Bilkent University
CS 353
Design Report

Group 26
School Library Database

Cemhan Kaan Özaltan - 21902695 - Section 1
Hissam Mahmoud Elsayed Faramawy - 21901253 - Section 3
Servet Gülnaroğlu - 21902474 - Section 2
Taha Batur Şenli - 21901857 - Section 3

Table of Contents

1. Revised ER Diagram

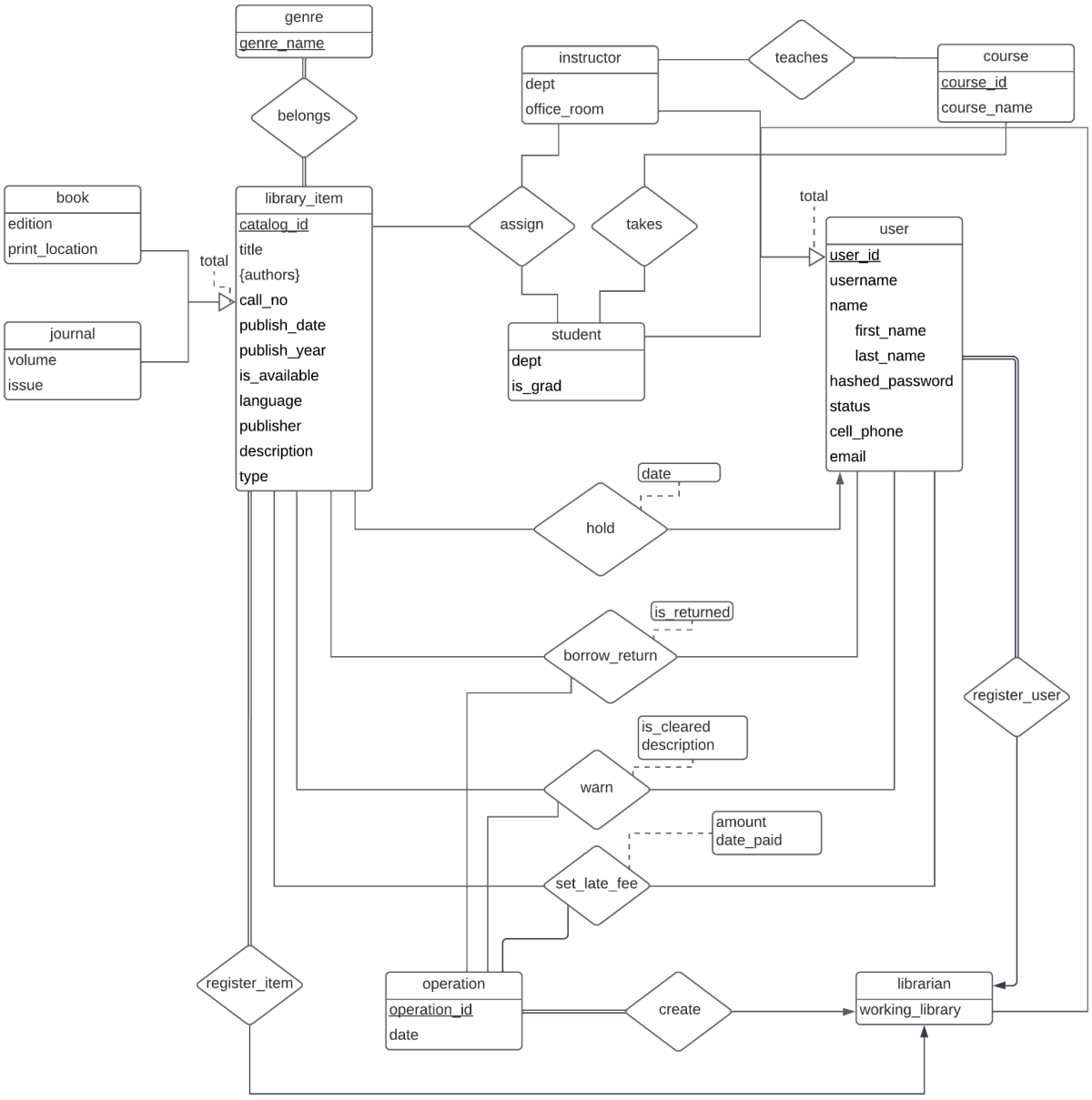


Figure 1: Revised ER Diagram

Note: Librarian actions are done through operations to make actions on the same item at different dates unique. The hold relationship set has a date attribute which will be added to its primary key in the schema to make holds on the same item at different dates unique. Register relationship sets will not be translated into schemas since they only exist to make functionalities explicit in the ER diagram and do not hold useful data.

2. Database Schema

The following relation schemas define our database, the attributes of the relations, their domains and referential integrity information. We have also verified that the relations are at least BCNF, which automatically makes them a part of 3NF.

2.1. library_item

Relational Schema
library_item(catalog_id: int, title: varchar(20), call_no: int, publish_date: date, publish_year: int, is_available: boolean, language: varchar(20), type: varchar(10), publisher: varchar(20), description: varchar(50))

Candidate Keys
{(catalog_id)}

Functional Dependencies
catalog_id → title call_no publish_date publish_year is_available language type publisher description

Normal Form
BCNF

Creation

```
CREATE TABLE library_item(
    catalog_id char(10),
    title varchar(20),
    call_no int,
    publish_date date,
    publish_year int,
    is_available boolean,
    language varchar(20),
    type varchar(10),
    publisher varchar(20),
    description varchar(50),
    PRIMARY KEY (catalog_id)
);
```

2.2. authors

Relational Schema
authors(catalog_id: int, author: varchar(20))
catalog_id: Foreign key to library_item

Candidate Keys

{(catalog_id, author)}

Functional Dependencies
None

Normal Form
BCNF

Creation

```
CREATE TABLE authors(
    catalog_id int,
    author varchar(20),
    PRIMARY KEY (catalog_id, author),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT
);
```

Note: Authors is a multivalued attribute, therefore translated to a schema.

2.3. genre

Relational Schema
genre(genre_name: varchar(20))

Candidate Keys
{(genre_name)}

Functional Dependencies
None

Normal Form
BCNF

Creation

```
CREATE TABLE genre(
    genre_name varchar(20),
    PRIMARY KEY (genre_name)
);
```

2.4. belongs

Relational Schema
belongs(catalog_id: int, genre_name: varchar(20))
catalog_id: Foreign key to library_item
genre_name: Foreign key to genre

Candidate Keys
{(catalog_id, genre_name)}

Functional Dependencies
None

Normal Form
BCNF

Creation

```
CREATE TABLE belongs(
    catalog_id int,
    genre_name varchar(20),
    PRIMARY KEY (catalog_id, genre_name),
    FOREIGN KEY(catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY(genre_name) REFERENCES genre(genre_name) ON UPDATE CASCADE
ON DELETE RESTRICT
);
```

2.5. book

Relational Schema
book(catalog_id: int, edition: int, print_location: varchar(20))
catalog_id: Foreign key to library_item

Candidate Keys
{(catalog_id)}

Functional Dependencies
catalog_id → edition print_location

Normal Form
BCNF

Creation

```
CREATE TABLE book(
    catalog_id int,
    edition int,
    print_location varchar(20),
    PRIMARY KEY (catalog_id),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT
);
```

2.6. journal

Relational Schema

journal(catalog_id: int, volume: int, issue: int)
catalog_id: Foreign key to library_item

Candidate Keys

{(catalog_id)}

Functional Dependencies

catalog_id → volume issue

Normal Form

BCNF

Creation

```
CREATE TABLE journal(
    catalog_id int,
    volume int,
    issue int,
    PRIMARY KEY (catalog_id),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT
);
```

2.7. user

Relational Schema

user(user_id: int, username: varchar(20), first_name: varchar(20), last_name: varchar(20), hashed_password: varchar(20), status: boolean, cell_phone: varchar(12), email: varchar(20))

Candidate Keys

{(user_id)}

Functional Dependencies

user_id → username first_name last_name hashed_password status cell_phone, email

Normal Form

BCNF

Creation

```
CREATE TABLE user(
    user_id int,
    username varchar(20) NOT NULL,
```

```
    first_name varchar(20) NOT NULL,
    last_name varchar(20) NOT NULL,
    hashed_password varchar(20) NOT NULL,
    status boolean,
    cell_phone varchar(12),
    email varchar(20),
    PRIMARY KEY (user_id)
);
```

2.8. instructor

Relational Schema

instructor(user_id: int, dept: varchar(5), office_room: varchar(10))
user_id: Foreign key to user

Candidate Keys

{(user_id)}

Functional Dependencies

user_id → dept office_room

Normal Form

BCNF

Creation

```
CREATE TABLE instructor(
    user_id int,
    dept varchar(5),
    office_room varchar(10),
    PRIMARY KEY (user_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.9. student

Relational Schema

student(user_id: int, dept: varchar(5), is_grad: boolean)
user_id: Foreign key to user

Candidate Keys

{(user_id)}

Functional Dependencies
user_id → dept is_grad

Normal Form
BCNF

Creation
```
CREATE TABLE student(
    user_id int,
    dept varchar(5),
    is_grad boolean,
    PRIMARY KEY (user_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.10. librarian

Relational Schema
librarian(user_id: int, working_library: varchar(10))
user_id: Foreign key to user

Candidate Keys
{(user_id)}

Functional Dependencies
user_id → working_library

Normal Form
BCNF

Creation
```
CREATE TABLE librarian(
    user_id int,
    working_library varchar(10),
    PRIMARY KEY (user_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.11. operation

Relational Schema
operation(operation_id: int, date: date, user_id: int)

user_id: Foreign key to librarian

Candidate Keys
{(operation_id)}

Functional Dependencies
operation_id → date user_id

Normal Form
BCNF

Creation

```
CREATE TABLE operation(
    operation_id int,
    date date,
    user_id int,
    PRIMARY KEY (operation_id),
    FOREIGN KEY (user_id) REFERENCES librarian(user_id) ON UPDATE CASCADE
ON DELETE RESTRICT
);
```

Note: create relationship set is removed and the primary key of the one side is added to the many side with total participation.

2.12. hold

Relational Schema
hold(catalog_id: int, user_id: int)
catalog_id: Foreign key to library_item
user_id: Foreign key to user

Candidate Keys
{(catalog_id, user_id)}

Functional Dependencies
None

Normal Form
BCNF

Creation

```
CREATE TABLE hold(
    catalog_id int,
    user_id int,
    date date,
    is_cleared boolean,
```

```
    PRIMARY KEY (catalog_id, user_id, date),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.13. borrow_return

Relational Schema

borrow_return(catalog_id: int, operation_id: int, user_id: int, is_returned: boolean)
catalog_id: Foreign key to library_item
operation_id: Foreign key to operation
user_id: Foreign key to user

Candidate Keys

{(catalog_id, operation_id, user_id)}

Functional Dependencies

catalog_id operation_id user_id → is_returned

Normal Form

BCNF

Creation

```
CREATE TABLE borrow_return(
    catalog_id int,
    operation_id int,
    user_id int,
    is_returned boolean,
    PRIMARY KEY (catalog_id, operation_id, user_id),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (operation_id) REFERENCES operation(operation_id) ON
UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.15. warn

Relational Schema

warn(catalog_id: int, operation_id: int, user_id: int, description: varchar(50), is_cleared: boolean)
user_id: Foreign key to user
catalog_id: Foreign key to library_item
operation_id: Foreign key to operation

Candidate Keys
{(catalog_id, operation_id, user_id)}

Functional Dependencies
catalog_id operation_id user_id → is_cleared description

Normal Form
BCNF

Creation

