COE 418 Project: Apollo Script

*Project Submission Report*

Mahmoud R. Kebbi, Abed Al Rahman B. Mneimneh, and Jad E. Baradei

**Abstract**—This reports offers a description of the design na d the development process of the Apollo Script desktop application. The application serves as a library management system for a local library. It was developed using Java and a MySQL server database.

**Index Terms**—library, database, entity, relationship, Java, SQL, query.

Adapted from the IEEE Computer Society template

—————————— ◆ ——————————

# contents

[contents 1](#_Toc153643703)

[1 Introduction 1](#_Toc153643704)

[2 Background 1](#_Toc153643705)

[3 Proposal 2](#_Toc153643706)

[6 Helpful Hints 6](#_Toc153643707)

[7 Conclusion 6](#_Toc153643708)

[**References** 7](#_Toc153643709)

# 1 Introduction

T

HIS document is a project report of the Apollo Script project which includes a thorough explanation of the process of the development of the desktop application. The project is made up of a relational database and a graphical user interface. The software developed is simply a management system for a local library which handles data coming in and out of the database and offers client and staff users an easy and efficient interface. The database contains information about entities and relationships including but not limited to different types of users (staff, admin, and client), books, transactions, reservations, studyrooms, and computers. Books will have multiple copies, each identified by a copy number, and will be rented and returned by client users.

# 2 Background

## 2.1 Context

Given that the objective of the database systems project is to developa software that handles data flow coming from and going to the relational database through an interactive grapgical user interface, the functionalities and features of the application were chosen and iomplemented accordingly.

## 2.2 Problem Analysis

A local library usually accommodates 3 types of users: Clients – Staff (librarians) – Admins. Each type of user possesses different privilegs in the system and therefore requires the application to offer a each a different of features and capabilities.

The admin user should be able to:

1. Add, remove, and update staff to the Staff list.
2. Add and remove computers from the computer list.
3. Add and remove rooms from the Studyroom list.
4. Modify the database and/or the application.
5. Solve any bugs or problems that might be revealed during operation.

Staff users should be able to:

1. Remove users using their username or userIds.
2. Add, remove, and modify books.
3. Know which computers are currently available for use.
4. Reserve studrooms for clients and observe the reservations.
5. Perform transactions (rent, return) on book copies for clients.

Client users should be able to:

1. Log in to their accounts.
2. Sign up for an account.
3. Modify their accounts.
4. Observe the books in the database.
5. Search for books using their titles, author, ISBN, and publisher.

## Information needs

Information needs of this project include details of (attributes of each) books, transactions, users, staff, studyrooms, reservations, branches, and computers.

