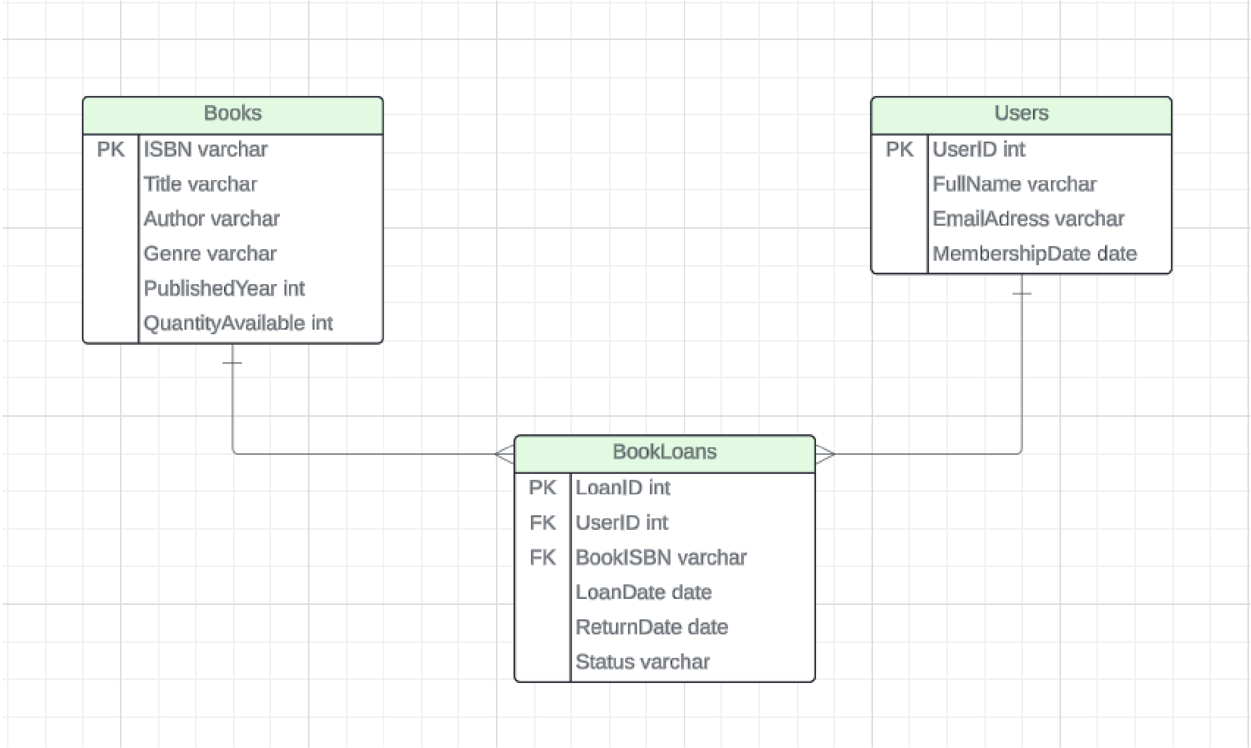
SE 2141 - LAB 4

Cassandra Ysabel Gallo
BSSE -2

Part 1: Conceptual Design

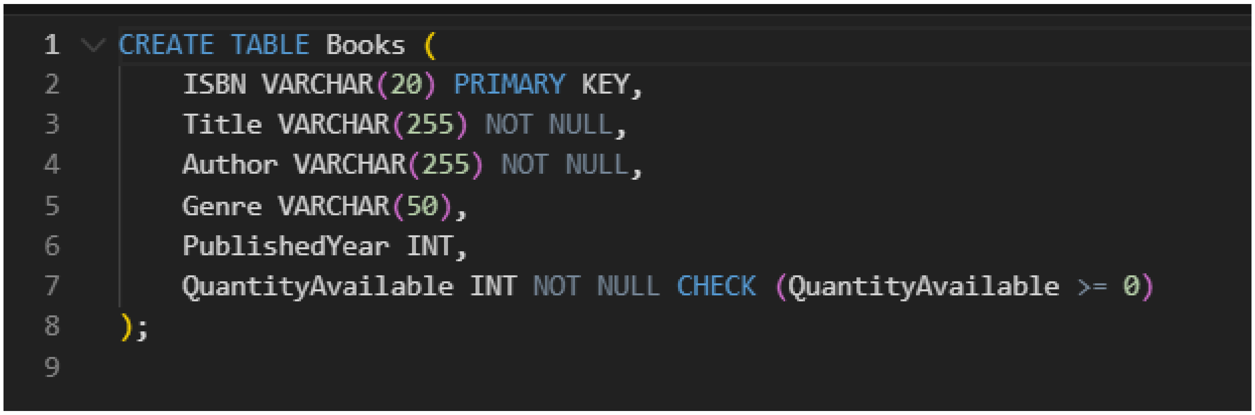


Books	
PK	ISBN varchar
	Title varchar
	Author varchar
	Genre varchar
	PublishedYear int
	QuantityAvailable int

Users	
PK	UserID int
	FullName varchar
	EmailAdress varchar
	MembershipDate date

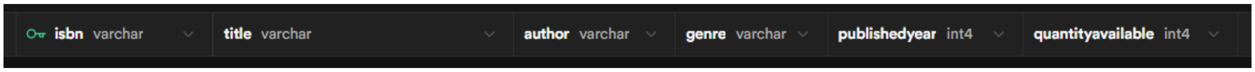
BookLoans	
PK	LoanID int
FK	UserID int
FK	BookISBN varchar
	LoanDate date
