Instructions:

1. Read the case study carefully and answer the questions based on the requirements described.
2. Use **ER diagrams**, **SQL schema definitions**, and written explanations where applicable.
3. Complete the exam by **12/11/2024 19:00**.

Case Study:

You have been tasked to design a database for an **Online Library Management System**. The system should keep track of books, users, and book loans. Below are the requirements:

1. **Books**: The library has a collection of books. Each book has the following details:
    - Title
    - Author
    - ISBN (unique identifier)
    - Genre (e.g., Fiction, Non-Fiction)
    - Published Year
    - Quantity Available
2. **Users**: Users of the library can borrow books. Each user has:
    - A unique ID
    - Full Name
    - Email Address
    - Membership Date
3. **Book Loans**: Users can borrow books. Each loan should record:

- User ID
- Book ISBN
- Loan Date
- Return Date
- Status (e.g., "borrowed", "returned", "overdue")

4. **Rules**:
   - A user can borrow multiple books, but the loan status must be updated when books are returned.
   - The library should not allow loans for unavailable books (i.e., if all copies of a book are borrowed).

Part 1: Conceptual Design - 25pts

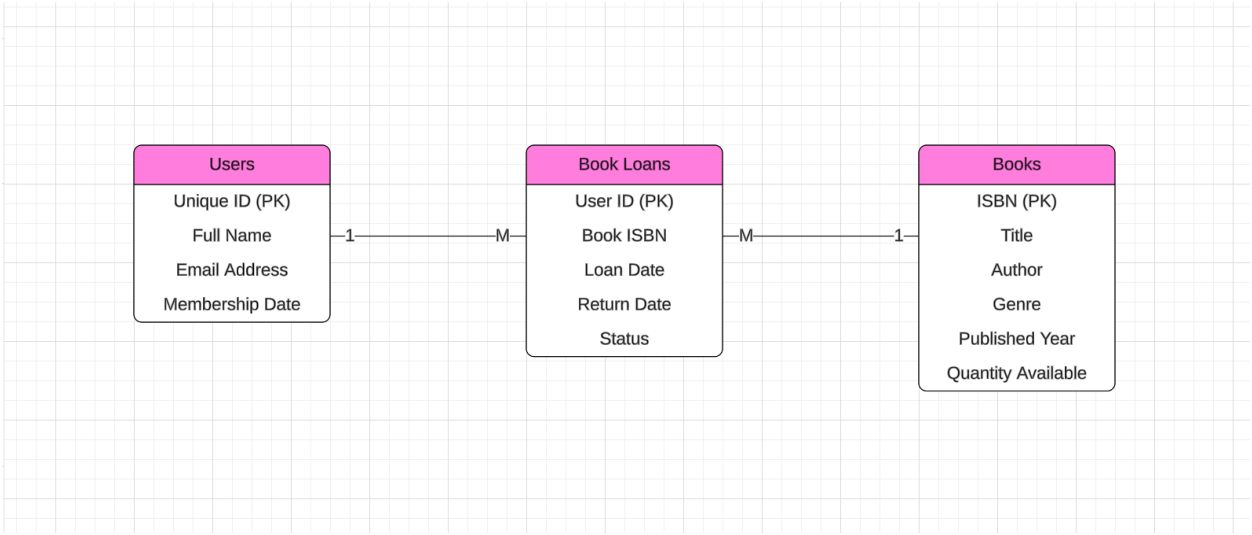
1. Draw an **Entity-Relationship (ER) Diagram** for the system based on the given requirements. Ensure you specify:
   - Entities
   - Attributes
   - Primary Keys
   - Relationships with cardinalities (e.g., one-to-many, many-to-many)

Specification:

- In each table below, their corresponding entities are placed in the upper part with the purple shaded, example: (Users, Book Loan, and Books. Below these entities are their corresponding attributes.

