DATABASE SETUP:

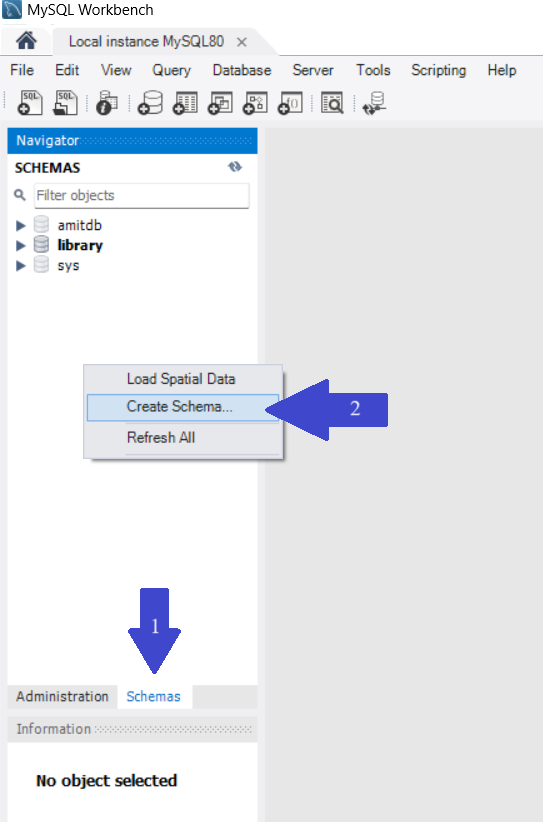
*MySQL Installation:*

If you haven't installed MySQL, please download and install it from the official website:

<https://dev.mysql.com/downloads/installer/>

During installation, **please use this tutorial and make note of the MySQL root user credentials (username: root and password: 172901120103 (please use this password as this is the password inserted into my java classes))**:  
<https://www.youtube.com/watch?v=OM4aZJW_Ojs&t=321s>

After installation, open MySQL Workbench from search bar and click on *Local instance MySQL80*



Click on “Schemas” and then right-click and choose “Create Schema”; After that, a new page will be opened and it will require you to add a name for the new schema: this name should be “**library”** then click on “Apply”, and “Apply” again. After that, right-click on the “library” schema and choose “Set as Default Schema”;

**Open SQL Files:**

Navigate to File > Open SQL Script in MySQL Workbench.

Open each .sql file (from folder database) one by one, in this **exact order**:

