Final Project Report

Library Database

Team 17: Evelyn Drake (jcd171), William Cankar (whc44), Fei Triolo (pht13)

Problem Description

This database stores information about a specific library's book catalog, the list of authors the library has books for, and the registered members of the library. The information per book will contain its title, ISBN, edition, publication date, publisher, copyright year, and genre. The database also stores a list of authors, storing their first name, last name, date of birth, and current status (currently writing or no longer writing). The database will link these authors to specific books, with each book having at least one primary author and zero to many secondary authors. There will be keyword searching functionality to find books by specific keywords. In addition to the high-level catalog of books, the database will also track physical copies of each book (there is one book entry for a specific book, but there can be multiple copies of it). The list of members allows the library to track which people are currently members of the library, tracking their first name, last name, date of birth, and the date when they registered for their library account. Finally, the database includes functionality to allow users to check out specific copies of books and hold them to check out in the future. The database also enables the library to track which books are overdue.

Project Recreation Walkthrough

Name of Database Backup:
Group17LibraryDB.bak
Located within the zip folder titled 'DatabasesFinalFiles.zip'

Java Zip File Name:
Group17LibraryDBJava.zip
Located within the zip folder titled 'DatabasesFinalFiles.zip'

TA Session:
Shravani Suram
12/3/2024
4:30PM - 5PM

Virtual Environment Information:
Host: William Cankar, whc44
Server Name: cxp-sql-03\whc44
Password: p54s3LSq5LepoZ

Database Usernames And Passwords:
Login: member_login
Password: MemberPassword!

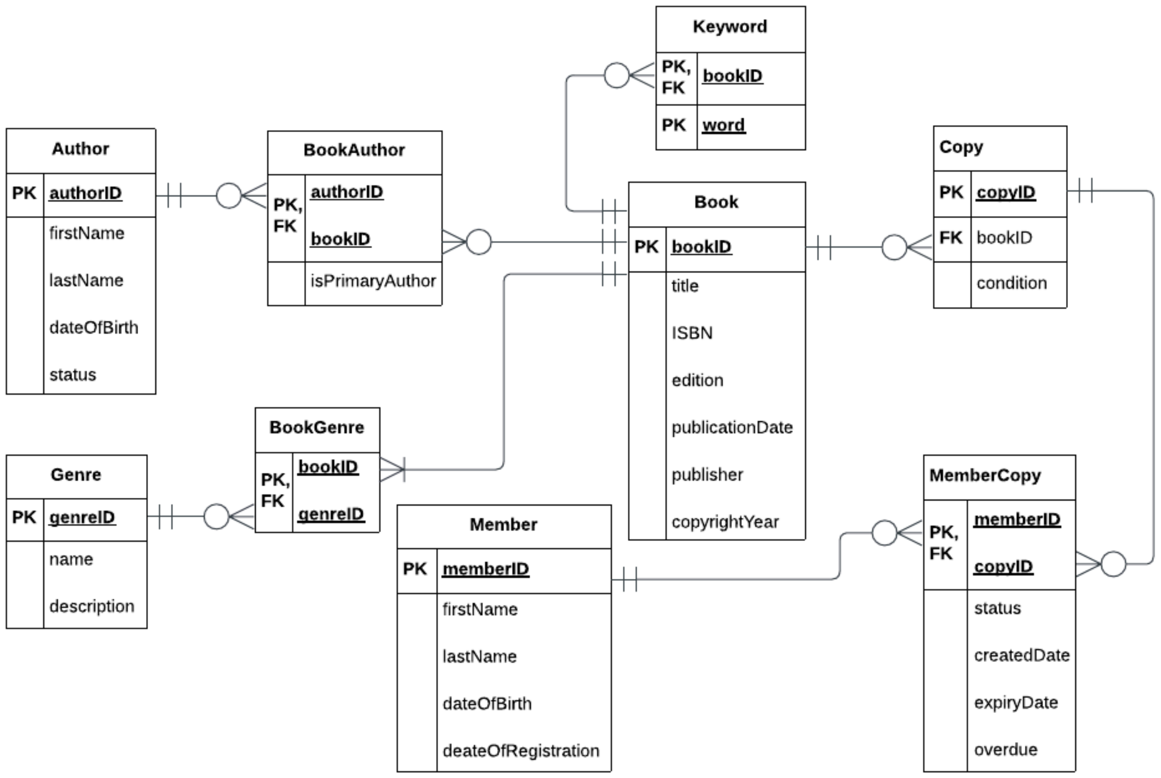
Login: employee_login
Password: EmployeePassword!

Login: curator_login

Password: CuratorPassword!

Library Database Description

Revised ER Diagram



Tables And Their Attributes

- **Book**(<u>bookID</u>, title, ISBN, edition, publicationDate, publisher, copyrightYear)
    - BookID - Primary key that uniquely identifies a book (although the ISBN should be a unique identifier as well, the library could theoretically hold books written before ISBNs were used), int, (this just represents a book in the library's database; the library may own one or more copies of this book, so it does not represent the physical book in the library), this will be incremented each time a book is added
    - Title - The title of the book, varchar
    - ISBN - ISBN, also a unique identifier, int (length 13 to match the ISBN-13 specification)
    - Edition - Edition of the book, int, can be null if edition is unknown
    - PublicationDate - Publication date of the book, date, can be null if publication date is unknown
    - Publisher - Publishing company, varchar, can be null if publisher is unknown

  - CopyrightYear - Copyright year for this edition, int, can be null if year unknown
- **Author**(authorID, firstName, lastName, dateOfBirth, status)
  - AuthorID - Primary key that uniquely identifies an author, int, this will be incremented each time an author is added
  - FirstName - First name of the author, varchar, can be null if unknown
  - LastName - Last name of the author, varchar, presumably this shouldn't be null because if a book has an unknown author, it can just have no BookAuthor entries instead
  - DateOfBirth - Author's date of birth, date, can be null if unknown
  - Status - Status of the author, varchar, "active" or "inactive"
- **BookAuthor**(bookID, authorID, isPrimaryAuthor)
  - BookID - Primary and foreign key corresponding to the ID of the book this relation references, int
  - AuthorID - Primary and foreign key corresponding to the ID of the author this relation references, int
  - IsPrimaryAuthor - Books in the library database can have multiple authors, but only one is designated as the "primary author," this bit/boolean (0=false, 1=true) must not be null, and there should be checks to ensure that a single BookID is not associated with multiple primary authors
- **Keyword**(bookID, word)
  - BookID - Primary and foreign key that uniquely identifies the book this keyword is associated with, int
  - Word - Primary key representing the keyword, varchar, cannot be null, must be checked when added to ensure that a single book does not have multiple instances of the same keyword
- ~~**Reference**(referencingBookID, referencedBookID)~~
  - Reference table was removed because it added too much complexity to the database and was an unnecessary feature for our use cases
- **Genre**(genreID, name, description)
  - GenreID - Primary key that uniquely identifies a genre, int, this will be automated to increment each time a genre is added
  - Name - Name of the genre, varchar, cannot be null (named genreName in database)
  - Description - Brief description of the genre, varchar
- **BookGenre**(bookID, genreID)

  Exists because books can have multiple genres
  - BookID - Primary and foreign key corresponding to the ID of the book this relation references, int
  - GenreID - Primary and foreign key corresponding to the ID of the genre this relation references, int
- **Member**(memberID, firstName, lastName, dateOfBirth, dateOfRegistration)
  - MemberID - Primary key that uniquely identifies this member of the library, int, this will be incremented each time a member is added
  - FirstName - First name of the member, varchar, should not be null
  - LastName - Last name of the member, varchar, should not be null

- DateOfBirth - Member's date of birth, date, should not be null (named memdob in database)
- DateOfRegistration - The date the member registered for the library, date, should not be null (named memdor in database)

- **Copy**(<u>copyID</u>, bookID, condition)
  - CopyID - Primary key corresponding to this specific copy, int, cannot be null, this will be incremented each time a copy is added
  - BookID - Foreign key corresponding to the book this copy is an instance of, int, cannot be null
  - Condition - Represents the condition of this copy, varchar, "good", "neutral", or "poor", cannot be null and defaults to "good"
- **MemberCopy**(<u>memberID</u>, <u>copyID</u>, status, createdDate, expiryDate)
  - MemberID - Primary and foreign key corresponding to the ID of the member this relation references, int
  - CopyID - Primary and foreign key corresponding to the ID of the copy of the book this relation references, int
  - Status - Status of the copy, varchar, "held" (member has only requested this book) or "checkedOut" (member has checked out this copy of the book), cannot be null, and no copy of a book can be held or checked out by more than one unique member (named memStatus is database)
  - CreatedDate - Date that this book has been either held or checked out (if a held book becomes checked out, this updates to the date at which it was checked out), date, cannot be null
  - ExpiryDate - Date that this book must be returned by (or if held, the date at which the hold expires), date, cannot be null

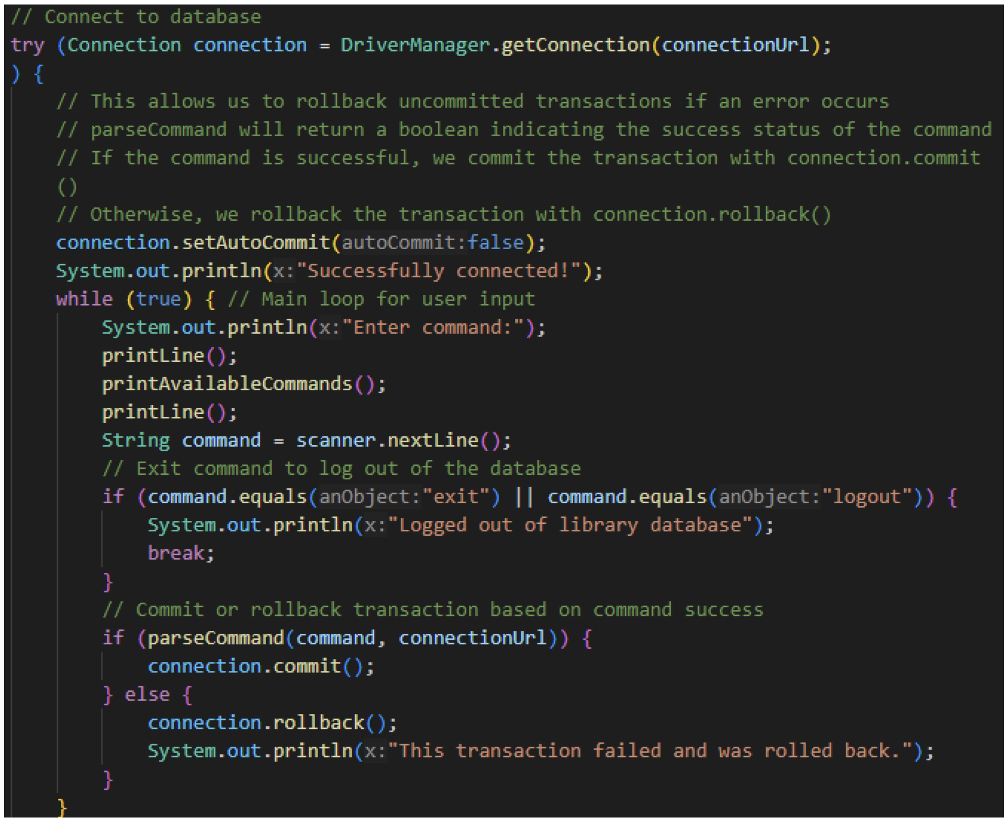
Relations

- Author to BookAuthor: One and only one to zero or many. Each author stored in the database has many or no books they have written in the database represented by BookAuthor.
- BookAuthor to Book: Zero or many to one and only one. Each book can be associated with many authors or none if the author is unknown.
- Keyword to Book: Zero or many to one and only one. Each book can have many keywords associated with it or none if the book was newly added to the database.
- Book to Copy: One and only one to zero or many. Each book will have zero or many instances of a Copy present in the library.
- Copy to MemberCopy: One and only one to zero or many. Since a MemberCopy represents the library's transactions which can be on hold or checked out, one and only one Copy can have zero or multiple MemberCopy instances.
- Genre to BookGenre: One and only one to zero or many. Each genre may have none or multiple books in the collection with it as its genre.
- BookGenre to Book: One or many to one and only one. Each Book will have at least one Genre attached to it through BookGenre. Each bookID in BookGenre will correspond to one and exactly one instance of the Book.

- Member to MemberCopy: One and only one to zero or many. Each member can have zero or multiple MemberCopy relations, and each MemberCopy corresponds to exactly one member.