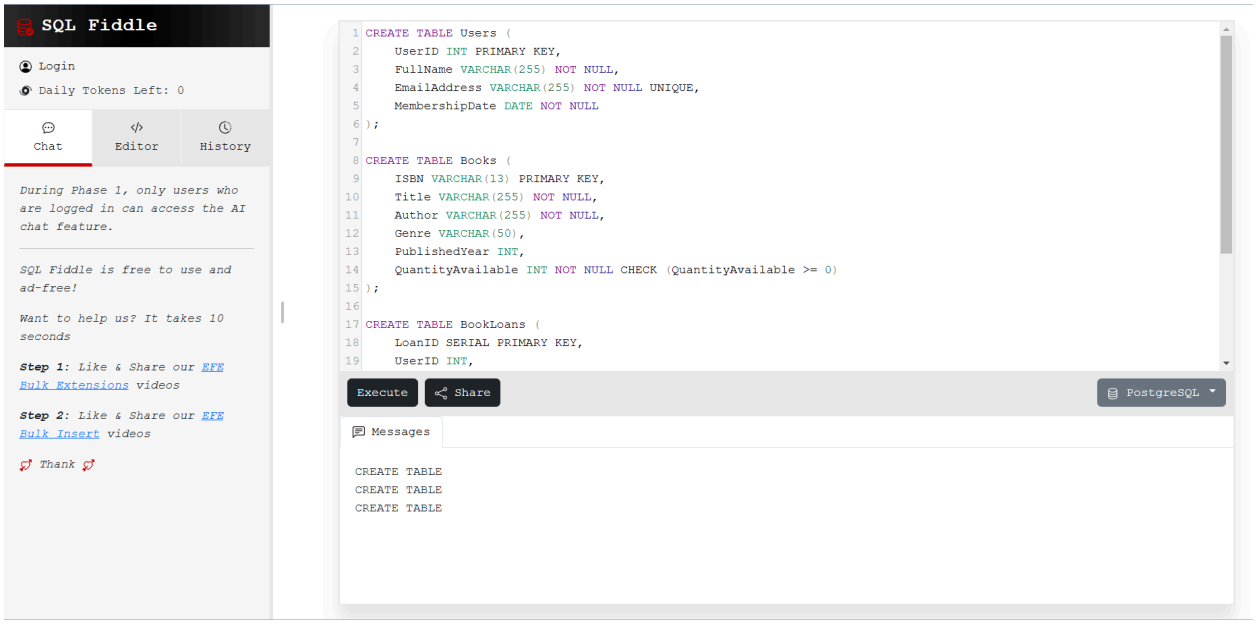
Relationships:

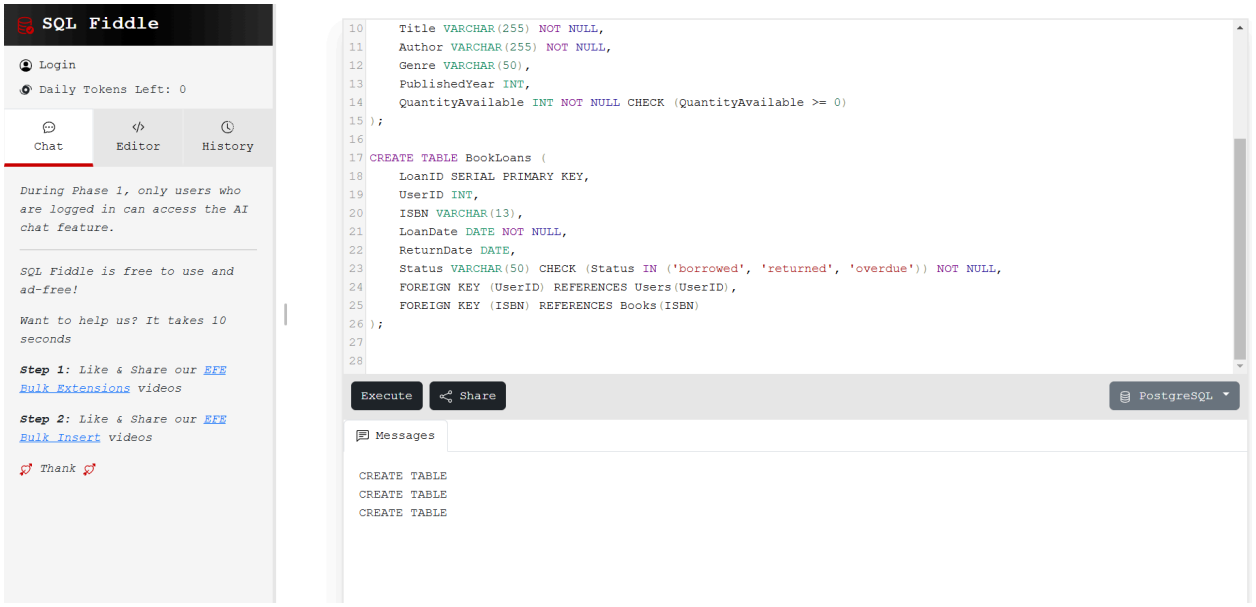
- **(Users to Book Loans):** One user can have many book loans. A user can borrow multiple books, and each borrowing record is tracked in the Book Loans table. **(One-to-Many).**
- **(Books to Book Loans):** One book can be loaned to many users over time. A single book can be borrowed multiple times by different users, with each borrowing event represented as a record in the Book Loans table. **(One-to-Many).**
- **(Users to Books):** Users can borrow many books, and each book can be borrowed by many users. This relationship is mediated by the Book Loans table, which connects the Users and Books entities. **(Many-to-Many)**. Also, there is no direct connection between Users and Books that is why it was not shown in the ERD below.