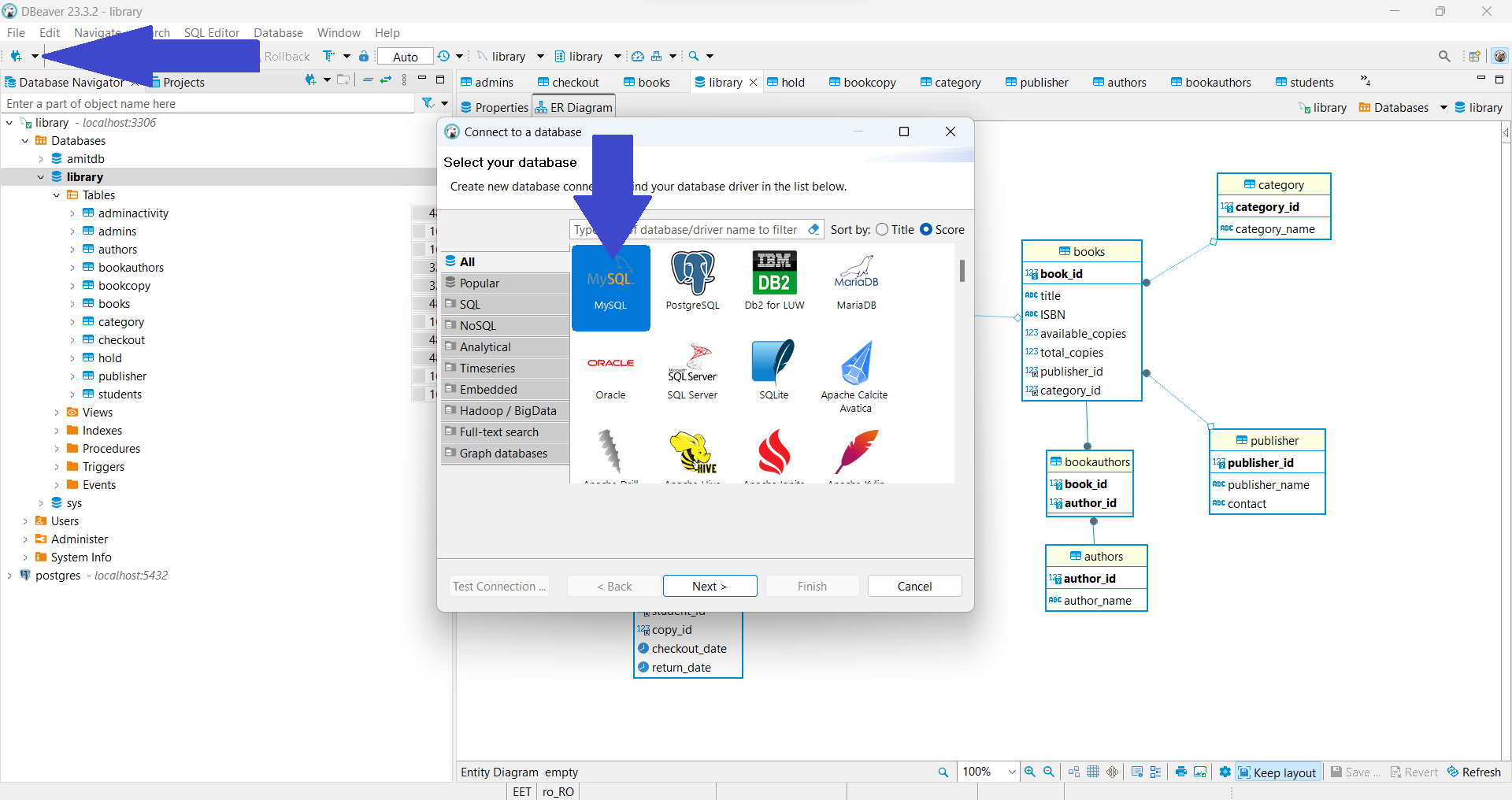
*category. publisher. authors. bookauthors. books. admins. adminactivity. bookcopy. students. hold. checkout.*

I suggest you execute each SQL script by navigating to Query > Execute (All or Selection) right after you have opened it (so open *category*, execute, open *publisher*, execute an so on);

*Database Tool (DBeaver) Installation:*

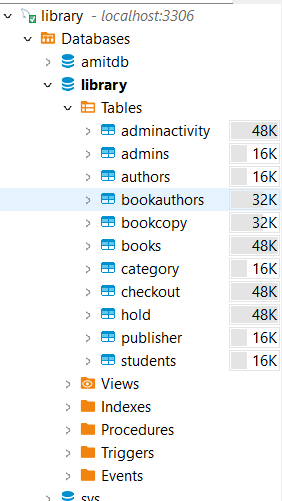
Download and install DBeaver, a universal database tool, from DBeaver Downloads of from Microsoft Store.

Database Initialization:

