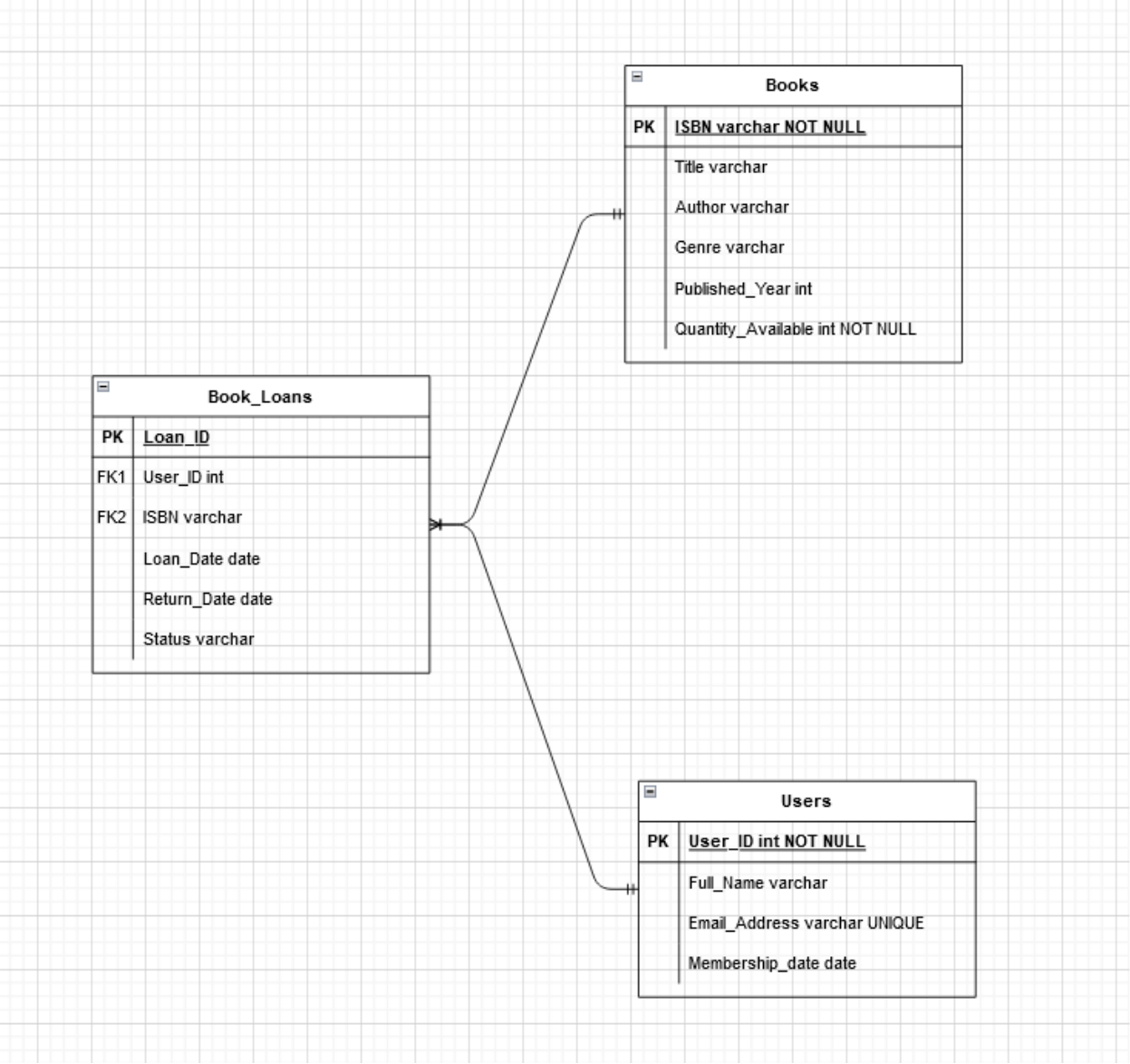
1. The ER Diagram that is required of us



2. Here we will create the tables for our books, users and book loans

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

CREATE TABLE Users (
    User_ID SERIAL PRIMARY KEY,
    Full_Name VARCHAR(255) NOT NULL,
    Email_Address VARCHAR(255) UNIQUE NOT NULL,
    Membership_Date DATE NOT NULL
);