Part 2: Logical Design - 25pts

2. Translate the ER diagram into relational tables. Define:
    - Table schemas (list all attributes, data types, and constraints such as primary keys, foreign keys, and NOT NULL).



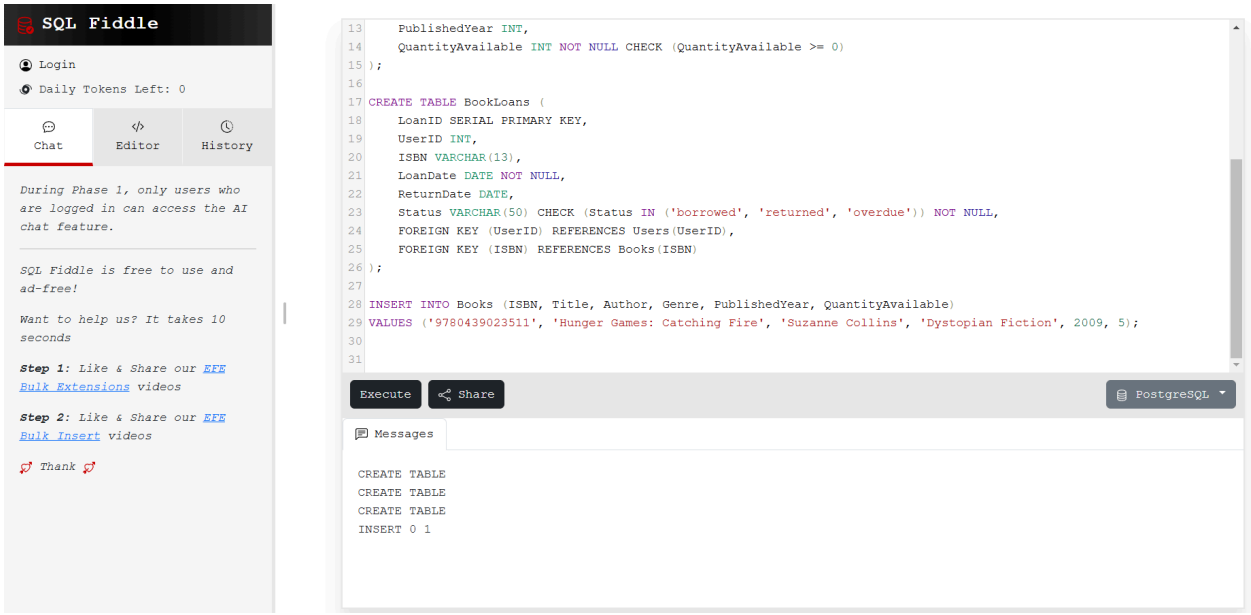


Part 3: SQL Queries

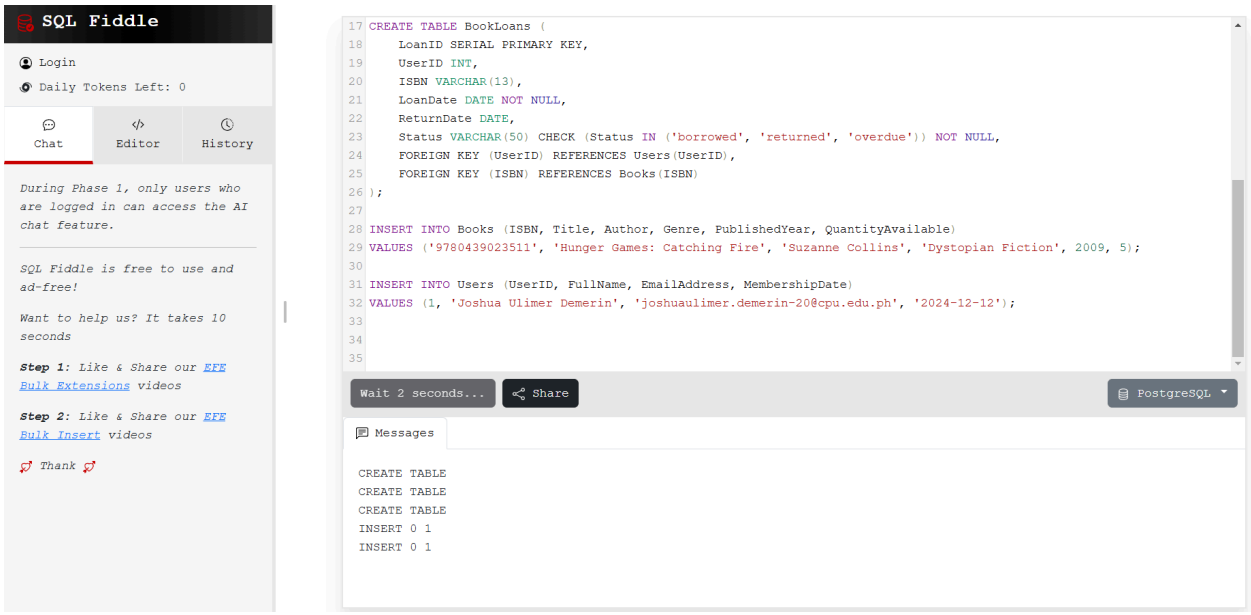
3. Write SQL queries for the following scenarios (15pts each):
    - a. Insert a new book into the library with a quantity of 5.
    - b. Add a new user to the system.

- c. Record a book loan for a user.
- d. Find all books borrowed by a specific user.
- e. List all overdue loans.