# Proposal

## Conceptaul Data Model

### Relational Data Model

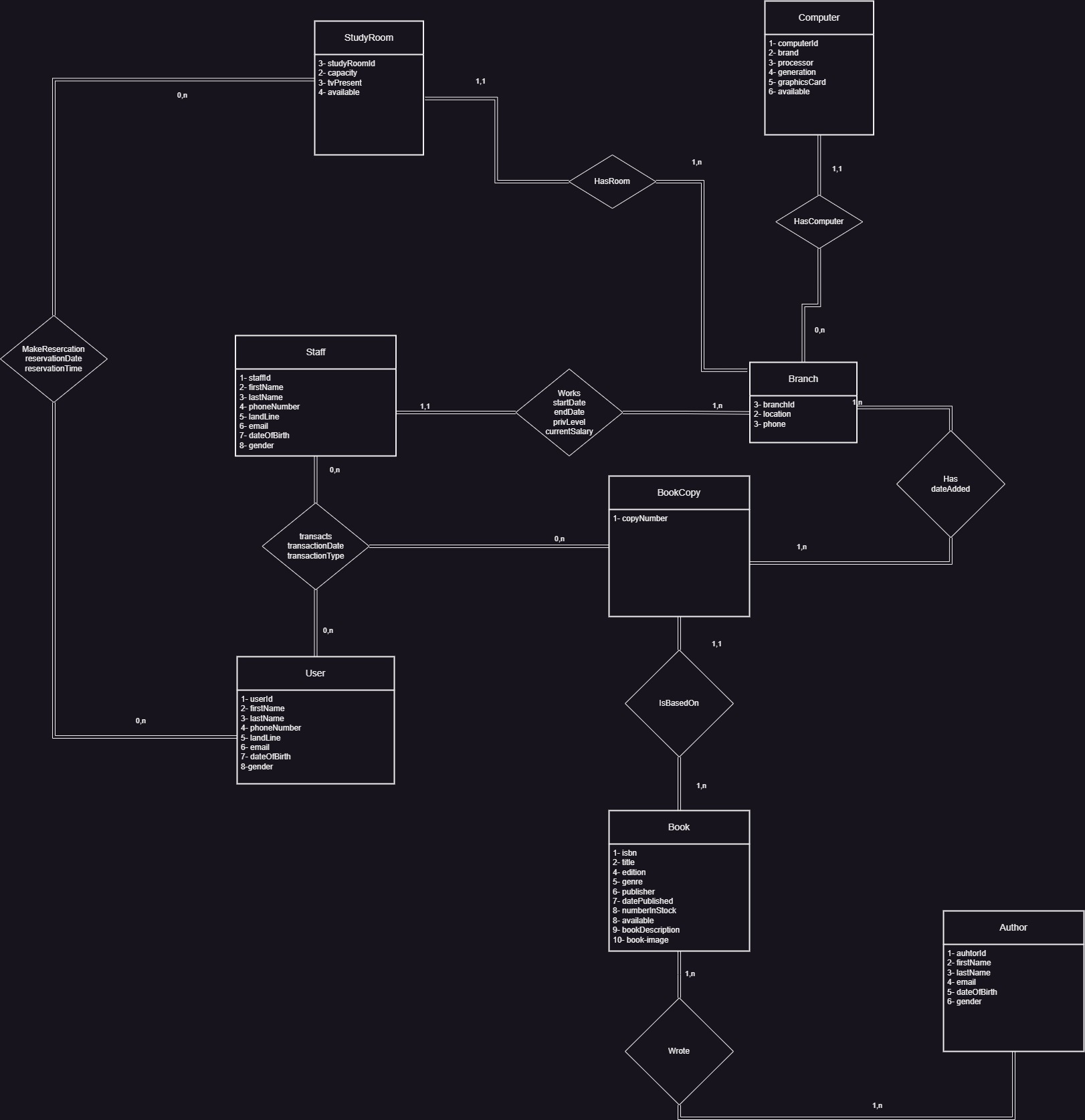
The following conceptual data model describes the relationships between the main entities in the database.

Figure Realtional Data Model of the library database

Description and Notes:

* There are no rent and return tables because the transaction contains the attribute transaction type which can only have two values (‘Rent’, ‘Return’).
* The attribute “available” in Computer is just instantaneous Boolean value which will be used to display available or unavailable.
* Users and Staff do not include “username” and “password” attributes because it is unrelated to the schema and will just be used for log in purposes.
* The reservation time is at most 2 hours.
* The Branch entity was later not used in the project.

