SE 2141 - Laboratory 4
Online Library Management System

Name: Constantino, Michael Jay C.
Course/Year: BSSE - 2

Date: 11/12/24

Part 1: Conceptual Design

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

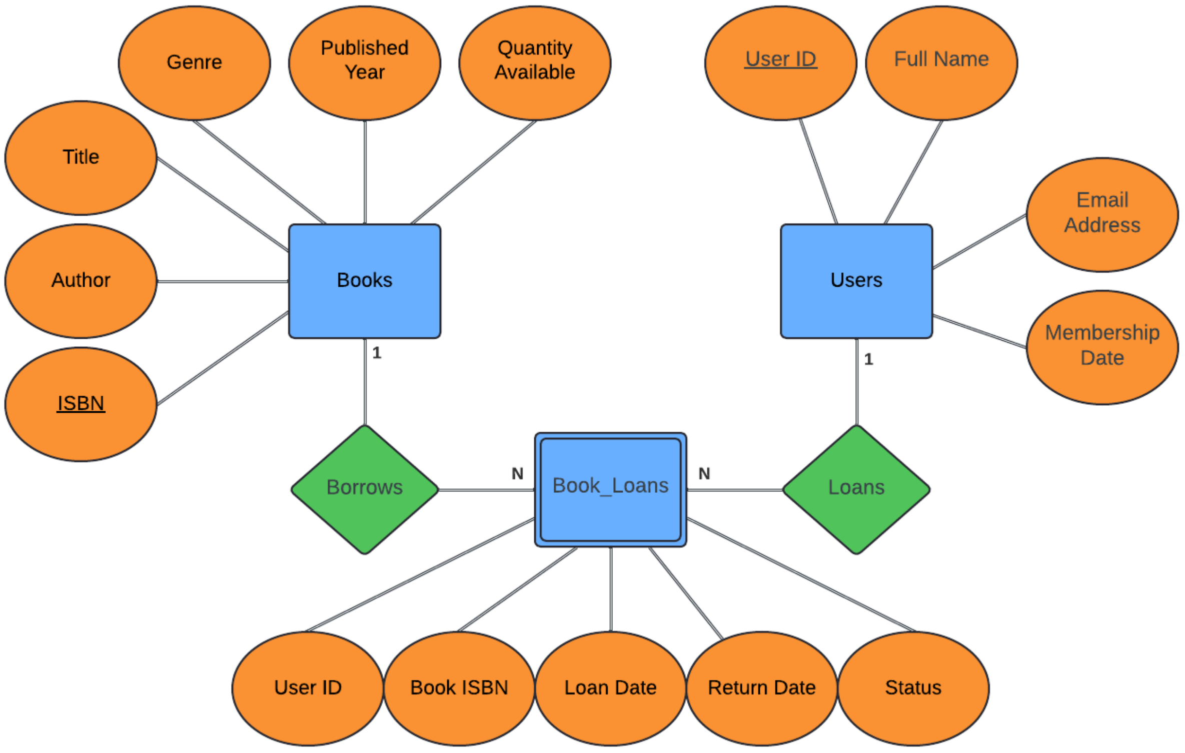
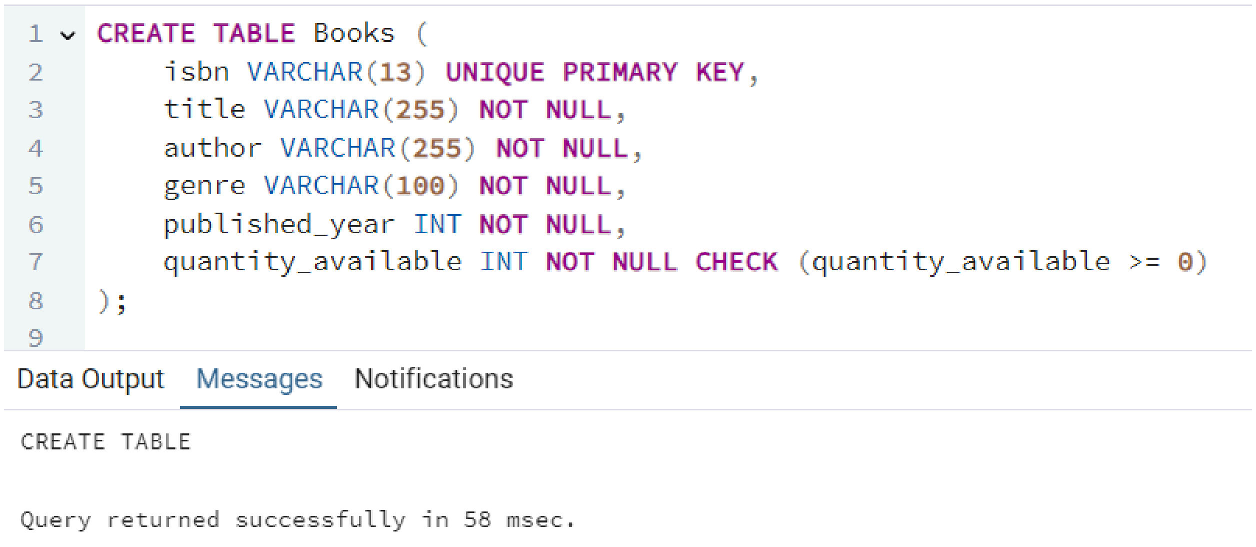