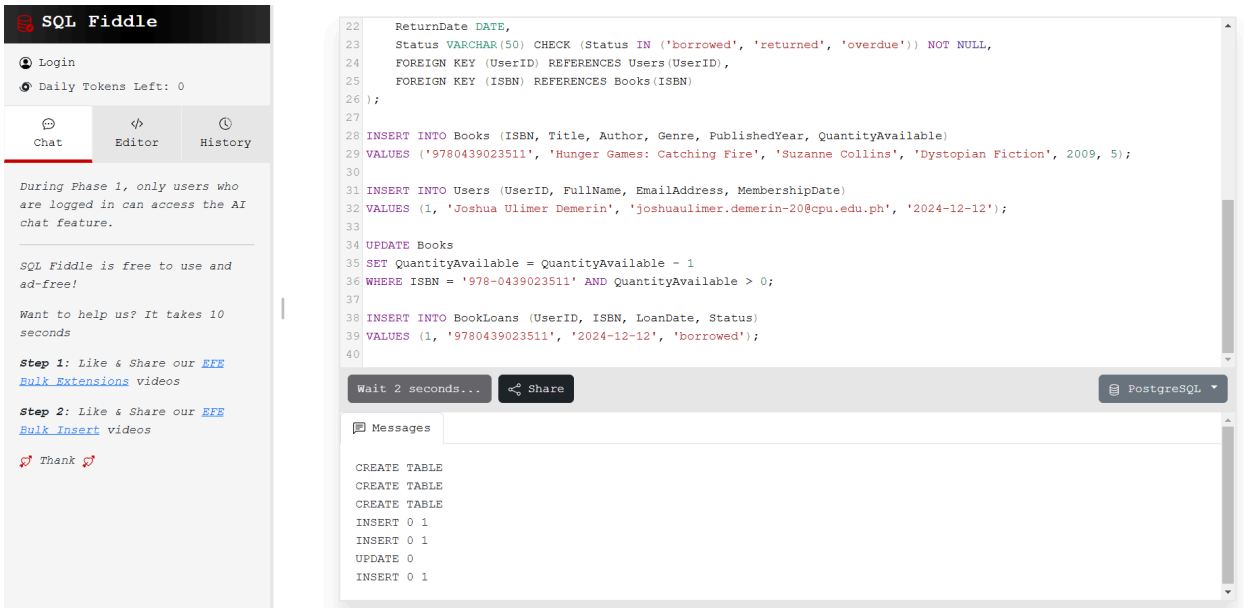
A.



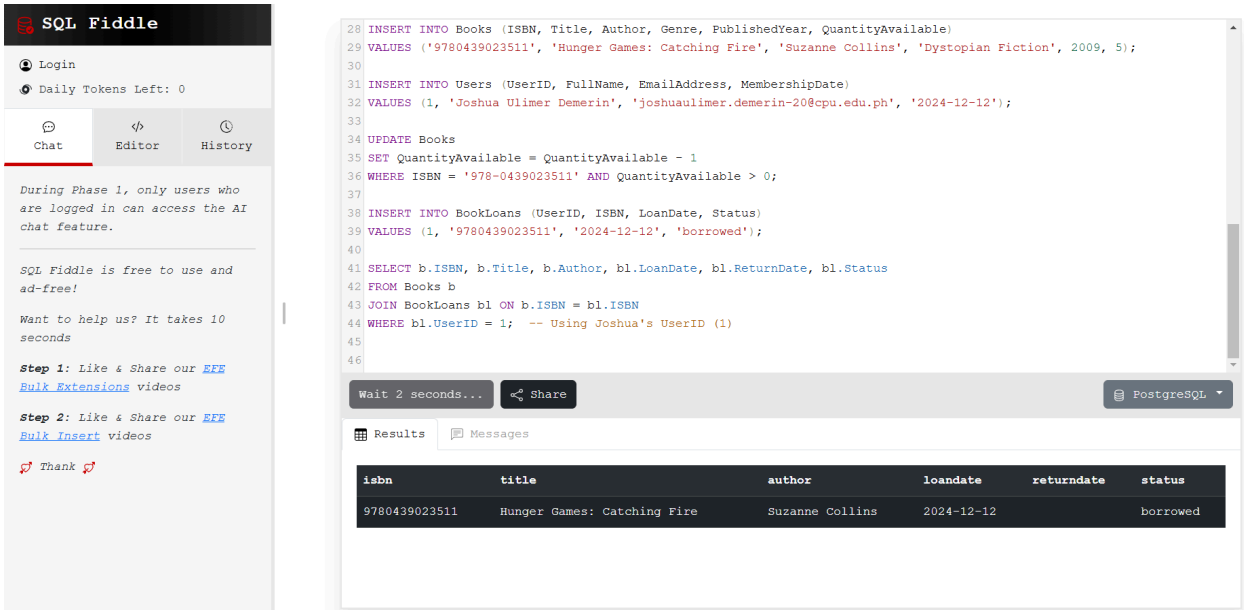
B.