CREATE TABLE Book_Loans (
    Loan_ID SERIAL PRIMARY KEY,
    User_ID INT NOT NULL,
    ISBN VARCHAR(20) NOT NULL,
    Loan_Date DATE NOT NULL,
    Return_Date DATE,
    Status VARCHAR(20) NOT NULL CHECK (Status IN ('borrowed', 'returned', 'overdue'))
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
    FOREIGN KEY (ISBN) REFERENCES Books(ISBN)
);
```

3. a,b,c) Here are the SQL Queries to insert users into the users table, books into the books table, and loan tables

```sql
INSERT INTO Users (Full_Name, Email_Address, Membership_Date)
VALUES ('Ryn Gabriel', 'deer@deer.com', '2024-1-01')
```

```sql
INSERT INTO Books (ISBN, Title, Author, Genre, Published_Year, Quantity_Available)
VALUES ('1234567890', 'Girls Frontline', 'Yuzhong', 'Fiction', 2016, 5);
```

```sql
INSERT INTO Book_Loans (User_ID, ISBN, Loan_Date, Status)
VALUES (1, '1234567890', '2024-12-11', 'borrowed');
```

D) Here we can see the books borrowed by the user

Query | Query History

```
1  SELECT B.Title, B.Author, BL.Loan_Date, BL.Status
2  FROM Books B
3  JOIN Book_Loans BL ON B.ISBN = BL.ISBN
4  WHERE BL.User_ID = 1;
```

Data Output | Messages | Notifications

	title character varying (255)	author character varying (255)	loan_date date	status character varying (20)
1	Girls Frontline	Yuzhong	2024-12-11	borrowed

E) And here we can see if the user has any overdue books

Query | Query History

```
1  SELECT U.Full_Name, B.Title, BL.Loan_Date, BL.Return_Date
2  FROM Book_Loans BL
3  JOIN Users U ON BL.User_ID = U.User_ID
4  JOIN Books B ON BL.ISBN = B.ISBN
5  WHERE BL.Status = 'borrowed'
6  AND BL.Return_Date < CURRENT_DATE;
7
```

Data Output | Messages | Notifications

	full_name character varying (255)	title character varying (255)	loan_date date	return_date date

4. a) We can prevent the borrowing of books when no copies are available by simply checking if there are any copies available to borrow before allowing a loan to recorded into the database and we can do that by making a trigger function is Postgresql.

```
CREATE OR REPLACE FUNCTION prevent_borrowing_when_no_copies()
RETURNS TRIGGER AS $$
DECLARE
    available_quantity INT;
BEGIN

    SELECT Quantity_Available INTO available_quantity
    FROM Books
    WHERE ISBN = NEW.ISBN;

    IF available_quantity <= 0 THEN
        RAISE EXCEPTION 'No copies available for this book.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_book_availability
BEFORE INSERT ON Book_Loans
FOR EACH ROW
EXECUTE FUNCTION prevent_borrowing_when_no_copies();
```

b) We can ensure the fast retrieval of loans by using indexing

```
CREATE INDEX idx_status_return_date ON Book_Loans (Status, Return_Date);
```

Query | Query History

```
1  SELECT U.Full_Name, B.Title, BL.Loan_Date, BL.Return_Date
2  FROM Book_Loans BL
3  JOIN Users U ON BL.User_ID = U.User_ID
4  JOIN Books B ON BL.ISBN = B.ISBN
5  WHERE BL.Status = 'overdue'
6  AND BL.Return_Date < CURRENT_DATE;
7
```

By running this query, we can determine how fast the query can take

Query | Query History

```
1  EXPLAIN ANALYZE
2  SELECT U.Full_Name, B.Title, BL.Loan_Date, BL.Return_Date
3  FROM Book_Loans BL
4  JOIN Users U ON BL.User_ID = U.User_ID
5  JOIN Books B ON BL.ISBN = B.ISBN
6  WHERE BL.Status = 'overdue'
7  AND BL.Return_Date < CURRENT_DATE;
8
```

This is execution time is before indexing

```
11    Execution Time: 0.093 ms
```

And this execution time is after indexing

```
11    Execution Time: 0.060 ms
```

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

I think on the challenges that may manifest when the database is to handle many users is the sheer scale and amount of users and books. The system may slow down due to the amount of data being stored. One solution which is already present in this laboratory is using indexes to speed up search queries and such.

Another challenge I can think of if the database were to handle millions of users and books is the amount of data that the database can store. A solution that may work is to split tables into their own category like “Books” splitting into genres or year published, or “Book Loans” being separated according to the loan duration.