Use Cases

Note: The Java implementation for all use cases does make use of transactions. The individual methods return a boolean based on the transaction's success status. The program's primary while loop uses this boolean to determine whether to commit or rollback the transaction.



```
// Connect to database
try (Connection connection = DriverManager.getConnection(connectionUrl);
) {
    // This allows us to rollback uncommitted transactions if an error occurs
    // parseCommand will return a boolean indicating the success status of the command
    // If the command is successful, we commit the transaction with connection.commit
    ()
    // Otherwise, we rollback the transaction with connection.rollback()
    connection.setAutoCommit(autoCommit:false);
    System.out.println(x:"Successfully connected!");
    while (true) { // Main loop for user input
        System.out.println(x:"Enter command:");
        printLine();
        printAvailableCommands();
        printLine();
        String command = scanner.nextLine();
        // Exit command to log out of the database
        if (command.equals(anObject:"exit") || command.equals(anObject:"logout")) {
            System.out.println(x:"Logged out of library database");
            break;
        }
        // Commit or rollback transaction based on command success
        if (parseCommand(command, connectionUrl)) {
            connection.commit();
        } else {
            connection.rollback();
            System.out.println(x:"This transaction failed and was rolled back.");
        }
    }
```

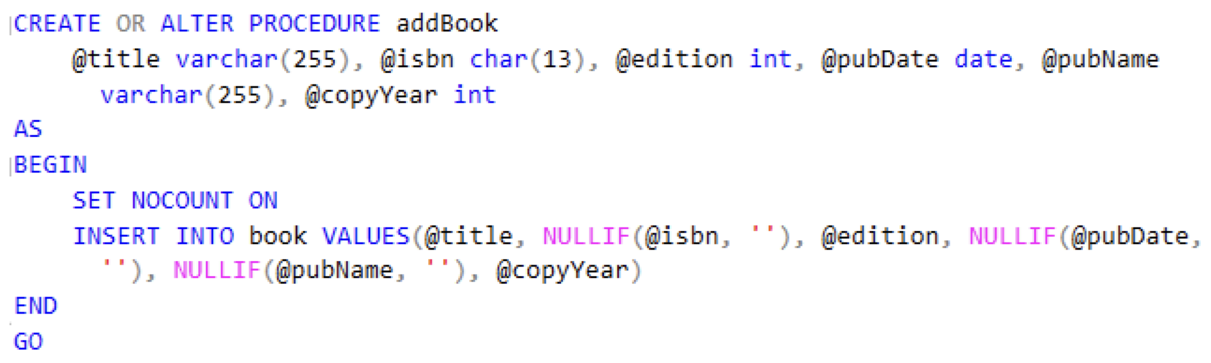
Group

1.
    a. **Title:** Adding a new book to the database
    b. **Description:**
        i. After getting a new book, a library curator needs to add the book to the database in order to allow library patrons to check it out or put it on hold
        ii. The library curator can enter as much or as little information as possible
        iii. This does not add a reference to the physical copy the library owns–just the information about the book itself
    c. **User Requirements:**
        i. The library curator must connect to the database from an account with library curator permissions

ii. In the software, the library curator must select the option to add a new book to the library database

iii. Next, they must follow the instructions they are prompted with, entering as much or as little information about the book as possible

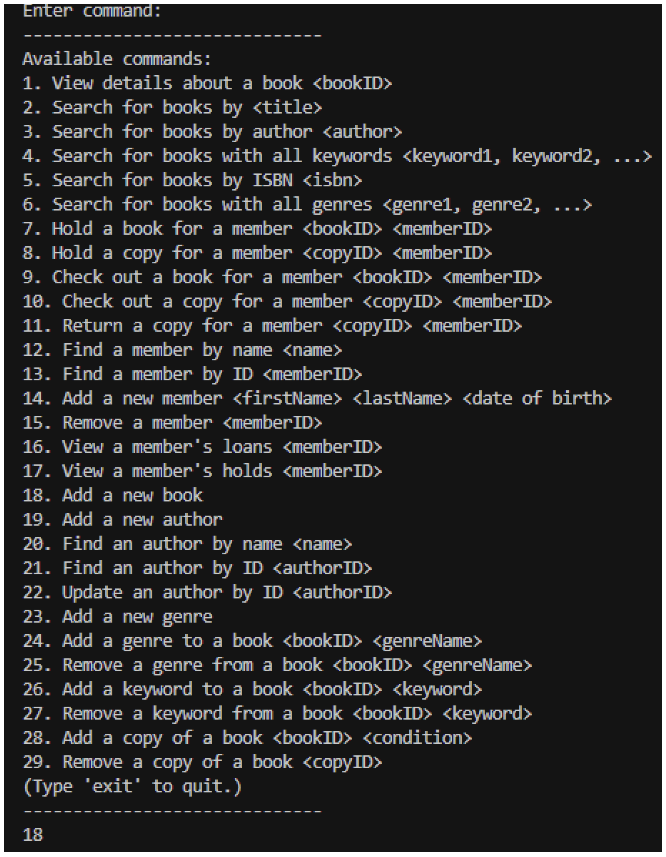
d. **SQL Queries:**



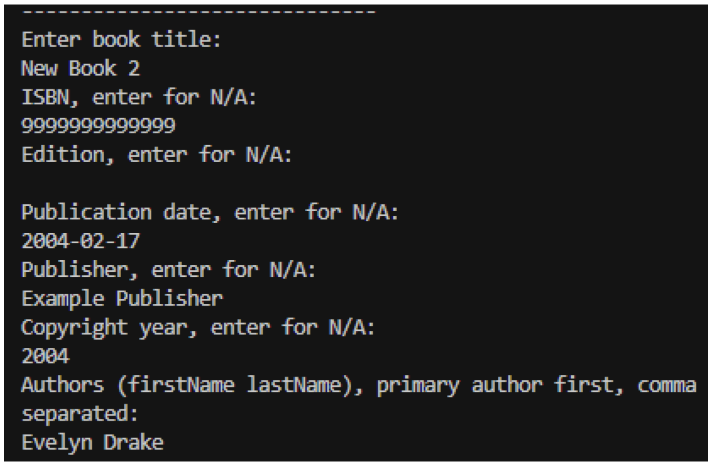
```
CREATE OR ALTER PROCEDURE addBook
    @title varchar(255), @isbn char(13), @edition int, @pubDate date, @pubName
      varchar(255), @copyYear int
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO book VALUES(@title, NULLIF(@isbn, ''), @edition, NULLIF(@pubDate,
      ''), NULLIF(@pubName, ''), @copyYear)
END
GO
```

e. **Screenshots:**

Adding a new book to the database

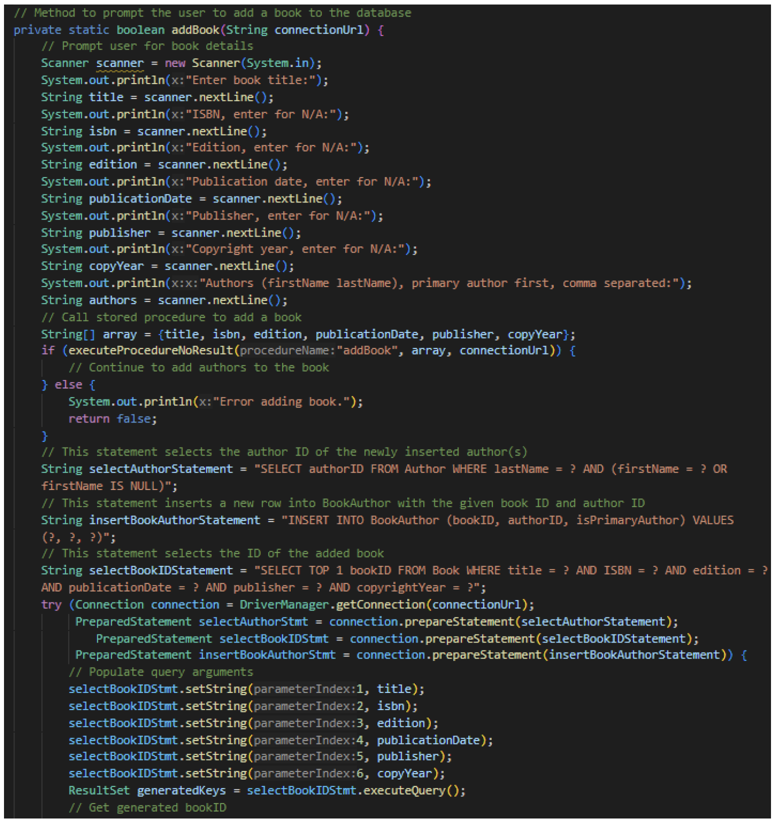


```
Enter command:
------------------------------
Available commands:
1. View details about a book <bookID>
2. Search for books by <title>
3. Search for books by author <author>
4. Search for books with all keywords <keyword1, keyword2, ...>
5. Search for books by ISBN <isbn>
6. Search for books with all genres <genre1, genre2, ...>
7. Hold a book for a member <bookID> <memberID>
8. Hold a copy for a member <copyID> <memberID>
9. Check out a book for a member <bookID> <memberID>
10. Check out a copy for a member <copyID> <memberID>
11. Return a copy for a member <copyID> <memberID>
12. Find a member by name <name>
13. Find a member by ID <memberID>
14. Add a new member <firstName> <lastName> <date of birth>
15. Remove a member <memberID>
16. View a member's loans <memberID>
17. View a member's holds <memberID>
18. Add a new book
19. Add a new author
20. Find an author by name <name>
21. Find an author by ID <authorID>
22. Update an author by ID <authorID>
23. Add a new genre
24. Add a genre to a book <bookID> <genreName>
25. Remove a genre from a book <bookID> <genreName>
26. Add a keyword to a book <bookID> <keyword>
27. Remove a keyword from a book <bookID> <keyword>
28. Add a copy of a book <bookID> <condition>
29. Remove a copy of a book <copyID>
(Type 'exit' to quit.)
------------------------------
18
```

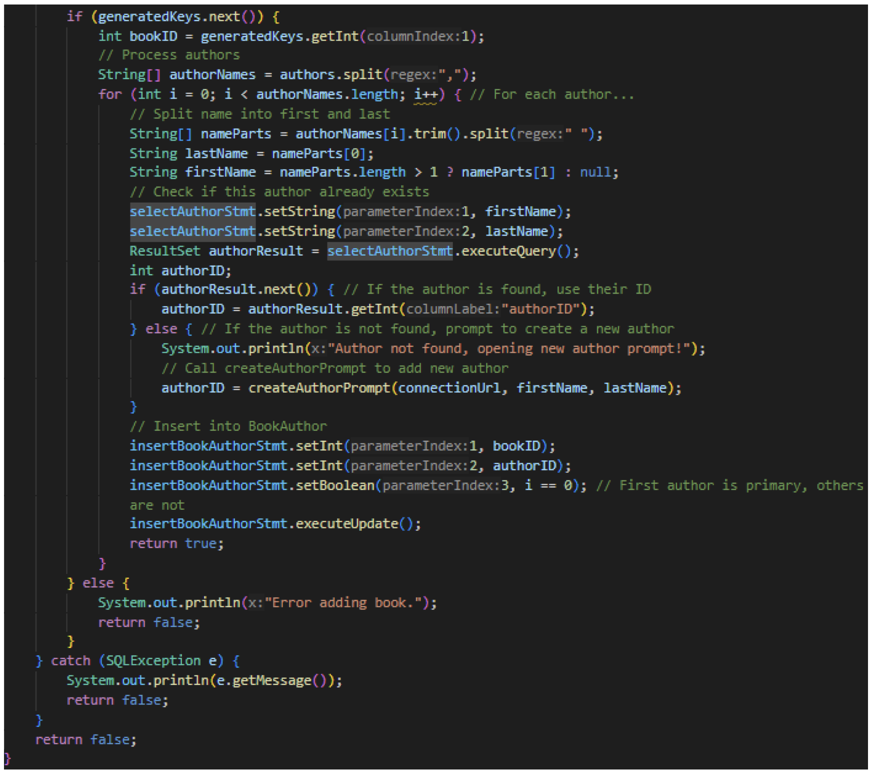


```
------------------------------
Enter book title:
New Book 2
ISBN, enter for N/A:
9999999999999
Edition, enter for N/A:

Publication date, enter for N/A:
2004-02-17
Publisher, enter for N/A:
Example Publisher
Copyright year, enter for N/A:
2004
Authors (firstName lastName), primary author first, comma
separated:
Evelyn Drake
```

f. **Java Implementation:**



```java
// Method to prompt the user to add a book to the database
private static boolean addBook(String connectionUrl) {
    // Prompt user for book details
    Scanner scanner = new Scanner(System.in);
    System.out.println(x:"Enter book title:");
    String title = scanner.nextLine();
    System.out.println(x:"ISBN, enter for N/A:");
    String isbn = scanner.nextLine();
    System.out.println(x:"Edition, enter for N/A:");
    String edition = scanner.nextLine();
    System.out.println(x:"Publication date, enter for N/A:");
    String publicationDate = scanner.nextLine();
    System.out.println(x:"Publisher, enter for N/A:");
    String publisher = scanner.nextLine();
    System.out.println(x:"Copyright year, enter for N/A:");
    String copyYear = scanner.nextLine();
    System.out.println(x:x:"Authors (firstName lastName), primary author first, comma separated:");
    String authors = scanner.nextLine();
    // Call stored procedure to add a book
    String[] array = {title, isbn, edition, publicationDate, publisher, copyYear};
    if (executeProcedureNoResult(procedureName:"addBook", array, connectionUrl)) {
        // Continue to add authors to the book
    } else {
        System.out.println(x:"Error adding book.");
        return false;
    }
    // This statement selects the author ID of the newly inserted author(s)
    String selectAuthorStatement = "SELECT authorID FROM Author WHERE lastName = ? AND (firstName = ? OR
    firstName IS NULL)";
    // This statement inserts a new row into BookAuthor with the given book ID and author ID
    String insertBookAuthorStatement = "INSERT INTO BookAuthor (bookID, authorID, isPrimaryAuthor) VALUES
    (?, ?, ?)";
    // This statement selects the ID of the added book
    String selectBookIDStatement = "SELECT TOP 1 bookID FROM Book WHERE title = ? AND ISBN = ? AND edition = ?
    AND publicationDate = ? AND publisher = ? AND copyrightYear = ?";
    try (Connection connection = DriverManager.getConnection(connectionUrl);
         PreparedStatement selectAuthorStmt = connection.prepareStatement(selectAuthorStatement);
            PreparedStatement selectBookIDStmt = connection.prepareStatement(selectBookIDStatement);
         PreparedStatement insertBookAuthorStmt = connection.prepareStatement(insertBookAuthorStatement)) {
        // Populate query arguments
        selectBookIDStmt.setString(parameterIndex:1, title);
        selectBookIDStmt.setString(parameterIndex:2, isbn);
        selectBookIDStmt.setString(parameterIndex:3, edition);
        selectBookIDStmt.setString(parameterIndex:4, publicationDate);
        selectBookIDStmt.setString(parameterIndex:5, publisher);
        selectBookIDStmt.setString(parameterIndex:6, copyYear);
        ResultSet generatedKeys = selectBookIDStmt.executeQuery();
        // Get generated bookID
```



```java
        if (generatedKeys.next()) {
            int bookID = generatedKeys.getInt(columnIndex:1);
            // Process authors
            String[] authorNames = authors.split(regex:",");
            for (int i = 0; i < authorNames.length; i++) { // For each author...
                // Split name into first and last
                String[] nameParts = authorNames[i].trim().split(regex:" ");
                String lastName = nameParts[0];
                String firstName = nameParts.length > 1 ? nameParts[1] : null;
                // Check if this author already exists
                selectAuthorStmt.setString(parameterIndex:1, firstName);
                selectAuthorStmt.setString(parameterIndex:2, lastName);
                ResultSet authorResult = selectAuthorStmt.executeQuery();
                int authorID;
                if (authorResult.next()) { // If the author is found, use their ID
                    authorID = authorResult.getInt(columnLabel:"authorID");
                } else { // If the author is not found, prompt to create a new author
                    System.out.println(x:"Author not found, opening new author prompt!");
                    // Call createAuthorPrompt to add new author
                    authorID = createAuthorPrompt(connectionUrl, firstName, lastName);
                }
                // Insert into BookAuthor
                insertBookAuthorStmt.setInt(parameterIndex:1, bookID);
                insertBookAuthorStmt.setInt(parameterIndex:2, authorID);
                insertBookAuthorStmt.setBoolean(parameterIndex:3, i == 0); // First author is primary, others
                are not
                insertBookAuthorStmt.executeUpdate();
                return true;
            }
        } else {
            System.out.println(x:"Error adding book.");
            return false;
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
    return false;
}
```