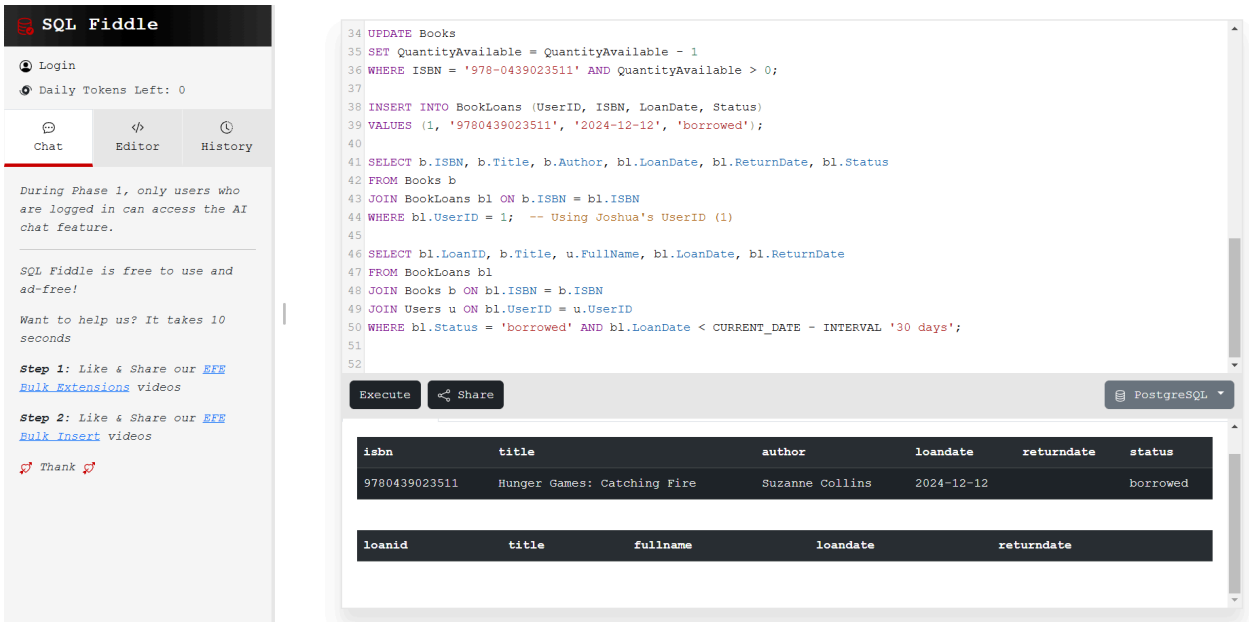
C.


SQL Fiddle
Login
Daily Tokens Left: 0
Chat
Editor
History
During Phase 1, only users who are logged in can access the AI chat feature.
SQL Fiddle is free to use and ad-free!
Want to help us? It takes 10 seconds
Step 1: Like & Share our EFE Bulk Extensions videos
Step 2: Like & Share our EFE Bulk Insert videos
Thank
ReturnDate DATE,
Status VARCHAR(50) CHECK (Status IN ('borrowed', 'returned', 'overdue')) NOT NULL,
FOREIGN KEY (UserID) REFERENCES Users(UserID),
FOREIGN KEY (ISBN) REFERENCES Books(ISBN)
);
INSERT INTO Books (ISBN, Title, Author, Genre, PublishedYear, QuantityAvailable)
VALUES ('9780439023511', 'Hunger Games: Catching Fire', 'Suzanne Collins', 'Dystopian Fiction', 2009, 5);
INSERT INTO Users (UserID, FullName, EmailAddress, MembershipDate)
VALUES (1, 'Joshua Ulimer Demerin', 'joshuaulimer.demerin-20@cpu.edu.ph', '2024-12-12');
UPDATE Books
SET QuantityAvailable = QuantityAvailable - 1
WHERE ISBN = '978-0439023511' AND QuantityAvailable > 0;
INSERT INTO BookLoans (UserID, ISBN, LoanDate, Status)
VALUES (1, '9780439023511', '2024-12-12', 'borrowed');
Wait 2 seconds...
Share
PostgreSQL
Messages
CREATE TABLE
CREATE TABLE
CREATE TABLE
INSERT 0 1
INSERT 0 1
UPDATE 0
INSERT 0 1

D.


