

SE 2141 LABORATORY 4

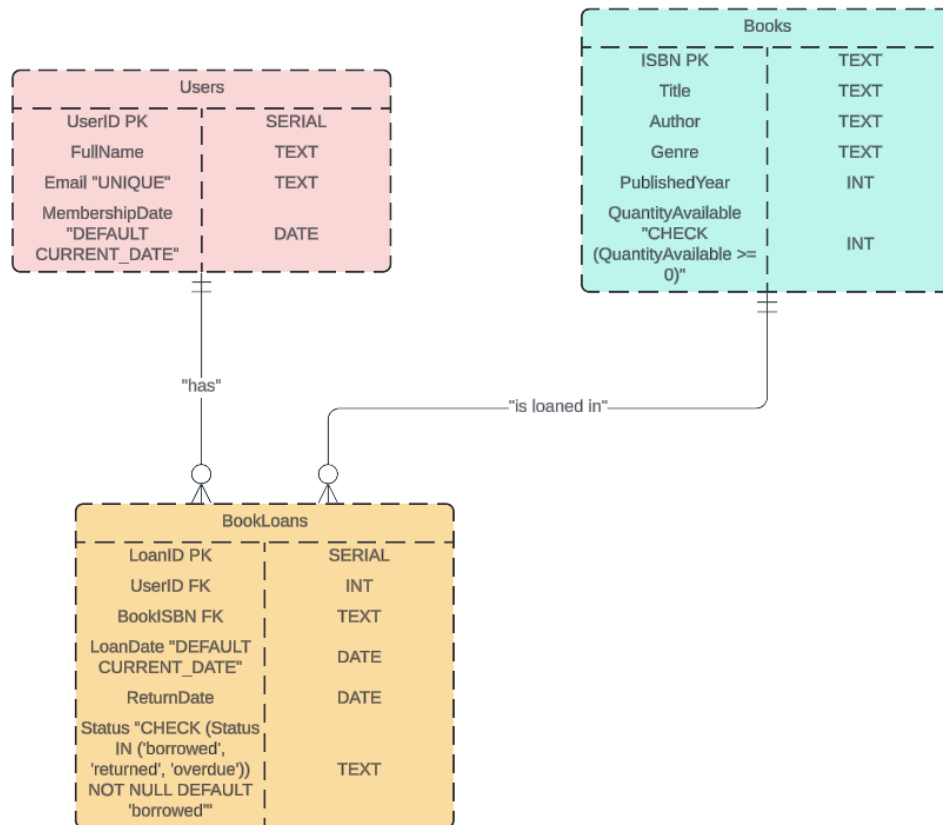
REGINE THERESE N. BARTE

DECEMBER 11, 2024

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).

```
1 CREATE TABLE Books (  
2     ISBN TEXT PRIMARY KEY,  
3     Title TEXT NOT NULL,  
4     Author TEXT NOT NULL,  
5     Genre TEXT,  
6     PublishedYear INT,  
7     QuantityAvailable INT NOT NULL CHECK (QuantityAvailable >= 0)  
8 );  
9  
10 CREATE TABLE Users (  
11     UserID SERIAL PRIMARY KEY,  
12     FullName TEXT NOT NULL,  
13     Email TEXT UNIQUE NOT NULL,  
14     MembershipDate DATE DEFAULT CURRENT_DATE  
15 );  
16  
17 CREATE TABLE BookLoans (  
18     LoanID SERIAL PRIMARY KEY,  
19     UserID INT NOT NULL REFERENCES Users(UserID),  
20     BookISBN TEXT NOT NULL REFERENCES Books(ISBN),  
21     LoanDate DATE DEFAULT CURRENT_DATE,  
22     ReturnDate DATE,  
23     Status TEXT CHECK (Status IN ('borrowed', 'returned', 'overdue')) NOT NULL DEFAULT 'borrowed'  
24 );  
25
```

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.

```
4
5  INSERT INTO Books (ISBN, Title, Author, Genre, PublishedYear,
6  QuantityAvailable)
7  VALUES ('1112346782104', 'Geronimo Stilton Lost Treasure of the Emerald
  Eye', 'Geronimo Stilton', 'Fiction', 1887, 5);
```