### Logical Data Model

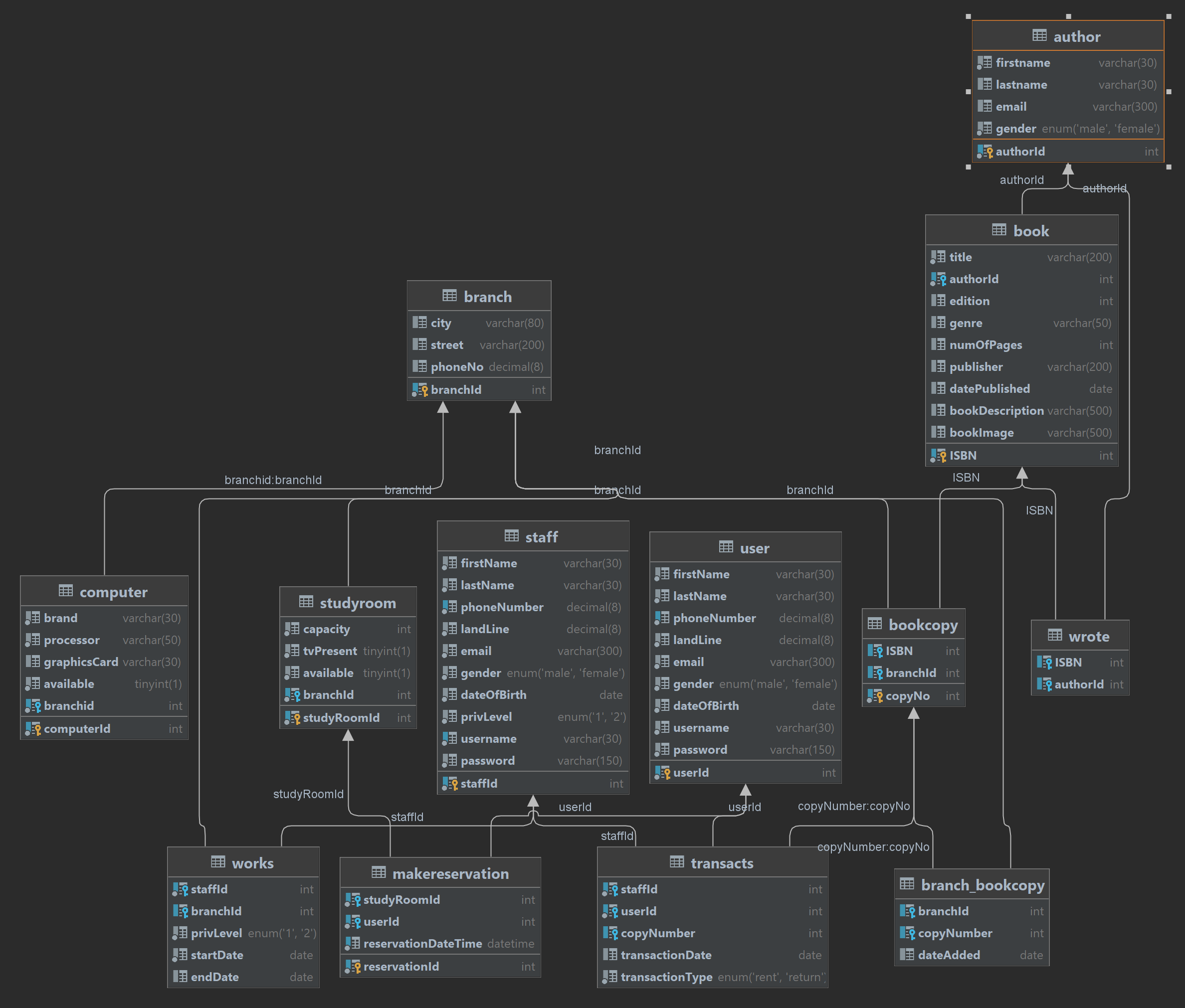
The logical database was obtained by following the rules and procedural steps of RDM to LDM conversion in the COE 418 course. Some attributes including “username” and “password” were added to Staff and User for log in purposes, also a privilege level attribute was added to the Staff entity which has two values ‘1’ and ‘2’ to designate librarians and admins. Each book has multiple copies that is based on it which customers can ask librarians to perform a rent request for them if there is an available copy. No purchase operation is allowed, customers can only come in the library to read books, or they can rent and return books within a time limit of 2 weeks. After that they will be notified to return the book through email.

Figure Logical Data Model of the library Database

# Software Implementation

## Languages:

### Java

Java was used to develop desktop applications.

### SQL

SQL was used to define and manipulate the project’s relational Database.

## Frame Works:

### Spring Boot:

A screenshot of a computer

Description automatically generatedSpring Framework is an open-source Java framework that facilitates building and launching Java applications and services. Spring Boot comes with a default embedded web server, namely Apache Tomcat, which enables developers to develop production-grade applications. It also provides starter dependencies to simplify the build configuration. Additionally, it offers production-ready features (dependency injection, metrics, health checks). To use the Spring Boot framework the Sprin Initializr was used through the following link [Spring Initializr](https://start.spring.io/) and the Spring project was built according to this configuration:

Figure Spring Inatializr configuration.

The Java Mail Sender API was added to the dependencies of the project. This framework was used in specifically to enable sending emails in the java application, in addition to its dependency injection capabilities.