SQL Fiddle
Login
Daily Tokens Left: 0
Chat
Editor
History
During Phase 1, only users who are logged in can access the AI chat feature.
SQL Fiddle is free to use and ad-free!
Want to help us? It takes 10 seconds
Step 1: Like & Share our EFE Bulk Extensions videos
Step 2: Like & Share our EFE Bulk Insert videos
Thank
INSERT INTO Books (ISBN, Title, Author, Genre, PublishedYear, QuantityAvailable)
VALUES ('9780439023511', 'Hunger Games: Catching Fire', 'Suzanne Collins', 'Dystopian Fiction', 2009, 5);
INSERT INTO Users (UserID, FullName, EmailAddress, MembershipDate)
VALUES (1, 'Joshua Ulimer Demerin', 'joshuaulimer.demerin-20@cpu.edu.ph', '2024-12-12');
UPDATE Books
SET QuantityAvailable = QuantityAvailable - 1
WHERE ISBN = '978-0439023511' AND QuantityAvailable > 0;
INSERT INTO BookLoans (UserID, ISBN, LoanDate, Status)
VALUES (1, '9780439023511', '2024-12-12', 'borrowed');
SELECT b.ISBN, b.Title, b.Author, bl.LoanDate, bl.ReturnDate, bl.Status
FROM Books b
JOIN BookLoans bl ON b.ISBN = bl.ISBN
WHERE bl.UserID = 1; -- Using Joshua's UserID (1)
Wait 2 seconds...
Share
PostgreSQL
Results
Messages
isbn
title
author
loandate
returndate
status
9780439023511
Hunger Games: Catching Fire
Suzanne Collins
2024-12-12
borrowed

E.



Part 4: Data Integrity and Optimization

4. Explain how you would ensure:

○ The prevention of borrowing books when no copies are available. (15 pts)

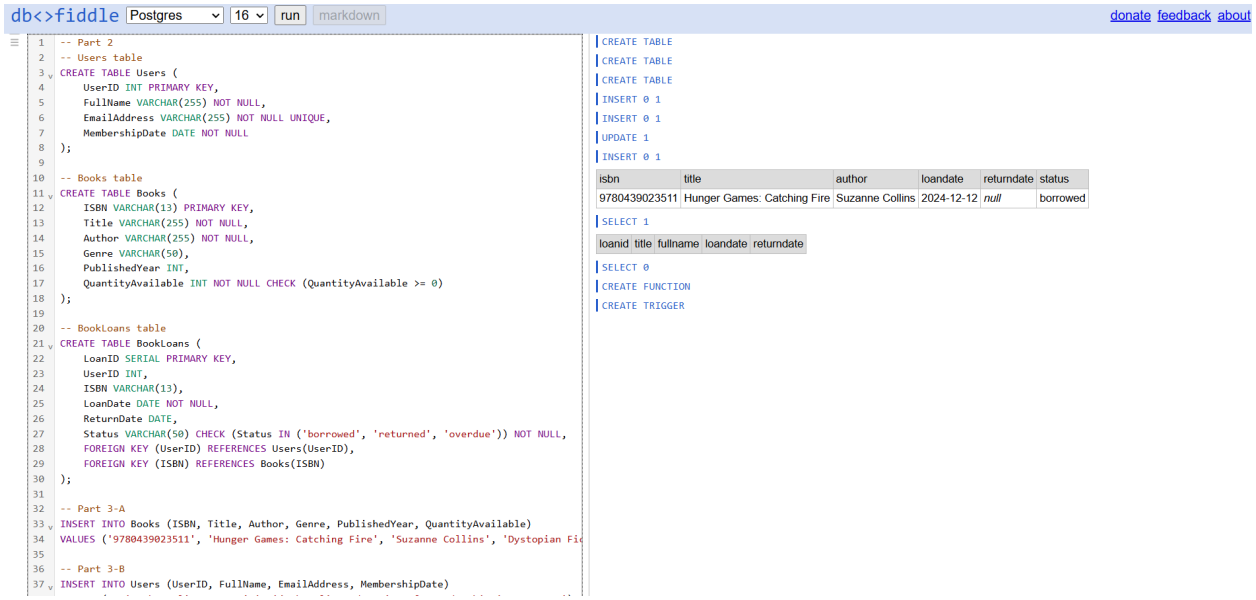
- I ensured that a user cannot borrow a book when there are no available copies, I created a trigger to prevent borrowing when no copies are available.
- The `prevent_borrow_when_no_copies()` function checks if the `QuantityAvailable` for a book is greater than 0 before allowing a borrow operation to happen. If the quantity is 0 or less, the function raises an exception and prevents the insertion of the loan.
- The trigger (`check_availability_before_insert`) is applied before inserting a new loan into the `BookLoans` table. This ensures that every time a book loan is attempted, the trigger runs first to check if the book has available copies.
- If a user tries to borrow a book with `QuantityAvailable = 0`, the trigger will raise an exception and prevent the loan from being created, thus ensuring that borrowing is only allowed when there are available copies.