Diagram 1: Online Library Management System (Conceptual Model)

Here is the conceptual model, in here you can see the three entities the books, the users and the book loans, in this model the book loans serves as the associative table to handle the many-to-many relationship of the books to the users, the users have the attributes: a unique ID, full name, email address, and membership date, where the User Id serves as the unique identifier. Then, the books which has the attributes: title, author, ISBN (unique identifier), genre (e.g., Fiction, Non-Fiction), published year, and the quantity available. Then the last entity is the book loans which has foreign keys from both the books and users table, it has the attributes: user ID, book ISBN, loan date, return date , and the status (e.g., "borrowed", "returned", "overdue"). Then the book loans handles the relationship of the books to the users where the user can borrow a book.

Part 2: Logical Design

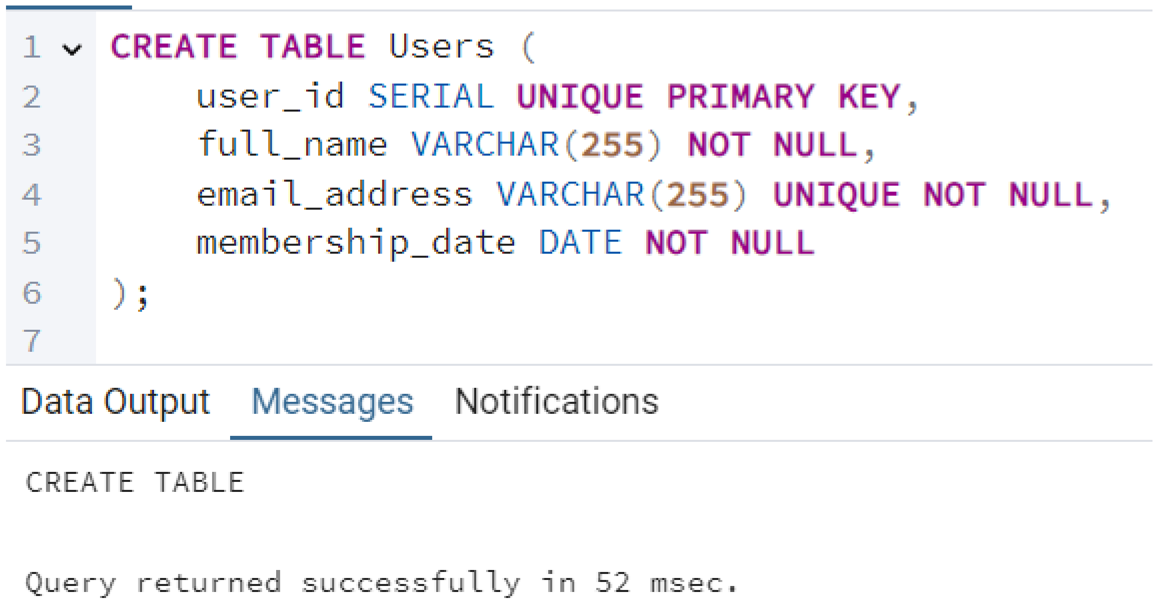
2. Translate the ER diagram into relational tables.



```sql
CREATE TABLE Books (
    isbn VARCHAR(13) UNIQUE PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    genre VARCHAR(100) NOT NULL,
    published_year INT NOT NULL,
    quantity_available INT NOT NULL CHECK (quantity_available >= 0)
);
```

```
CREATE TABLE

Query returned successfully in 58 msec.
```