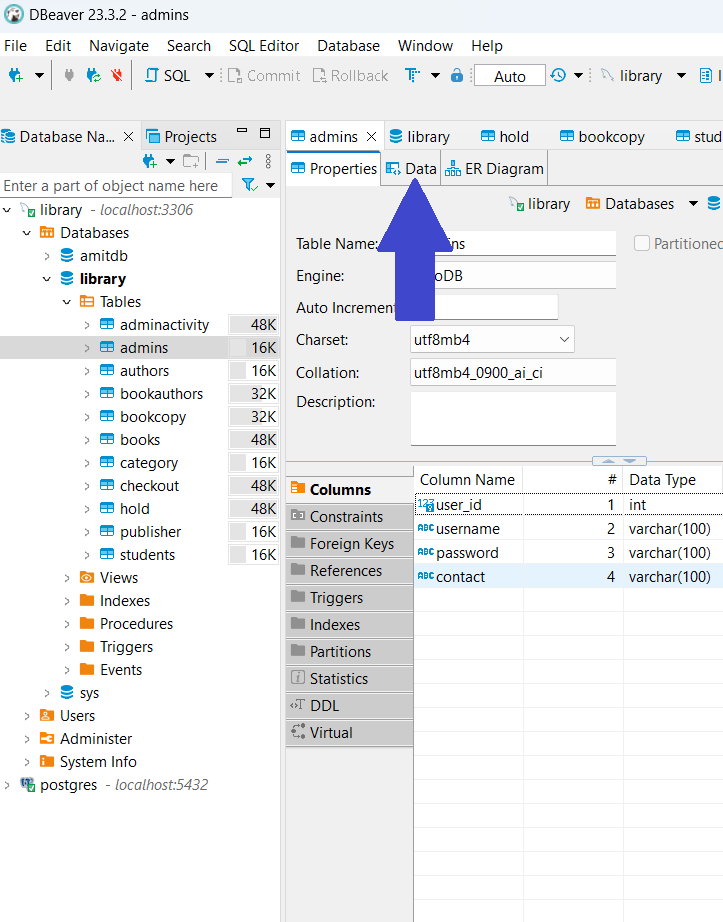


Open DBeaver and configure a new MySQL connection using the MySQL root credentials obtained during installation.

Once DBeaver is connected to MySQL, you will be able to see all schemas from the Workbench; Click on “library” > Tables and from here, you can view the data from each table by double-clicking on it



After clicking on a table, click on “Data” in order to be able to view the data from the database;

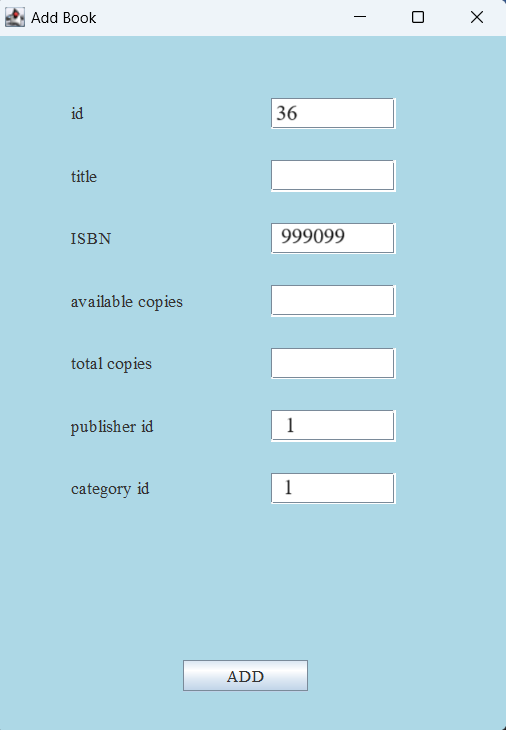


|  |  |
| --- | --- |
| Project title: | Library Management System |
| Name: | Danci Patricia Ioana |
| Group: | 30424 |

1. **TASK DESCRIPTION**

Embark on a journey to develop a sophisticated Library Management System in Java, combining efficiency, security, and user-friendliness. Firstly, we have a login page where we have to input username and password; the username and passwords available can be viewed in the database table “admin”; each username has a corresponding password and the login will only succeed if both are correct. For a demo, I suggest using username **“john\_doe”** with password **“0000”**. Once we have logged in, a dashboard panel appears with different functions; **I suggest using the SHOWDATA button first thing, as the tables provided by this function will be useful in order to insert data when required *(take into account that as we refresh our tables in the database, we also have to reopen the ones from the GUI – SHOW DATA*);**

1. *ADD BOOK BUTTON:*

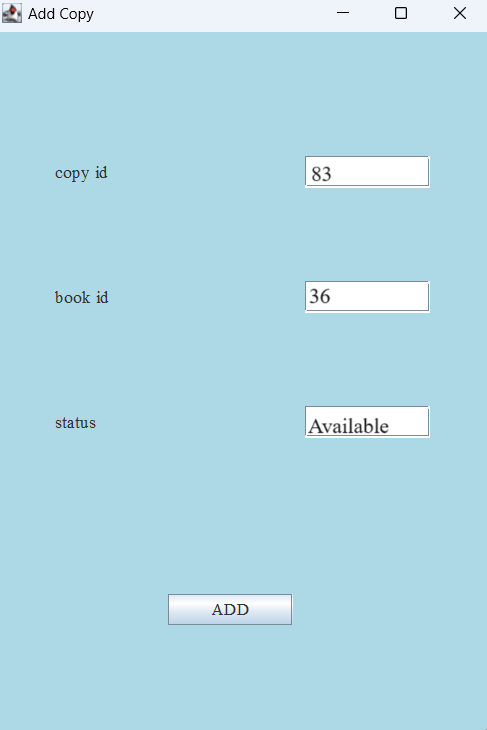


Upon successful login, users are greeted with an intuitive dashboard housing six powerful functionalities:

If we click on the “ADD BOOK” button, a new panel opens with several text fields for input; by this function, we can add a new book to our library with specific details: book\_id (for a demo, I suggest adding “36” here as we already have 35 books in the library), title, ISBN (for a demo, add “999099”), available\_copies (I suggest “1”), total\_copies (I suggest “1”), publisher\_id (here, our “books” table is linked to table “publisher”, so in order to successfully add a book, we have to insert an available publisher\_id; for a demo, I suggest “1”), category\_id (works the same as publisher\_id, here I suggest adding “1” as well);

Once we added the book, click on table “books” > data and then on refresh.

1. *ADD COPY BUTTON:*



After we have added book with book\_id “36”, we also have to add the copy into the “bookcopy” table; since we already have 82 copies, copy\_id should be “83” for our demo;   
you can add the data I have presented in the photo, **or, you can also add a new copy of a different book**;

Suppose a new copy of the book “The Shining”, which has book\_id “10”, has arrived and we have to add it into the database: copy\_id will be “83” (in case an error appers, try copy\_id “100” or any other number >100), book\_id will be “10” and status will be “Available”; in this case, we can refresh table “bookcopy” and our new copy will appear, and we can also refresh table “books” and we will see that the number of available copies, as well as the number of total copies for “The Shining” will be incremented;

1. *CHECKOUT BUTTON:*

When we press the “CHECKOUT” button, a new panel will be opened which requires us to enter the Title of the book we want to checkout: for a demo, I suggest “The Shining”; Once we have entered the title, a new pop up will tell us which copy\_ids of that specific title are available; for “The Shining”, we have copy\_id 75 and 76 available and we can choose which one we want to checkout; checkout\_id should be “26” for our demo, student\_id can be any number in interval (1,30) as we have 30 students; checkout\_date will have format year-month-day, and return\_date represents the date when the book should be returned; for my current data, I have chosen one month as borrowing period for the book; once we have done the checkout, refresh table “checkout” and also table “book\_copy” to see how the status changes from “Available” to “Not available”; if we also refresh table “books”, we can see that now the “available\_copies\_ will be decremented by 1;

1. *RETURN BUTTON:*

If we press button “RETURN”, we have to input the “checkout\_id”; for a demo, I suggest “22”, which is a checkout for the book “The Song of Achilles”; now, if this specific book is on hold by several students, a pop up will tell us the id of the student who first placed this book on hold; for this example, we have 3 students who have this book on hold (this can be viewed in the “hold” table); in the “hold” table, the status represents the order in which the students will get the copy of the book, based on the order they placed it on hold; when we return a book that is on hold, it will automatically be given to that student who has status 1; now, we can refresh table “hold” and see that the student who had status 1 will be deleted, and all the other statuses will be decremented for the other students who are waiting for that copy as well;

Another example for a book that is not on hold is checkout\_id “1”, which was done for copy\_id “6”; after the return, we can refresh table “bookcopy” and see how the status for copy\_id “6” changes from “Not Available” to “Available”, and also in table “books”, the number of available\_copies for book\_id “1” will be incremented by 1.

1. *HOLD BUTTON:*

If we press the “HOLD” button, a new panel will be opened which will require us to enter: hold\_id (for a demo, I suggest “4”), student\_id (for a demo, I suggest”29”), copy\_id (for a demo, I suggest “78”) and hold\_date; now, after we have successfully done a hold, we can refresh the “hold” table to view our new data, and we can see that the status is automatically inserted, based on how many other students have that specific copy on hold; based on the example I have provided, the status should be “1”, as we have no other hold for copy\_id 78.