○ Fast retrieval of overdue loans. (20 pts - with CODE and actual screenshot of performance).

- In order to acheive retrieving overdue loans efficiently, first create an index on the `BookLoans` table for the `Status` and `LoanDate` columns.
- The index will speed up the retrieval of overdue loans by allowing faster lookups for loans that are borrowed and have a `LoanDate` more than 30 days ago.
- The index on `Status` and `LoanDate` helps the database efficiently query overdue loans, significantly improving performance as the number of loans grows.
- Thus, If I have a large number of loans, querying overdue loans will be much faster because the index allows for quicker filtering based on `Status` and `LoanDate`.

db<>fiddle Postgres 16 run markdown

```
FROM BookLoans bl
JOIN Books b ON bl.ISBN = b.ISBN
JOIN Users u ON bl.UserID = u.UserID
WHERE bl.Status = 'borrowed' AND bl.LoanDate < CURRENT_DATE - INTERVAL '30 days';

-- Part 4
CREATE OR REPLACE FUNCTION prevent_borrow_when_no_copies()
RETURNS TRIGGER AS $$
BEGIN
    -- Debugging: Check if the trigger is being executed and the quantity is being checke
    RAISE NOTICE 'Checking availability for ISBN: %, QuantityAvailable: %', NEW.ISBN, (SE

    -- If no copies available, raise an exception
    IF (SELECT QuantityAvailable FROM Books WHERE ISBN = NEW.ISBN) <= 0 THEN
        RAISE EXCEPTION 'No copies available for this book';
    END IF;

    -- Update the QuantityAvailable for the book after a successful loan
    UPDATE Books
    SET QuantityAvailable = QuantityAvailable - 1
    WHERE ISBN = NEW.ISBN;

    -- Debugging: Confirm that the quantity is updated
    RAISE NOTICE 'Updated QuantityAvailable for ISBN: %, New QuantityAvailable: %', NEW.I

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER check_availability_before_insert
BEFORE INSERT ON BookLoans
FOR EACH ROW
EXECUTE FUNCTION prevent_borrow_when_no_copies();
```



Part 5: Reflection (25 pts)

5. What challenges might arise when scaling this database to handle millions of users and books? Suggest one solution for each challenge.

    - When scaling this database to handle millions of users and books, several challenges might arise. First, as the number of users and books grows, queries may slow down, especially when retrieving overdue loans or checking book availability. A solution to this would be indexing critical columns like Status and LoanDate to speed up queries. Second, managing data integrity and ensuring no books are borrowed when unavailable could become more complex as the database expands. Implementing triggers like the one to check book availability ensures that the system can still prevent errors. Lastly, as more data is added, the database might face performance issues during data updates, such as reducing QuantityAvailable on book loans. A possible solution here would be partitioning large tables, like BookLoans, to distribute the data across multiple storage locations, improving performance.

Deliverables:

- ER Diagram (hand-drawn or created using software).
- SQL table definitions and queries.
- Written responses to conceptual and reflection questions.
- Any assumptions you made.