SE 2141 – Laboratory 4

Online Library Management System

Name: Kimly John Vergara

Course & Year: BSSE – 2

Date: December 11, 2024

Part 1: Draw an Entity-Relationship (ER) Diagram for the system based on the given requirements.

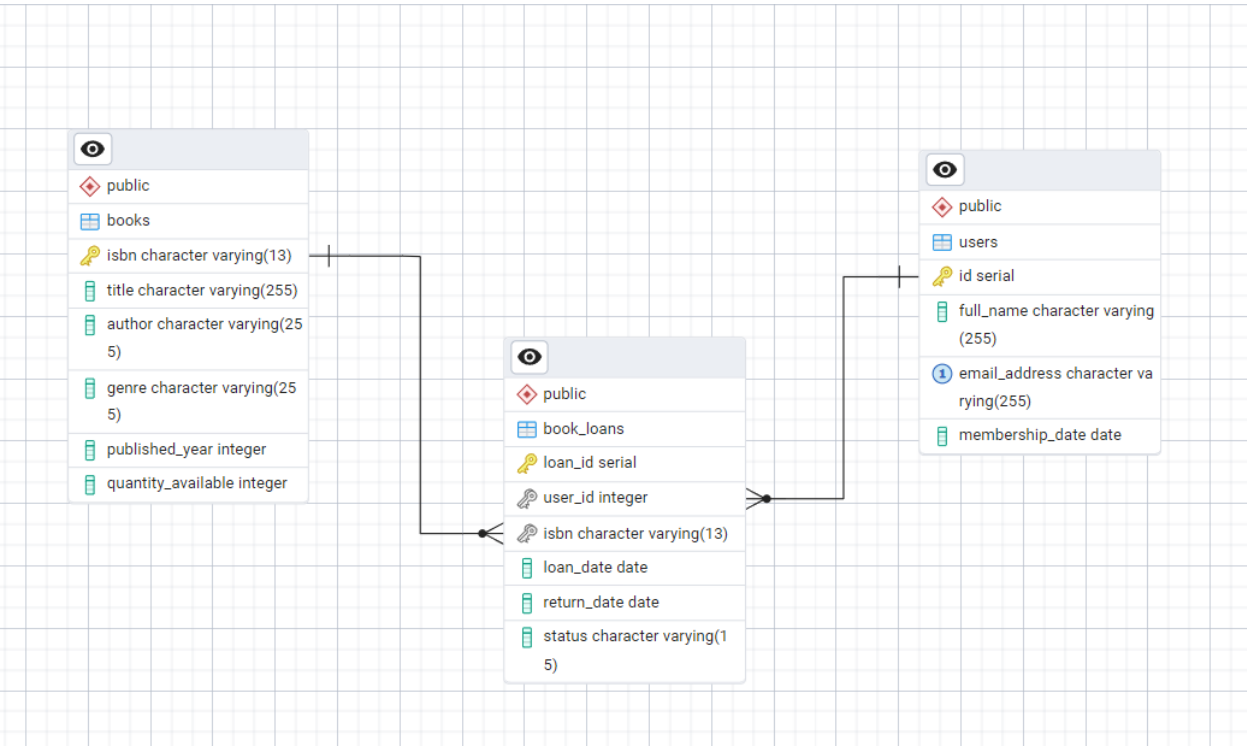


Diagram 1: Online Library Management System

The Entity-Relationship (ER) Diagram illustrates the structure of the library management system, highlighting its entities, attributes, primary keys, and relationships. Key entities include Books (with attributes like ISBN, Title, and Quantity Available), Users (with attributes like ID, Full Name, and Email Address), and Book Loans (with attributes like Loan ID, Loan Date, and Status). The diagram shows a one-to-many relationship between Users and Book Loans (a user can have multiple loans) and between Books and Book Loans (a book can be borrowed multiple times). These relationships and attributes form the foundation for implementing the database.

Part 2: Translate the ER diagram into relational tables.

```sql
CREATE TABLE Books (
    isbn VARCHAR(13) UNIQUE PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    genre VARCHAR(255) NOT NULL,
    published_year INT NOT NULL,
    quantity_available INT NOT NULL CHECK (quantity_available >= 0)
);
```

Diagram 2.1: Create Books Table

The Books table stores details about each book in the library, including Book_ID (Primary Key), Title, Author, ISBN, Genre, Published_Year, and Quantity_Available. The Book_ID is unique, ensuring each book can be identified distinctly. The other fields are mandatory, with Quantity_Available tracking the number of available copies for borrowing. Constraints ensure that each book has a unique identifier and all relevant information is provided.

```sql
CREATE TABLE Users (
    id SERIAL UNIQUE PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    email_address VARCHAR(255) UNIQUE NOT NULL,
    membership_date DATE NOT NULL CHECK (membership_date <= CURRENT_DATE)
);
```