```
CREATE TABLE warn(
    catalog_id int,
    operation_id int,
    user_id int,
    is_cleared boolean,
    description varchar(50),
    PRIMARY KEY (catalog_id, operation_id, user_id),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (operation_id) REFERENCES operation(operation_id) ON
UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.16. set_late_fee

Relational Schema
set_late_fee(catalog_id: int, operation_id: int, user_id: int, amount: int, date_paid: date)
user_id: Foreign key to user
catalog_id: Foreign key to library_item
operation_id: Foreign key to operation

Candidate Keys
{(catalog_id, operation_id, user_id)}

Functional Dependencies
catalog_id operation_id user_id → amount date_paid

Normal Form
BCNF

Creation

```
CREATE TABLE set_late_fee(
    catalog_id int,
    operation_id int,
    user_id int,
    amount int,
    date_paid date,
    PRIMARY KEY (catalog_id, operation_id, user_id),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (operation_id) REFERENCES operation(operation_id) ON
UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.17. course

Relational Schema
course(course_id: int, course_name: varchar(20))

Candidate Keys
{(course_id)}

Functional Dependencies
course_id → course_name

Normal Form
BCNF

Creation

```
CREATE TABLE course(
    course_id int,
    course_name varchar(20),
    PRIMARY KEY (course_id)
);
```

2.18. teaches

Relational Schema

teaches(course_id: int, user_id: int)
course_id: Foreign key to course
user_id: Foreign key to user

Candidate Keys
{(course_id, user_id)}

Functional Dependencies
None

Normal Form
BCNF

Creation

```
CREATE TABLE teaches(
    course_id int,
    user_id int,
    PRIMARY KEY (course_id, user_id),
    FOREIGN KEY (course_id) REFERENCES course(course_id) ON UPDATE CASCADE
ON DELETE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.19. takes

Relational Schema
takes(course_id: int, user_id: int)
course_id: Foreign key to course
user_id: Foreign key to user

Candidate Keys
{(course_id, user_id)}

Functional Dependencies
None

Normal Form
BCNF

Creation

```
CREATE TABLE takes(
    course_id int,
    user_id int,
    PRIMARY KEY (course_id, user_id),
```

```
    FOREIGN KEY (course_id) REFERENCES course(course_id) ON UPDATE CASCADE
ON DELETE RESTRICT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON UPDATE CASCADE ON
DELETE RESTRICT
);
```

2.20. assign

Relational Schema

assign(<u>catalog_id: int, student_user_id: int, instructor_user_id: int</u>)

catalog_id: Foreign key to library_item
student_user_id: Foreign key to student(user_id)
instructor_user_id: Foreign key to instructor(user_id)

Candidate Keys

{(catalog_id, student_user_id, instructor_user_id)}

Functional Dependencies

None

Normal Form

BCNF

Creation

```
CREATE TABLE assign(
    catalog_id int,
    student_user_id int,
    instructor_user_id int,
    PRIMARY KEY (catalog_id, student_user_id, instructor_user_id),
    FOREIGN KEY (catalog_id) REFERENCES library_item(catalog_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (student_user_id) REFERENCES student(user_id) ON UPDATE
CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (instructor_user_id) REFERENCES instructor(user_id) ON
UPDATE CASCADE ON DELETE RESTRICT
);
```

Note: All IDs are integers as instructor and student IDs have different lengths and char cannot be used. Also, randomized unique IDs are needed for library items, which can much easily be done through a global integer value. All schemas are in BCNF, therefore also in 3NF which is its superset.

3. Functional Components

Below are the use cases of our system, along with the required algorithms for these functionalities.

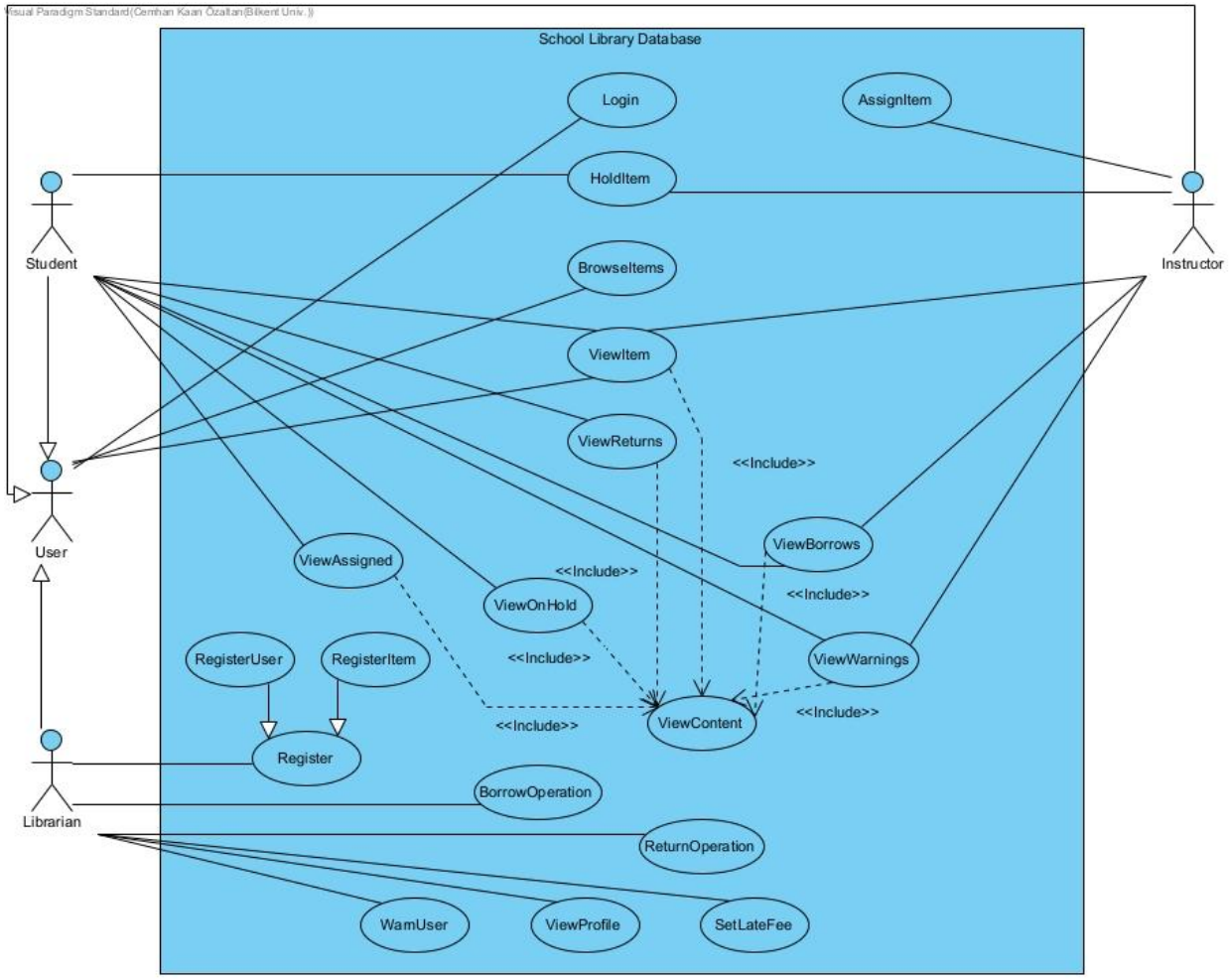


Figure 2: Use Case Diagram

3.1. Login

Prototype of the function:
boolean login(int user_id, String password)

High-level algorithm of the function:
Check if such a user exists
Return true if operation is successful (user exists and credentials are correct)

Use case:
Participating Actor:
➡ User
Entry Condition:

➡ User enters to system

Exit Condition:

➡ User logs in

Flow of Events:

➡ User enters user_id
➡ User enters password
➡ User clicks “Login” button
➡ User is directed to dashboard

3.2. Assigning a Library Item (AssignItem)

Prototype of the function:
boolean assignALibraryItemToAStudent(int student_user_id, int instructor_user_id)

High-level algorithm of the function:
Insert row into “assign” relation with student’s and instructor’s id
Insert row into “create” relation with librarian’s id and operation id
Return true if the assigning was successful

Use case:
Participating Actor:

➡ Instructor

Entry Condition:

➡ Instructor clicks “Assign to students” button which is located on a library item

Exit Condition:

➡ Instructor cancels the operation || Assigning is successful

Flow of Events:

➡ Instructor clicks “Assign to students” button which is located on a library item
➡ Instructor enters user_id of the Student
➡ Instructor clicks “Assign” button
➡ A dialog box shows up
➡ Instructor confirms assigning by clicking “OK” button

3.3. Hold a Library Item (HoldItem)

Prototype of the function:
boolean holdLibraryItem(int catalog_id, int user_id)

High-level algorithm of the function:
Create a hold relation between the user and the library item

Use case:
Participating Actor:

➡ Student, Instructor

Entry Condition:

➡ Actor clicks “Hold” or “Hold Next” Button

Exit Condition:

➡ Actor holds the library item

Flow of Events:

➡ Actor clicks “Hold” or “Hold Next” Button

➡ Actor fills input boxes such as title, author, genre, or published year

➡ Actor clicks “Search” button

➡ Matched library items are listed

Special/Quality Requirements:

➡ Actors may leave input boxes empty if the property (title, author, genre, or published year) does not matter.

3.4. Browsing Library Items (BrowseItems)

Prototype of the function:

LibraryItem[] browseLibraryItem(String title, String author, String genre, int publishedYear)

High-level algorithm of the function:

Find library items according to the search specifications

Use case:

Participating Actor:

➡ User

Entry Condition:

➡ User enters to search page

Exit Condition:

➡ Matched Library Items are listed

Flow of Events:

➡ User clicks “Filter” button

➡ User fills input boxes such as title, author, genre, or published year

➡ User clicks “Search” button

➡ Matched Library Items are listed

Special/Quality Requirements:

➡ Users may leave input boxes empty if the property (title, author, genre, or published year) does not matter.

3.5. BorrowOperation

Prototype of the function:

boolean borrowLibraryItem(int librarian_user_id, int borrower_user_id, int library_item_id)

High-level algorithm of the function:

Insert row into “borrow” relation with user, library, item and operation

Insert row into “create” relation with librarian’s id and operation id

Use case:

Participating Actor:

➡ Librarian
Entry Condition:
➡ Library item is not already borrowed or is on-hold to another student.
Exit Condition:
➡ Borrows successfully or borrowing fails
Flow of Events:
➡ Student asks for the library item that is held by them (face-to-face, outside the system)
➡ Librarian registers the borrowing action to the system

3.6. ReturnOperation

Prototype of the function:
boolean returnLibraryItem(int librarian_user_id, int borrower_user_id, int library_item_id)

High-level algorithm of the function:
Insert row into “return” relation with user, library, item and operation
Insert row into “create” relation with librarian’s id and operation id

Use case:
Participating Actor:
➡ Librarian
Entry Condition:
➡ Someone has a library item to return, approaches librarian
Exit Condition:
➡ Returns successfully
Flow of Events:
➡ Person returns the book to a librarian (face-to-face, outside the system)
➡ Librarian registers the returning action to the system

3.7. Viewing Warning Messages (ViewWarnings)

Prototype of the function:
String[] getWarningMesssages(int student_user_id)

High-level algorithm of the function:
Select warnings with specified user_id from “warn” relation

Use case
Participating Actor:
➡ Student, Instructor
Entry Condition:
➡ Actor has a warning
Exit Condition:
➡ Actor exits warning messages page
Flow of Events:

➡ Actor clicks the “Warnings” button

3.8. Viewing On-Hold Library Items (ViewOnHold)

Prototype of the function:
LibraryItem[] getOnHoldLibraryItems(int student_user_id)

High-level algorithm of the function:
Select library items from “hold” relation with specified user id

Use case:
Participating Actor:
➡ Student, Instructor
Entry Condition:
➡ Actor has on-hold library items
Exit Condition:
➡ Actor exits on-hold page
Flow of Events:
➡ Actor clicks the “On-Hold Library Items” button
Special/Quality Requirements:
➡ Actors are able to only view their current holds.