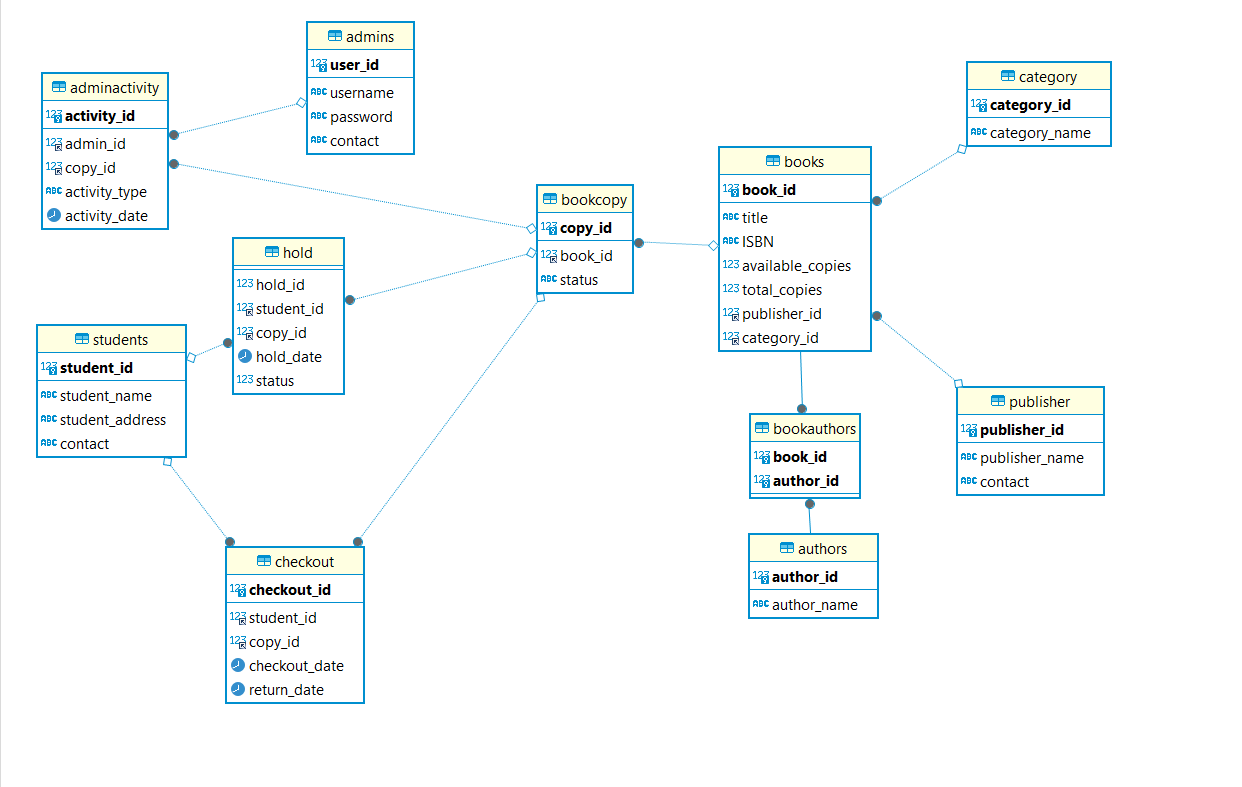
1. *SHOW DATA BUTTON:*

This button will lead us to another page with 3 additional buttons: BOOKS, COPIES, STUDENTS;  
Depending on which one we click on, we can see the full data from that specific table in our database; in case we insert new data into these 3 tables, they will have to be reopened from the GUI.

This Library Management System isn't just about functionality; it's about delivering a secure and elegant solution.

By developing this Library Management System, the goal is to create a robust and user-friendly application that streamlines library operations, enhances resource tracking, and provides valuable insights through reporting functionalities.

