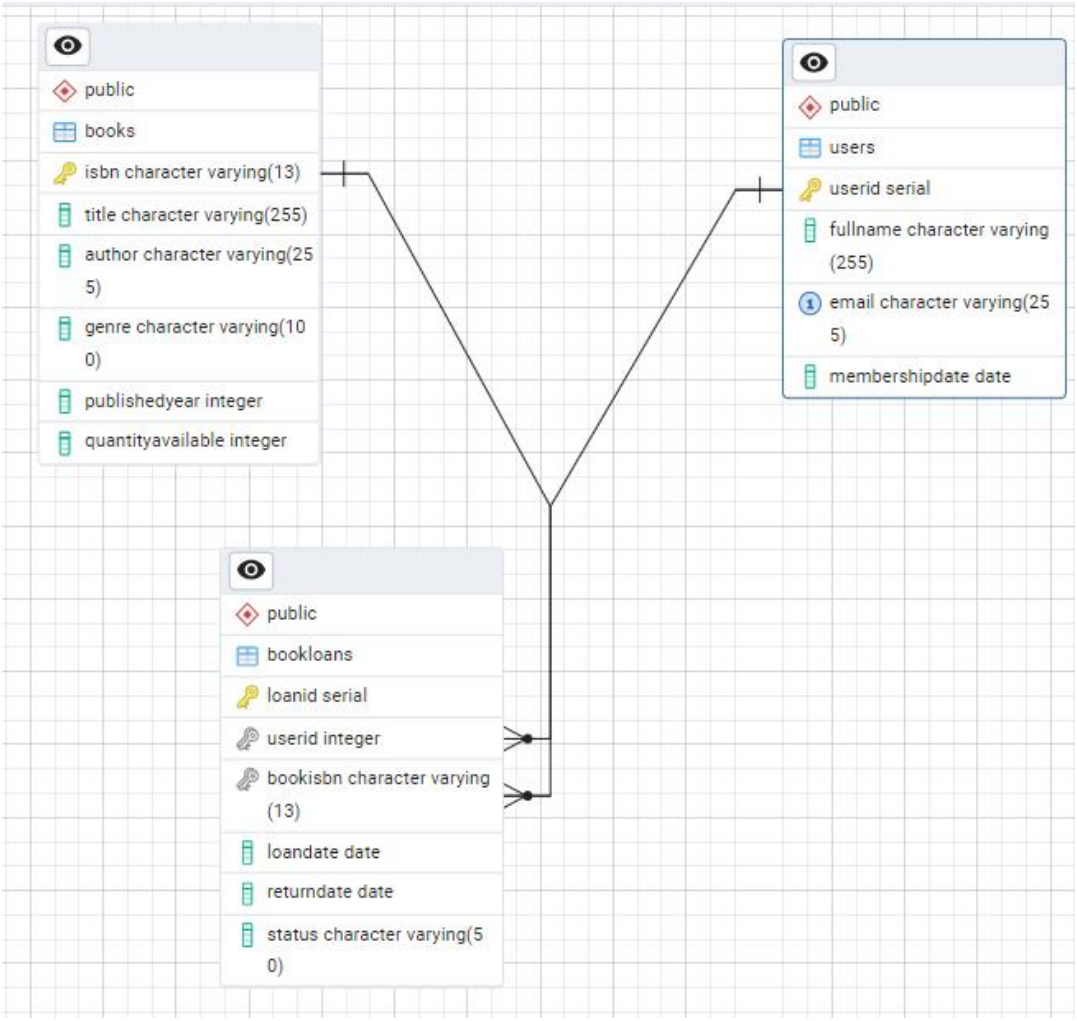
SE 2141
DATA BASE SYSTEMS

Laboratory #4
Online Library Management System

Sidryl T. Gerardo
Student

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)



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).

```
Query  Query History
1  CREATE TABLE Books (
2      ISBN VARCHAR(13) PRIMARY KEY,
3      Title VARCHAR(255) NOT NULL,
4      Author VARCHAR(255),
5      Genre VARCHAR(100),
6      PublishedYear INT,
7      QuantityAvailable INT NOT NULL
8  );
9
10 CREATE TABLE Users (
11     UserID SERIAL PRIMARY KEY,
12     FullName VARCHAR(255) NOT NULL,
13     Email VARCHAR(255) UNIQUE NOT NULL,
14     MembershipDate DATE NOT NULL
15 );
16
17 CREATE TABLE BookLoans (
18     LoanID SERIAL PRIMARY KEY,
19     UserID INT NOT NULL REFERENCES Users(UserID),
20     BookISBN VARCHAR(13) NOT NULL REFERENCES Books(ISBN),
21     LoanDate DATE NOT NULL,
22     ReturnDate DATE,
23     Status VARCHAR(50) NOT NULL
24 );
25

Data Output  Messages  Notifications
CREATE TABLE

Query returned successfully in 142 msec.
```

```
Query  Query History
1  CREATE TABLE Books (
2      ISBN VARCHAR(13) PRIMARY KEY,
3      Title VARCHAR(255) NOT NULL,
4      Author VARCHAR(255),
5      Genre VARCHAR(100),
6      PublishedYear INT,
7      QuantityAvailable INT NOT NULL
8  );
```

```
Query  Query History
1  CREATE TABLE Users (
2      UserID SERIAL PRIMARY KEY,
3      FullName VARCHAR(255) NOT NULL,
4      Email VARCHAR(255) UNIQUE NOT NULL,
5      MembershipDate DATE NOT NULL
6  );
```

```
Query  Query History
1  CREATE TABLE BookLoans (
2      LoanID SERIAL PRIMARY KEY,
3      UserID INT NOT NULL REFERENCES Users(UserID),
4      BookISBN VARCHAR(13) NOT NULL REFERENCES Books(ISBN),
5      LoanDate DATE NOT NULL,
6      ReturnDate DATE,
7      Status VARCHAR(50) NOT NULL
8  );
9
```