3.9. Viewing Borrowed Library Items (ViewBorrows)

Prototype of the function:
LibraryItem[] getBorrowedLibraryItems(int student_user_id)

High-level algorithm of the function:
Select library items from “borrow” relation with specified user

Use case:
Participating Actor:
➡ Student, Instructor
Entry Condition:
➡ Actor has borrowed a library item in the past
Exit Condition:
➡ Actor exits borrowed library items page
Flow of Events:
➡ Actor clicks the “Borrowed Library Items” button
Special/Quality Requirements:
➡ Actors are able to view their previous borrowings along with currents.

3.10. Viewing Returned Library Items (ViewReturns)

Prototype of the function:
LibraryItem[] getReturnedLibraryItems(int student_user_id)

High-level algorithm of the function:
Select library items from “return” relation with specified user id

Use case:
Participating Actor:
➡ Student, Instructor
Entry Condition:
➡ Actor has returned a library item in the past
Exit Condition:
➡ Actor exits returned library items page
Flow of Events:
➡ Actor clicks the “Returned Library Items” button
Special/Quality Requirements:
➡ Actors are able to view their previous returns along with currents.

3.11. Viewing Assigned Library Items (ViewAssigned)

Prototype of the function:
LibraryItem[] getAssignedLibraryItems(int student_user_id)

High-level algorithm of the function:
Select library items from “assign” relation with specified user id

Use case:
Participating Actor:
➡ Student
Entry Condition:
➡ Student has been assigned a book by an instructor
Exit Condition:
➡ Student exits assigned library items page
Flow of Events:
➡ Student clicks the “Assigned Library Items” button

3.12. Registering a New User (RegisterUser)

Prototype of the function:
boolean registerANewAccount(int librarian_user_id, User new_user)

High-level algorithm of the function:
Insert row into “register_user” relation with properties of “new_user”

Use case:
Participating Actor:
➡ Librarian
Entry Condition:
➡ User is not already registered
Exit Condition:

➡ User registered successfully

Flow of Events:

➡ Librarian asks for username, name, cell phone number, email to user (face-to-face, outside the system)

➡ Librarian enters input

➡ Librarian clicks “register” button

3.13. Registering a New Library Item (RegisterItem)

Prototype of the function:
boolean registerANewLibraryItem(int librarian_user_id, LibraryItem library_item)

High-level algorithm of the function:
Insert row into “register_item” relation with properties of “library_item”

Use case:
Participating Actor:

➡ Librarian

Entry Condition:

➡ Librarian is on “Register New Library Item” page

➡ Library item is not already registered

Exit Condition:

➡ Register operation is successful

Flow of Events:

➡ Librarian enters specifications of the library item

➡ Librarian clicks “Register Library Item” button

3.14. View a Library Item (ViewItem)

Prototype of the function:
LibraryItem getALibraryItem(int catalog_id)

High-level algorithm of the function:
Return the specified library item

Use case:
Participating Actor:

➡ User

Entry Condition:

➡ User searched a library item

Exit Condition:

➡ Library item is shown

Flow of Events:

➡ User clicks “Details” button which locates on each library item

➡ The details of the library item show up

3.15. View User's Profiles (ViewProfile)

Prototype of the function:
User getAUser(int user_id)

High-level algorithm of the function:
Select a specific user
Use case:
Participating Actor:
➡ Librarian
Entry Condition:
➡ Librarian is in the "All Users" page
Exit Condition:
➡ Librarian sees users' account
Flow of Events:
➡ Librarian clicks "See Account" button which located on each row of students

3.16. Warning Users (WarnUser)

Prototype of the function:
boolean warnAUser(int librarian_user_id, int user_id_to_be_warned)

High-level algorithm of the function:
Insert row into the "warn" relation

Use case:
Participating Actor:
➡ Librarian
Entry Condition:
➡ Librarian is on the user's account page
Exit Condition:
➡ Librarian sends a warning
Flow of Events:
➡ Librarian clicks "Warn" button which is located in the user's account
➡ A popup shows up
➡ Librarian enters description
➡ Librarian clicks "Warn" button

3.17. Fining Late Users (SetLateFee)

Prototype of the function:
boolean fineALateUser(int librarian_user_id, int user_id, int library_item_catalog_id, float amount)

High-level algorithm of the function:
Check if the user has the library item
Insert row into "set_late_fee" with the parameters

Use case:
Participating Actor:
- ➡ Librarian

Entry Condition:
- ➡ User doesn’t return the book on time
- ➡ Librarian is on the users’ account page

Exit Condition:
- ➡ Librarian fines the user

Flow of Events:
- ➡ Librarian clicks “Late fee” Button
- ➡ A popup shows up
- ➡ Librarian enters amount
- ➡ Librarian confirms late fee

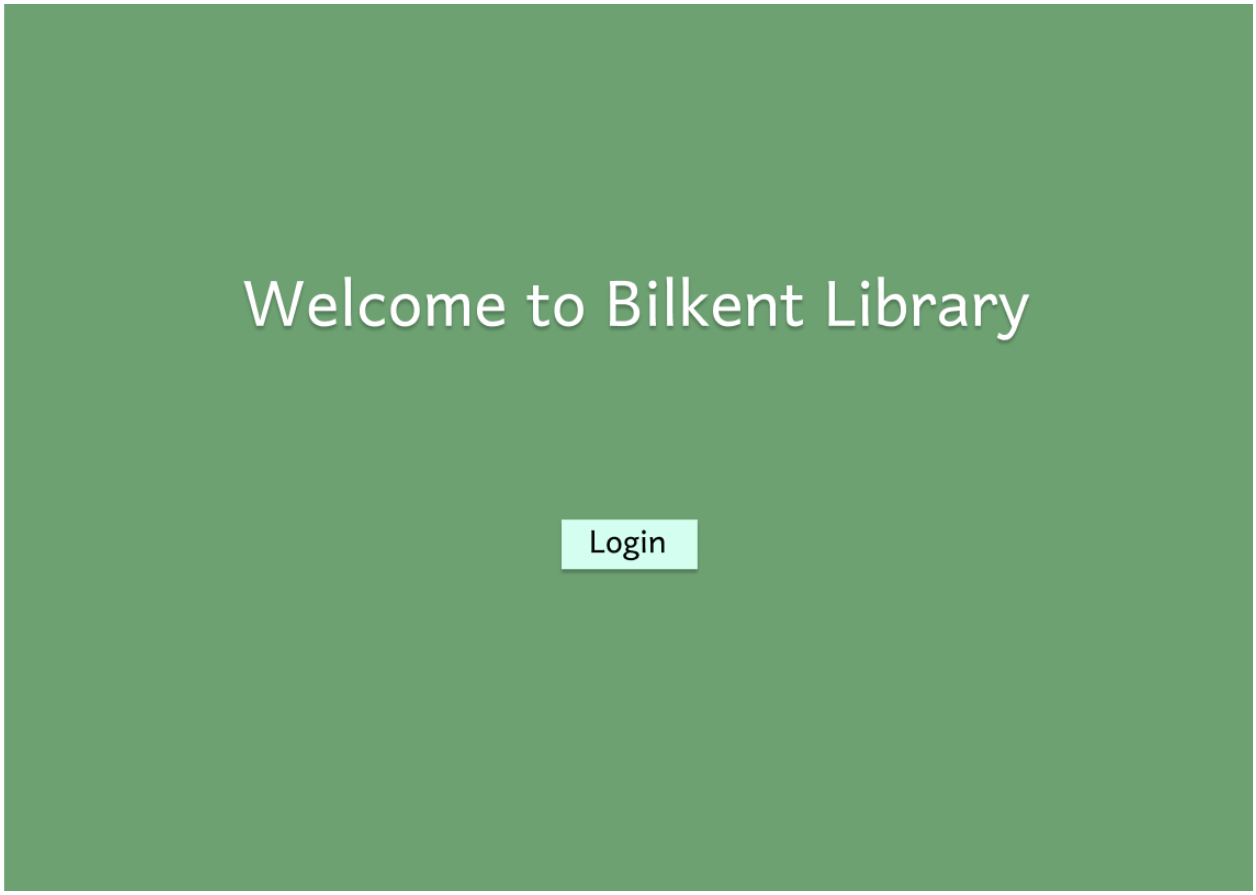
3.18. Algorithms

The used algorithms are given with their use cases. With these algorithms, the array data structure of JavaScript, which is similar to a dynamic array, will be used along with a few additional model classes such as LibraryItem, Book and Journal for more structured code. Further algorithms will be used for the purpose of checking constraints such as password length, which will handle and prevent undesired exceptions and perform the necessary warnings and actions through high-level algorithms implemented in code according to the constraints of the system and other requirements.

4. User Interface Design & SQL Statements

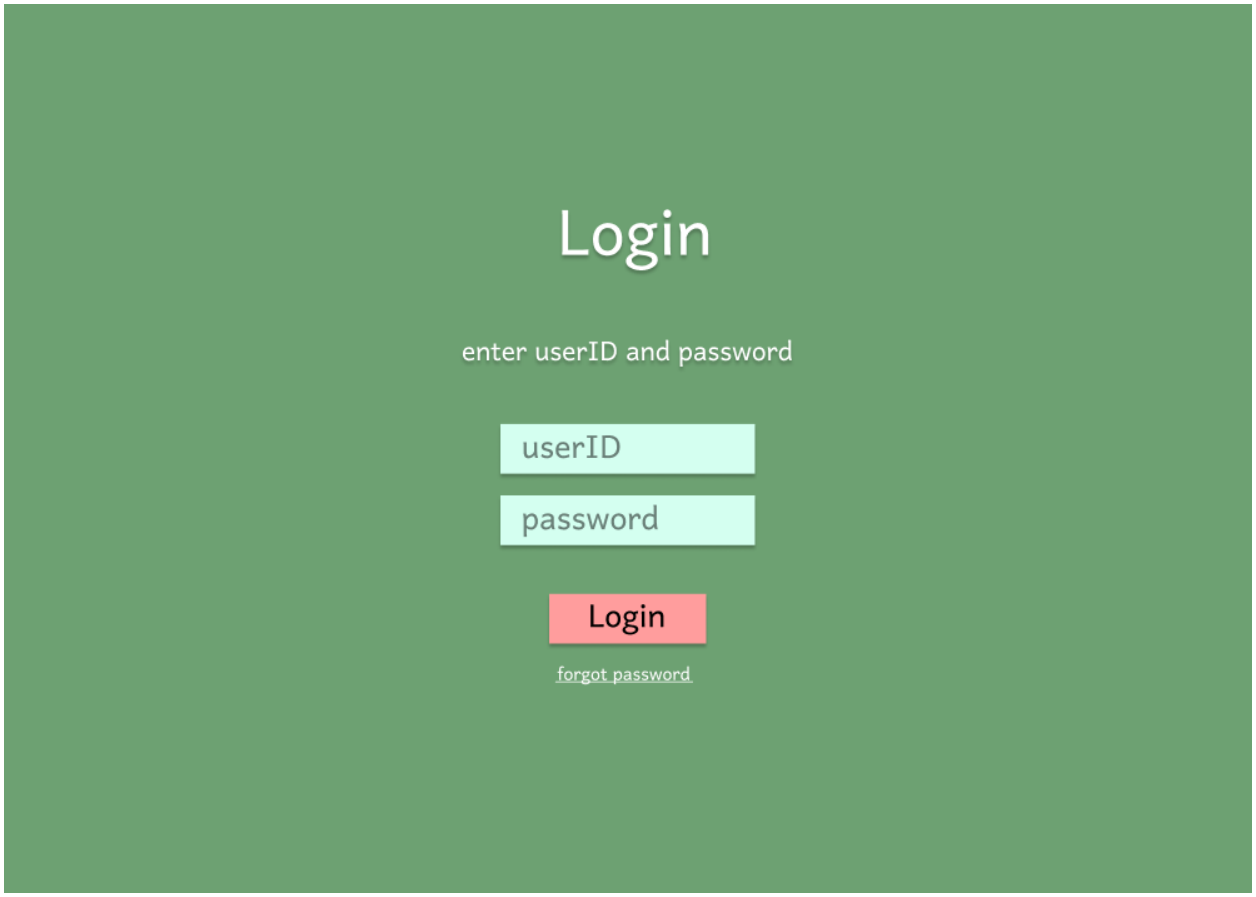
The mockup design of our user interface is as follows, along with the required SQL statements for the functionalities of each page.