*DATABASE DIAGRAM:*

**

1. **CLASS DISCOVERY**

**Main.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborations |
| - Initialize GUI Components  - Handle User Authentication  - Display Dashboard  - Error Handling | - Login: opens login panel  - Dashboard: open dashboard panel |

**Login.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - User Input Retrieval: Retrieves the entered username and password from the text fields for further processing.  - ActionListener Handling: Provides a mechanism to set an external ActionListener for the "Login" button. | - none |

**Dashboard.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - Initialize GUI Components  - Interact with Commands Class  - Handle User Actions  - Display Additional Panels | - Commands (checkAvailableBooks)  - AddBookD  - AddCopyD  - CheckBook  - CheckoutD  - ReturnD  - HoldD  - DataD |

**Commands.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - Database Interaction  - Book Management Operations  - Checkout and Return Operations  - Hold Operations  - Data Retrieval | - JDBC |

**AddBookD.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Input Validation: Collects user input for book details and validates the entered information.  - Integration with Commands Class: Utilizes the Commands class to execute the database operation for adding a new book. | - Commands (addBook) |

**AddCopyD.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Input Validation: Collects user input for copy details and validates the entered information.  - Integration with Commands Class: Utilizes the Commands class to execute the database operation for adding a new copy. | - Commands (addCopy) |

**CheckBook.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Input Retrieval: Retrieves the book title entered by the user.  - Integration with Commands Class: Utilizes the Commands class to execute the database operation for checking available copies. | - Commands (checkAvailableBooks) |

**CheckoutD.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Input Retrieval: Retrieves checkout-related data entered by the user, such as checkout ID, student ID, copy ID, checkout date, and return date.  - Integration with Commands Class: Utilizes the Commands class to execute the database operation for checking out a book. | - Commands (checkoutBook) |

**DataD.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Data Retrieval and Display: Fetches data from the database based on the selected table (bookcopy, students, books) and displays it in a JTable within a new JFrame  - Integration with Commands Class: Utilizes the Commands class to execute SQL queries and retrieve data from the database. | - Commands (executeQuery) |