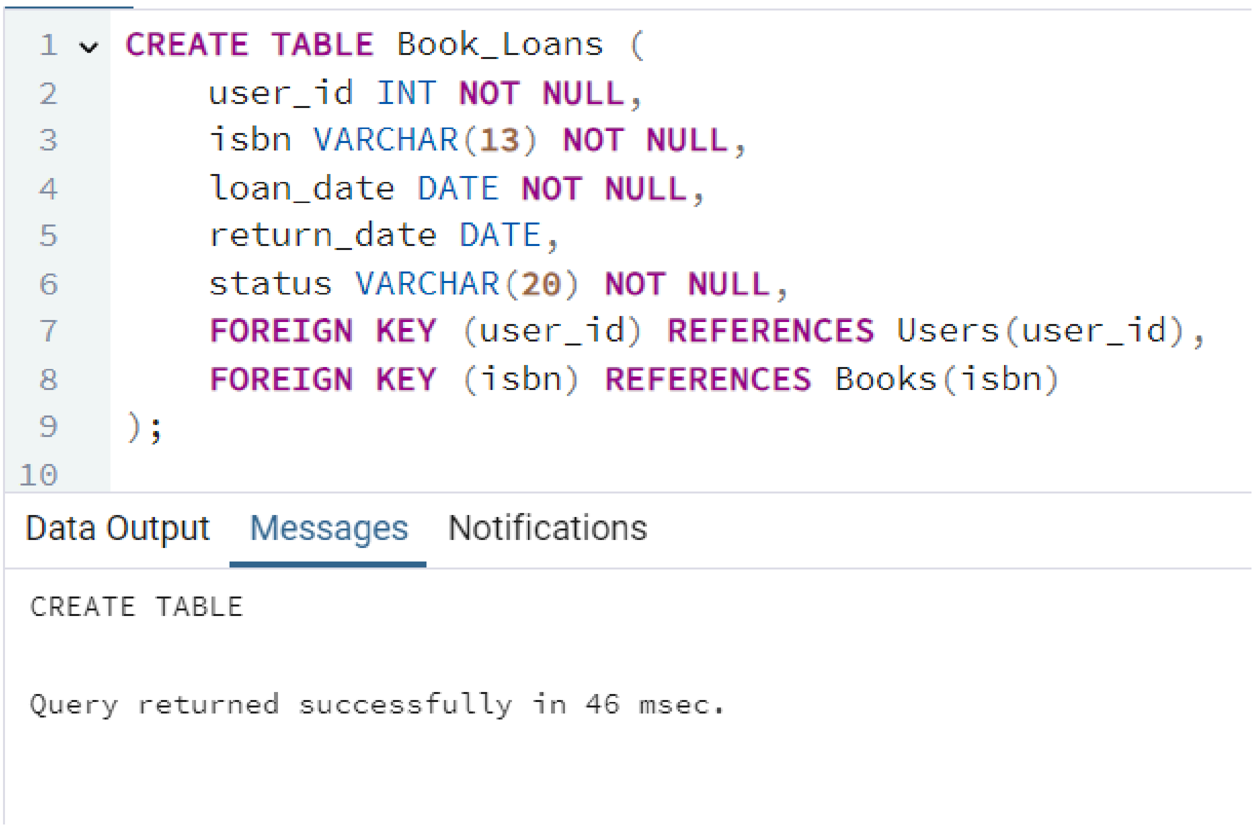
In here I created a Books table with the attributes isbn, title, author, genre, published_year, and quantity_available, it each has its corresponding constraints. In the isbn I added the unique constraint since no different books have the same isbn and made it varchar(13) to follow the ISBN 13 format, I also added a check in the quantity_available to handle the prevention of borrowing books when no copies are available.



```
CREATE TABLE Users (
    user_id SERIAL UNIQUE PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    email_address VARCHAR(255) UNIQUE NOT NULL,
    membership_date DATE NOT NULL
);