4.1. Welcome Page



This is the welcome page. It will appear to all types of users if they are not logged in. Users can click Login to start logging in.

4.2. Login Page



This is the Login page. Users enter their userID and password.

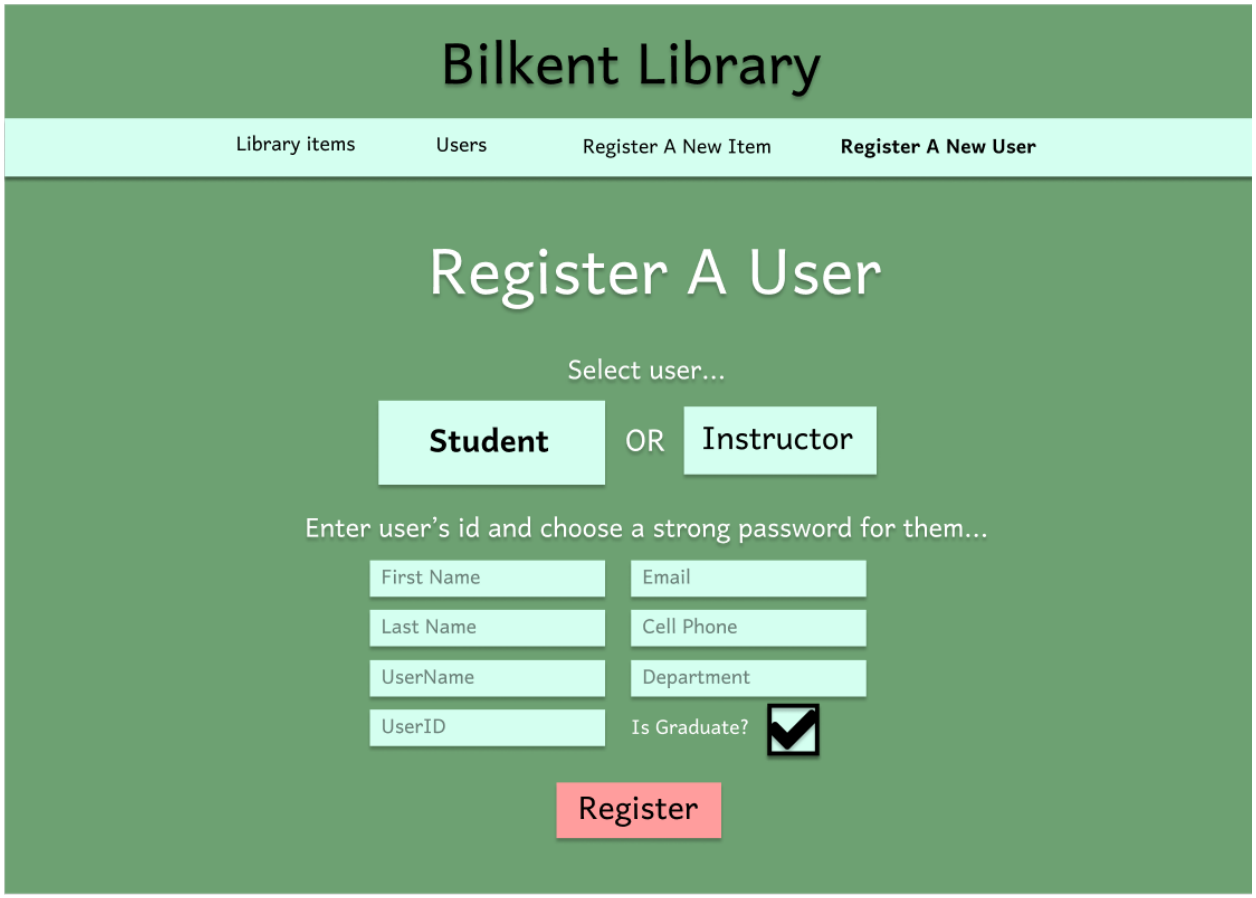
Query for Checking Credentials

```
SELECT *
FROM user
WHERE user_id = @user_id AND hashed_password = @hashed_password;
```

This query returns the user where it matches the information entered.

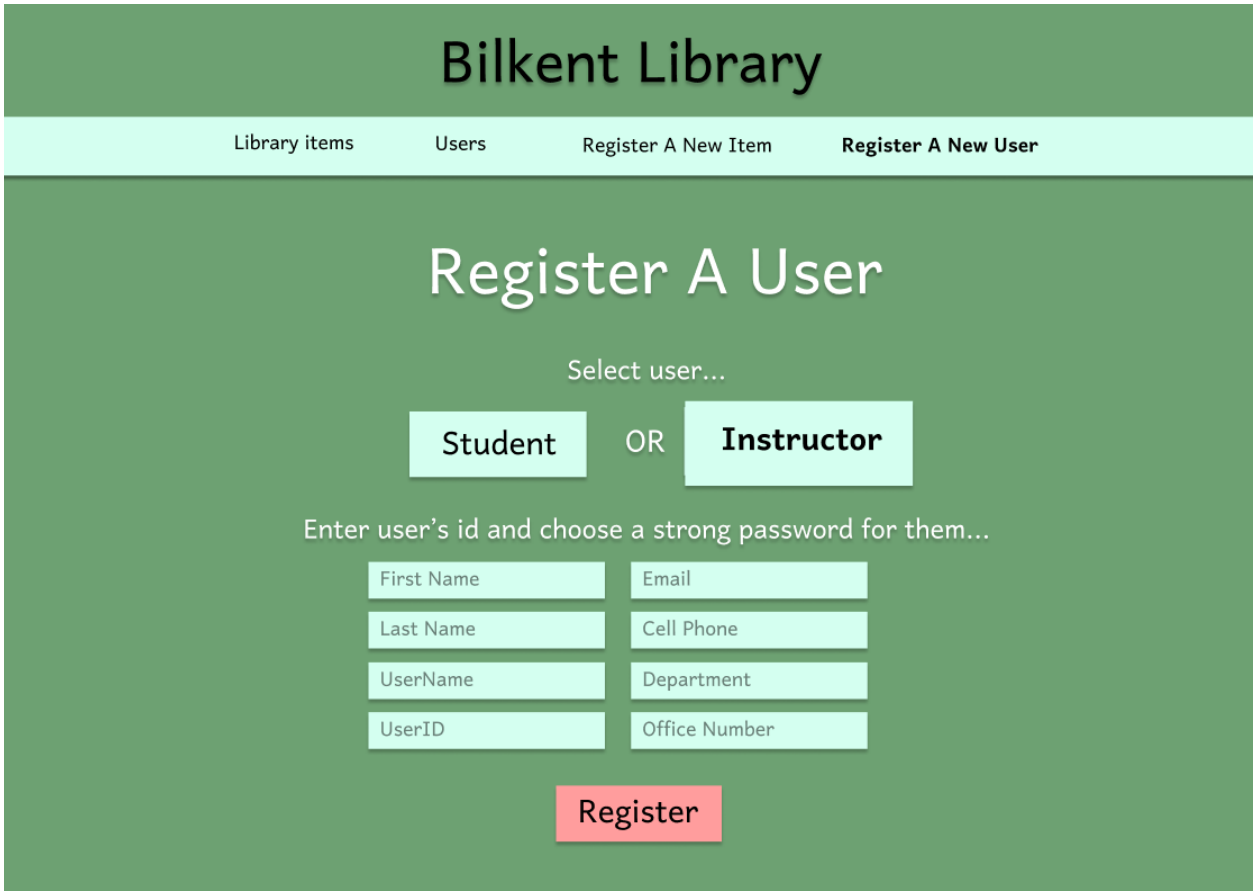
4.3. Register A New User (from Librarian account)

For Student




```
INSERT INTO user
FROM user
WHERE user_id = @user_id AND hashed_password = @hashed_password;
```

For Instructor



Statements for Registering New User

```
INSERT INTO user VALUES (@user_id, @username, @first_name, @last_name,
@hsashed_pasword, true, @cell_phone, @email);
```

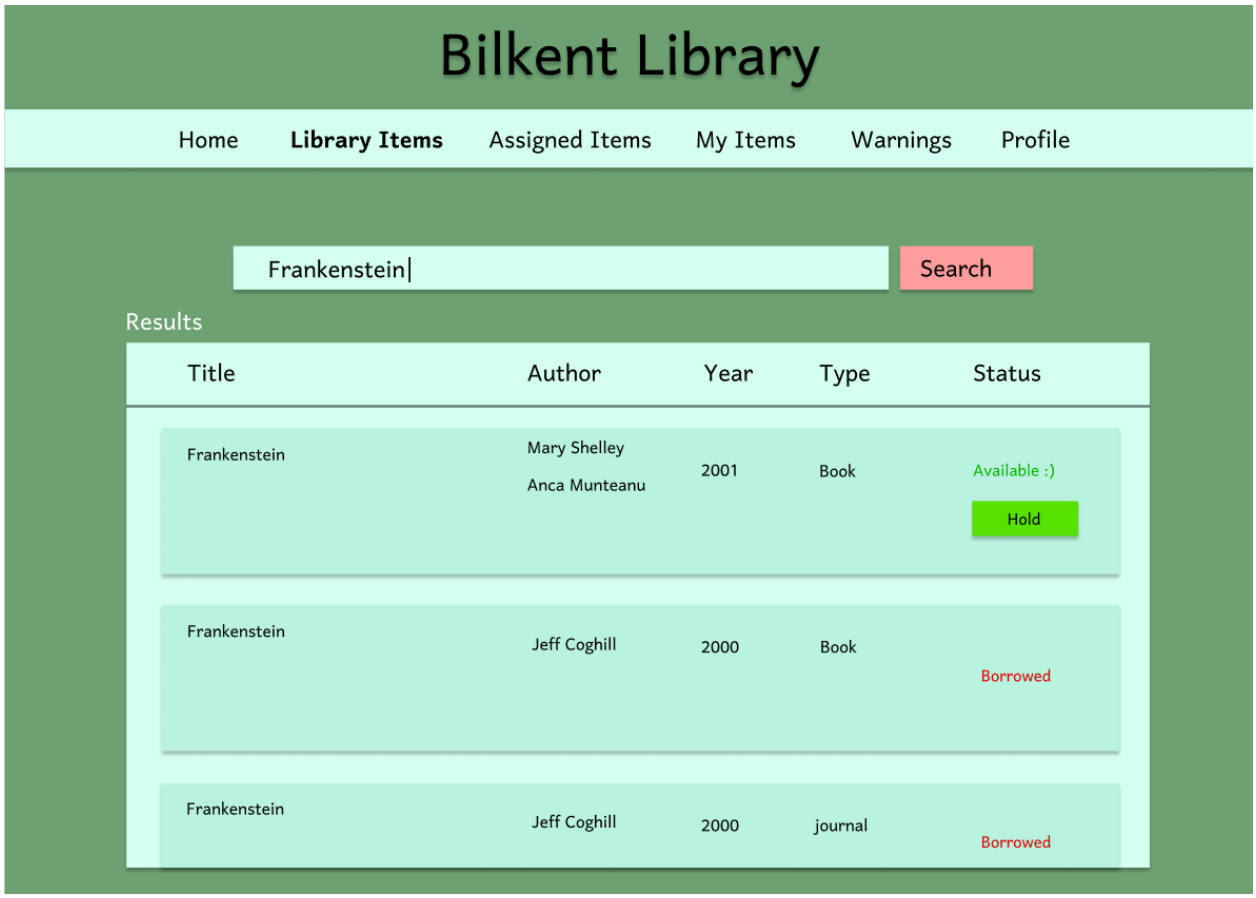
Initially, status is clear, therefore true is inserted. Depending on the selected type, one of the following will be executed right after (password is randomly generated):

```
INSERT INTO student VALUES (@user_id, @dept, @is_grad);
```

```
INSERT INTO instructor VALUES (@user_id, @dept, @office_room);
```

