
```



```
 #  Time      Action                                          Message                                                                  Duration / Fetch
 1  21:02:47  use wiprodb                                     Error Code: 1044. Access denied for user 'vaishu'@'localhost' to data...  0.000 sec
 2  21:02:52  grant all privileges on wiprodb.* to 'vaishu'@'localhost'  Error Code: 1044. Access denied for user 'vaishu'@'localhost' to data...  0.000 sec
```

Explanation of Commands

1. CREATE USER: This command creates a new user with the specified username and password. The ''vaishu'@'localhost' specifies that the user can connect from the local host.

2. GRANT: This command grants the specified privileges to the user. In this case, the user is granted SELECT, INSERT, UPDATE, and DELETE privileges on all tables in the LibrarySystem database.

3. REVOKE: This command revokes a specified privilege from the user. Here, we revoke the DELETE privilege, which means the user can no longer delete records from the tables in the LibrarySystem database.

4. DROP USER: This command deletes the user from the database, removing all associated privileges.

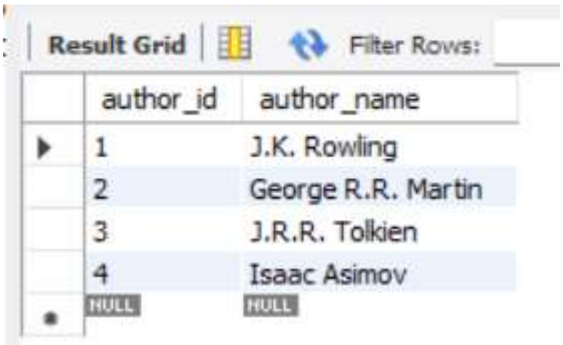
Assignment 7:

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. Inserting New Records

Insert into Authors

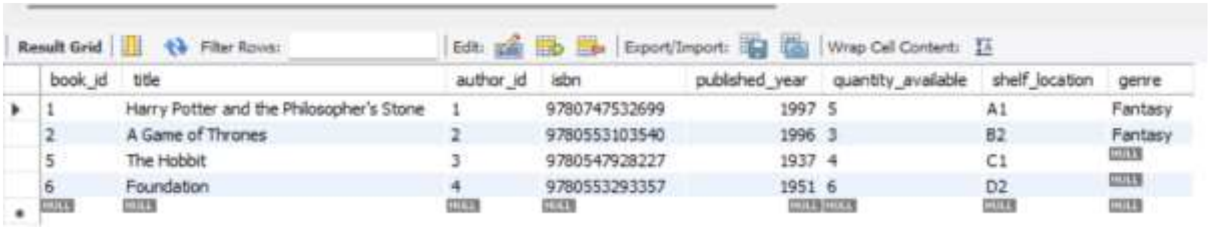
```
INSERT INTO Authors (author_name)

VALUES

('J.R.R. Tolkien'),

('Isaac Asimov');
```



author_id	author_name
1	J.K. Rowling
2	George R.R. Martin
3	J.R.R. Tolkien
4	Isaac Asimov
NULL	NULL

Insert into Books

```
INSERT INTO Books (title, author_id, isbn, published_year, quantity_available, shelf_location)

VALUES

('The Hobbit', (SELECT author_id FROM Authors WHERE author_name = 'J.R.R. Tolkien'),
'9780547928227', 1937, 4, 'C1'),