```

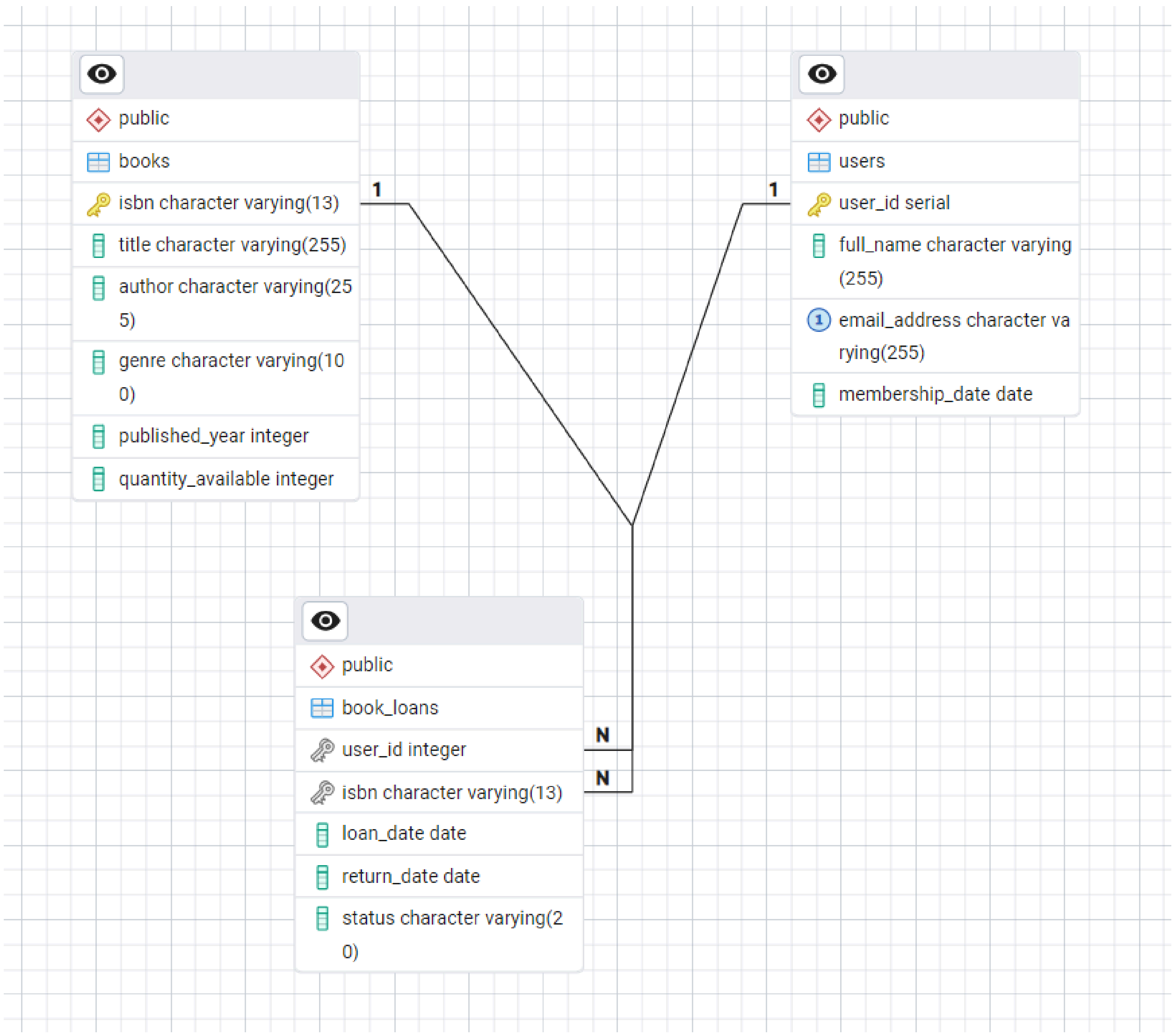
```
CREATE TABLE

Query returned successfully in 52 msec.
```

In here I created a Users table with the attributes user_id, full_name, email_address, and membership_date, it each has its corresponding constraints. In the isbn I added the unique constraint since the user_id for each user should be unique and made it a SERIAL type so that it auto increments. I also added the unique constraint for the email_address since different users cant have the email address.


```sql
CREATE TABLE Book_Loans (
    user_id INT NOT NULL,
    isbn VARCHAR(13) NOT NULL,
    loan_date DATE NOT NULL,
    return_date DATE,
    status VARCHAR(20) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (isbn) REFERENCES Books(isbn)
);
```

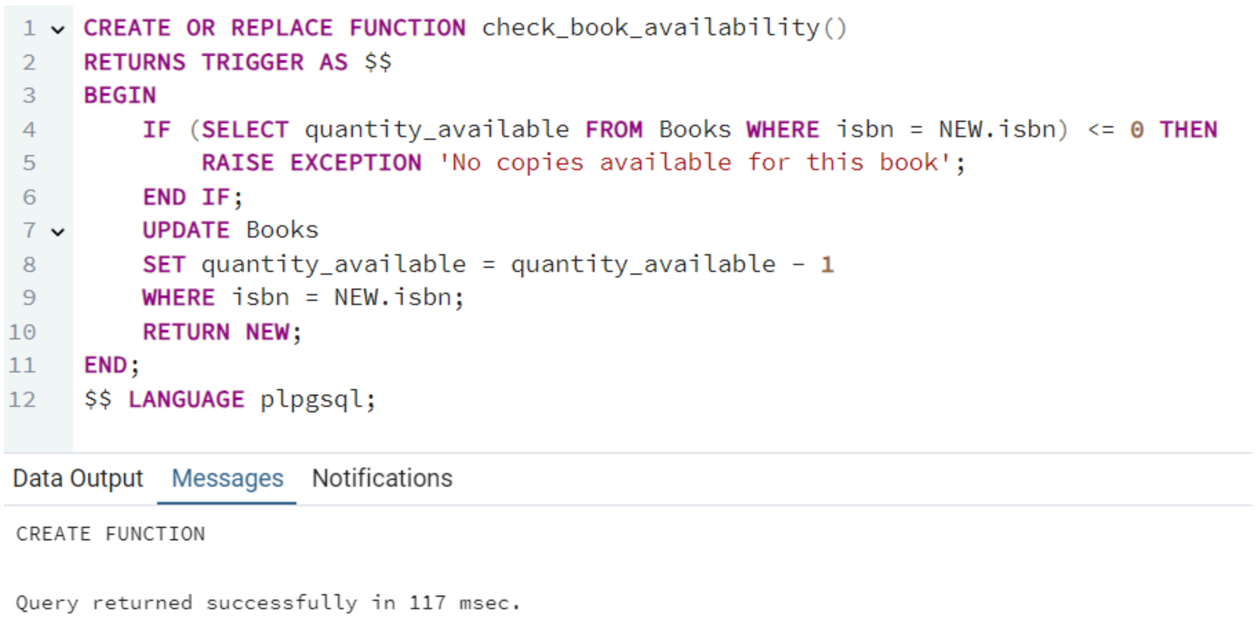
```
CREATE TABLE


Query returned successfully in 46 msec.
```