4.4. Browse And Hold Library Items

From Student account



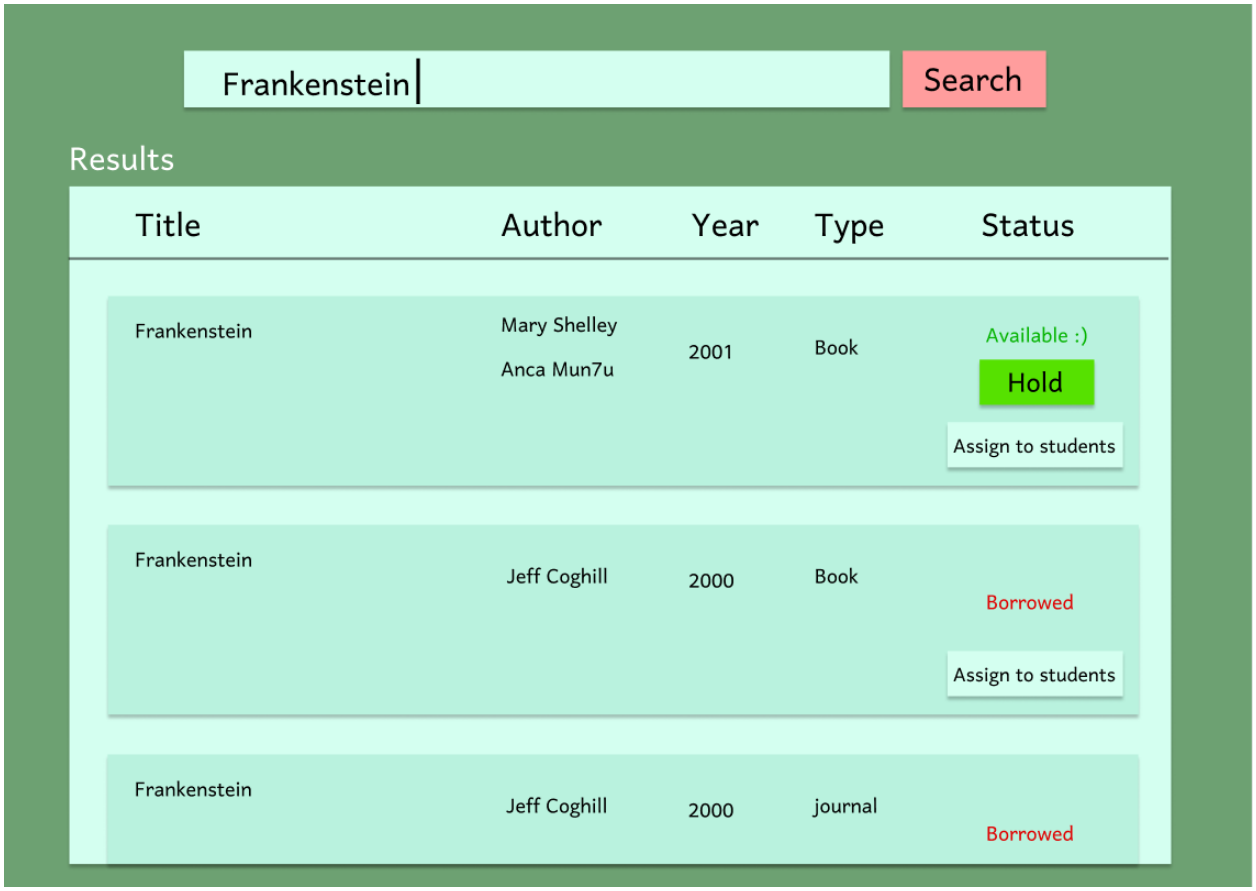
Statements for Holding Item For Student

```
SELECT *
FROM library_item
WHERE title = @search_title;

INSERT INTO hold VALUES (@catalog_id, @user_id, @date)

UPDATE library_item SET is_available = false WHERE catalog_id =
@catalog_id;
```

From Instructor account



Statements for Holding Item For Instructor

```
INSERT INTO hold VALUES (@catalog_id, @user_id, @date);

UPDATE library_item SET is_available = false WHERE catalog_id =
@catalog_id;
```

This inserts a new row in the hold table with id of the item/catalog, user id, and date.

4.5. Assigning Books To Students (from instructor account)

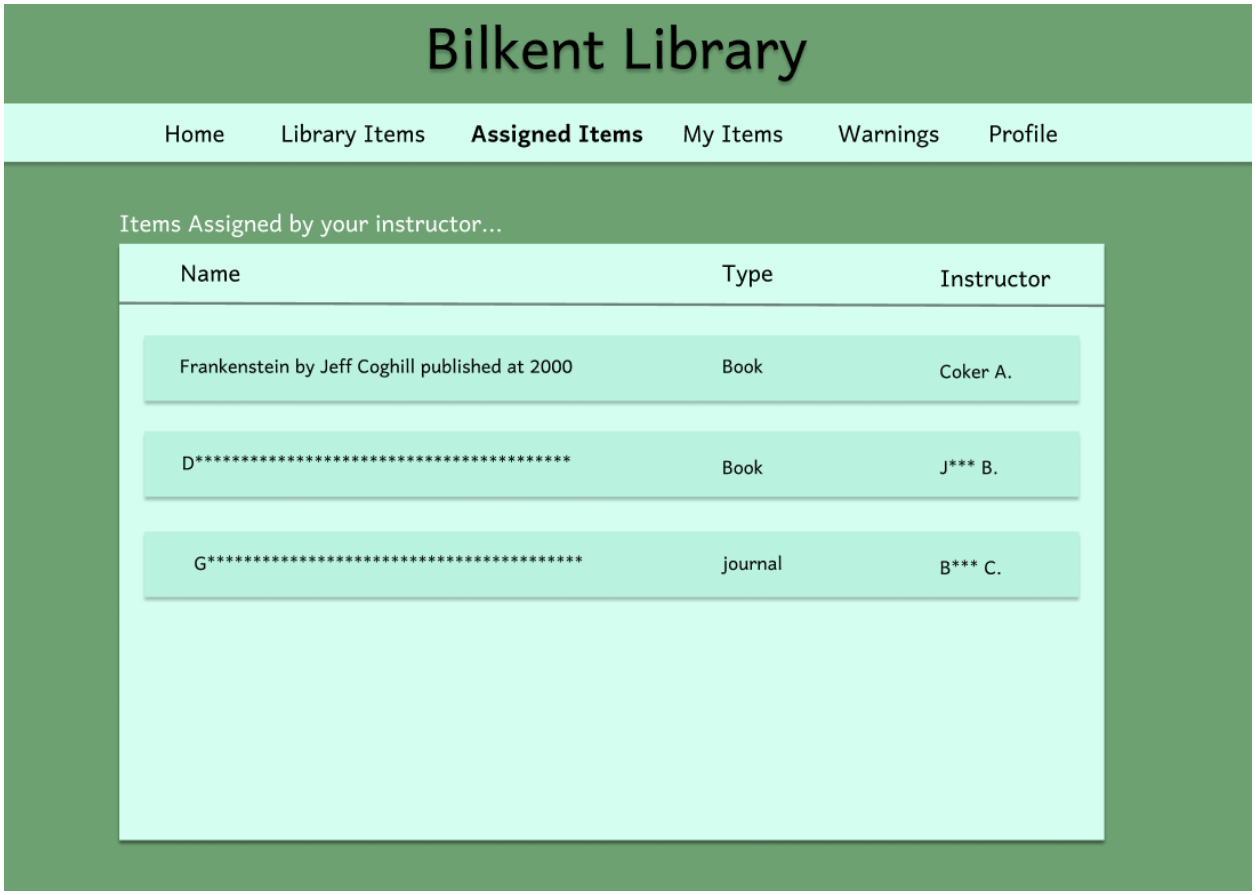
After finding the desired items, the instructor can click “Assign to students” to open the assigning window where the instructor can find all their students, search for some of them and assign the book to them.

Statements for Assigning Item For Student

```
INSERT INTO assign VALUES (@catalog_id, @student_user_id,
@instructor_user_id);
```

This inserts a new row in the assign table with id of the item/catalog, user id, and instructor id.

4.6. Viewing Assigned Books (from student account)



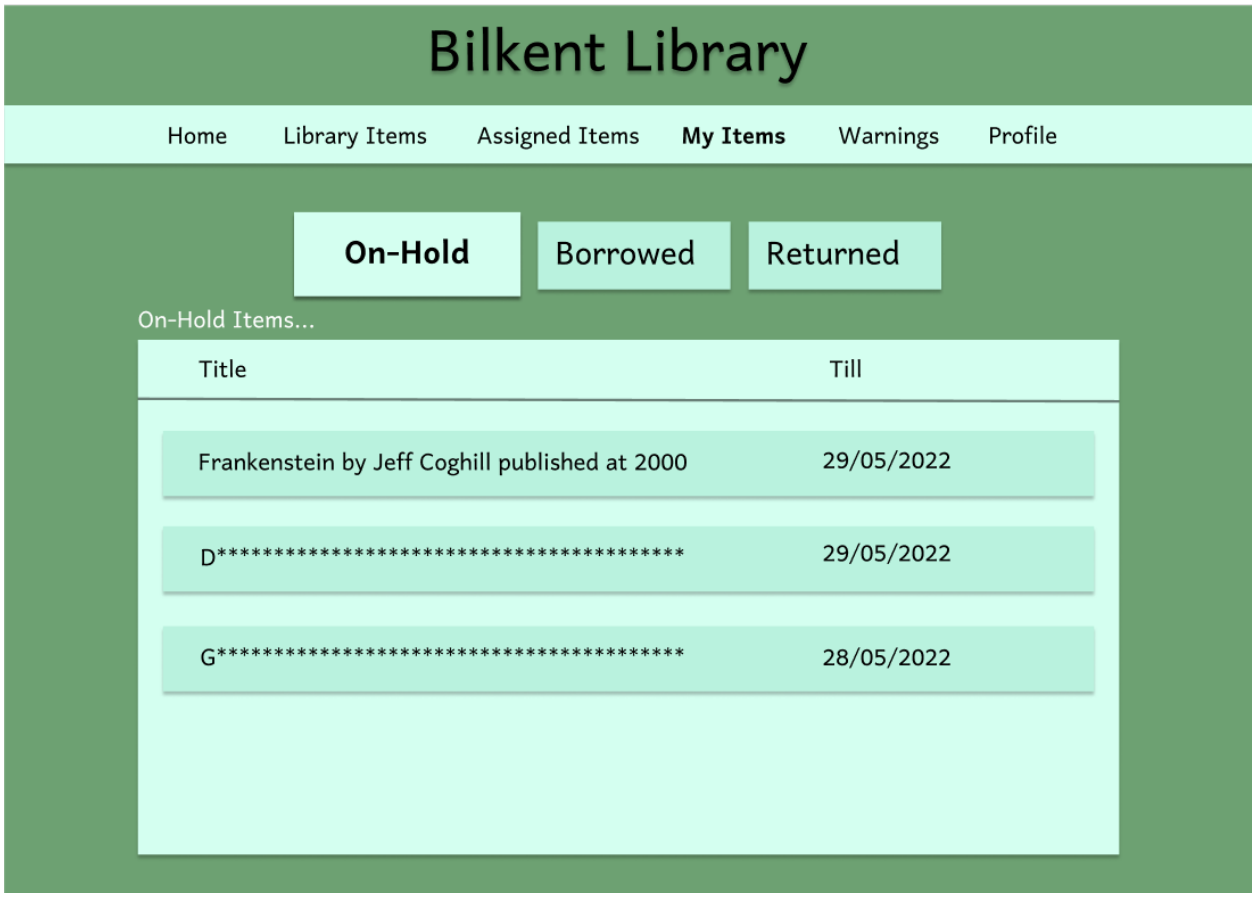
Students can view the assigned book from the “Assigned Items” section.