### JavaFX:

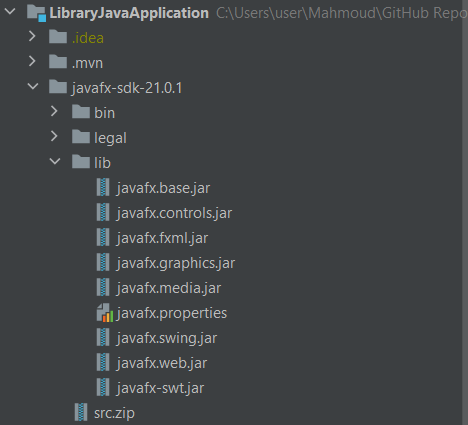
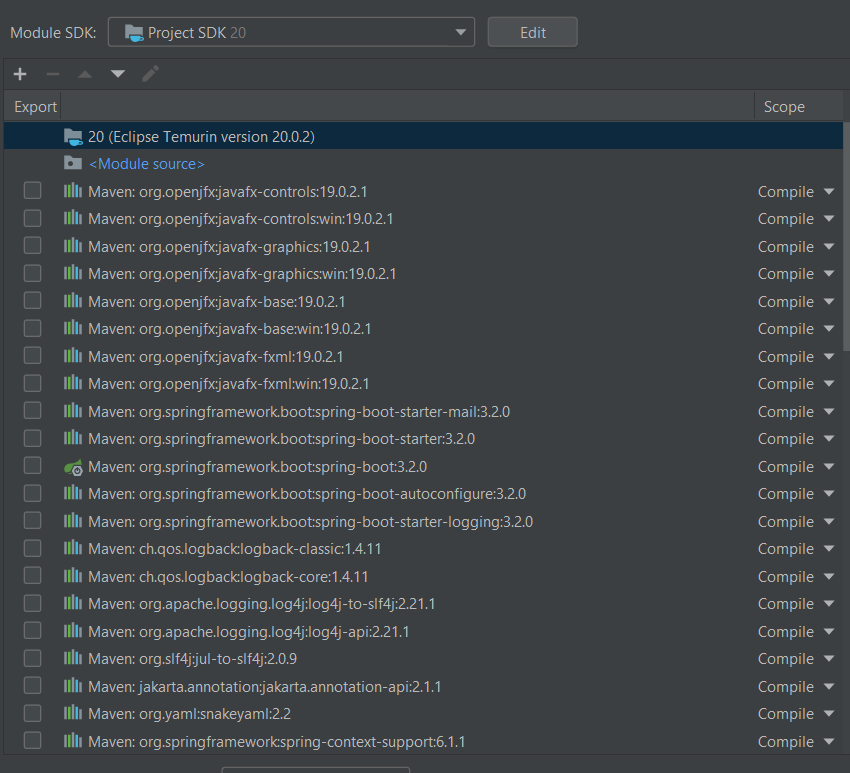
JavaFX is a Java framework that allows developers to create rich client applications through APIs for building GUI applications. JavaFX framework is also highly compatible with the Spring Boot framework. To use JavaFX in the Spring project the JavaFX SDK folder was added to the project folder (not necessary), then dependencies on the JavaFX libraries were added to the project object model xml file. Next, the JavaFX libraries were added to the project modules.

Figure JavaFX SDK file and project dependencies.

## Tools

### GitHub and Git:

The collaboration on the project was done via GitHub through an online repositor, and Git was used for version control. Using GitHub and Git facilitated the development process by allowing all team members to track changes in the project, and allowing each member to work on separate branches to implement features one by one, The collaboration was done the following repository [AbedAlRahmanMneimneh/Library-Management-System (github.com)](https://github.com/AbedAlRahmanMneimneh/Library-Management-System).

### MySQL:

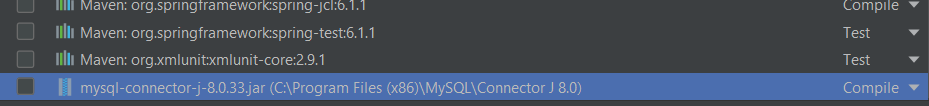
To connect the MySQL Server to the Java project the J Connector JAR was added to the project module.

Figure 5 MySQL-Connector dependency.

The MySQL database features were used to facilitate data retrieval, increase security, and maintain data integrity and consistency. The features utilized include access control features, views, triggers, and indexing.