2.

a. **Title:** Adding and removing physical copies of a book to/from the database

b. **Description:**

i. After a book's information is added to the library database, a curator will need to add the copy/copies of this book to the database to allow patrons to check it out or put it on hold

ii. If this copy no longer exists for whatever reason, the its representation in the database must be deleted

iii. When adding a copy, the curator enters the ID of the book they're making a copy of and the condition of the book (good, neutral, or poor)

iv. When removing a copy, the curator enters the ID of the copy they're removing

c. **User Requirements:**

i. The library curator must connect to the database from an account with library curator permissions

ii. In the software, the library curator must select the option to add a new copy of a book or the option to remove a copy of a book

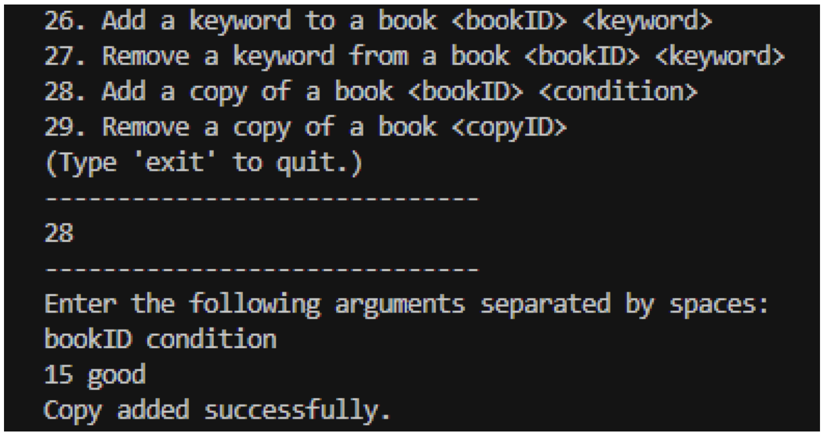
iii. Next, they must enter all the information required by the prompts

d. **SQL Queries:**

```sql
CREATE OR ALTER PROCEDURE addCopy @bookID int, @condition varchar(7)
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO copy VALUES (@bookID, @condition)
END
GO

CREATE OR ALTER PROCEDURE removeCopy @copyID int
AS
BEGIN
    SET NOCOUNT ON
    DELETE FROM copy WHERE copyID = @copyID
END
GO
```

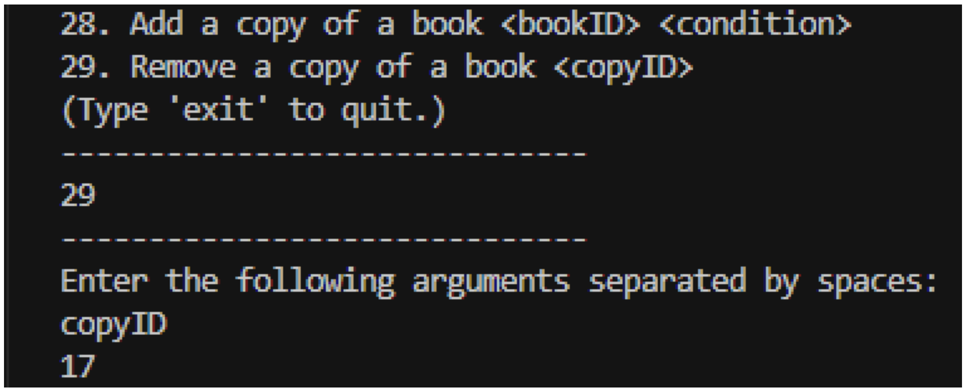
e. **Screenshots**:

Adding a new copy of a book with ID 15



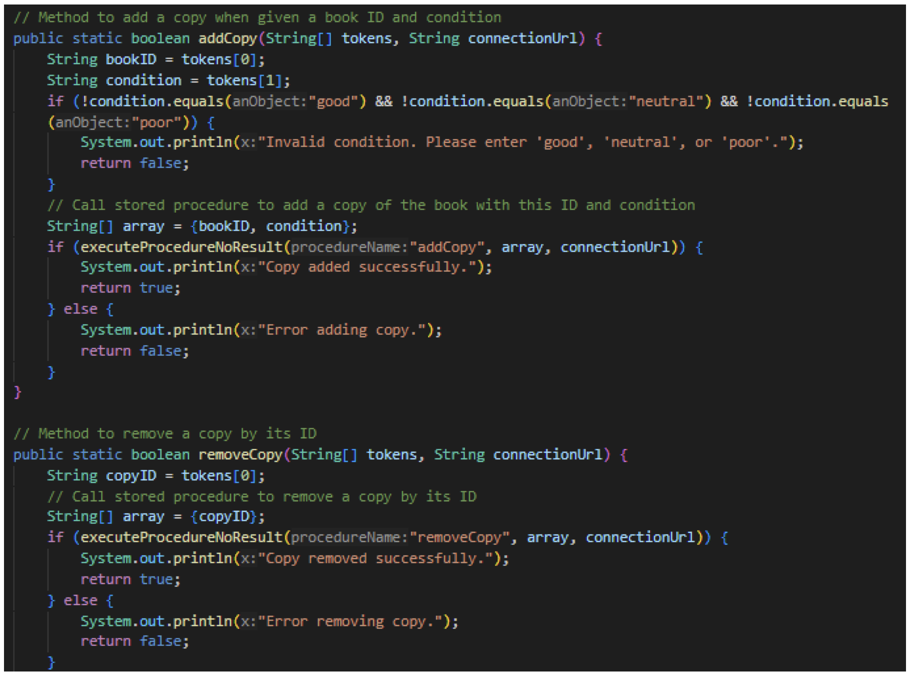
```
26. Add a keyword to a book <bookID> <keyword>
27. Remove a keyword from a book <bookID> <keyword>
28. Add a copy of a book <bookID> <condition>
29. Remove a copy of a book <copyID>
(Type 'exit' to quit.)
------------------------------
28
------------------------------
Enter the following arguments separated by spaces:
bookID condition
15 good
Copy added successfully.
```

Removing the copy with ID 17 from the library database



```
28. Add a copy of a book <bookID> <condition>
29. Remove a copy of a book <copyID>
(Type 'exit' to quit.)
------------------------------
29
------------------------------
Enter the following arguments separated by spaces:
copyID
17
```

f. **Java Implementation:**



```java
// Method to add a copy when given a book ID and condition
public static boolean addCopy(String[] tokens, String connectionUrl) {
    String bookID = tokens[0];
    String condition = tokens[1];
    if (!condition.equals(anObject:"good") && !condition.equals(anObject:"neutral") && !condition.equals
    (anObject:"poor")) {
        System.out.println(x:"Invalid condition. Please enter 'good', 'neutral', or 'poor'.");
        return false;
    }
    // Call stored procedure to add a copy of the book with this ID and condition
    String[] array = {bookID, condition};
    if (executeProcedureNoResult(procedureName:"addCopy", array, connectionUrl)) {
        System.out.println(x:"Copy added successfully.");
        return true;
    } else {
        System.out.println(x:"Error adding copy.");
        return false;
    }
}

// Method to remove a copy by its ID
public static boolean removeCopy(String[] tokens, String connectionUrl) {
    String copyID = tokens[0];
    // Call stored procedure to remove a copy by its ID
    String[] array = {copyID};
    if (executeProcedureNoResult(procedureName:"removeCopy", array, connectionUrl)) {
        System.out.println(x:"Copy removed successfully.");
        return true;
    } else {
        System.out.println(x:"Error removing copy.");
        return false;
    }
```

3.

a. **Title:** Adding new members (patrons) to the database and removing existing members

b. **Description:**

   i. Employees and curators can add new members to the database

   ii. Employees and curators can also remove members from the database

   iii. A prospective member approaches the library's front desk and gives the information to the employee, who enters it into the database, or they tell the employee they're no longer interested in having a library account

c. **User Requirements:**

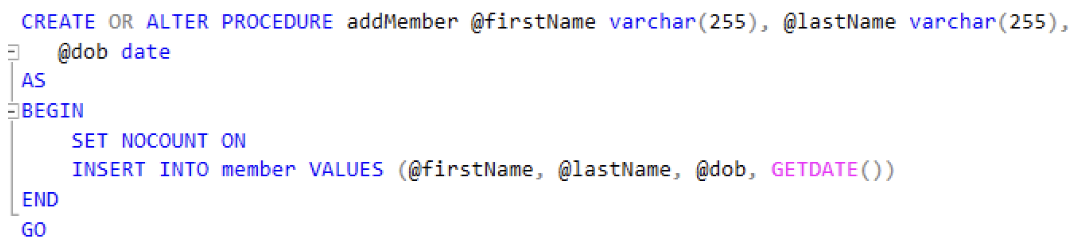
   i. The user must connect to the database from an account with library employee or curator permissions

   ii. In the software, the user must select the option to add a new member or remove an existing member

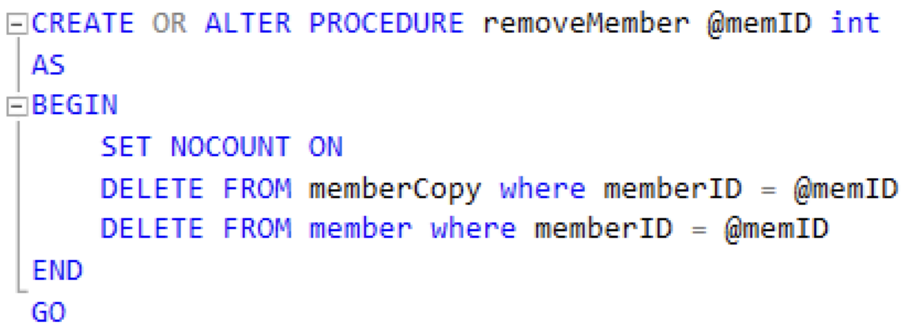
iii. When adding a new member, they must enter all information requested by the prompts (first name, last name, and date of birth)--note that their registration date is added automatically based on the current date

iv. When removing an existing member, they must enter the ID of the member

d. **SQL Queries:**



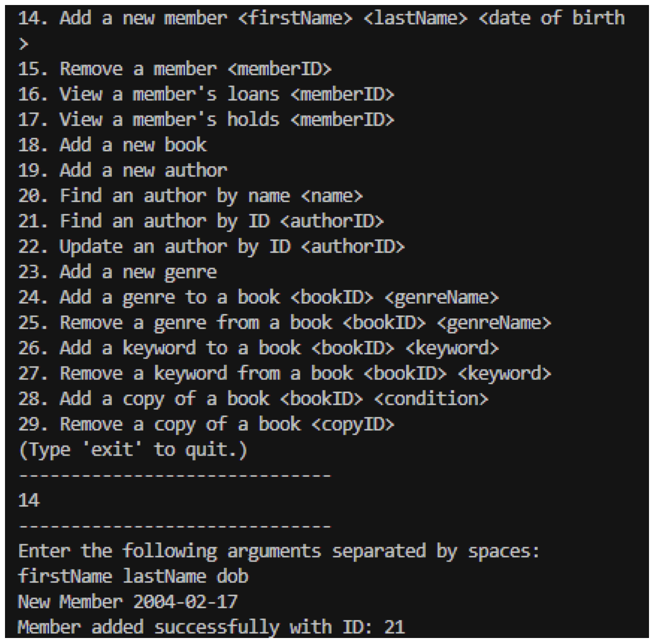
```
CREATE OR ALTER PROCEDURE addMember @firstName varchar(255), @lastName varchar(255),
  @dob date
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO member VALUES (@firstName, @lastName, @dob, GETDATE())
END
GO
```



```
CREATE OR ALTER PROCEDURE removeMember @memID int
AS
BEGIN
    SET NOCOUNT ON
    DELETE FROM memberCopy where memberID = @memID
    DELETE FROM member where memberID = @memID
END
GO
```

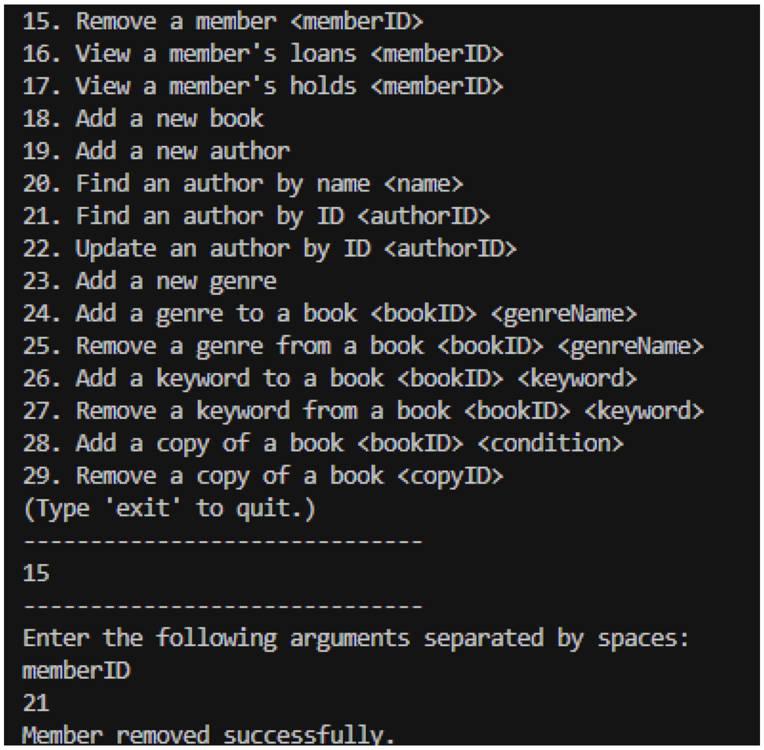
e. **Screenshots**:

Adding a new member to the library database



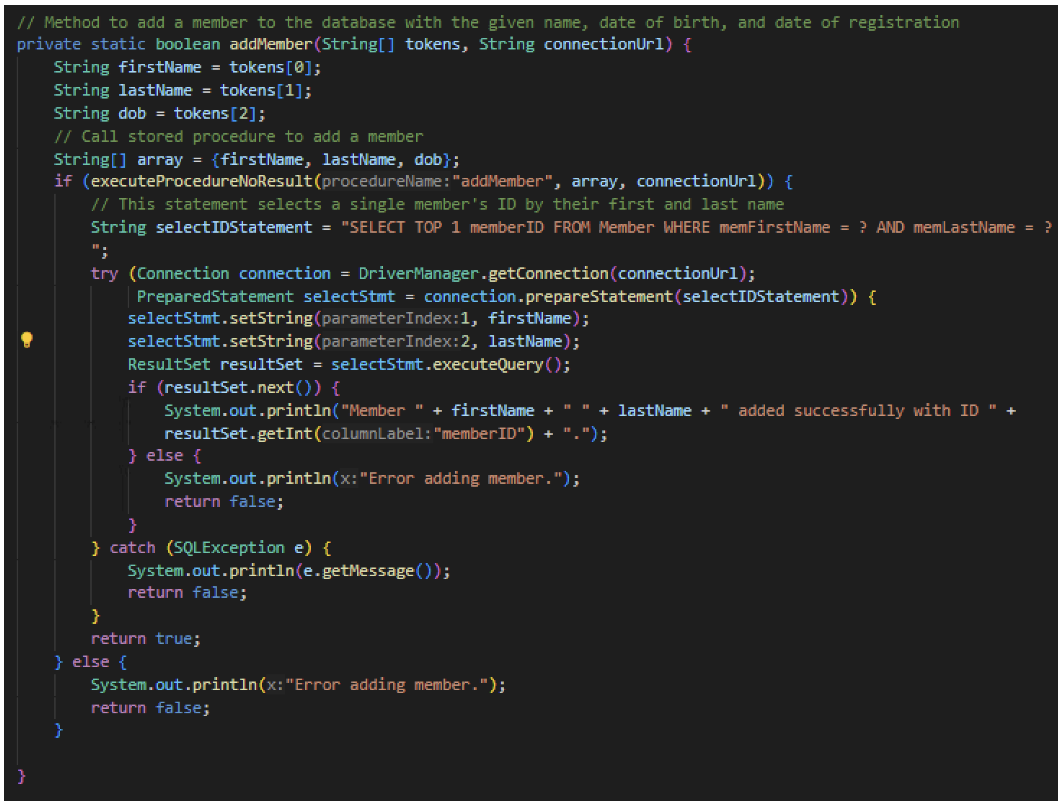
```
14. Add a new member <firstName> <lastName> <date of birth
>
15. Remove a member <memberID>
16. View a member's loans <memberID>
17. View a member's holds <memberID>
18. Add a new book
19. Add a new author
20. Find an author by name <name>
21. Find an author by ID <authorID>
22. Update an author by ID <authorID>
23. Add a new genre
24. Add a genre to a book <bookID> <genreName>
25. Remove a genre from a book <bookID> <genreName>
26. Add a keyword to a book <bookID> <keyword>
27. Remove a keyword from a book <bookID> <keyword>
28. Add a copy of a book <bookID> <condition>
29. Remove a copy of a book <copyID>
(Type 'exit' to quit.)
------------------------------
14
------------------------------
Enter the following arguments separated by spaces:
firstName lastName dob
New Member 2004-02-17
Member added successfully with ID: 21
```

Removing an existing member from the library database



```
15. Remove a member <memberID>
16. View a member's loans <memberID>
17. View a member's holds <memberID>
18. Add a new book
19. Add a new author
20. Find an author by name <name>
21. Find an author by ID <authorID>
22. Update an author by ID <authorID>
23. Add a new genre
24. Add a genre to a book <bookID> <genreName>
25. Remove a genre from a book <bookID> <genreName>
26. Add a keyword to a book <bookID> <keyword>
27. Remove a keyword from a book <bookID> <keyword>
28. Add a copy of a book <bookID> <condition>
29. Remove a copy of a book <copyID>
(Type 'exit' to quit.)
------------------------------
15
------------------------------
Enter the following arguments separated by spaces:
memberID
21
Member removed successfully.
```

f. **Java Implementation:**



```java
// Method to add a member to the database with the given name, date of birth, and date of registration
private static boolean addMember(String[] tokens, String connectionUrl) {
    String firstName = tokens[0];
    String lastName = tokens[1];
    String dob = tokens[2];
    // Call stored procedure to add a member
    String[] array = {firstName, lastName, dob};
    if (executeProcedureNoResult(procedureName:"addMember", array, connectionUrl)) {
        // This statement selects a single member's ID by their first and last name
        String selectIDStatement = "SELECT TOP 1 memberID FROM Member WHERE memFirstName = ? AND memLastName = ?
        ";
        try (Connection connection = DriverManager.getConnection(connectionUrl);
             PreparedStatement selectStmt = connection.prepareStatement(selectIDStatement)) {
            selectStmt.setString(parameterIndex:1, firstName);
            selectStmt.setString(parameterIndex:2, lastName);
            ResultSet resultSet = selectStmt.executeQuery();
            if (resultSet.next()) {
                System.out.println("Member " + firstName + " " + lastName + " added successfully with ID " +
                resultSet.getInt(columnLabel:"memberID") + ".");
            } else {
                System.out.println(x:"Error adding member.");
                return false;
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            return false;
        }
        return true;
    } else {
        System.out.println(x:"Error adding member.");
        return false;
    }

}
```

4.

a. **Title:** Looking up member information from their name or member ID

b. **Description:**
   i. To find a member's ID (e.g. for removal), a library employee or patron needs to be able to search for their ID based on their name
   ii. This also displays all of the member's other information (name, date of birth, registration date, etc.)
   iii. Similarly, the employee needs to be able to find this information from a member's ID

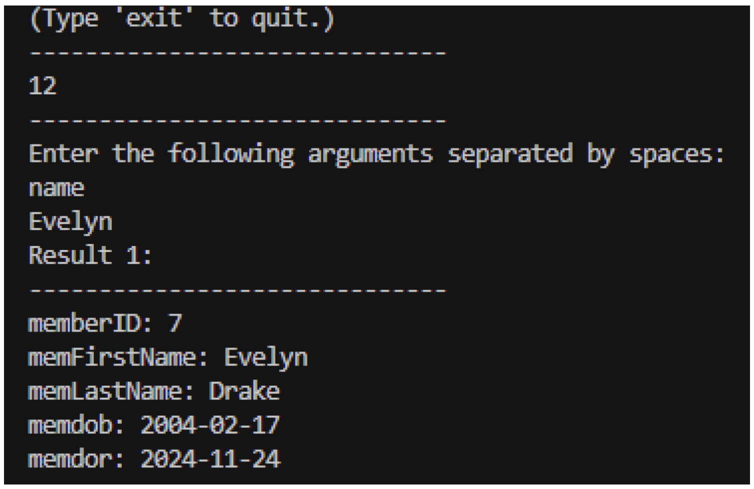
c. **User Requirements:**
   i. The user must connect to the database from an account with library employee or curator permissions
   ii. In the software, the user must select the option to find a member by name or by ID
   iii. They must enter the patron's name or their member ID

d. **SQL Queries:**

```sql
CREATE OR ALTER PROCEDURE findMemberByName @name varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM member
    WHERE memFirstName like '%'+@name+'%' OR memLastName like '%'+@name+'%'
END
GO

CREATE OR ALTER PROCEDURE findMemberByID @memID varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM member
    WHERE memberID = @memID;
END
GO
```

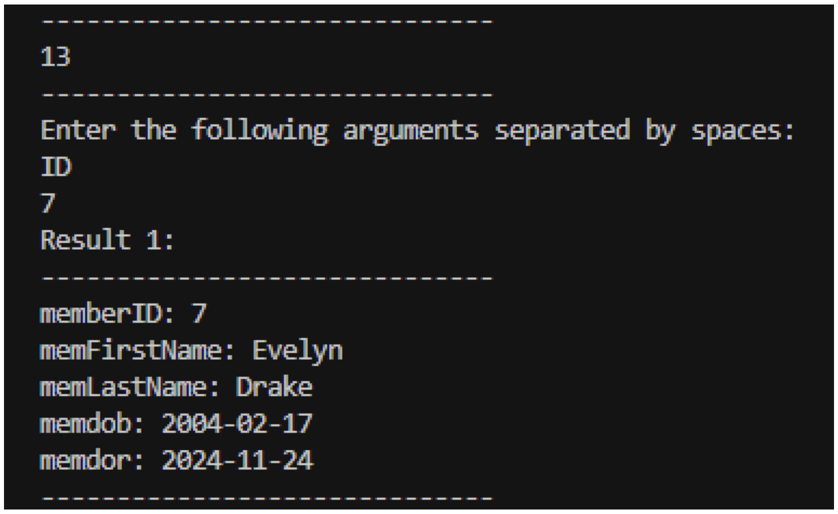
e. **Screenshots**:

Finding a member's information by name



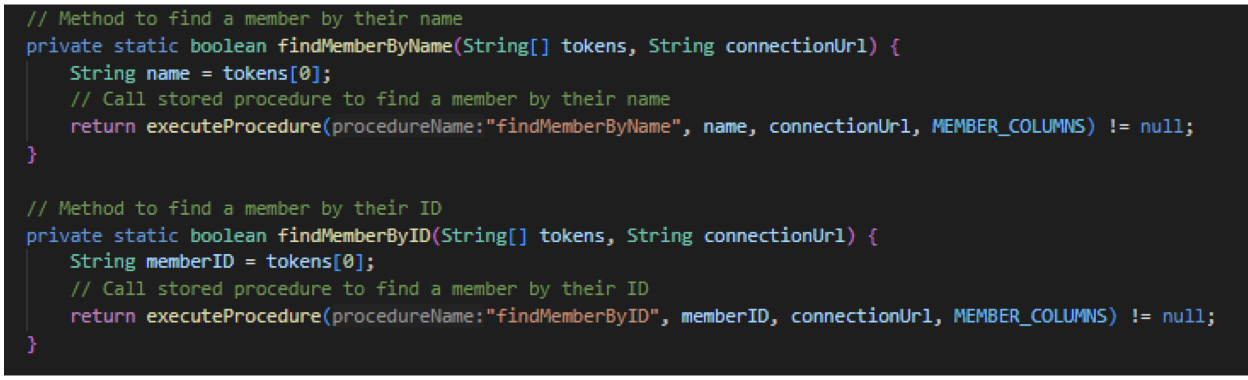
```
(Type 'exit' to quit.)
------------------------------
12
------------------------------
Enter the following arguments separated by spaces:
name
Evelyn
Result 1:
------------------------------
memberID: 7
memFirstName: Evelyn
memLastName: Drake
memdob: 2004-02-17
memdor: 2024-11-24
```

Finding a member's information by name



```
------------------------------
13
------------------------------
Enter the following arguments separated by spaces:
ID
7
Result 1:
------------------------------
memberID: 7
memFirstName: Evelyn
memLastName: Drake
memdob: 2004-02-17
memdor: 2024-11-24
------------------------------
```

f. **Java Implementation:**



```java
// Method to find a member by their name
private static boolean findMemberByName(String[] tokens, String connectionUrl) {
    String name = tokens[0];
    // Call stored procedure to find a member by their name
    return executeProcedure(procedureName:"findMemberByName", name, connectionUrl, MEMBER_COLUMNS) != null;
}

// Method to find a member by their ID
private static boolean findMemberByID(String[] tokens, String connectionUrl) {
    String memberID = tokens[0];
    // Call stored procedure to find a member by their ID
    return executeProcedure(procedureName:"findMemberByID", memberID, connectionUrl, MEMBER_COLUMNS) != null;
}
```

5.

a. **Title:** Putting a book on hold or checking out a book for a specific member

b. **Description:**

   i. When a member wants to check out a book or put a book on hold, they must bring it to the library employee, who can use the software to add the book to their account

   ii. The employee can search for the book by title, author, keywords, ISBN, or genres to find its ID, which will automatically find an available copy of the book for the library member

   iii. Otherwise, the employee can directly supply a specific copy of a book by its copy ID

   iv. Then, the employee can check out the book or put it on hold for the member using this information

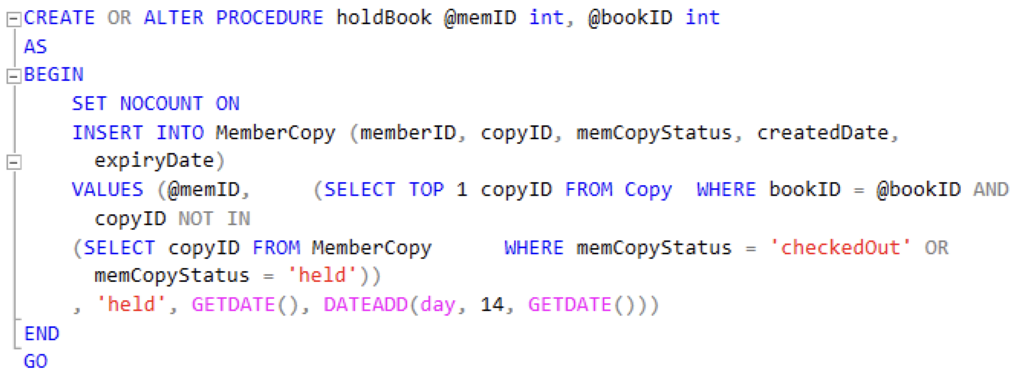
c. **User Requirements:**

   i. The user must connect to the database from an account with library employee or curator permissions