In here I created a Book_Loans table with the attributes user_id, isbn, loan_date, and return_date, and status, it each has its corresponding constraints. The user_id and isbn are both FOREIGN KEYS from the books and users table, and in here I assumed that the return_date can be null, since a book that has not been returned has no return_date.


public
books
isbn character varying(13)
title character varying(255)
author character varying(255)
genre character varying(100)
published_year integer
quantity_available integer
public
users
user_id serial
full_name character varying (255)
email_address character varying(255)
membership_date date
public
book_loans
user_id integer
isbn character varying(13)
loan_date date
return_date date
status character varying(20)
1
1
N
N

Here is the logical model created from the creation table queries from earlier.

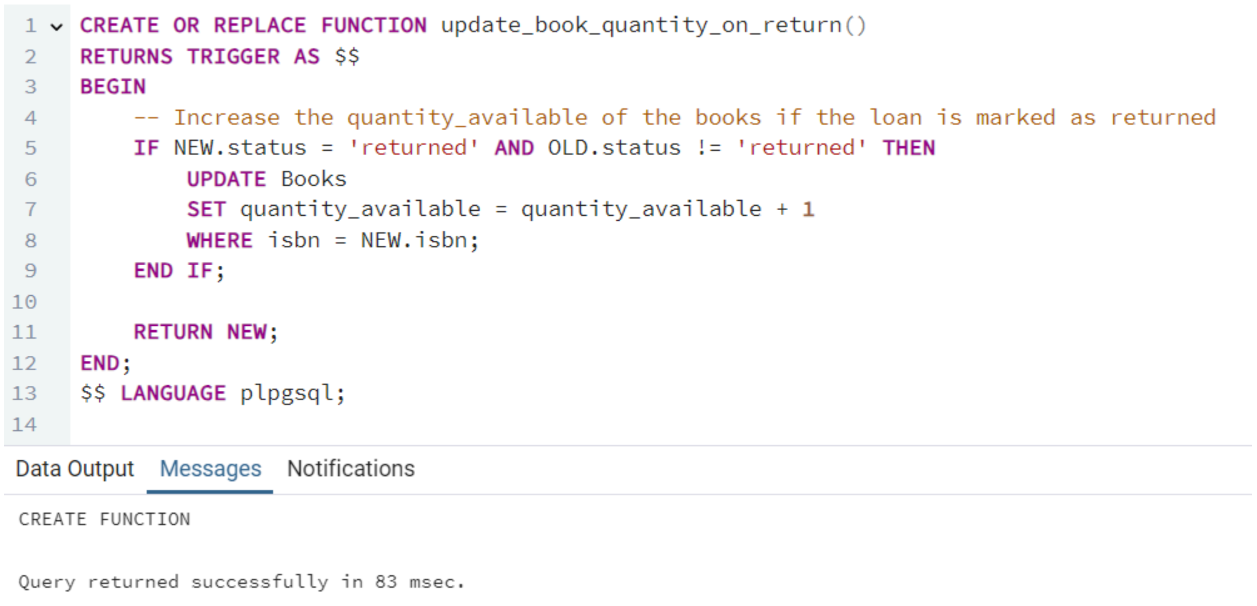
RULES



```
CREATE OR REPLACE FUNCTION check_book_availability()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT quantity_available FROM Books WHERE isbn = NEW.isbn) <= 0 THEN
        RAISE EXCEPTION 'No copies available for this book';
    END IF;
    UPDATE Books
    SET quantity_available = quantity_available - 1
    WHERE isbn = NEW.isbn;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION

Query returned successfully in 117 msec.
```