	ReturnDate date
	Status varchar

Part 2: Logical Design

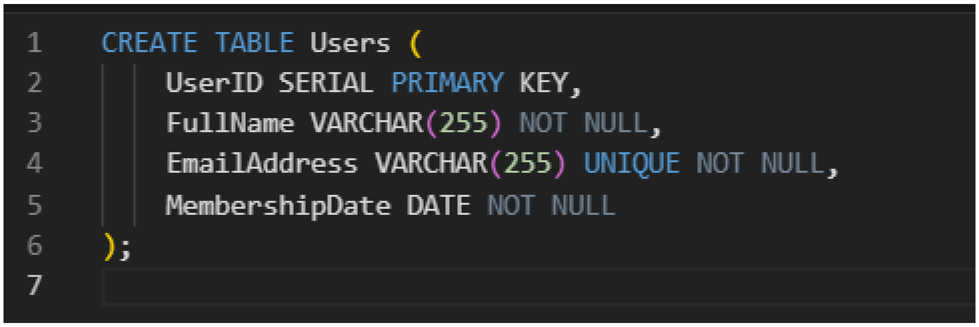


```
CREATE TABLE Books (
    ISBN VARCHAR(20) PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255) NOT NULL,
    Genre VARCHAR(50),
    PublishedYear INT,
    QuantityAvailable INT NOT NULL CHECK (QuantityAvailable >= 0)
);

```

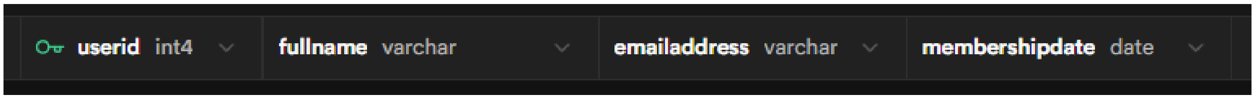


isbn varchar	title varchar	author varchar	genre varchar	publishedyear int4	quantityavailable int4