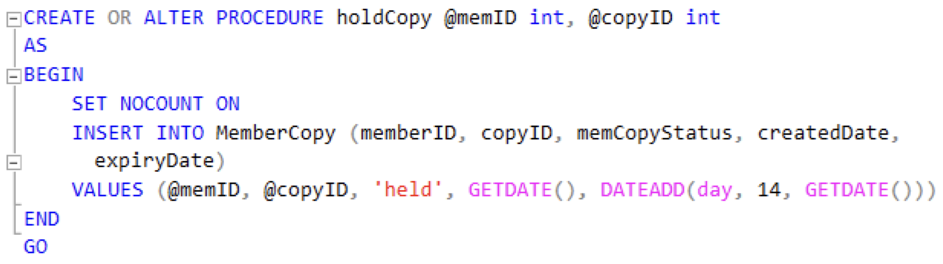
   ii. In the software, the user must select the option to check out a book/copy or put a book/copy on hold for a member

   iii. They must enter the book/copy ID and the ID of the member

d. **SQL Queries:**



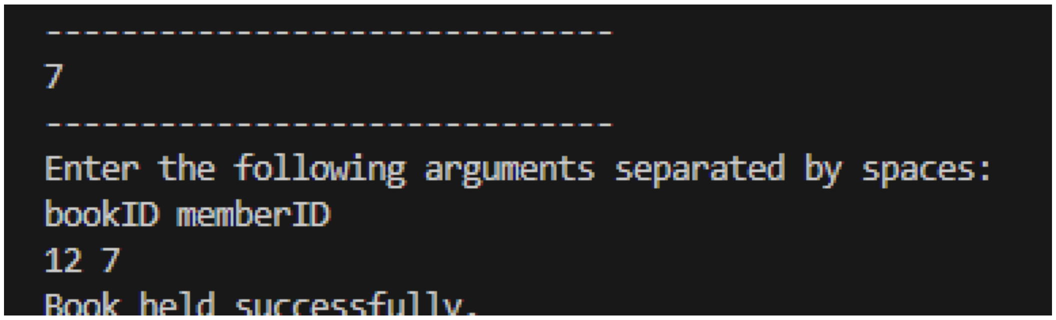
```
CREATE OR ALTER PROCEDURE holdBook @memID int, @bookID int
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO MemberCopy (memberID, copyID, memCopyStatus, createdDate,
      expiryDate)
    VALUES (@memID,     (SELECT TOP 1 copyID FROM Copy  WHERE bookID = @bookID AND
      copyID NOT IN
    (SELECT copyID FROM MemberCopy      WHERE memCopyStatus = 'checkedOut' OR
      memCopyStatus = 'held'))
    , 'held', GETDATE(), DATEADD(day, 14, GETDATE()))
END
GO
```



```
CREATE OR ALTER PROCEDURE holdCopy @memID int, @copyID int
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO MemberCopy (memberID, copyID, memCopyStatus, createdDate,
      expiryDate)
    VALUES (@memID, @copyID, 'held', GETDATE(), DATEADD(day, 14, GETDATE()))
END
GO
```

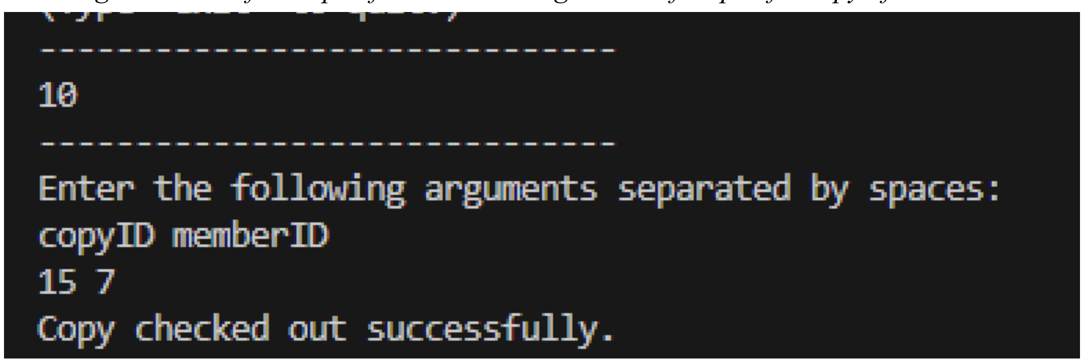
e. **Screenshots**:

Holding a book for a specific member by the book's ID, which automatically finds a suitable copy



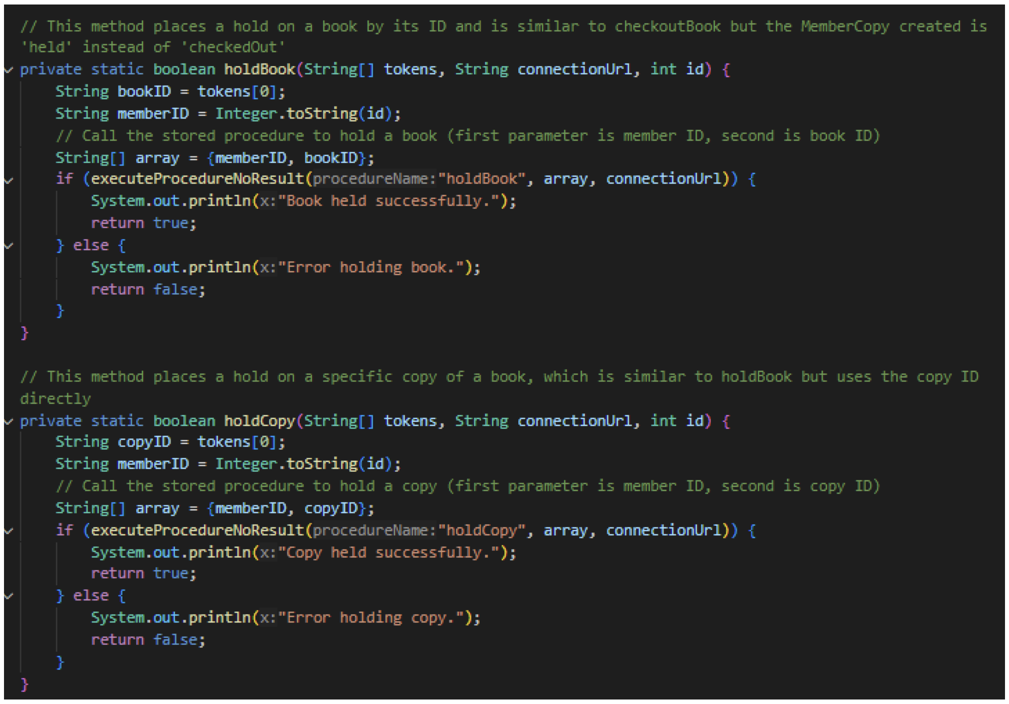
```
--------------------------------
7
--------------------------------
Enter the following arguments separated by spaces:
bookID memberID
12 7
Book held successfully.
```

Checking out a book for a specific number using the ID of a specific copy of the book



```
--------------------------------
10
--------------------------------
Enter the following arguments separated by spaces:
copyID memberID
15 7
Copy checked out successfully.
```

f. **Java Implementation:**



```
// This method places a hold on a book by its ID and is similar to checkoutBook but the MemberCopy created is
'held' instead of 'checkedOut'
private static boolean holdBook(String[] tokens, String connectionUrl, int id) {
    String bookID = tokens[0];
    String memberID = Integer.toString(id);
    // Call the stored procedure to hold a book (first parameter is member ID, second is book ID)
    String[] array = {memberID, bookID};
    if (executeProcedureNoResult(procedureName:"holdBook", array, connectionUrl)) {
        System.out.println(x:"Book held successfully.");
        return true;
    } else {
        System.out.println(x:"Error holding book.");
        return false;
    }
}

// This method places a hold on a specific copy of a book, which is similar to holdBook but uses the copy ID
directly
private static boolean holdCopy(String[] tokens, String connectionUrl, int id) {
    String copyID = tokens[0];
    String memberID = Integer.toString(id);
    // Call the stored procedure to hold a copy (first parameter is member ID, second is copy ID)
    String[] array = {memberID, copyID};
    if (executeProcedureNoResult(procedureName:"holdCopy", array, connectionUrl)) {
        System.out.println(x:"Copy held successfully.");
        return true;
    } else {
        System.out.println(x:"Error holding copy.");
        return false;
    }
}
```

6.

a. **Title:** Adding a new genre to a book and removing a genre from a book

b. **Description:**

i. Because books can belong to several different genres, library curators need to be able to easily add and remove genres from books

ii. If the genres do not already exist in the database, they can be created when necessary

c. **User Requirements:**

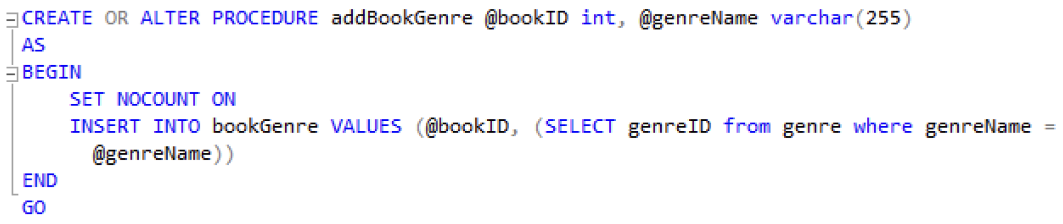
i. The user must connect to the database from an account with library curator permissions

ii. In the software, the user must provide the necessary information

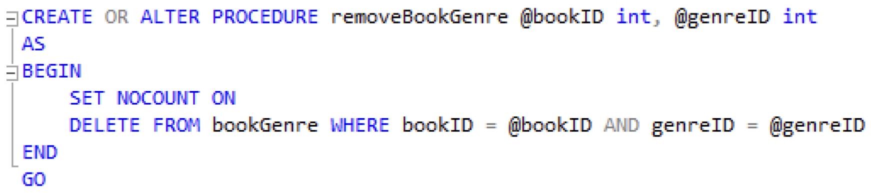
iii. When adding a genre to a book, the book’s ID and genre name must be entered, and if a genre with that name does not already exist, they must provide a description for it as well

iv. When removing a genre from a book, the book’s ID and the name of the genre are required

d. **SQL Queries:**



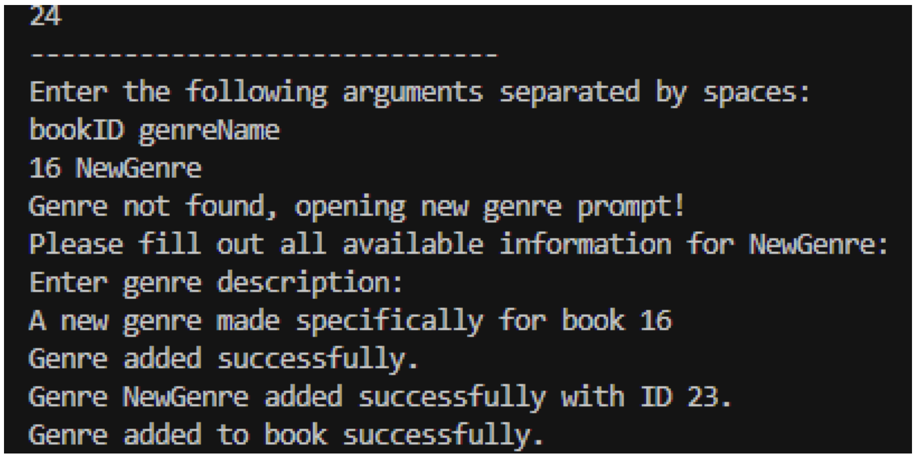
```sql
CREATE OR ALTER PROCEDURE addBookGenre @bookID int, @genreName varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO bookGenre VALUES (@bookID, (SELECT genreID from genre where genreName =
      @genreName))
END
GO
```



```sql
CREATE OR ALTER PROCEDURE removeBookGenre @bookID int, @genreID int
AS
BEGIN
    SET NOCOUNT ON
    DELETE FROM bookGenre WHERE bookID = @bookID AND genreID = @genreID
END
GO
```

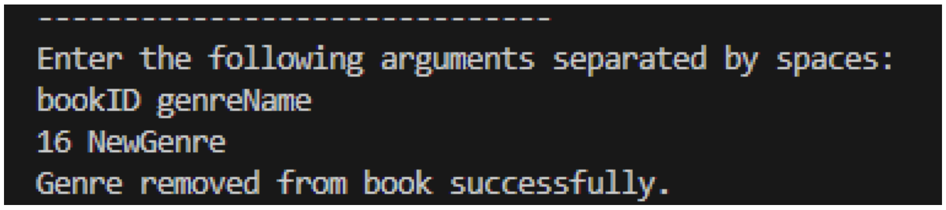
e. **Screenshots**:

Adding a new genre which does not exist in the database to a book



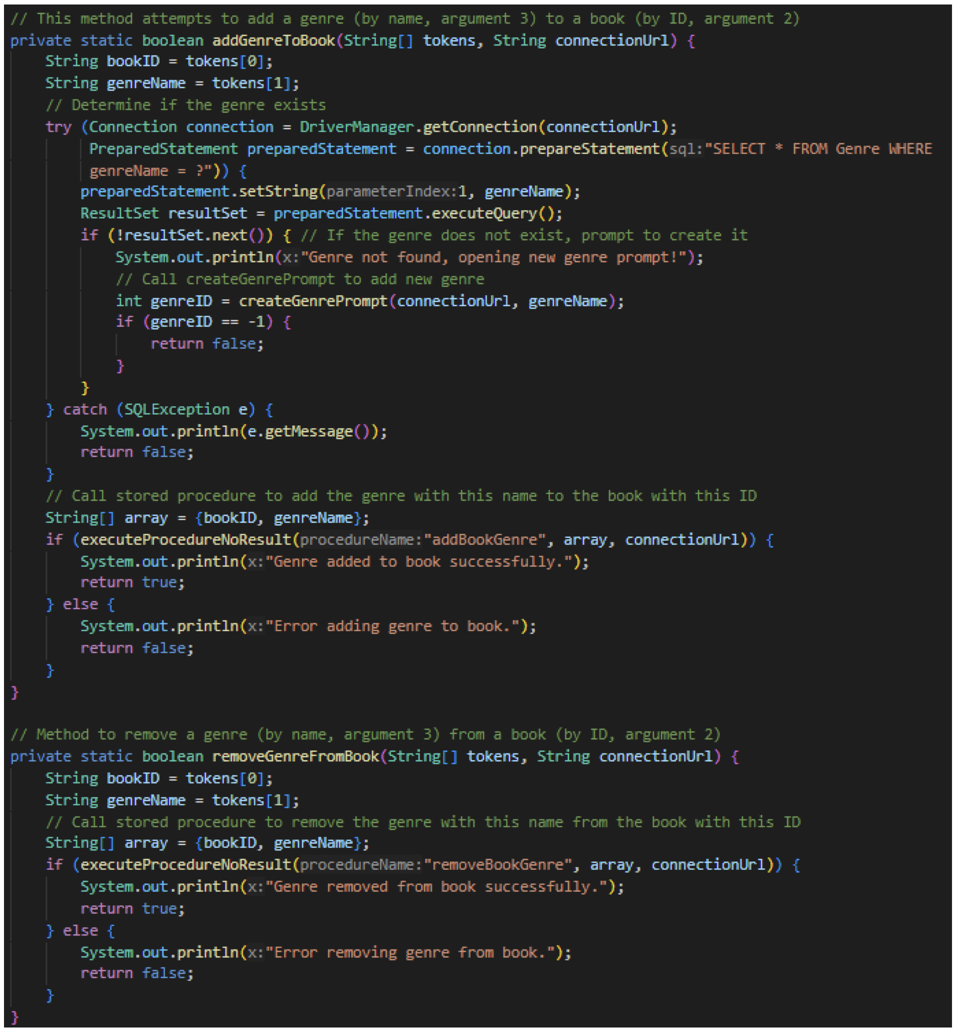
```
24
------------------------------
Enter the following arguments separated by spaces:
bookID genreName
16 NewGenre
Genre not found, opening new genre prompt!
Please fill out all available information for NewGenre:
Enter genre description:
A new genre made specifically for book 16
Genre added successfully.
Genre NewGenre added successfully with ID 23.
Genre added to book successfully.
```