('Foundation', (SELECT author_id FROM Authors WHERE author_name = 'Isaac Asimov'),
'9780553293357', 1951, 6, 'D2');
```



book_id	title	author_id	isbn	published_year	quantity_available	shelf_location	genre
1	Harry Potter and the Philosopher's Stone	1	9780747532699	1997	5	A1	Fantasy
2	A Game of Thrones	2	9780553103540	1996	3	B2	Fantasy
5	The Hobbit	3	9780547928227	1937	4	C1	NULL
6	Foundation	4	9780553293357	1951	6	D2	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

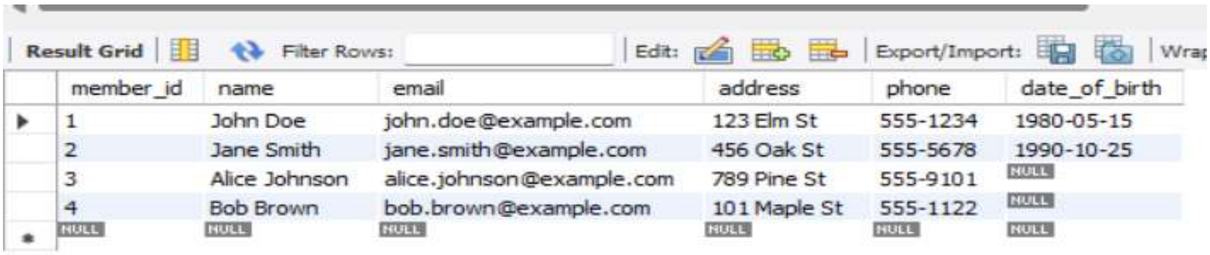
Insert into members

INSERT INTO Members (name, email, address, phone)

VALUES

('Alice Johnson', 'alice.johnson@example.com', '789 Pine St', '555-9101'),

('Bob Brown', 'bob.brown@example.com', '101 Maple St', '555-1122');



member_id	name	email	address	phone	date_of_birth
1	John Doe	john.doe@example.com	123 Elm St	555-1234	1980-05-15
2	Jane Smith	jane.smith@example.com	456 Oak St	555-5678	1990-10-25
3	Alice Johnson	alice.johnson@example.com	789 Pine St	555-9101	NULL
4	Bob Brown	bob.brown@example.com	101 Maple St	555-1122	NULL
NULL	NULL	NULL	NULL	NULL	NULL

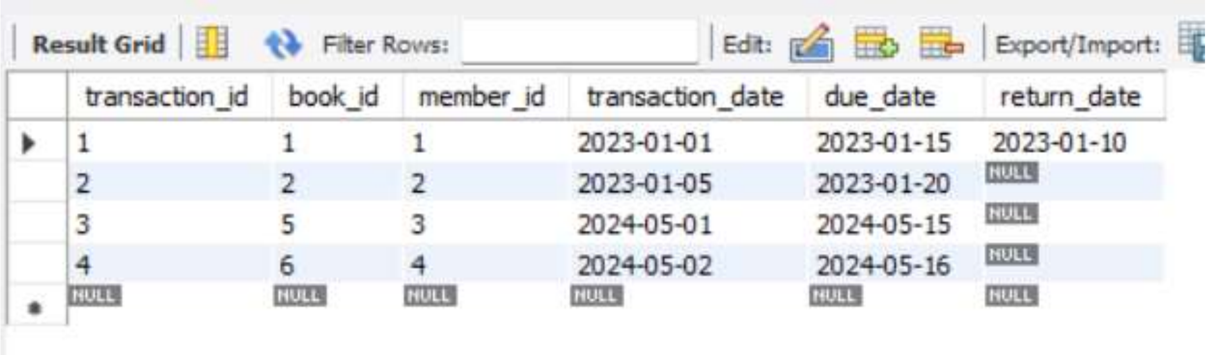
Insert into Transactions

INSERT INTO Transactions (book_id, member_id, transaction_date, due_date, return_date)

VALUES

((SELECT book_id FROM Books WHERE title = 'The Hobbit'), (SELECT member_id FROM Members WHERE name = 'Alice Johnson'), '2024-05-01', '2024-05-15', NULL),

((SELECT book_id FROM Books WHERE title = 'Foundation'), (SELECT member_id FROM Members WHERE name = 'Bob Brown'), '2024-05-02', '2024-05-16', NULL);

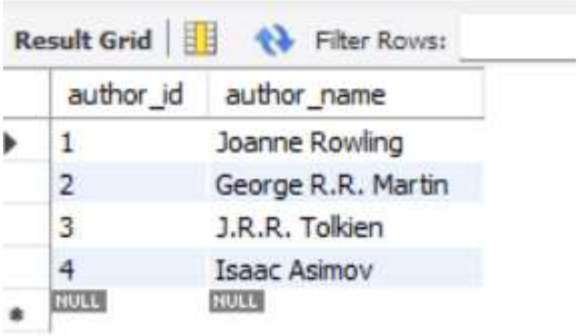


transaction_id	book_id	member_id	transaction_date	due_date	return_date
1	1	1	2023-01-01	2023-01-15	2023-01-10
2	2	2	2023-01-05	2023-01-20	NULL
3	5	3	2024-05-01	2024-05-15	NULL
4	6	4	2024-05-02	2024-05-16	NULL
NULL	NULL	NULL	NULL	NULL	NULL

2. Updating Existing Records

Update the name of an author

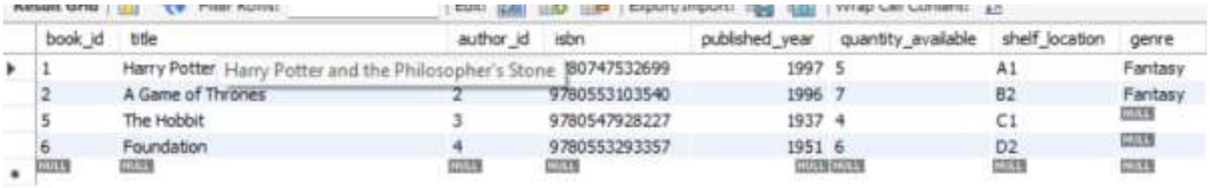
```
UPDATE Authors

SET author_name = 'Joanne Rowling'