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

Query Query History

```
1 INSERT INTO Books (ISBN, Title, Author, Genre, PublishedYear, QuantityAvailable)
2 VALUES ('9781234567890', 'The Great Library', 'John Doe', 'Fiction', 2022, 5);
3
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 94 msec.
```

Query Query History

```
1 INSERT INTO Users (FullName, Email, MembershipDate)
2 VALUES ('Alice Smith', 'alice.smith@example.com', CURRENT_DATE);
3
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 91 msec.
```

Query Query History

```
1 INSERT INTO BookLoans (UserID, BookISBN, LoanDate, Status)
2 VALUES (1, '9781234567890', CURRENT_DATE, 'borrowed');
3
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 81 msec.
```

Query Query History

```
1 SELECT b.Title, b.Author, bl.LoanDate, bl.Status
2 FROM BookLoans bl
3 JOIN Books b ON bl.BookISBN = b.ISBN
4 WHERE bl.UserID = 1 AND bl.Status = 'borrowed';
```

Data Output Messages Notifications

	title character varying (255)	author character varying (255)	loandate date	status character varying (50)
1	The Great Library	John Doe	2024-12-11	borrowed

Query Query History

```
1 SELECT bl.LoanID, u.FullName, b.Title, bl.LoanDate, bl.ReturnDate
2 FROM BookLoans bl
3 JOIN Users u ON bl.UserID = u.UserID
4 JOIN Books b ON bl.BookISBN = b.ISBN
5 WHERE bl.Status = 'overdue';
6
```

Data Output Messages Notifications

	loanid integer	fullname character varying (255)	title character varying (255)	loandate date	returndate date

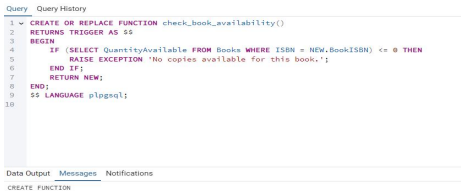
Part 4: Data Integrity and Optimization

4. Explain how you would ensure:

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

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

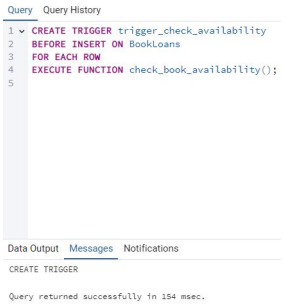
To prevent borrowing books when no copies are available, the system uses a combination of constraints and triggers. The Books table includes a QuantityAvailable column with a CHECK constraint to ensure the value never drops below zero. A trigger function validates book availability before any loan is processed, raising an exception if no copies are available. Another trigger automatically updates the quantity when books are borrowed or returned, ensuring real-time accuracy. This approach enforces data integrity, prevents errors, and maintains a consistent record of book availability.



```
CREATE OR REPLACE FUNCTION check_book_availability()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT QuantityAvailable FROM Books WHERE ISBN = NEW.BookISBN) <= 0 THEN
        RAISE EXCEPTION 'No copies available for this book.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION

Query returned successfully in 200 msec.
```



```
CREATE TRIGGER trigger_check_availability
BEFORE INSERT ON BookLoans
FOR EACH ROW
EXECUTE FUNCTION check_book_availability();
```

```
CREATE TRIGGER

Query returned successfully in 154 msec.
```

```
CREATE OR REPLACE FUNCTION update_quantity_on_loan()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Status = 'borrowed' THEN
        UPDATE Books SET QuantityAvailable = QuantityAvailable - 1 WHERE ISBN = NEW.BookISBN;
    ELSIF NEW.Status = 'returned' THEN
        UPDATE Books SET QuantityAvailable = QuantityAvailable + 1 WHERE ISBN = NEW.BookISBN;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION

Query returned successfully in 238 msec.
```

```
CREATE TRIGGER trigger_update_quantity
AFTER INSERT OR UPDATE ON BookLoans
FOR EACH ROW
EXECUTE FUNCTION update_quantity_on_loan();
```

```
CREATE TRIGGER

Query returned successfully in 238 msec.
```

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.

There are a number of difficulties in scaling an online library management system's database to accommodate millions of users and volumes. Caching and indexing are essential for effective retrieval, particularly for past-due loans and book availability, as query performance may deteriorate with increasing data volume. In order to maintain smooth scalability, distributed database systems or cloud-based solutions may be necessary because the sheer volume of data may exceed single-server storage limits. By using row-level locking and streamlining transaction processing, transaction conflicts caused by concurrent borrowing by a wide user base can be avoided. Although maintaining data integrity across large datasets is more difficult, the stability and dependability of the system can be protected using automated monitoring tools, frequent data audits, and solid backup plans.