Removing this genre from the book



```
------------------------------
Enter the following arguments separated by spaces:
bookID genreName
16 NewGenre
Genre removed from book successfully.
```

f. **Java Implementation:**



```java
// This method attempts to add a genre (by name, argument 3) to a book (by ID, argument 2)
private static boolean addGenreToBook(String[] tokens, String connectionUrl) {
    String bookID = tokens[0];
    String genreName = tokens[1];
    // Determine if the genre exists
    try (Connection connection = DriverManager.getConnection(connectionUrl);
         PreparedStatement preparedStatement = connection.prepareStatement(sql:"SELECT * FROM Genre WHERE
         genreName = ?")) {
        preparedStatement.setString(parameterIndex:1, genreName);
        ResultSet resultSet = preparedStatement.executeQuery();
        if (!resultSet.next()) { // If the genre does not exist, prompt to create it
            System.out.println(x:"Genre not found, opening new genre prompt!");
            // Call createGenrePrompt to add new genre
            int genreID = createGenrePrompt(connectionUrl, genreName);
            if (genreID == -1) {
                return false;
            }
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
    // Call stored procedure to add the genre with this name to the book with this ID
    String[] array = {bookID, genreName};
    if (executeProcedureNoResult(procedureName:"addBookGenre", array, connectionUrl)) {
        System.out.println(x:"Genre added to book successfully.");
        return true;
    } else {
        System.out.println(x:"Error adding genre to book.");
        return false;
    }
}

// Method to remove a genre (by name, argument 3) from a book (by ID, argument 2)
private static boolean removeGenreFromBook(String[] tokens, String connectionUrl) {
    String bookID = tokens[0];
    String genreName = tokens[1];
    // Call stored procedure to remove the genre with this name from the book with this ID
    String[] array = {bookID, genreName};
    if (executeProcedureNoResult(procedureName:"removeBookGenre", array, connectionUrl)) {
        System.out.println(x:"Genre removed from book successfully.");
        return true;
    } else {
        System.out.println(x:"Error removing genre from book.");
        return false;
    }
}
```

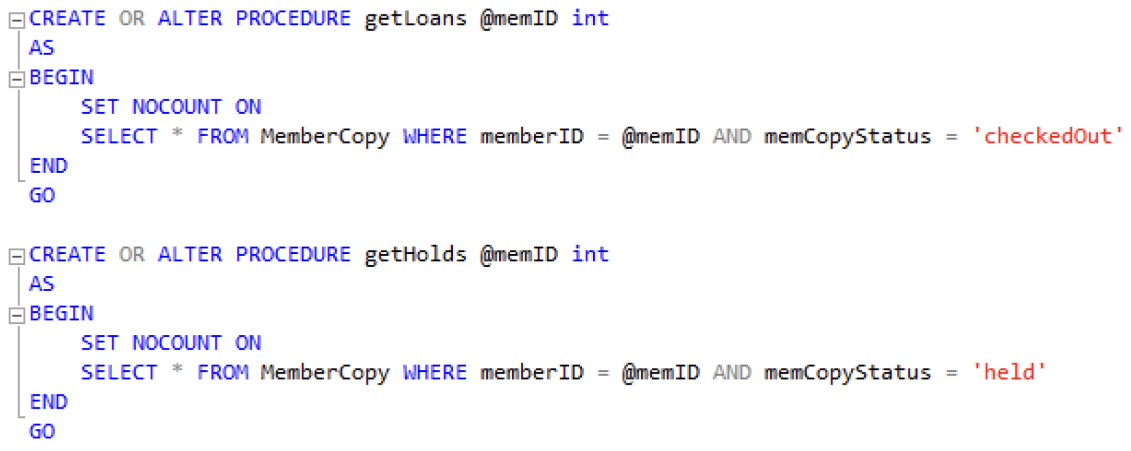
7.

a. **Title:** Viewing a member's holds and checked out books

b. **Description:**
   i. A library employee/curator can view all books held or checked out by a specific member
   ii. This provides them with information about the copies they have checked out/held, including their issue and expiration dates
   iii. When this book is overdue, a warning will be displayed to the user

c. **User Requirements:**
   i. The user must connect to the database from an account with library employee or curator permissions
   ii. In the software, the user must provide the necessary information (the member's ID)
   iii. The user must know the ID of the member in question, which can be looked up as specified in use case #4

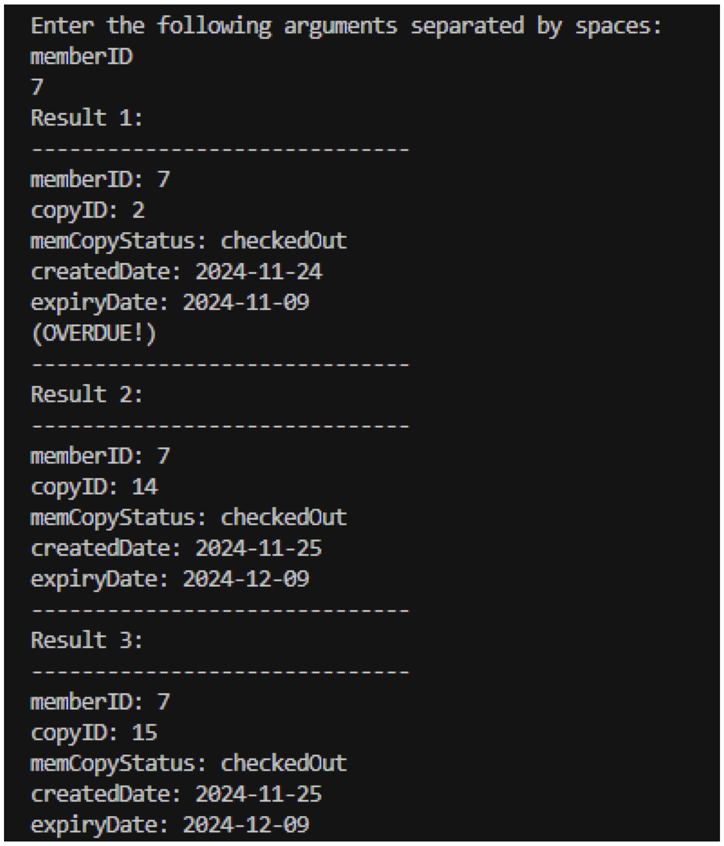
d. **SQL Queries:**



```
CREATE OR ALTER PROCEDURE getLoans @memID int
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM MemberCopy WHERE memberID = @memID AND memCopyStatus = 'checkedOut'
END
GO

CREATE OR ALTER PROCEDURE getHolds @memID int
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM MemberCopy WHERE memberID = @memID AND memCopyStatus = 'held'
END
GO
```

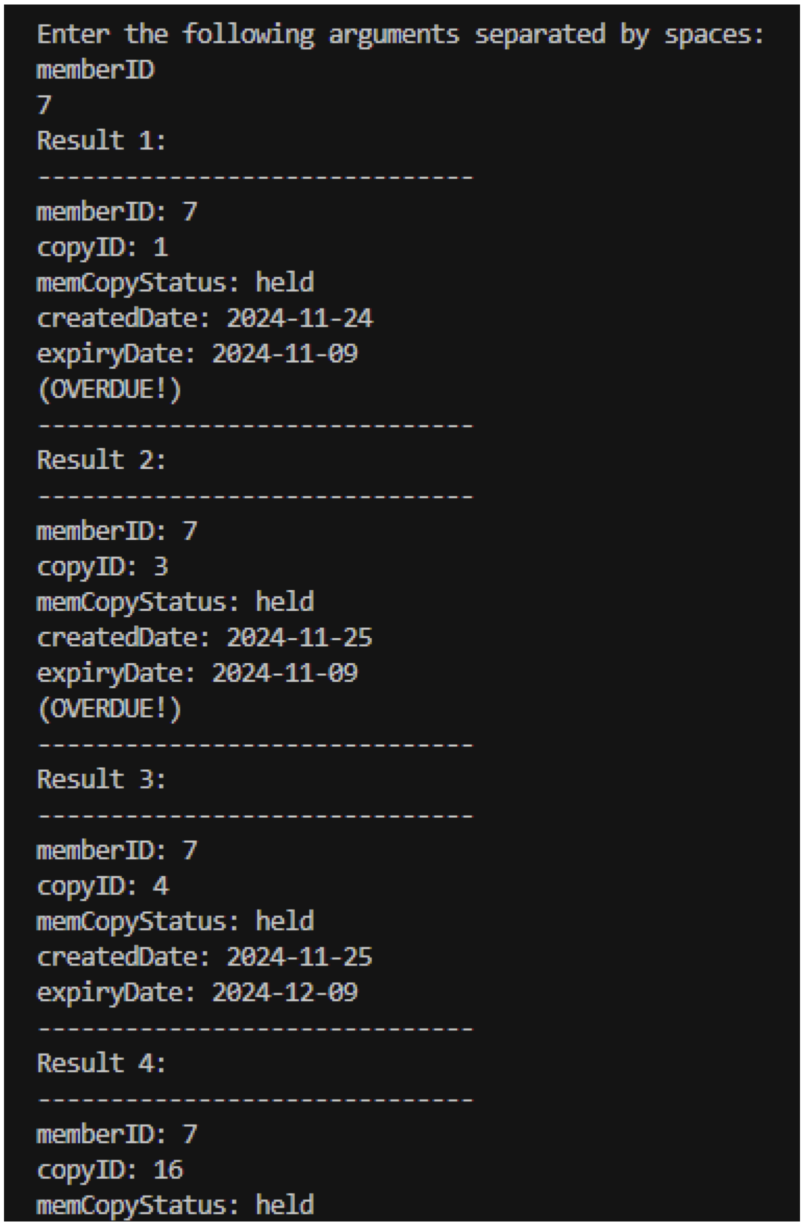
e. **Screenshots**:

Viewing a member's loans (checked out books)



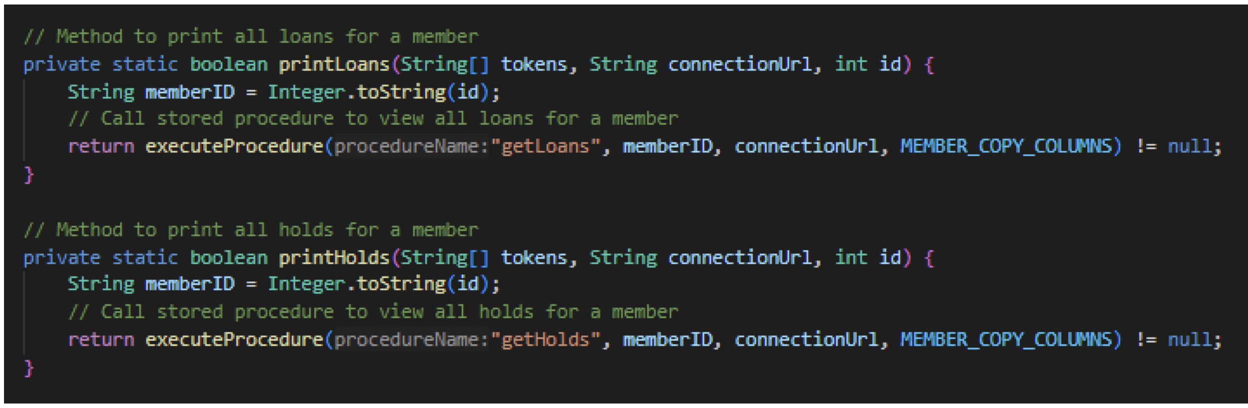
```
Enter the following arguments separated by spaces:
memberID
7
Result 1:
------------------------------
memberID: 7
copyID: 2
memCopyStatus: checkedOut
createdDate: 2024-11-24
expiryDate: 2024-11-09
(OVERDUE!)
------------------------------
Result 2:
------------------------------
memberID: 7
copyID: 14
memCopyStatus: checkedOut
createdDate: 2024-11-25
expiryDate: 2024-12-09
------------------------------
Result 3:
------------------------------
memberID: 7
copyID: 15
memCopyStatus: checkedOut
createdDate: 2024-11-25
expiryDate: 2024-12-09
```

Viewing a member's held books



```
Enter the following arguments separated by spaces:
memberID
7
Result 1:
------------------------------
memberID: 7
copyID: 1
memCopyStatus: held
createdDate: 2024-11-24
expiryDate: 2024-11-09
(OVERDUE!)
------------------------------
Result 2:
------------------------------
memberID: 7
copyID: 3
memCopyStatus: held
createdDate: 2024-11-25
expiryDate: 2024-11-09
(OVERDUE!)
------------------------------
Result 3:
------------------------------
memberID: 7
copyID: 4
memCopyStatus: held
createdDate: 2024-11-25
expiryDate: 2024-12-09
------------------------------
Result 4:
------------------------------
memberID: 7
copyID: 16
memCopyStatus: held
```

f. **Java Implementation:**



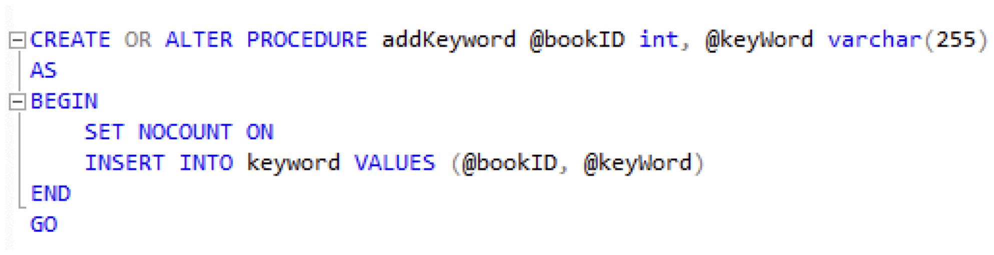
```java
// Method to print all loans for a member
private static boolean printLoans(String[] tokens, String connectionUrl, int id) {
    String memberID = Integer.toString(id);
    // Call stored procedure to view all loans for a member
    return executeProcedure(procedureName:"getLoans", memberID, connectionUrl, MEMBER_COPY_COLUMNS) != null;
}

// Method to print all holds for a member
private static boolean printHolds(String[] tokens, String connectionUrl, int id) {
    String memberID = Integer.toString(id);
    // Call stored procedure to view all holds for a member
    return executeProcedure(procedureName:"getHolds", memberID, connectionUrl, MEMBER_COPY_COLUMNS) != null;
}
```

Individual

William Cankar's Use Cases

W1.