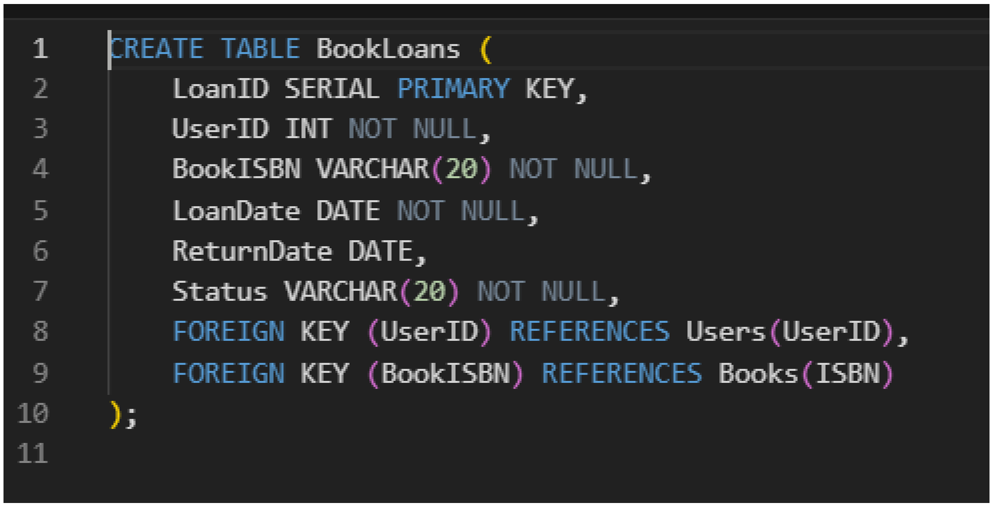


```
CREATE TABLE Users (
    UserID SERIAL PRIMARY KEY,
    FullName VARCHAR(255) NOT NULL,
    EmailAddress VARCHAR(255) UNIQUE NOT NULL,
    MembershipDate DATE NOT NULL
);

```

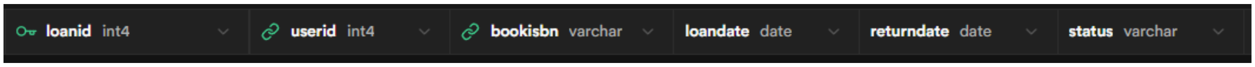


userid int4	fullname varchar	emailaddress varchar	membershipdate date



```
CREATE TABLE BookLoans (
    LoanID SERIAL PRIMARY KEY,
    UserID INT NOT NULL,
    BookISBN VARCHAR(20) NOT NULL,
    LoanDate DATE NOT NULL,
    ReturnDate DATE,
    Status VARCHAR(20) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (BookISBN) REFERENCES Books(ISBN)
);

```

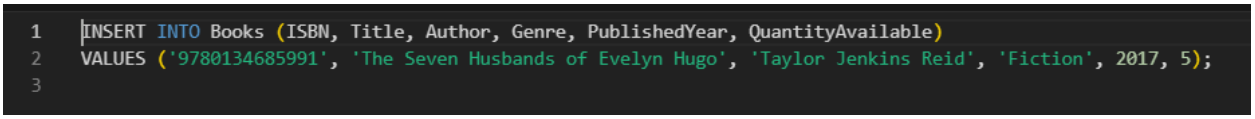


loanid int4	userid int4	bookisbn varchar	loandate date	returndate date	status varchar

Part 3: SQL Queries

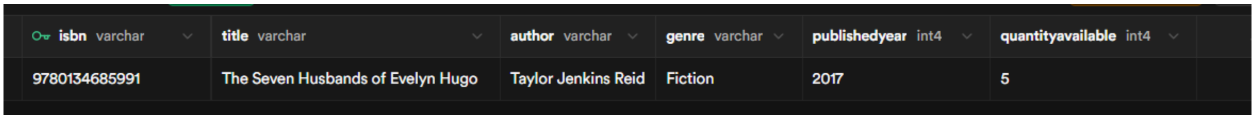
3. Write SQL queries for the following scenarios