In here I created a function to check the book availability, where if the quantity_available is less than or equal to zero then it would raise an exception that there are no more copies available for the book, then it would also subtract from the quantity available if the book is borrowed.



```
CREATE OR REPLACE FUNCTION update_book_quantity_on_return()
RETURNS TRIGGER AS $$
BEGIN
    -- Increase the quantity_available of the books if the loan is marked as returned
    IF NEW.status = 'returned' AND OLD.status != 'returned' THEN
        UPDATE Books
        SET quantity_available = quantity_available + 1
        WHERE isbn = NEW.isbn;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

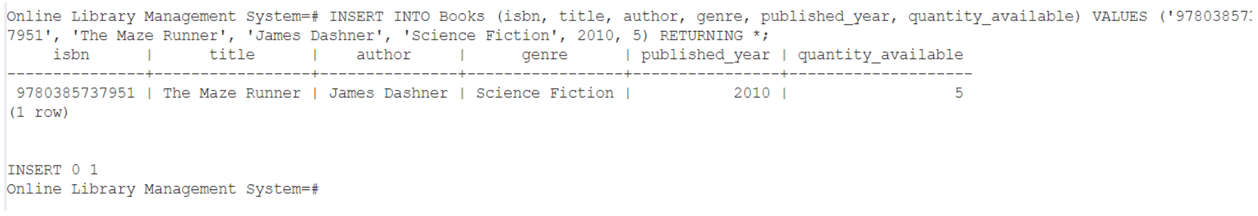
```
CREATE FUNCTION

Query returned successfully in 83 msec.
```

Then, in here i created a function to handle the returned books where it would add the quantity_available of the table if a book loan has an status ‘returned’. Meaning that if the user returns a book a book would be added back to the library meaning the quantity available increases by one.

Part 3: SQL Queries

3. Write SQL queries for the following scenarios

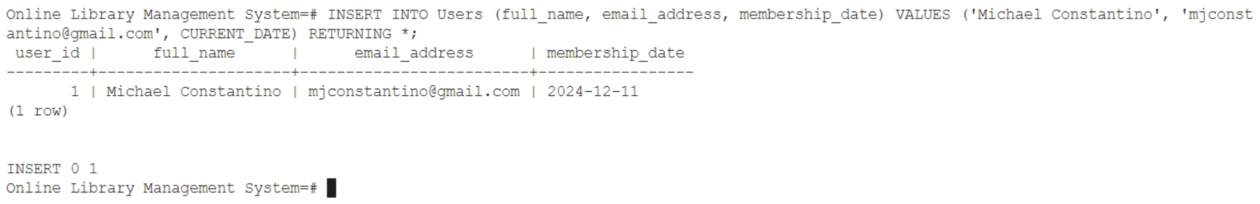
○ a. Insert a new book into the library with a quantity of 5.



```
Online Library Management System=# INSERT INTO Books (isbn, title, author, genre, published_year, quantity_available) VALUES ('97803857
7951', 'The Maze Runner', 'James Dashner', 'Science Fiction', 2010, 5) RETURNING *;
     isbn      |      title      |    author     |      genre      | published_year | quantity_available
---------------+-----------------+---------------+-----------------+----------------+--------------------
 9780385737951 | The Maze Runner | James Dashner | Science Fiction |           2010 |                  5
(1 row)


INSERT 0 1
Online Library Management System=#
```

So in this query I used the INSERT method to insert values into the books table, then I inputted values needed to meet the requirements and added the value of 5 to the quantity_available, then I used RETURNING * to show the inputted value into the table..

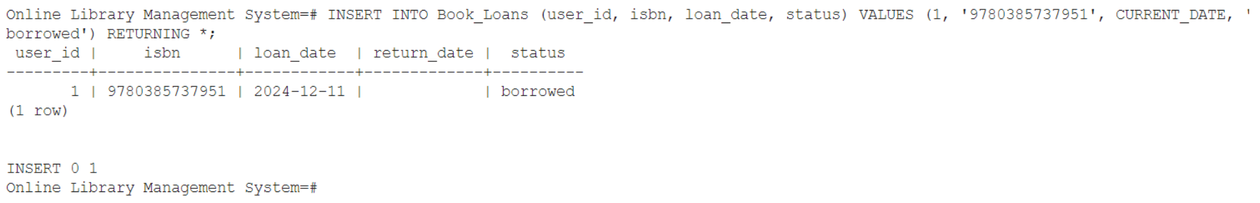
○ b. Add a new user to the system.



```
Online Library Management System=# INSERT INTO Users (full_name, email_address, membership_date) VALUES ('Michael Constantino', 'mjconst
antino@gmail.com', CURRENT_DATE) RETURNING *;
 user_id |      full_name      |      email_address       | membership_date
---------+---------------------+--------------------------+-----------------
       1 | Michael Constantino | mjconstantino@gmail.com  | 2024-12-11
(1 row)


INSERT 0 1
Online Library Management System=# 
```

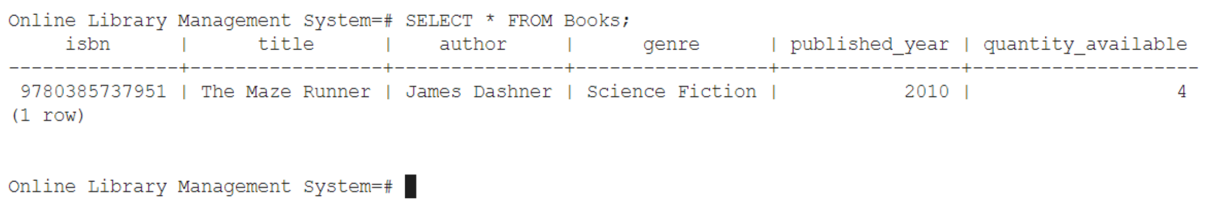
So in this query I used the INSERT method to insert a new user into the users table, then I inputted values needed to meet the requirements needed to register a new user ‘Michael Constantino’ to the database, then I used RETURNING * to show the inputted value into the table.

○ c. Record a book loan for a user.