- b. Add a new user to the system.

```
4
5  INSERT INTO Users (FullName, Email, MembershipDate)
6  VALUES ('Regine Therese Barte', 'Chokiniko@gmail.com', CURRENT_DATE);
7
```

- c. Record a book loan for a user.

```
1
2
3  INSERT INTO BookLoans (UserID, BookISBN, LoanDate, ReturnDate, Status)
4  VALUES (1, '1112346782104', CURRENT_DATE, '2024-12-10', 'borrowed');
5
```

Results Chart Export ▾



Source

Primary Database ▾

role postgres ▾

Success. No rows returned

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

SE 2141 LAB 4 Free / SE2141 LAB 4 (BARTE) Connect Enable branching

```
1 SELECT Books.Title, Books.Author, BookLoans.LoanDate
2 FROM Books
3 JOIN BookLoans ON Books.ISBN = BookLoans.BookISBN
4 WHERE BookLoans.UserID = 1;
5
```

Results Chart Export

title	author	loandate
"Geronimo Stilton Lost Treasure of the Emerald Eye"	"Geronimo Stilton"	"2024-12-11"

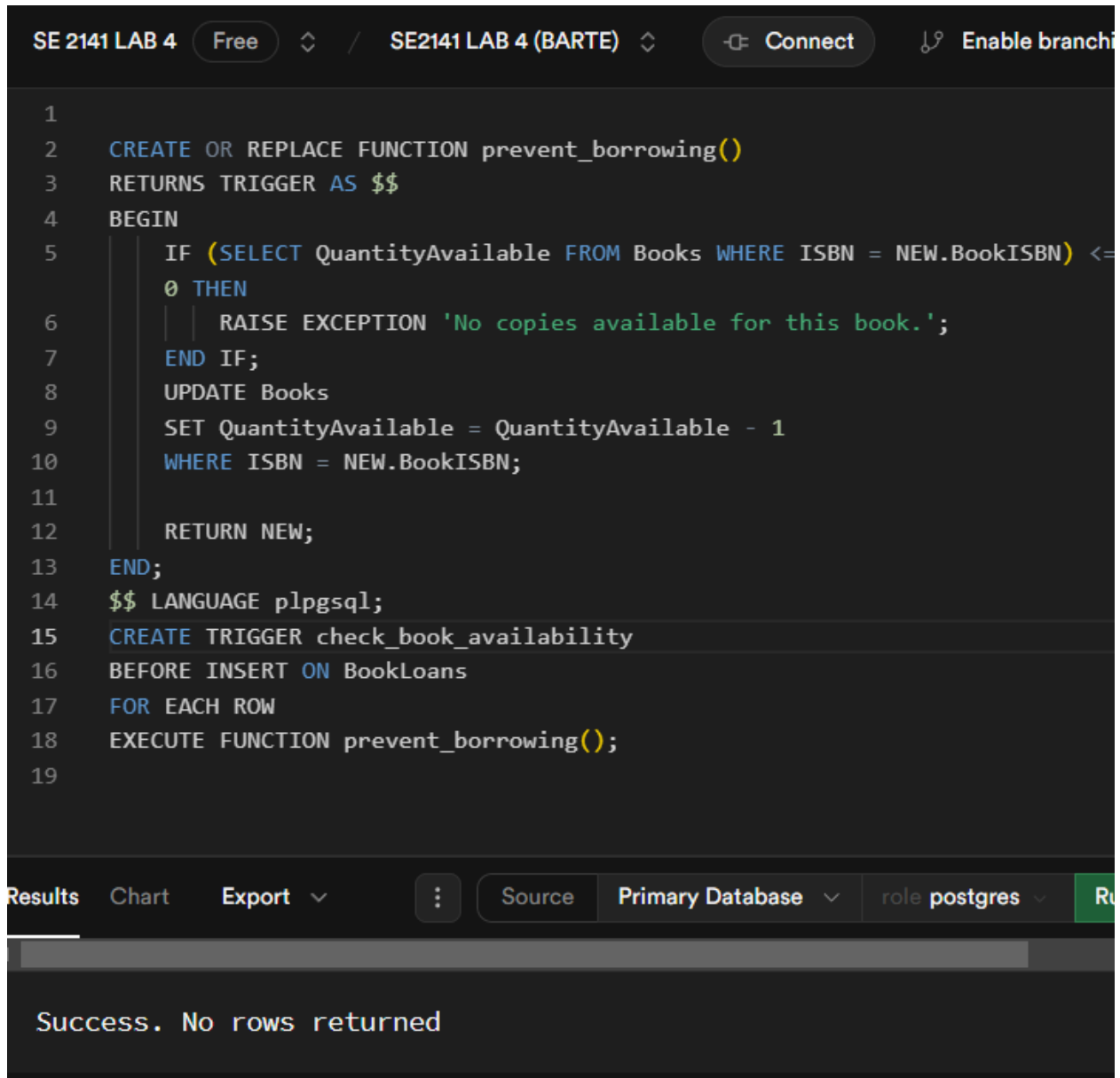
- e. List all overdue loans.

```
1
2
3 SELECT Users.FullName, Books.Title, BookLoans.LoanDate, BookLoans.ReturnDate
4 FROM BookLoans
5 JOIN Users ON BookLoans.UserID = Users.UserID
6 JOIN Books ON BookLoans.BookISBN = Books.ISBN
7 WHERE BookLoans.Status = 'overdue';
8
```