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



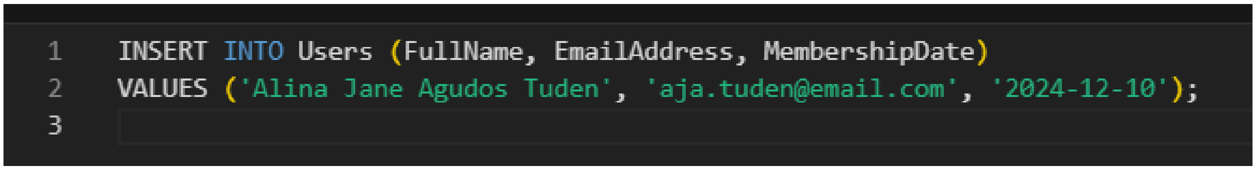
```
INSERT INTO Books (ISBN, Title, Author, Genre, PublishedYear, QuantityAvailable)
VALUES ('9780134685991', 'The Seven Husbands of Evelyn Hugo', 'Taylor Jenkins Reid', 'Fiction', 2017, 5);

```



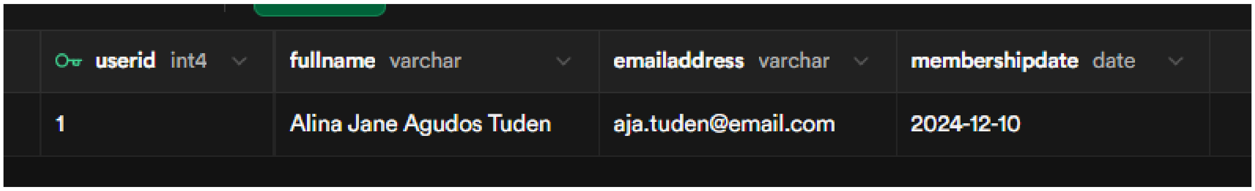
isbn varchar	title varchar	author varchar	genre varchar	publishedyear int4	quantityavailable int4
9780134685991	The Seven Husbands of Evelyn Hugo	Taylor Jenkins Reid	Fiction	2017	5

b. **Add a new user to the system.**



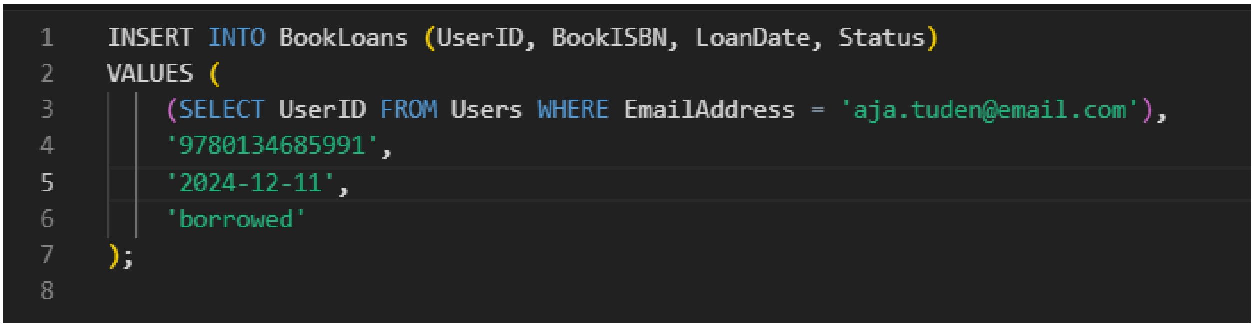
```sql
INSERT INTO Users (FullName, EmailAddress, MembershipDate)
VALUES ('Alina Jane Agudos Tuden', 'aja.tuden@email.com', '2024-12-10');

```



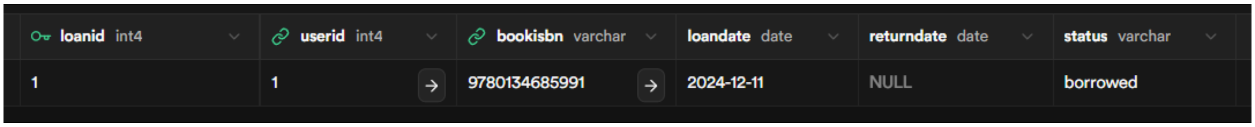
userid int4	fullname varchar	emailaddress varchar	membershipdate date
1	Alina Jane Agudos Tuden	aja.tuden@email.com	2024-12-10

c. Record a book loan for a user.