```
Online Library Management System=# INSERT INTO Book_Loans (user_id, isbn, loan_date, status) VALUES (1, '9780385737951', CURRENT_DATE, '
borrowed') RETURNING *;
 user_id |     isbn      | loan_date  | return_date |  status
---------+---------------+------------+-------------+----------
       1 | 9780385737951 | 2024-12-11 |             | borrowed
(1 row)


INSERT 0 1
Online Library Management System=#
```

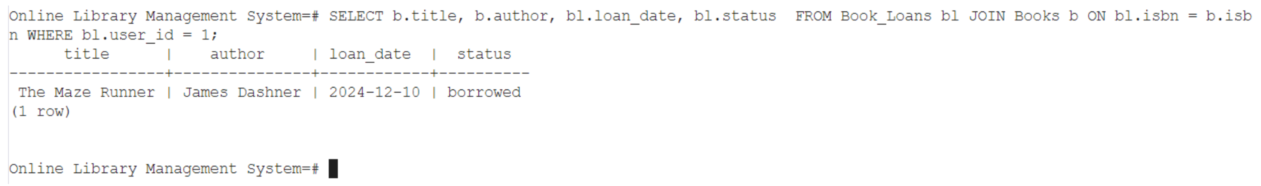
So in this query I used the INSERT method to insert a record of a book_loan, where it references the ISBN of the book entity from the query earlier and also the user id from the query earlier, after that I added the loan_date which is the current date by just putting the CURRENT_DATE into the values and also the status which is ‘borrowed’, as you can see that there the return_date is null because the book has not been returned yet, that is why in the table creation I did not put the NOT NULL constraint into the return_date row.



```
Online Library Management System=# SELECT * FROM Books;
     isbn      |      title      |    author     |      genre      | published_year | quantity_available
---------------+-----------------+---------------+-----------------+----------------+--------------------
 9780385737951 | The Maze Runner | James Dashner | Science Fiction |           2010 |                  4
(1 row)


Online Library Management System=#
```

Then as you can see here it updated the quantity_available from the books table since the user has borrowed the book.

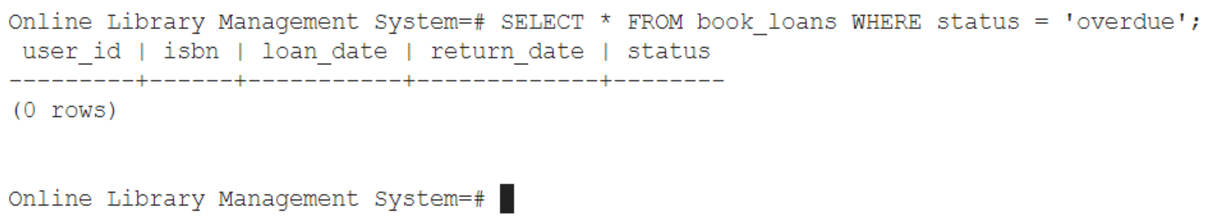
○ d. Find all books borrowed by a specific user.



```
Online Library Management System=# SELECT b.title, b.author, bl.loan_date, bl.status  FROM Book_Loans bl JOIN Books b ON bl.isbn = b.isb
n WHERE bl.user_id = 1;
      title      |    author     | loan_date  |  status
-----------------+---------------+------------+----------
 The Maze Runner | James Dashner | 2024-12-10 | borrowed
(1 row)


Online Library Management System=#
```

So in this query I selected the value from two tables namely the Books and the Book_Loans, then I used the JOIN query to join values from the Books and Book_Loans table to show me the books borrowed by the specific user.

○ e. List all overdue loans.