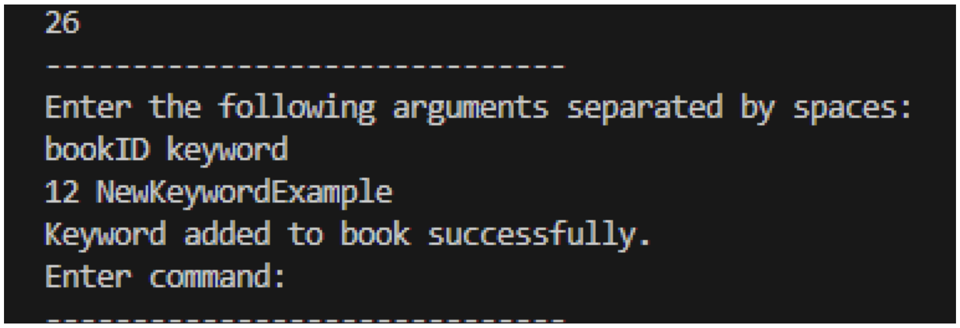
1.
   a. **Title:** Adding a keyword to book
   b. **Description:**
      i. A library employee/curator can add a new book they received
      ii. They then will be able to add a keyword that matches the book's content
      iii. Then the new book should be returned when they search for this keyword
   c. **User Requirements:**
      i. The user must connect to the database from an account with library employee or curator permissions
      ii. They can use a book's id with keyword to save it alongside a corresponding varchar representing a key characteristic
      iii. Users will be able to search by this keyword and be given the matching book
   d. **SQL Queries:**



```
CREATE OR ALTER PROCEDURE addKeyword @bookID int, @keyWord varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO keyword VALUES (@bookID, @keyWord)
END
GO
```

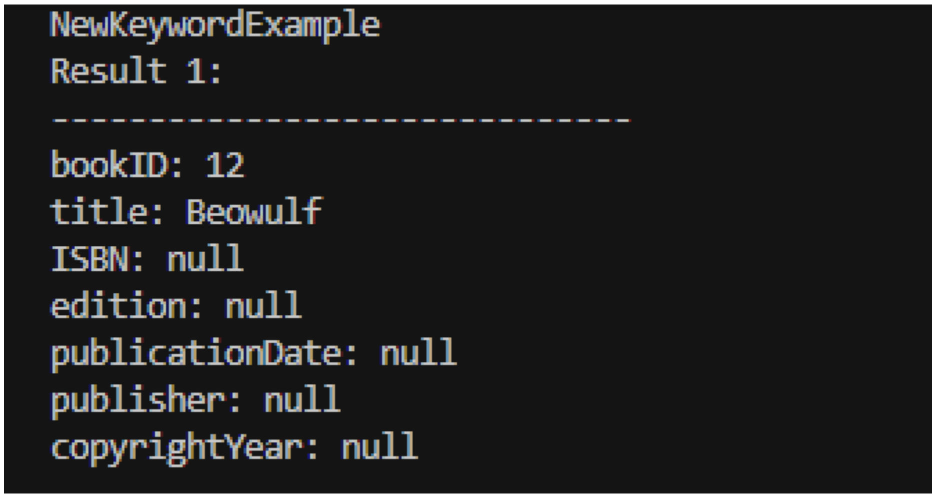
   e. **Screenshots**:

Adding a new keyword to a book



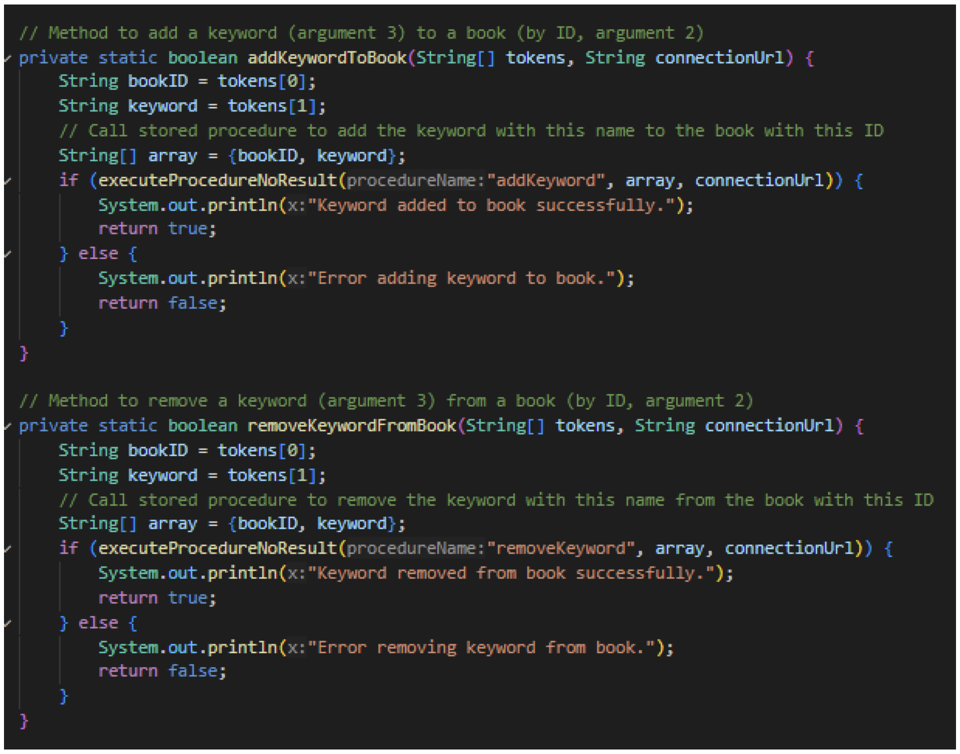
```
26
------------------------------
Enter the following arguments separated by spaces:
bookID keyword
12 NewKeywordExample
Keyword added to book successfully.
Enter command:
------------------------------
```

Searching for books with this keyword



```
NewKeywordExample
Result 1:
------------------------------
bookID: 12
title: Beowulf
ISBN: null
edition: null
publicationDate: null
publisher: null
copyrightYear: null
```

f. **Java Implementation:**



```java
// Method to add a keyword (argument 3) to a book (by ID, argument 2)
private static boolean addKeywordToBook(String[] tokens, String connectionUrl) {
    String bookID = tokens[0];
    String keyword = tokens[1];
    // Call stored procedure to add the keyword with this name to the book with this ID
    String[] array = {bookID, keyword};
    if (executeProcedureNoResult(procedureName:"addKeyword", array, connectionUrl)) {
        System.out.println(x:"Keyword added to book successfully.");
        return true;
    } else {
        System.out.println(x:"Error adding keyword to book.");
        return false;
    }
}

// Method to remove a keyword (argument 3) from a book (by ID, argument 2)
private static boolean removeKeywordFromBook(String[] tokens, String connectionUrl) {
    String bookID = tokens[0];
    String keyword = tokens[1];
    // Call stored procedure to remove the keyword with this name from the book with this ID
    String[] array = {bookID, keyword};
    if (executeProcedureNoResult(procedureName:"removeKeyword", array, connectionUrl)) {
        System.out.println(x:"Keyword removed from book successfully.");
        return true;
    } else {
        System.out.println(x:"Error removing keyword from book.");
        return false;
    }
}
```

W2.

2.

a. **Title:** Library patrons should see which holds and loans they currently have

b. **Description:**

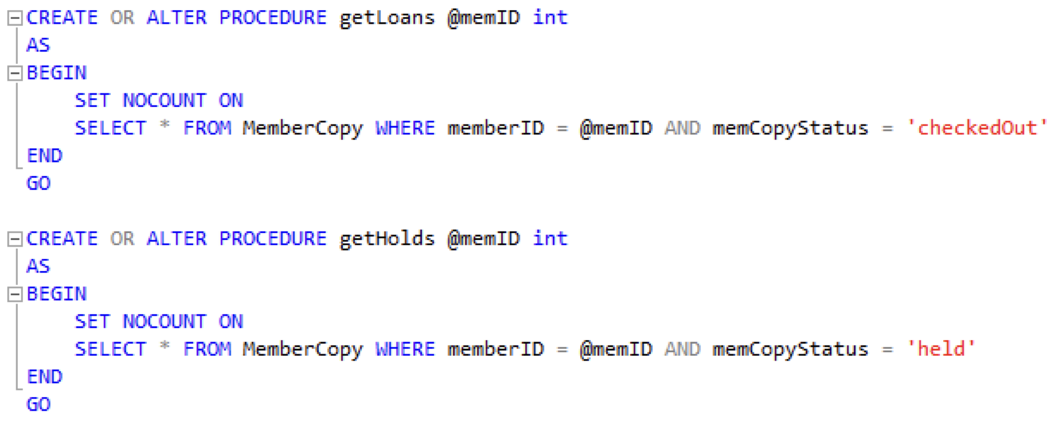
   i. A member of the library will be able to take out holds and loans on a particular copy

   ii. Then they can check their account to view the holds and loans they currently have made

    iii. They will also be warned if any of these holds/loans are overdue

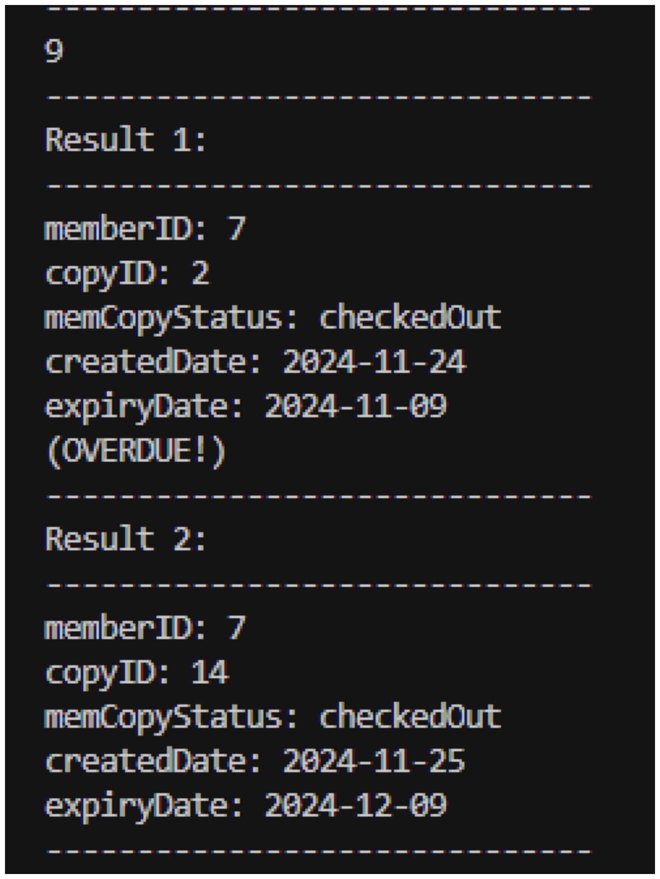
c. **User Requirements:**
    i. Log into an account with member permissions and give the corresponding login password
    ii. Check holds and loans by using the applicable command

d. **SQL Queries:** *(Same as an earlier use case, but these procedures are called from the front end using the currently logged in member's ID instead of an arbitrary employee-entered one)*



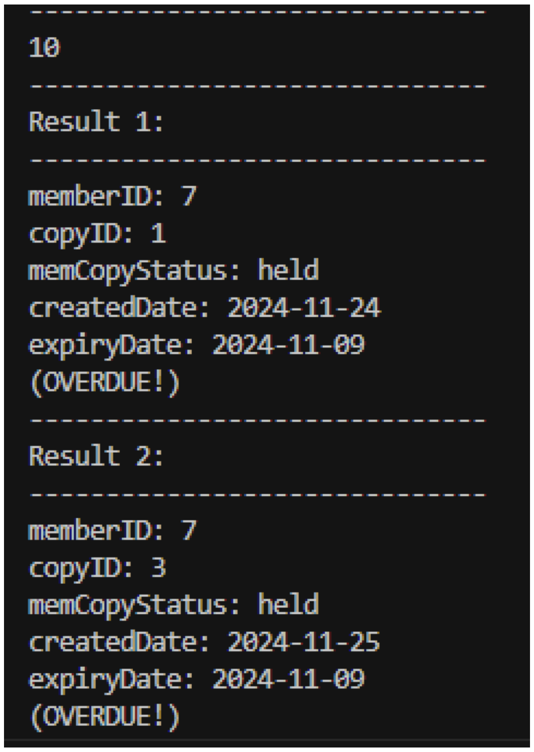
```sql
CREATE OR ALTER PROCEDURE getLoans @memID int
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM MemberCopy WHERE memberID = @memID AND memCopyStatus = 'checkedOut'
END
GO

CREATE OR ALTER PROCEDURE getHolds @memID int
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM MemberCopy WHERE memberID = @memID AND memCopyStatus = 'held'
END
GO
```

e. **Screenshots**:
Viewing loans when logged in as a member



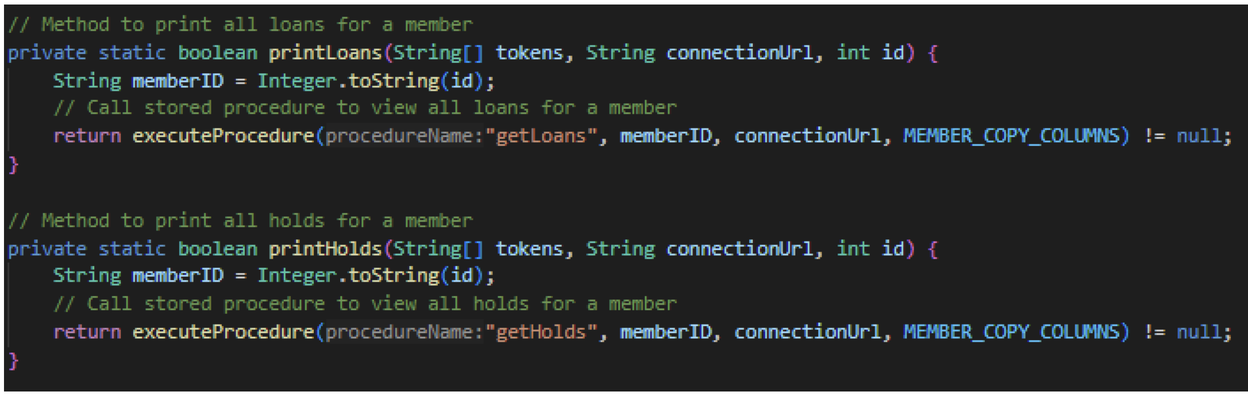
```
--------------------------------
9
--------------------------------
Result 1:
--------------------------------
memberID: 7
copyID: 2
memCopyStatus: checkedOut
createdDate: 2024-11-24
expiryDate: 2024-11-09
(OVERDUE!)
--------------------------------
Result 2:
--------------------------------
memberID: 7
copyID: 14
memCopyStatus: checkedOut
createdDate: 2024-11-25
expiryDate: 2024-12-09
--------------------------------
```