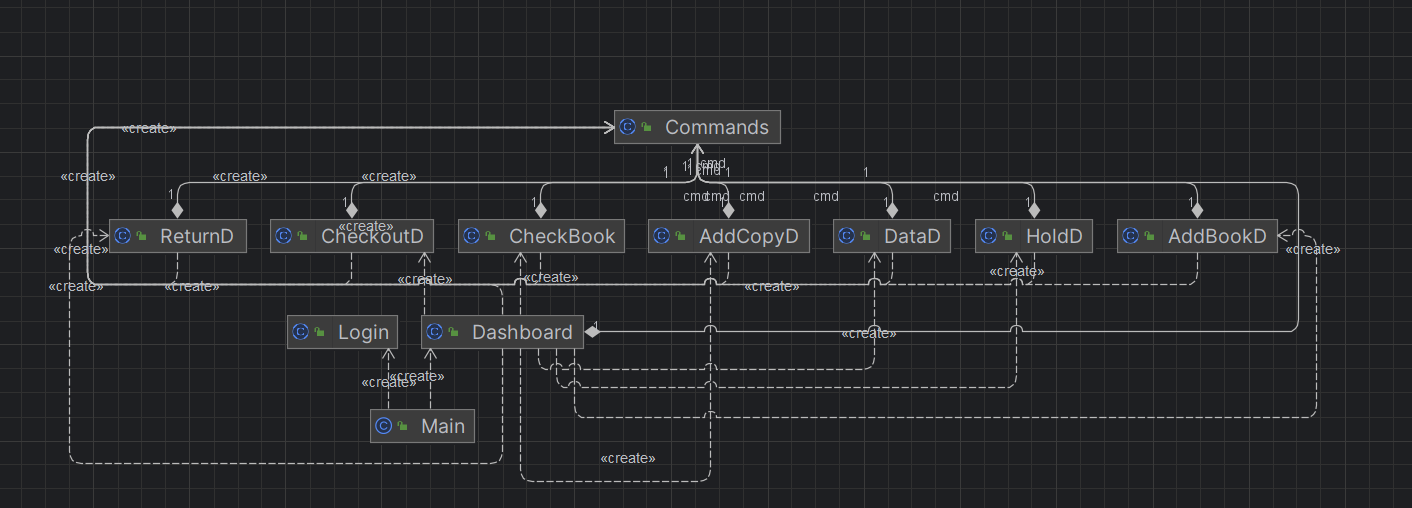
**HoldD.java**

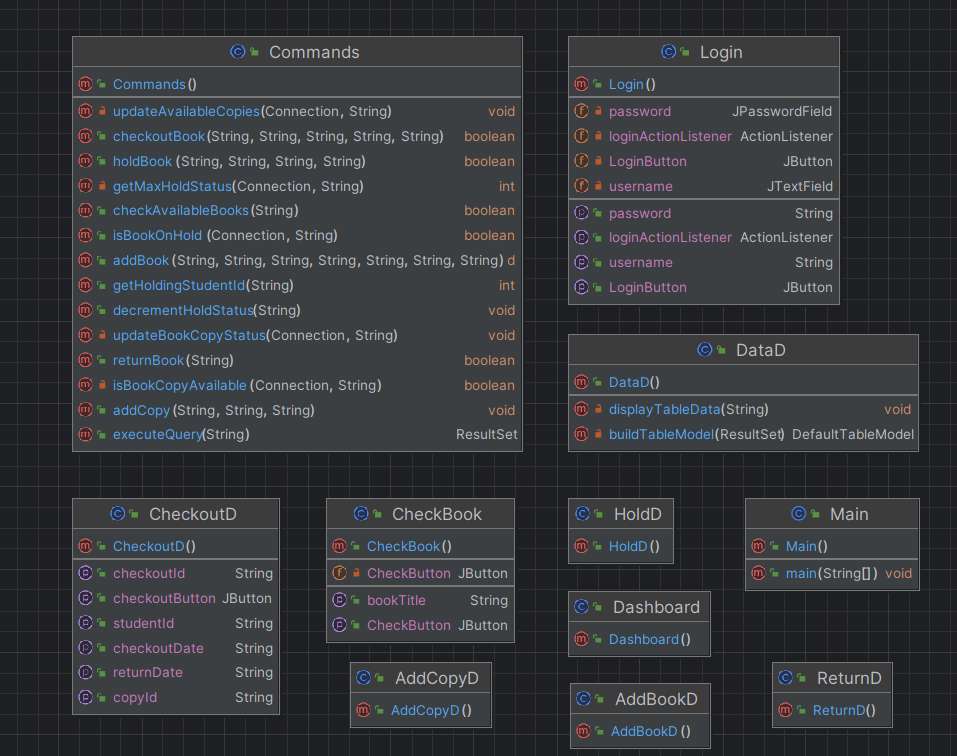
|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Integration with Commands Class: Utilizes the Commands class to execute the necessary logic for placing a book on hold.  - User Input Retrieval: Retrieves user input (holdId, studentId, copyId, holdDate) from text fields for processing. | - Commands (holdBook) |

**ReturnD.java**

|  |  |
| --- | --- |
| Responsabilities | Collaborators |
| - User Interface Components  - Event Handling  - Integration with Commands Class: Utilizes the Commands class to execute the logic for returning a book, checking if the book is on hold, and handling the corresponding actions  - User Input Retrieval: Retrieves user input (checkout\_id) from text fields for processing. | - Commands  (  returnBook  isBookOnHold  getHoldingStudentId  decrementHoldStatus  ) |

1. **CLASS DIAGRAM**





The class diagram for the Library Management System reflects a well-organized and interconnected structure, with a central focus on the "Commands" class. The "Commands" class serves as a pivotal component, encapsulating SQL queries and database interaction functionalities. All other classes within the diagram establish a relationship with the "Commands" class, indicating their dependency on the database operations encapsulated within.

**Key classes in the diagram include:**

*AddBookD, AddCopyD, CheckBook, CheckoutD, DataD, HoldD, ReturnD:*

These classes represent different aspects of the graphical user interface (GUI) for various operations in the library system. Each GUI class relies on the "Commands" class to execute corresponding SQL queries when interacting with the database.

*Dashboard:*

The "Dashboard" class acts as a hub for navigation within the system. It orchestrates the creation and display of different GUI panels and establishes connections with the "Commands" class to execute database operations seamlessly.

*Commands:*

The central "Commands" class encapsulates all SQL queries and database-related functionalities. It serves as the bridge between the GUI classes and the underlying database. This design promotes modularity, encapsulation, and separation of concerns, making it easier to maintain and extend the system.

In conclusion, the Library Management System project stands as a testament to the effective application of OOP concepts, Java programming, and database management. It provides a robust foundation for library administrators to manage resources seamlessly and offers a user-friendly interface for patrons, ultimately contributing to an enhanced library experience.