Statements for Viewing Assigned Books

```
SELECT title, type, username  FROM instructor NATURAL JOIN assign NATURAL
JOIN library_item
WHERE student_user_id = @user_id;
```

This Returns title, type and username, of the assigned books.

4.7. Viewing On-Hold Items



Student can view their On-Hold items from “**My Items/ On-Hold”**

Statements For Viewing Assigned Books

```
SELECT title, date FROM user NATURAL JOIN hold NATURAL JOIN library_item;
```

This returns the title and date of the on-hold book by the student.

4.8. Viewing Borrowed Items

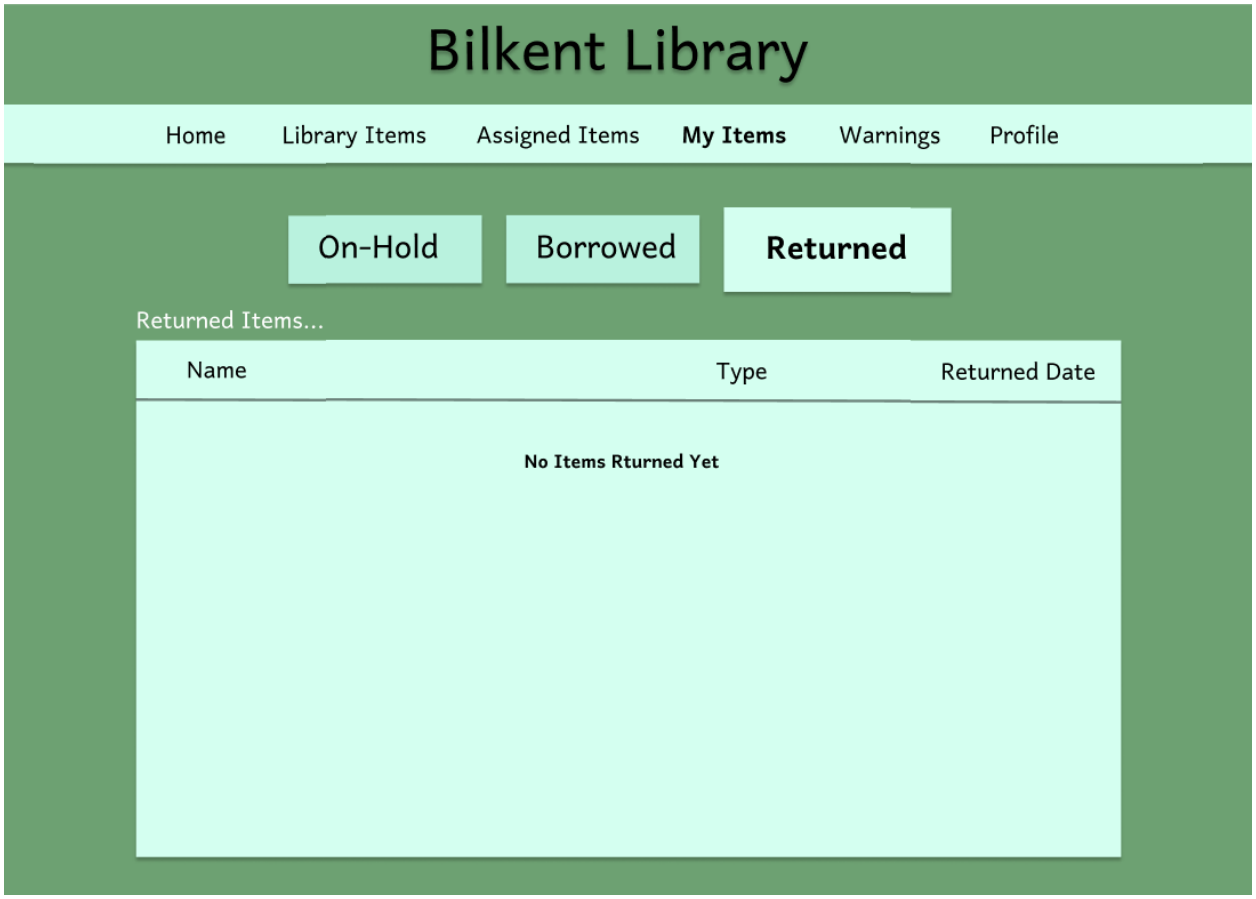
Students can view their Borrowed items from **“My Items/ Borrowed”.**

Statements for Viewing Borrowed Books

```
SELECT title, type, date FROM user NATURAL JOIN borrow_return NATURAL JOIN library_item;
```

This Returns title, type, date, of borrowed books.

4.9. Viewing Returned Items



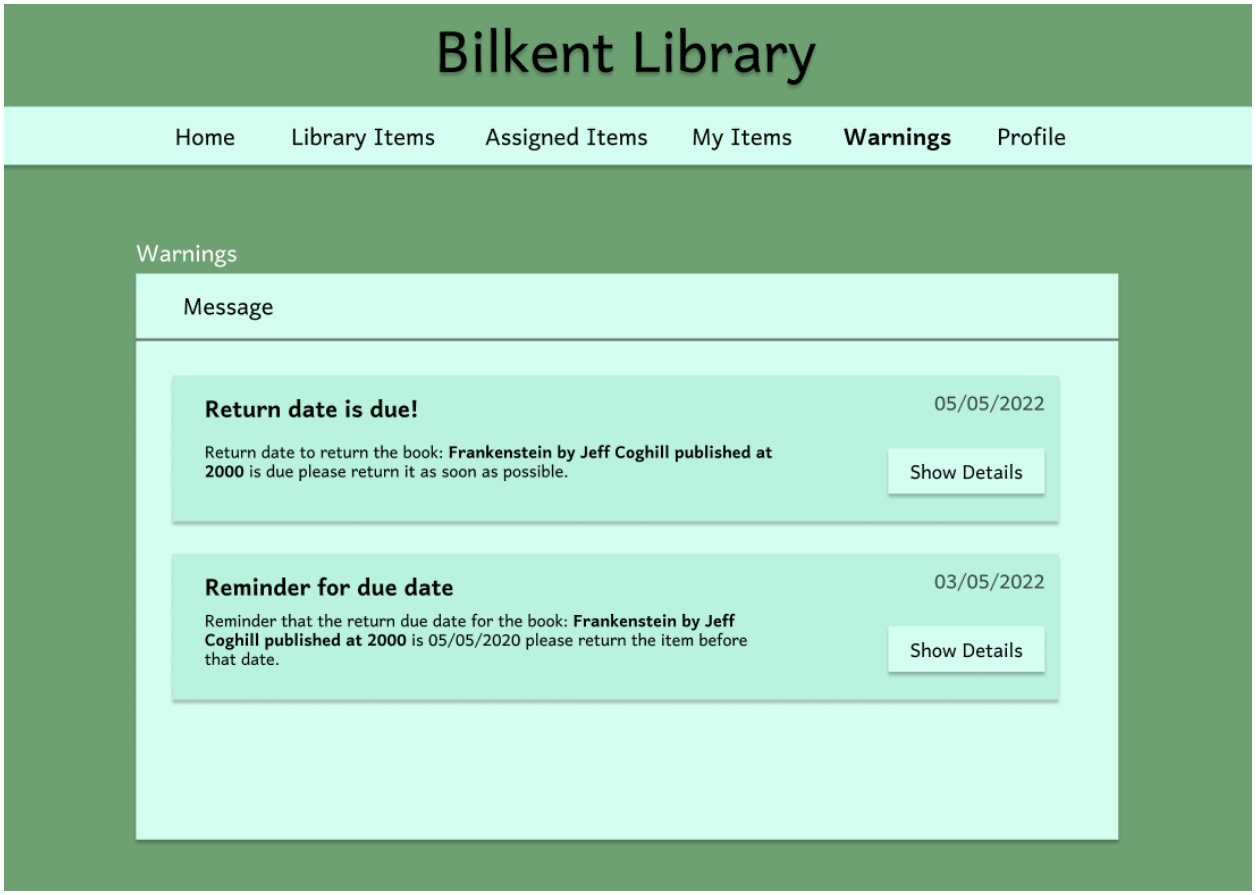
Students can view their Returned items from **“My Items/Returned”.**

Statements for Viewing Returned Books

```
SELECT title, type, date FROM user NATURAL JOIN borrow_return NATURAL JOIN
library_item WHERE is_returned = true
```

This Returns title, type, date, of returned books.

4.10. Viewing Warning Messages



Students can view their Warning messages from the **“Warning”** section .

Statements for Viewing Warnings

```
SELECT description, is_cleared, date FROM user NATURAL JOIN warn NATURAL
JOIN library_item NATURAL JOIN operation WHERE is_cleared = false
```

This Returns description/content, is_clear, and date of a warning.