WHERE author_name = 'J.K. Rowling';
```



author_id	author_name
1	Joanne Rowling
2	George R.R. Martin
3	J.R.R. Tolkien
4	Isaac Asimov
NULL	NULL

Update the quantity of a book

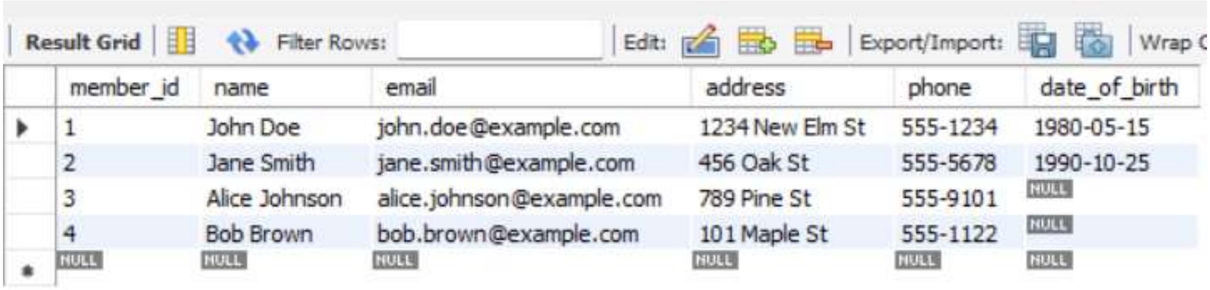
```
UPDATE Books

SET quantity_available = 7

WHERE title = 'A Game of Thrones';
```



book_id	title	author_id	isbn	published_year	quantity_available	shelf_location	genre
1	Harry Potter and the Philosopher's Stone		80747532699	1997	5	A1	Fantasy
2	A Game of Thrones	2	9780553103540	1996	7	B2	Fantasy
5	The Hobbit	3	9780547928227	1937	4	C1	NULL
6	Foundation	4	9780553293357	1951	6	D2	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Update a member's address

```
UPDATE Members

SET address = '1234 New Elm St'

WHERE email = 'john.doe@example.com';
```



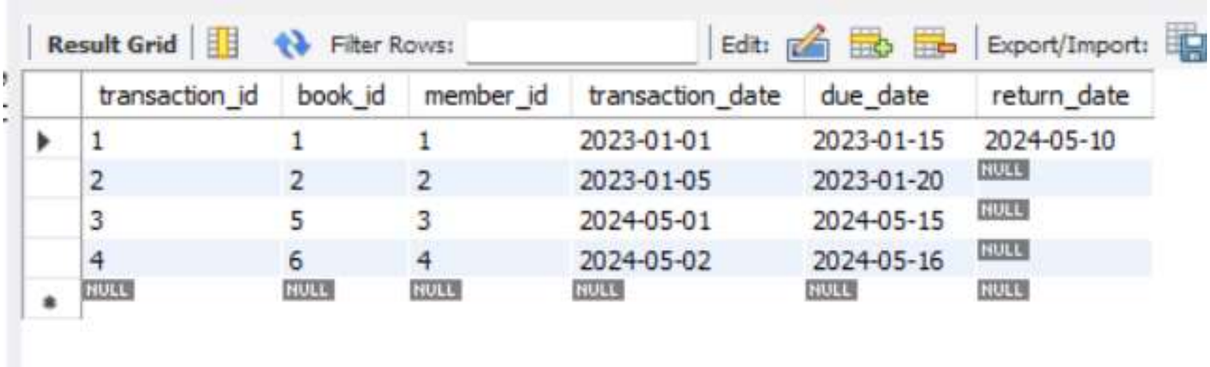
member_id	name	email	address	phone	date_of_birth
1	John Doe	john.doe@example.com	1234 New Elm St	555-1234	1980-05-15
2	Jane Smith	jane.smith@example.com	456 Oak St	555-5678	1990-10-25
3	Alice Johnson	alice.johnson@example.com	789 Pine St	555-9101	NULL
4	Bob Brown	bob.brown@example.com	101 Maple St	555-1122	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Update the return date of a transaction

UPDATE Transactions

SET return_date = '2024-05-10'

WHERE transaction_id = 1;



transaction_id	book_id	member_id	transaction_date	due_date	return_date
1	1	1	2023-01-01	2023-01-15	2024-05-10
2	2	2	2023-01-05	2023-01-20	NULL
3	5	3	2024-05-01	2024-05-15	NULL
4	6	4	2024-05-02	2024-05-16	NULL
NULL	NULL	NULL	NULL	NULL	NULL

3. Deleting Records

Delete an author

DELETE FROM Authors

WHERE author_name = 'George R.R. Martin';

Delete a book based on its title

DELETE FROM Books

WHERE title = 'Harry Potter and the Philosopher\'s Stone';

Delete a member based on their email

DELETE FROM Members

WHERE email = 'jane.smith@example.com';

Delete a transaction based on its ID

```
DELETE FROM Transactions

WHERE transaction_id = 2;
```