Diagram 2.2: Create Users Table

The Users table contains information about library members, including User_ID (Primary Key), Full_Name, Email_Address, and Membership_Date. The User_ID uniquely identifies each user, while Email_Address is unique and required for communication purposes. The Membership_Date helps track when the user joined the library. Constraints enforce uniqueness for User_ID and Email_Address, ensuring no duplicates.

```sql
CREATE TABLE Book_Loans (
    loan_id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    isbn VARCHAR(13) NOT NULL,
    loan_date DATE NOT NULL,
    return_date DATE,
    status VARCHAR(15) NOT NULL CHECK (status IN ('borrowed', 'returned', 'overdue')),
    FOREIGN KEY (user_id) REFERENCES Users(id),
    FOREIGN KEY (isbn) REFERENCES Books(isbn)
);
```

Diagram 2.3: Create Book Loans Table

The Book_Loans table records the details of books borrowed by users, including Loan_ID (Primary Key), User_ID (Foreign Key), Book_ISBN (Foreign Key), Loan_Date, Return_Date, and Status. The Loan_ID uniquely identifies each loan transaction, while User_ID and Book_ISBN create relationships with the Users and Books tables. The Status tracks the loan's current state, and the Return_Date is updated when the book is returned. Constraints ensure data integrity and accurate tracking of loans.

Part 3: Write SQL queries for the following scenarios.

- A. Insert a new book into the library with a quantity of 5.

```sql
-- Part 3a
INSERT INTO Books (isbn, title, author, genre, published_year, quantity_available)
VALUES ('9780439023528', 'The Hunger Games', 'Suzanne Collins', 'Fiction', 2008, 5);
```

This query inserts a new book record into the Books table with all necessary details, including the book's Title, Author, ISBN, Genre, Published_Year, and a Quantity_Available of 5. The ISBN ensures that the book is uniquely identified in the library's collection.

- B. Add a new user to the system.

```sql
-- Part 3b
INSERT INTO Users (full_name, email_address, membership_date)
VALUES ('Kimly John Vergara', 'gmver27@gmail.com', CURRENT_DATE);
```

The query adds a new user to the Users table, including the user's Full_Name, Email_Address, and Membership_Date. The Email_Address is required and must be unique, ensuring no duplicate accounts in the system.

- C. Record a book loan for a user.

```sql
-- Part 3c
INSERT INTO Book_Loans (user_id, isbn, loan_date, status)
VALUES (1, '9780439023528', CURRENT_DATE, 'borrowed');
```

This query records a book loan in the Book_Loans table by associating a user with a borrowed book. It includes the User_ID, Book_ISBN, Loan_Date, and an initial Status of "borrowed". The loan status is essential for tracking whether the book has been returned or is overdue.

- D. Find all books borrowed by a specific user.

```sql
-- Part 3d
SELECT b.title, b.author, bl.loan_date, bl.status
FROM Book_Loans bl
JOIN Books b ON bl.isbn = b.isbn
WHERE bl.user_id = 1
ORDER BY bl.loan_date DESC;
```

Data Output | Messages | Notifications

	title character varying (255)	author character varying (255)	loan_date date	status character varying (15)
1	The Hunger Games	Suzanne Collins	2024-12-11	borrowed

This query retrieves all books borrowed by a specific user by joining the Book_Loans and Books tables. It uses the User_ID to filter the results and returns information about the books, such as the title, author, and loan status.

- E. List all overdue loans.

```
102  -- Part 3e
103  SELECT u.full_name, b.title, bl.loan_date, bl.return_date
104  FROM Book_Loans bl
105  JOIN Users u ON bl.user_id = u.id
106  JOIN Books b ON bl.isbn = b.isbn
107  WHERE bl.status = 'overdue'
108  ORDER BY bl.loan_date ASC;
```

This query lists all overdue loans by selecting records from the Book_Loans table where the Return_Date is past the current date and the loan Status is "borrowed". It helps track overdue books that need attention.