Results Chart Export

Success. No rows returned

Part 4: Data Integrity and Optimization



The screenshot shows a SQL IDE interface with a dark theme. At the top, there are tabs for 'SE 2141 LAB 4' (labeled 'Free') and 'SE2141 LAB 4 (BARTE)'. To the right of these tabs are buttons for 'Connect' and 'Enable branch'. The main area displays SQL code for creating a trigger function and a trigger. The code is as follows:

```
1
2 CREATE OR REPLACE FUNCTION prevent_borrowing()
3 RETURNS TRIGGER AS $$
4 BEGIN
5     IF (SELECT QuantityAvailable FROM Books WHERE ISBN = NEW.BookISBN) <=
6         0 THEN
7         RAISE EXCEPTION 'No copies available for this book.';
8     END IF;
9     UPDATE Books
10    SET QuantityAvailable = QuantityAvailable - 1
11    WHERE ISBN = NEW.BookISBN;
12    RETURN NEW;
13 END;
14 $$ LANGUAGE plpgsql;
15 CREATE TRIGGER check_book_availability
16 BEFORE INSERT ON BookLoans
17 FOR EACH ROW
18 EXECUTE FUNCTION prevent_borrowing();
19
```

Below the code editor, there is a toolbar with buttons for 'Results', 'Chart', 'Export', and a menu icon. To the right of these buttons are dropdown menus for 'Source', 'Primary Database', and 'role postgres'. The 'Results' panel at the bottom shows the message: 'Success. No rows returned'.

- First, it makes a new trigger function `prevent_borrowing()` that will be executed before inserting a new loan into the book loans table. It will send out a trigger then if there are any copies of the books available to borrow by querying the books table for the quantity available.
- if the available quantity is less than or equal to zero then it raises an error that says "No copies available for this book"
- it would also check the availability and would loop to check if there are still books available to prevent loans if the book is not available anymore.

```

1
2 CREATE INDEX idx_bookloans_status ON BookLoans(Status);
3
4 SELECT Users.FullName, Books.Title, BookLoans.LoanDate, BookLoans.
   ReturnDate
5 FROM BookLoans
6 JOIN Users ON BookLoans.UserID = Users.UserID
7 JOIN Books ON BookLoans.BookISBN = Books.ISBN
8 WHERE BookLoans.Status = 'overdue';
9

```

- This creates an index on status column where indexes improve query performance for filtering or searching by specific columns.
- Then selecting from the required column joining the book borrowed and the user, using the ISBN to get the title of the book that the user wants to borrow.
- Filtering then is where book loans are being separated to be classified specifically the overdue ones.

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.

- I think the difficulty when handling millions of users would be the capacity of the database to carry out each information and give it more time to process each time it needs to be accessed. The consistency of data may also occur as loading of information through time may change and be updated so as the operations require it to process fast and accurately..