#### Accsess Control

SQL database management systems allow for multiple users to access a database, each with a different set of privileges and grants. In the Library App schema there are three types of users:

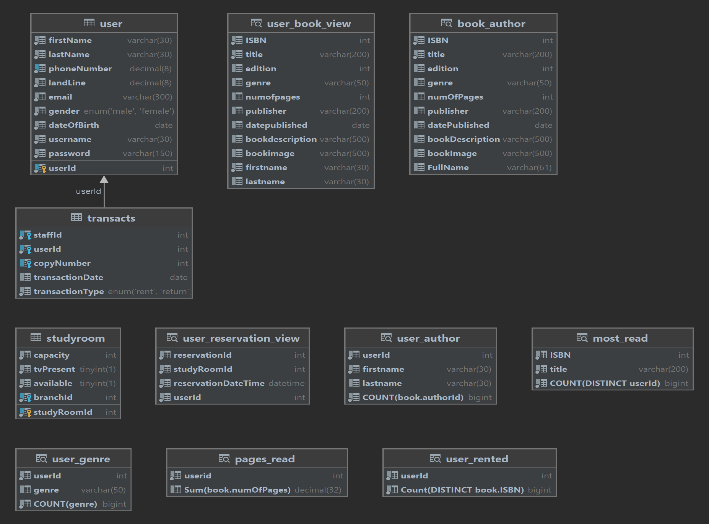
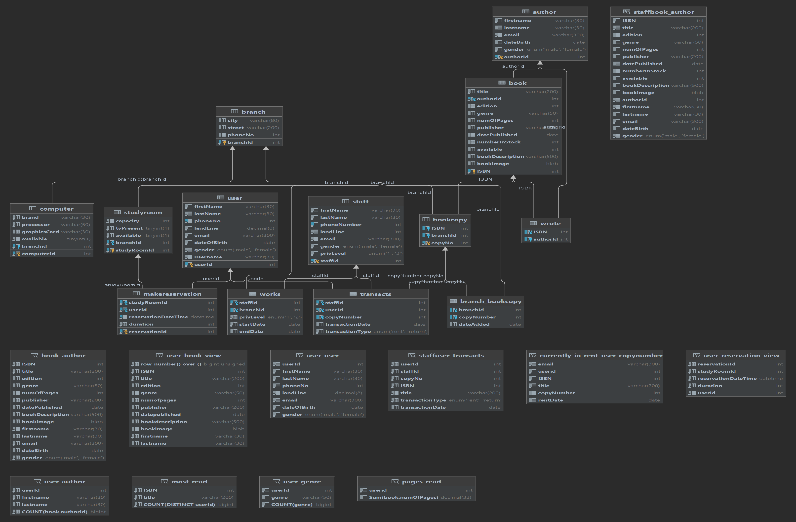
1. Client: A client user can select, insert, delete, and update the user and book tables in addition to selecting multiple other views.
2. Staff: A staff user can select, insert, delete, and update any table in the library app schema.
3. Admin: An admin user (usually the person maintaining the application) is granted all privileges on the entire library app schema.

Figure Client user schema and Staff user schema

#### Indexing

Indexing is a feature in SQL databases that allows for high-speed data retrieval from the database. A developer will not be affected by indexes when writing queries, however there will be less overhead on the SQL server when fetching data. Indexes were mostly made on primary keys, forgein keys, and unique keys.

#### Triggers

Triggers are event handlers that get activated by Data Modification Language (DML) events like INSERT, UPDATE, DELETE, and other Data Definition Language (DDL) events like ALTER and CREATE. Triggers were used in the SQL database where checks were not sufficient to enforce the rules set in the library application. Two triggers were used to check that a date added is not greater than than the current date.

Example:

create definer = root@localhost trigger before\_insert\_check\_DoP  
 before insert  
 on book  
 for each row  
BEGIN  
 IF NEW.datePublished >= *CURDATE*() THEN  
 SIGNAL SQLSTATE '45000'  
 SET MESSAGE\_TEXT = 'Added date must be less than the current date';  
 END IF;  
END;

#### Views

In the development of this project a a decision was made to utilize SQL views to minimize the complexity of SQL queries written in the Java application. This approach was adopted to enhance the readability and maintainability of the code while also achieving better security and performance.