```
Online Library Management System=# SELECT * FROM book_loans WHERE status = 'overdue';
 user_id | isbn | loan_date | return_date | status
---------+------+-----------+-------------+--------
(0 rows)


Online Library Management System=#
```

In here, I just made a SELECT * query that just gets all the book_loans with the status ‘overdue’.

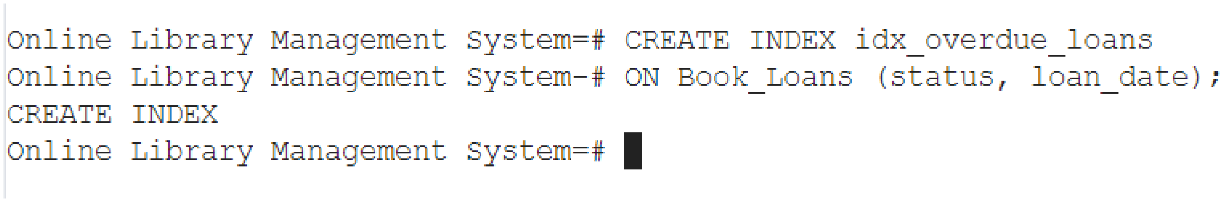
Part 4: Data Integrity and Optimization

4. Explain how you would ensure:

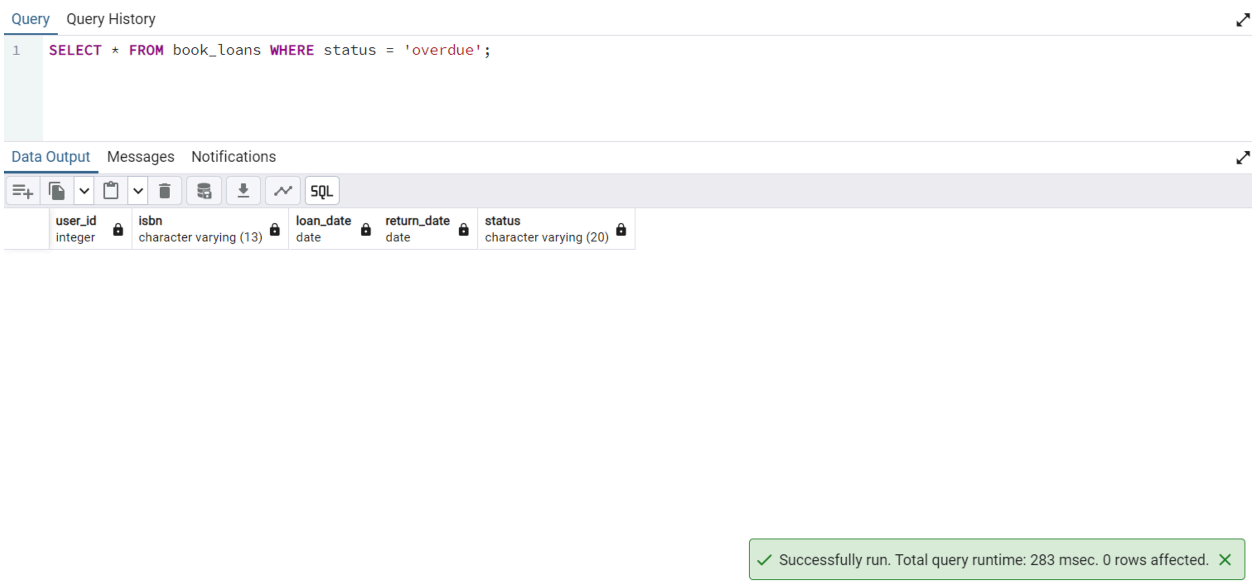
○ The prevention of borrowing books when no copies are available.

- To ensure the prevention of borrowing books when no copies are available I made a check in the available copies that it only works if the available copies are more than or equal zero, this means it checks if the books that are available are not a value lesser than 0. (refer to Diagram 1)
- Then I also created a function that checks the book availability, where it would raise an exception that there are no books available.

○ Fast retrieval of overdue loans.



```
Online Library Management System=# CREATE INDEX idx_overdue_loans
Online Library Management System-# ON Book_Loans (status, loan_date);
CREATE INDEX
Online Library Management System=#
```



```
SELECT * FROM book_loans WHERE status = 'overdue';
```

user_id integer	isbn character varying (13)	loan_date date	return_date date	status character varying (20)

Successfully run. Total query runtime: 283 msec. 0 rows affected.

- In here I created an index of the status and loan_dates so that they are fast to access.
- To retrieve overdue loans you just fetch all the loans and filter it to return only those with the status ‘overdue’. With this query, you wont have to do anymore other methods and just search for the book_loans with the ‘overdue’ status.
- As you can see it completed the request in 283 msec.

Part 5: Reflection

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

- When scaling the database to handle millions of users and books, performance issues and storage scalability become challenges for the online library management system. A solution to the performance issues is implementing database sharding, which distributes data across multiple servers. Then for the problem of storage scalability, the solution is to use a combination of database partitioning and cloud storage solutions.

Assumptions Made:

- I assumed that ISBN of the book is the Primary key for the table, since books have unique ISBN code and no different books have the same ISBN
- I also assumed that the ISBN format used is the 13 characters or the ISBN 13 format
- I also assumed that the return_date in the the book_loans table can be null since the return date can be null if the user has not returned the book yet.