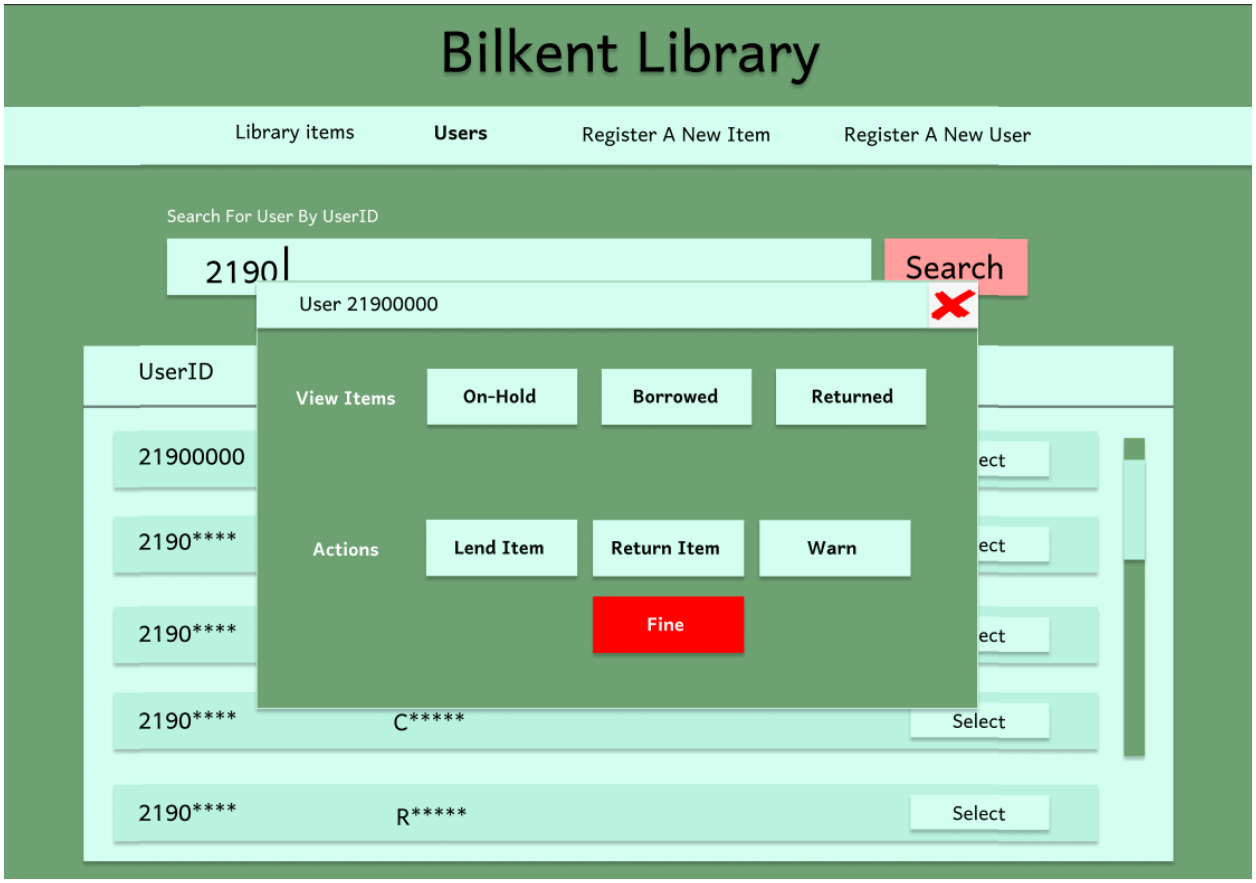
4.11. Viewing users (from Librarian account)

Statements for Browsing Users

```
SELECT user_id, name FROM user;
```

This Returns userID and name of users.

4.12. Selecting users (from Librarian account)

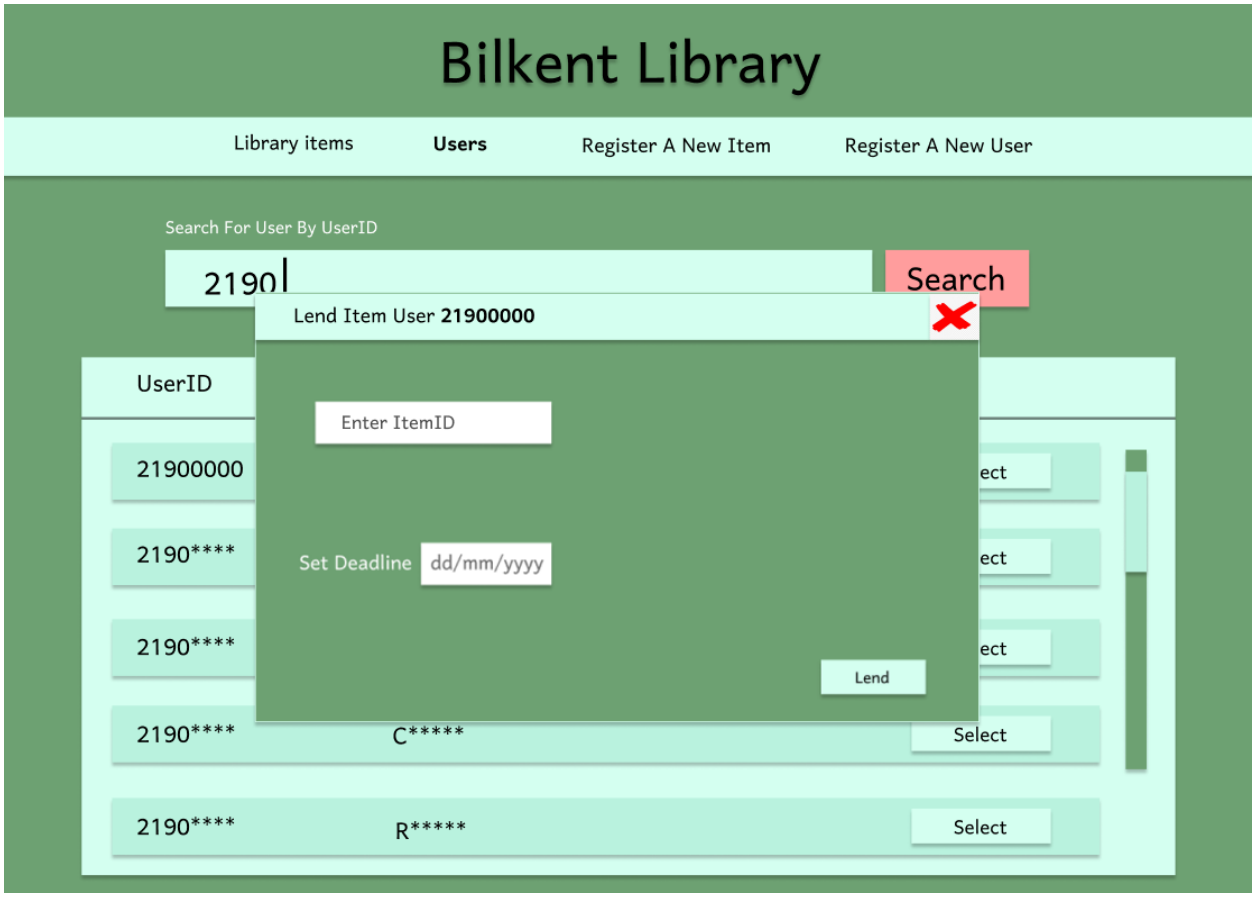


Statements for Selecting A User

```
SELECT user_id FROM user WHERE user_id = @user_id;
```

This returns the userID of the selected user.

4.13. Lending Item To User (from Librarian account)



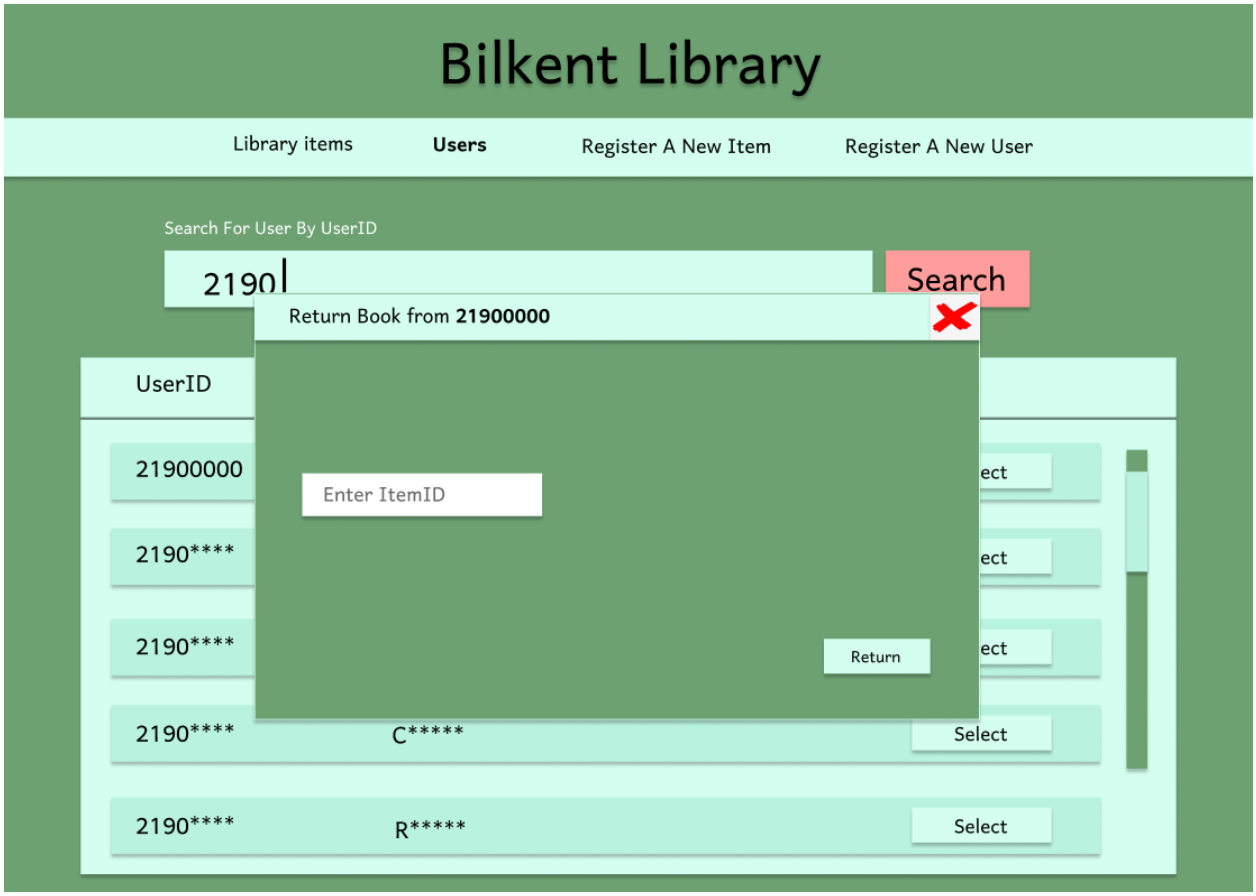
Statements for Lending An Item to A User

```
INSERT INTO borrow_return VALUES (@catalog_id, @operation_id, @user_id,
false)

UPDATE library_item SET is_available = false WHERE catalog_id =
@catalog_id
```

This inserts a new row to the borrow_return table with information of catalog_id/ itemID, operation id, and user id.