Example of code written without using views in the database:

sqlSelect = " select `libappschem`.`user`.`userId` AS `userId`,"  
+"`libappschem`.`book`.`genre` AS `genre`,"  
 +"*count*(`libappschem`.`book`.`genre`) AS `COUNT(genre)`"  
 +"from (((`libappschem`.`user` join `libappschem`.`transacts`) join `libappschem`.`book`) join `libappschem`.`bookcopy`)"  
+" where ((`libappschem`.`transacts`.`userId` = `libappschem`.`user`.`userId`) and"  
 +" (`libappschem`.`bookcopy`.`copyNo` = `libappschem`.`transacts`.`copyNumber`) and"  
 + " (`libappschem`.`bookcopy`.`ISBN` = `libappschem`.`book`.`ISBN`)) and (userId =" +*client*.clientId+")"  
 +"group by `libappschem`.`user`.`userId`, `libappschem`.`book`.`genre`"  
 +"order by count(`libappschem`.`book`.`genre`) desc;";

Example of code written while using views in the database:

sqlSelect = "Select \* from pages\_read where userId="+*client*.clientId;

# Experimental evaluation

This section will serve as an evaluation for querying bith as expert and non-expert users. A non-expert user usually uses a graphical user iinterface that is easy to use and does not require any knowledge of the database. On the other hand, an expert user can write SQL DML and DDL statements to get and manipulate the data they want in the database. Therefore, an expert user is bound to know all the details of the database. This principle is evaluated for the three types of users.

## Client:

Searching for a book:

Non-expert: A non-expert user can write the any attribute related to the vbook they want, and they will get a result as a card view of the book along with details about it.

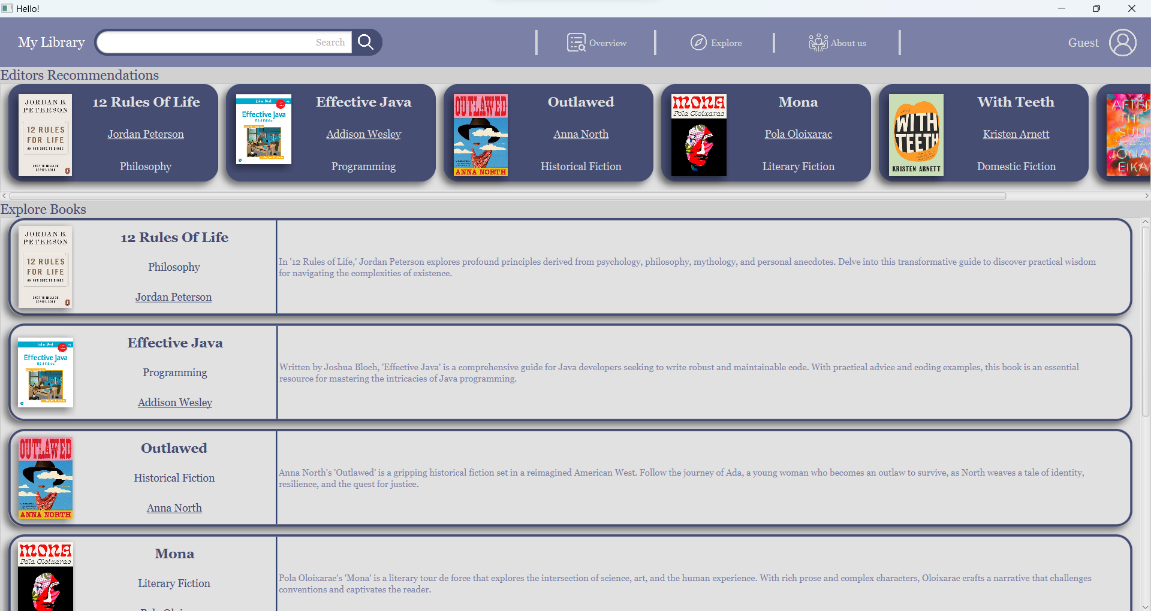


Figure serach bar for user

Expert: An expert can write the following query to get the result from MySQL:

"Select \* from book\_author where ISBN = '" + searchStatement +"' UNION "+"Select \* from book\_author where FullName like '" + searchStatement +"%' UNION " +"Select \* from book\_author where genre like '%" + searchStatement +"%' UNION "+"Select \* from book\_author where publisher like '%" + searchStatement +"%'";

Sign in:

# Conclusion

**References**