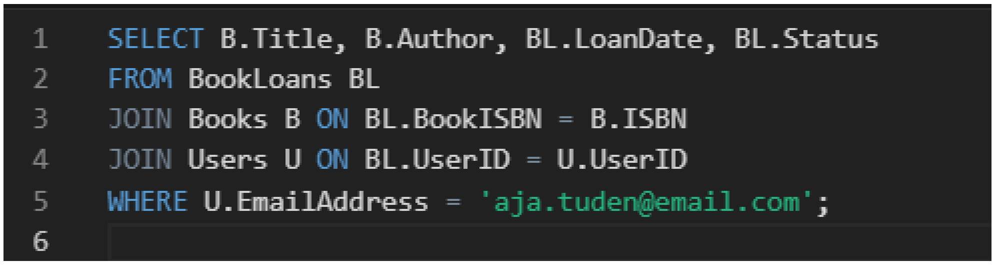
```sql
INSERT INTO BookLoans (UserID, BookISBN, LoanDate, Status)
VALUES (
    (SELECT UserID FROM Users WHERE EmailAddress = 'aja.tuden@email.com'),
    '9780134685991',
    '2024-12-11',
    'borrowed'
);

```



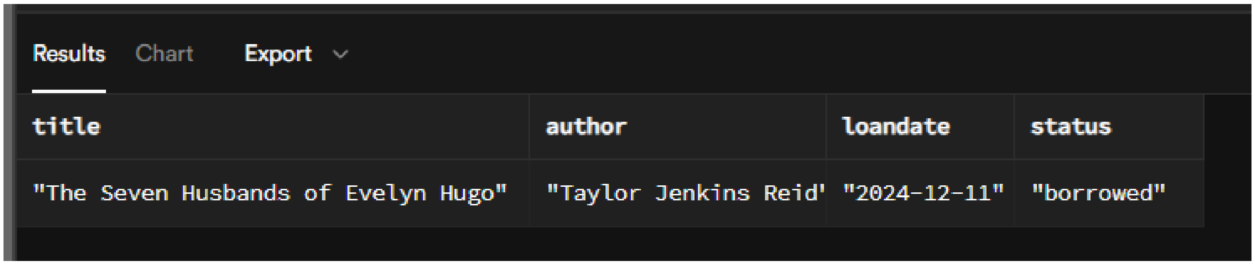
loanid int4	userid int4	bookisbn varchar	loandate date	returndate date	status varchar
1	1	9780134685991	2024-12-11	NULL	borrowed

d. Find all books borrowed by a specific user.



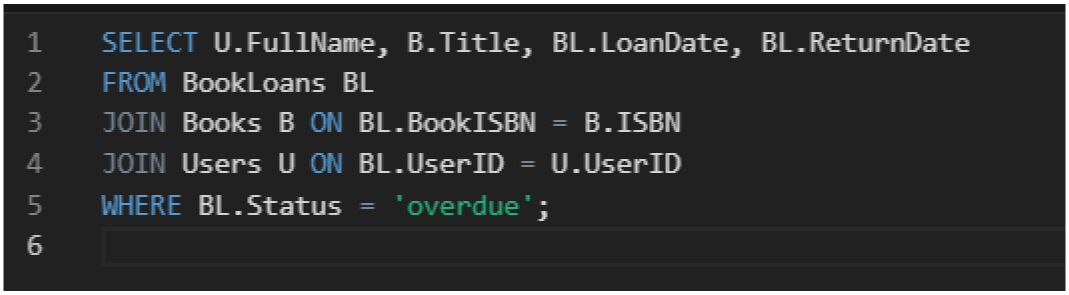
```sql
SELECT B.Title, B.Author, BL.LoanDate, BL.Status
FROM BookLoans BL
JOIN Books B ON BL.BookISBN = B.ISBN
JOIN Users U ON BL.UserID = U.UserID
WHERE U.EmailAddress = 'aja.tuden@email.com';