4.14. Returning Item To User (from Librarian account)

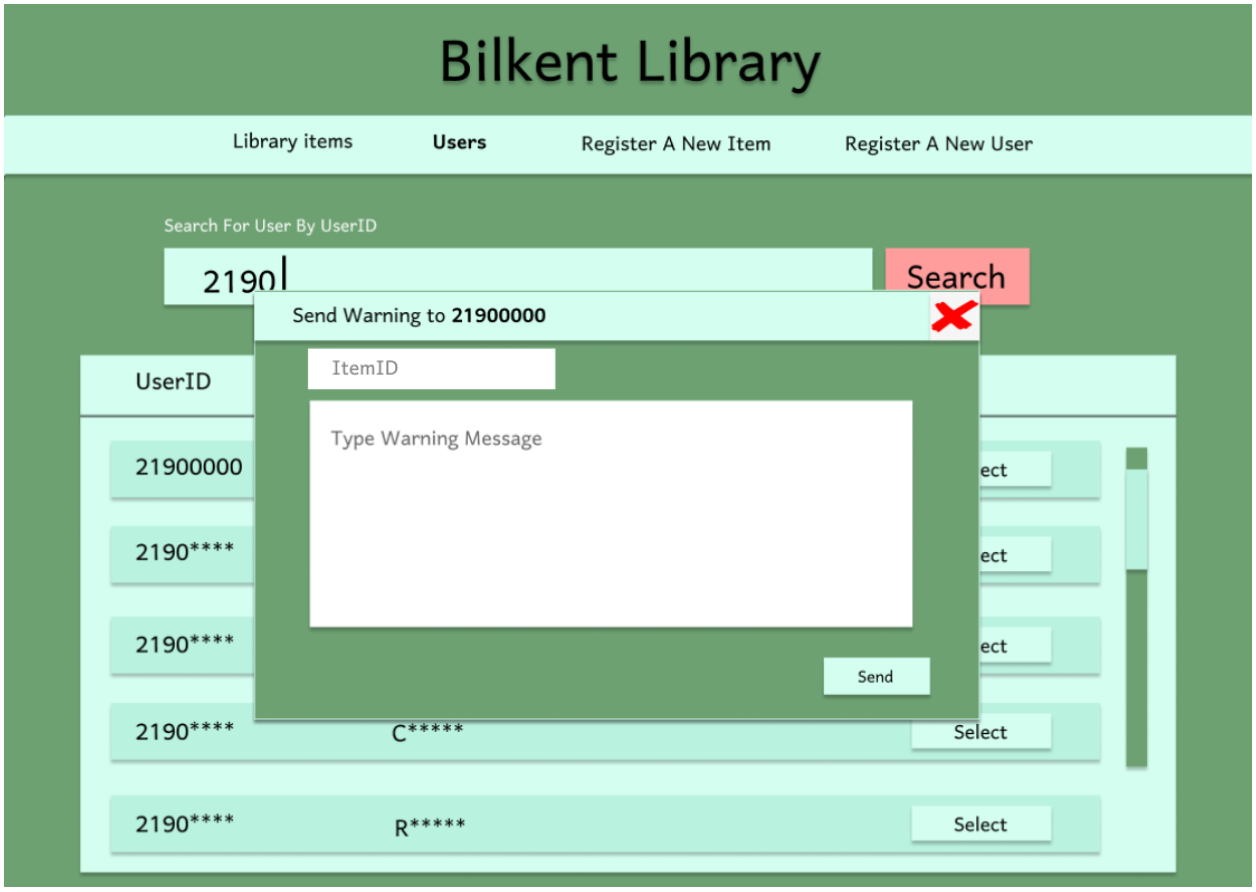


Statements for Returning An Item from A User

```
INSERT INTO borrow_return VALUES (@catalog_id, @operation_id,
@user_id,true)
UPDATE library_item SET is_available = true WHERE catalog_id = @catalog_id
```

This inserts a new row to the borrow_return table with information of catalog_id/ itemID, operation id, and user id.

4.15. Warning User (from Librarian account)



Statements for Sending Warning

```
INSERT INTO operation VALUES (@operation_id, @date, @user_id);
INSERT INTO warn VALUES @catalog_id, @operation_id, @user_id,
@description);
```

This inserts a new row to the warn table with information of catalog_id/ itemID, operation id, description, and user id.

4.16. Fine User (from Librarian account)

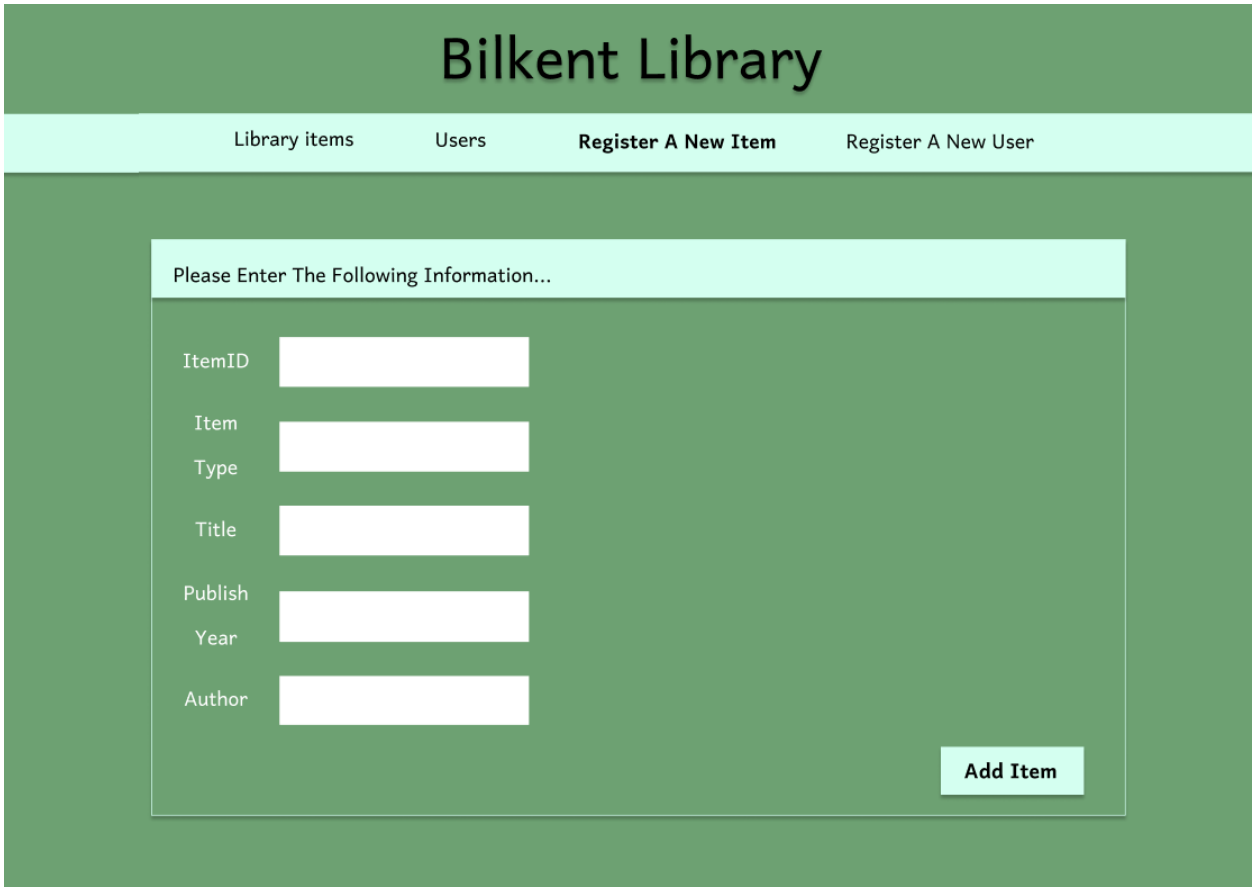
Statements for Setting Late Fee

```
INSERT INTO operation VALUES (@operation_id, @date, @user_id);
INSERT INTO set_late_fee VALUES (@catalog_id, @operation_id, @user_id,
@amount, @date_paid);
```

This inserts a new row to the set_late_fee table with information of catalog_id/ itemID, operation id, amount, date, and user id.

Note: operation_id randomly determined.

4.17. Registering A New Item (from Librarian account)



Statements for Registering A new Item

```
INSERT INTO library_item VALUES (@catalog_id, @title, @type,
@publish_year, @publisher )
UPDATE library_item SET catalog_id = @catalog_id, title= @title, type=
@type, publish_year= @publish_year, publisher= @publisher,
```

This inserts a new row to the library_item table with information of catalog_id/ itemID, title, type, publish year, and publisher.

5. Advanced Database Components

The advanced database components we will use, such as triggers, views and constraints, are listed as follows.

5.1. Reports

The following reports will provide interesting and needed statistics about the system.

Total Number of Library Items for Each Genre

```
SELECT genre_name, count(*) AS cnt
FROM library_item NATURAL JOIN belongs NATURAL JOIN genre
GROUP BY genre_name;
```

Total Number of Library Items That Are Currently Borrowed

```
SELECT count(*) as cur_borrowed_cnt
FROM borrow_return
WHERE is_returned = false;
```

Borrow Counts of Each Library Item Which Has Been Borrowed At Least Once

```
SELECT catalog_id, count(*) as borrow_cnt
FROM borrow_return
GROUP BY catalog_id;
```

5.2. Views

Users View for Librarians

Librarians will use the following view to see all users since they should not see login credentials.

```
CREATE VIEW users_for_librarian AS
SELECT user_id, username, first_name, last_name, status, cell_phone, email
FROM user;
```

Users With Restricted Status for Librarians

Librarians will use this view to access students and instructors who are currently not able to borrow books until they pay their fees.

```
CREATE VIEW restricted_users AS
SELECT *
FROM user
WHERE status = false;
```

List of Currently Borrowed Items for Librarians

Currently borrowed items will be listed with this view in order to make the librarian's task of detecting and setting late fees quicker.

```
CREATE VIEW currently_borrowed AS
SELECT catalog_id
```

```
FROM borrow_return
WHERE is_returned = false;
```

5.3. Triggers

The following triggers perform automatic operations that preserve the consistency of our system

Set Users With Late Fees as Restricted Users

```
DELIMITER //
CREATE TRIGGER set_restricted AFTER INSERT ON set_late_fee
FOR EACH ROW
BEGIN
    UPDATE user
    SET status = false
    WHERE NEW.user_id = user.user_id;
END //
DELIMITER ;
```

Set Users Who Paid Their Fees to Clear Status

```
DELIMITER //
CREATE TRIGGER remove_restricted AFTER UPDATE ON set_late_fee
FOR EACH ROW
BEGIN
    UPDATE user
    SET status = true
    WHERE NEW.user_id = user.user_id;
END //
DELIMITER ;
```

Note: Triggers are in MySQL syntax, where NEW denotes the new row, which is slightly different than examples given in the slides.

5.4. Constraints

1. Users must register to the system through a librarian in order to use it.
2. Passwords are at most 20 and at least 6 characters long.
3. A user can put 5 items on hold at a time.
4. A user can borrow 5 items at a time.

5. Users must pay their pending late fees if they are in the restricted status in order to make further holds or borrows. Otherwise, they will be unable to use the system.
6. An item cannot be held, borrowed or returned at the same time more than once (logical error otherwise, must be returned to be borrowed again and vice versa).
7. All fields must be filled when registering a user by a librarian (not the case for items as some details may be unknown or non-existent).

Other constraints are specified in the table creation statements, such as primary keys, foreign keys and conditions on what to do when a foreign key is updated or deleted, along with not null values.

5.5. Stored Procedures

The following stored procedures will be used due to the frequent need for them.

Login Check

Returns a table with 1 row if login successful, no rows if else.

```
DELIMITER //
CREATE PROCEDURE login(IN uid int, IN pass varchar(20)) BEGIN
    SELECT *
    FROM user
    WHERE user_id = uid AND hashed_password = pass;
END //
DELIMITER ;
```

Browse Library Items by Title

Most common search method, with the only filter being the title of the item.

```
DELIMITER //
CREATE PROCEDURE search_by_title(IN search_title varchar(20)) BEGIN
    SELECT *
    FROM library_item
    WHERE title = search_title;
END //
DELIMITER ;
```

6. Implementation Details

As our database management system (DBMS), we will use MySQL 8.0.28, which can be accessed through the MySQL Workbench software and supports all modern SQL features required by this course. For the backend implementation of our application, we will use the Node.js library of JavaScript, along with the React.js library for the frontend.

The testing of the application will be done locally through npm, and later will be deployed through GitHub facilities. The required libraries will also be acquired through npm. Finally, HTML and CSS will be used in the design and styling of the user interface.

7. Website

The website of this project can be accessed from the following link:
https://kaanozaltan.github.io/school-library-database

8. References

[1] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database system concepts. New York, NY: McGraw-Hill, 2020.