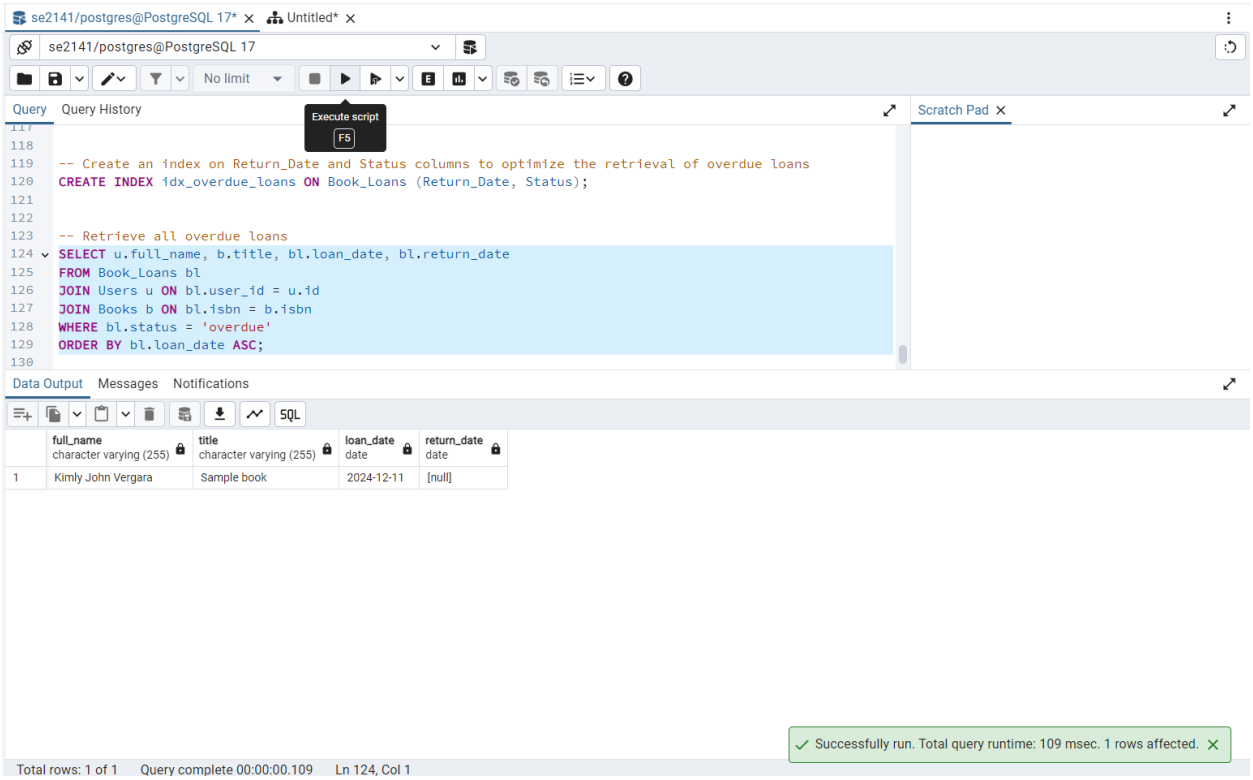
Part 4: Data Integrity and Optimization

- Explain how you would ensure the prevention of borrowing books when no copies are available.

```
-- Prevent Borrowing When Book Is Unavailable
CREATE OR REPLACE FUNCTION prevent_borrowing_unavailable_books()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if the book has any available copies
    IF (SELECT quantity_available FROM Books WHERE isbn = NEW.isbn) <= 0 THEN
        RAISE EXCEPTION 'Book is not available';
    END IF;

    -- Decrease the quantity of available books
    UPDATE Books
    SET quantity_available = quantity_available - 1
    WHERE isbn = NEW.isbn;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_borrowing
BEFORE INSERT ON Book_Loans
FOR EACH ROW
EXECUTE FUNCTION prevent_borrowing_unavailable_books();
```

This query defines a trigger function prevent_borrowing_unavailable_books that ensures a user cannot borrow a book if no copies are available. Before a new book loan is recorded in the Book_Loans table, the function checks if the quantity_available for the specified book (isbn) is greater than zero. If not, it raises an exception, preventing the loan. If copies are available, the quantity is decreased by 1 to reflect the borrowed book. The trigger before_borrowing is set to execute the function before each insert operation on the Book_Loans table.

- Explain how you would ensure fast retrieval of overdue loans.



This query creates an index on the Return_Date and Status columns in the Book_Loans table using the CREATE INDEX statement. The index, named idx_overdue_loans, optimizes the retrieval of overdue loans by allowing faster lookups based on these two columns. The query that follows retrieves overdue loans by joining the Book_Loans, Users, and Books tables. It filters for loans where the status is 'overdue' and orders the results by the loan_date in ascending order. This approach ensures that overdue loans are retrieved efficiently, even with large datasets, by utilizing the created index for faster query execution.

Part 5: Reflection.

When scaling the database to accommodate millions of users and books, a few challenges might come up. First, there's the issue of performance and query speed—retrieving large amounts of data, like overdue loans or book details, might slow things down. To fix this, indexing frequently searched columns like User_ID, ISBN, Return_Date, and Status can help speed up query execution and make data retrieval faster. Another concern is the database size and storage—storing tons of data can eventually lead to performance problems. A solution for this would be data partitioning, like splitting the Book_Loans table by date or region, which helps spread out the data and improves performance. Concurrency and data integrity might also be an issue, especially when lots of users are borrowing books at once. Using transaction management and locking mechanisms, like optimistic or pessimistic locking, can help avoid conflicts and keep the data consistent. As the system grows, maintaining availability is key, so implementing database replication and load balancing can ensure the system stays up and running without interruptions. Lastly, backup and recovery processes need to be efficient to handle all the growing data. Automating incremental backups and using cloud-based solutions for disaster recovery can help ensure a fast recovery and reduce data loss.

Assumptions made:

- I assumed that the Primary Key of the Books table is the ISBN since it is unique in every book.