```



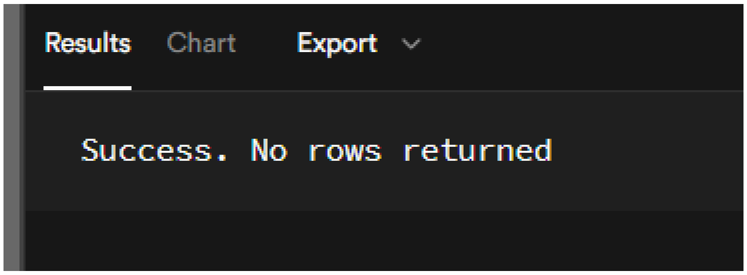
title	author	loandate	status
"The Seven Husbands of Evelyn Hugo"	"Taylor Jenkins Reid'	"2024-12-11"	"borrowed"

e. List all overdue loans.



```sql
SELECT U.FullName, B.Title, BL.LoanDate, BL.ReturnDate
FROM BookLoans BL
JOIN Books B ON BL.BookISBN = B.ISBN
JOIN Users U ON BL.UserID = U.UserID
WHERE BL.Status = 'overdue';

```



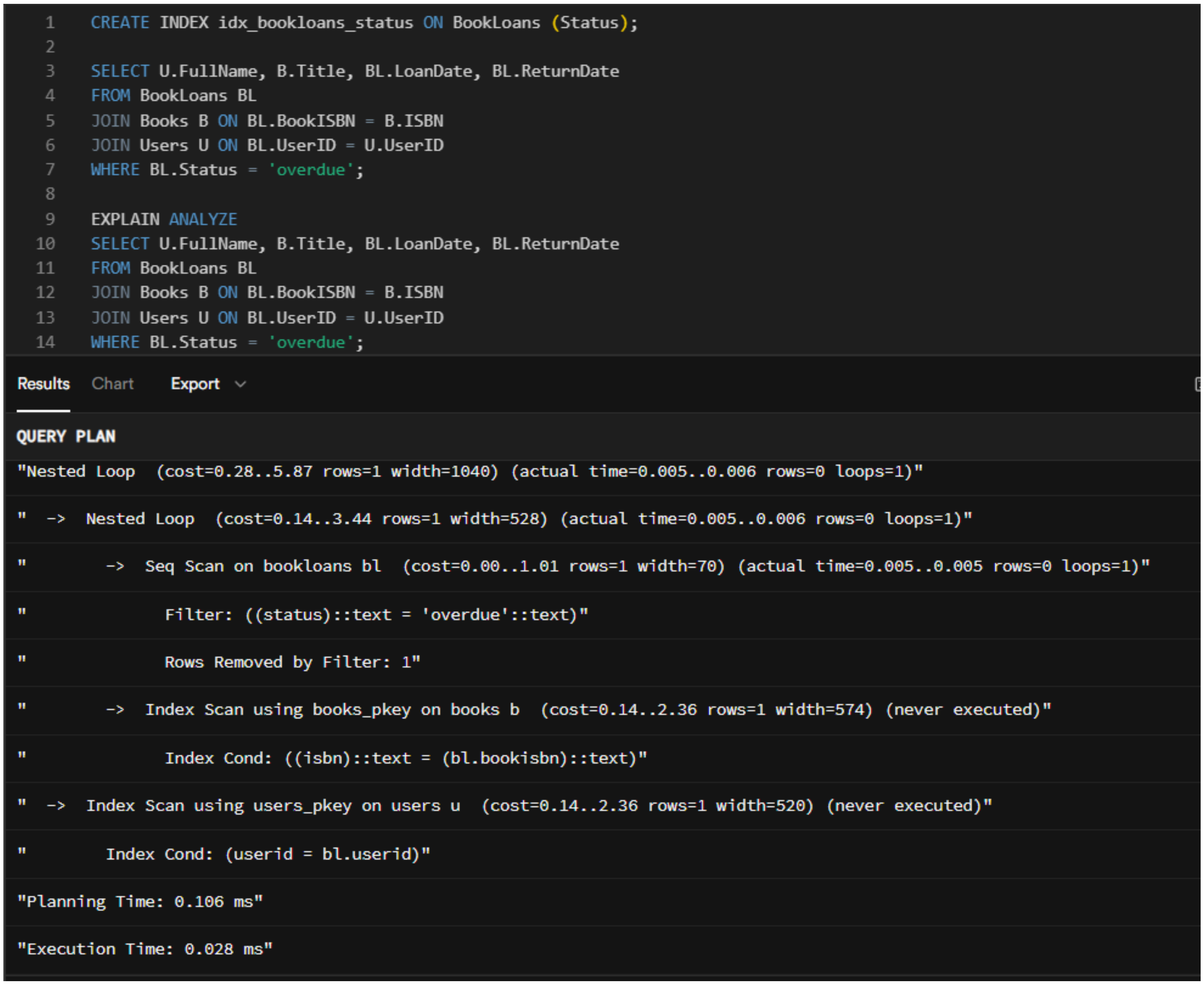
```
Success. No rows returned
```

Part 4: Data Integrity and Optimization

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

A trigger on the BookLoans table checks the availability of books in the Books table before processing a loan. If no copies are available, the trigger prevents the loan from being recorded by raising an exception. When a loan is successful, it reduces the QuantityAvailable by one to keep the inventory updated. An index on the ISBN column optimizes the process by ensuring quick availability checks. Additionally, the application can alert users about a book's availability before they submit a loan request, helping to maintain data integrity and improve the overall user experience.

○ **Fast retrieval of overdue loans.**



```sql
CREATE INDEX idx_bookloans_status ON BookLoans (Status);

SELECT U.FullName, B.Title, BL.LoanDate, BL.ReturnDate
FROM BookLoans BL
JOIN Books B ON BL.BookISBN = B.ISBN
JOIN Users U ON BL.UserID = U.UserID
WHERE BL.Status = 'overdue';

EXPLAIN ANALYZE
SELECT U.FullName, B.Title, BL.LoanDate, BL.ReturnDate
FROM BookLoans BL
JOIN Books B ON BL.BookISBN = B.ISBN
JOIN Users U ON BL.UserID = U.UserID
WHERE BL.Status = 'overdue';
```

```
QUERY PLAN
"Nested Loop  (cost=0.28..5.87 rows=1 width=1040) (actual time=0.005..0.006 rows=0 loops=1)"
"  ->  Nested Loop  (cost=0.14..3.44 rows=1 width=528) (actual time=0.005..0.006 rows=0 loops=1)"
"        ->  Seq Scan on bookloans bl  (cost=0.00..1.01 rows=1 width=70) (actual time=0.005..0.005 rows=0 loops=1)"
"              Filter: ((status)::text = 'overdue'::text)"
"              Rows Removed by Filter: 1"
"        ->  Index Scan using books_pkey on books b  (cost=0.14..2.36 rows=1 width=574) (never executed)"
"              Index Cond: ((isbn)::text = (bl.bookisbn)::text)"
"  ->  Index Scan using users_pkey on users u  (cost=0.14..2.36 rows=1 width=520) (never executed)"
"        Index Cond: (userid = bl.userid)"
"Planning Time: 0.106 ms"
"Execution Time: 0.028 ms"
```

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.

As the database grows, challenges like high query load can be handled by splitting the data across multiple servers. To keep data accurate when many users borrow or return books simultaneously, we can use locking or optimistic concurrency control. Searching for books can be sped up by adding indexes on commonly searched fields and using full-text search for more detailed queries. For storage limits, we can use cloud-based services like Google Cloud SQL or AWS RDS to optimize the performance of the database. These solutions help improve performance and make sure the system can handle increasing users and data.