Viewing holds when logged in as a member



```
------------------------------
10
------------------------------
Result 1:
------------------------------
memberID: 7
copyID: 1
memCopyStatus: held
createdDate: 2024-11-24
expiryDate: 2024-11-09
(OVERDUE!)
------------------------------
Result 2:
------------------------------
memberID: 7
copyID: 3
memCopyStatus: held
createdDate: 2024-11-25
expiryDate: 2024-11-09
(OVERDUE!)
```

f. **Java Implementation:**



```java
// Method to print all loans for a member
private static boolean printLoans(String[] tokens, String connectionUrl, int id) {
    String memberID = Integer.toString(id);
    // Call stored procedure to view all loans for a member
    return executeProcedure(procedureName:"getLoans", memberID, connectionUrl, MEMBER_COPY_COLUMNS) != null;
}

// Method to print all holds for a member
private static boolean printHolds(String[] tokens, String connectionUrl, int id) {
    String memberID = Integer.toString(id);
    // Call stored procedure to view all holds for a member
    return executeProcedure(procedureName:"getHolds", memberID, connectionUrl, MEMBER_COPY_COLUMNS) != null;
}
```

Evelyn Drake's Use Cases

E1.

3.

a. **Title:** Looking up information about an author by the author's name or ID

b. **Description:**

i. A library curator can view information about a specific author

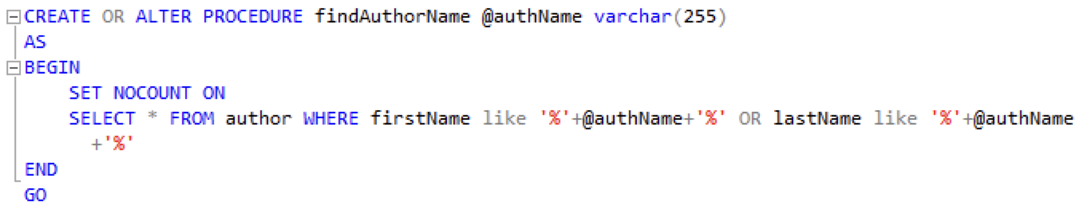
ii. If they know the author's ID, which is returned when doing things like looking up information about a book and its authors, they may want to find the author information associated with an ID

iii. If they need to know an author's ID to use elsewhere in the software, they can find this information by searching by the author's name

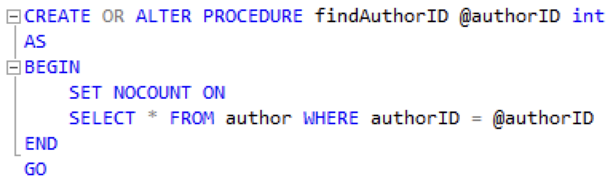
   iv. This provides them with all available information about the author (ID, first name, last name, date of birth, and status)

c. **User Requirements:**
   i. The user must connect to the database from an account with library curator permissions
   ii. In the software, the user will be prompted to either enter the author’s name or the author’s ID, and they must know at least one of these to find the author’s information

d. **SQL Queries:**



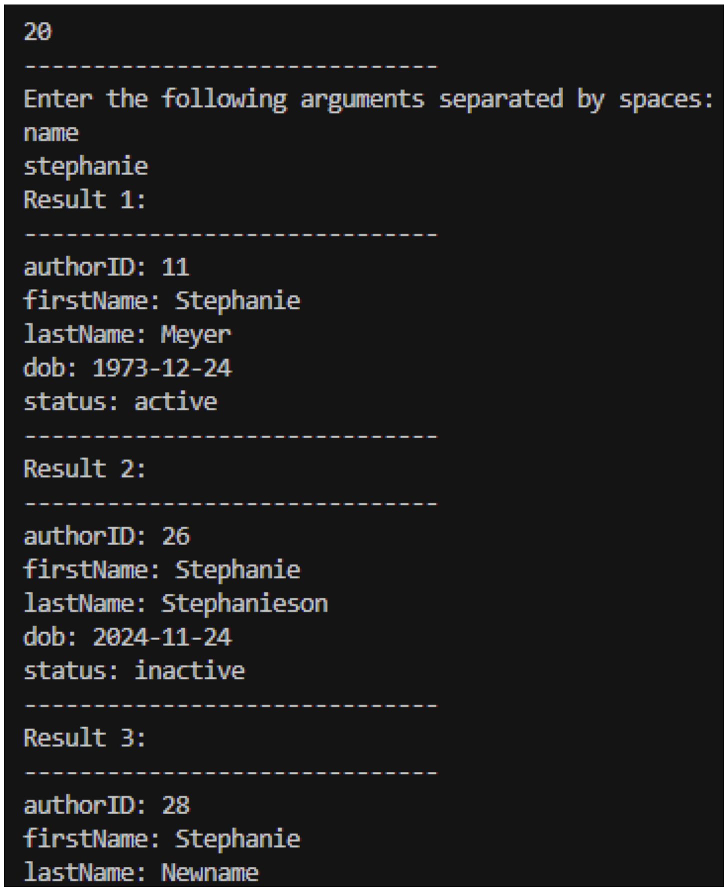
```
CREATE OR ALTER PROCEDURE findAuthorName @authName varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM author WHERE firstName like '%'+@authName+'%' OR lastName like '%'+@authName
      +'%'
END
GO
```



```
CREATE OR ALTER PROCEDURE findAuthorID @authorID int
AS
BEGIN
    SET NOCOUNT ON
    SELECT * FROM author WHERE authorID = @authorID
END
GO
```

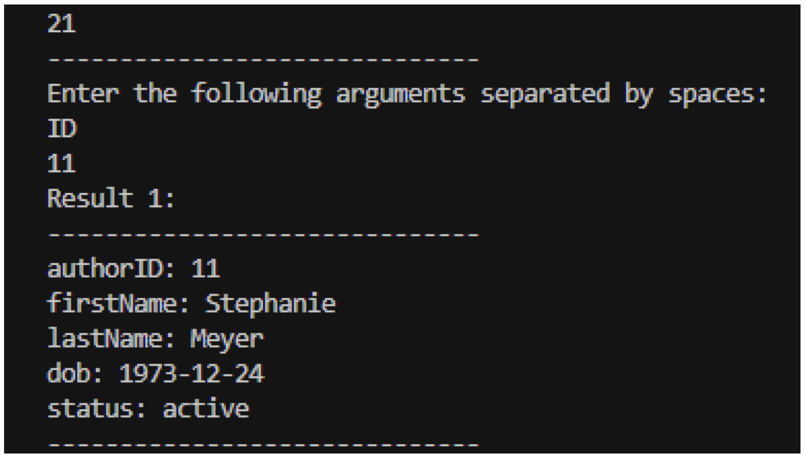
e. **Screenshots:**

Searching for an author's information by name



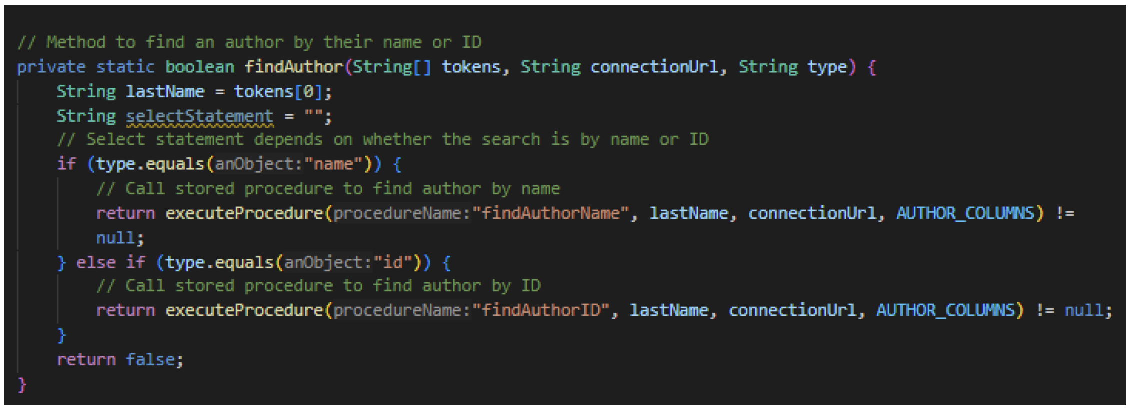
```
20
------------------------------
Enter the following arguments separated by spaces:
name
stephanie
Result 1:
------------------------------
authorID: 11
firstName: Stephanie
lastName: Meyer
dob: 1973-12-24
status: active
------------------------------
Result 2:
------------------------------
authorID: 26
firstName: Stephanie
lastName: Stephanieson
dob: 2024-11-24
status: inactive
------------------------------
Result 3:
------------------------------
authorID: 28
firstName: Stephanie
lastName: Newname
```

Searching for an author's information by ID



```
21
------------------------------
Enter the following arguments separated by spaces:
ID
11
Result 1:
------------------------------
authorID: 11
firstName: Stephanie
lastName: Meyer
dob: 1973-12-24
status: active
------------------------------
```

f. **Java Implementation:**



```java
// Method to find an author by their name or ID
private static boolean findAuthor(String[] tokens, String connectionUrl, String type) {
    String lastName = tokens[0];
    String selectStatement = "";
    // Select statement depends on whether the search is by name or ID
    if (type.equals(anObject:"name")) {
        // Call stored procedure to find author by name
        return executeProcedure(procedureName:"findAuthorName", lastName, connectionUrl, AUTHOR_COLUMNS) !=
        null;
    } else if (type.equals(anObject:"id")) {
        // Call stored procedure to find author by ID
        return executeProcedure(procedureName:"findAuthorID", lastName, connectionUrl, AUTHOR_COLUMNS) != null;
    }
    return false;
}
```

E2.

11.

a. **Title:** Adding a new author to the database

b. **Description:**

When a library gets a new book, a library curator will need to add its information to the database

Part of the book creation prompt asks the curator for a list of the book’s authors

If one of these authors is not recognized, an author creation prompt is opened to add the author to the database so that it can be added to the book

A library curator can also use this prompt on its own to add new authors to the database without creating any associated books

c.

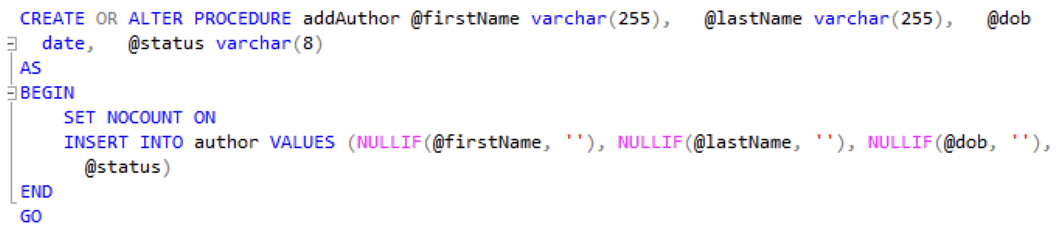
User Requirements:

The user must connect to the database from an account with library curator permissions

In the software, the user will be prompted for the author’s first name, last name, date of birth, and status (active, inactive, or unknown)

Some of this information can be skipped if unknown, but the user must enter at least the author’s last name

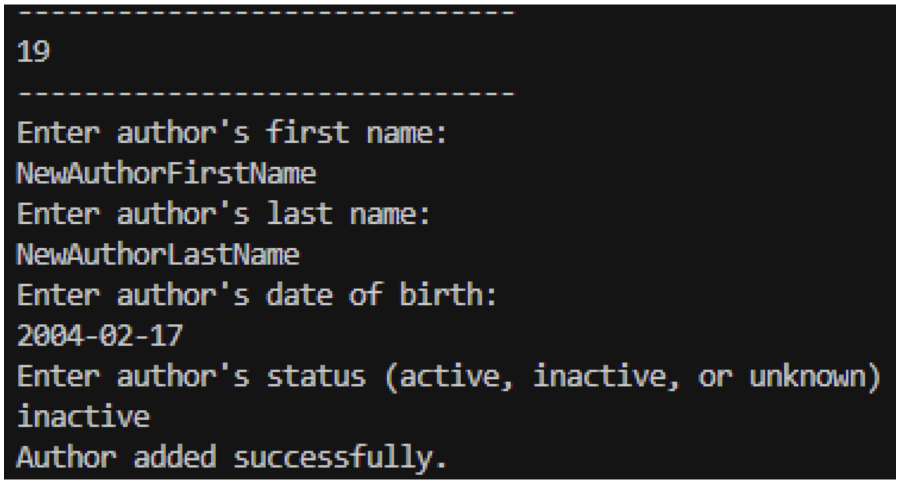
d. **SQL Queries:**



```sql
CREATE OR ALTER PROCEDURE addAuthor @firstName varchar(255),   @lastName varchar(255),   @dob
  date,   @status varchar(8)
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO author VALUES (NULLIF(@firstName, ''), NULLIF(@lastName, ''), NULLIF(@dob, ''),
      @status)
END
GO
```

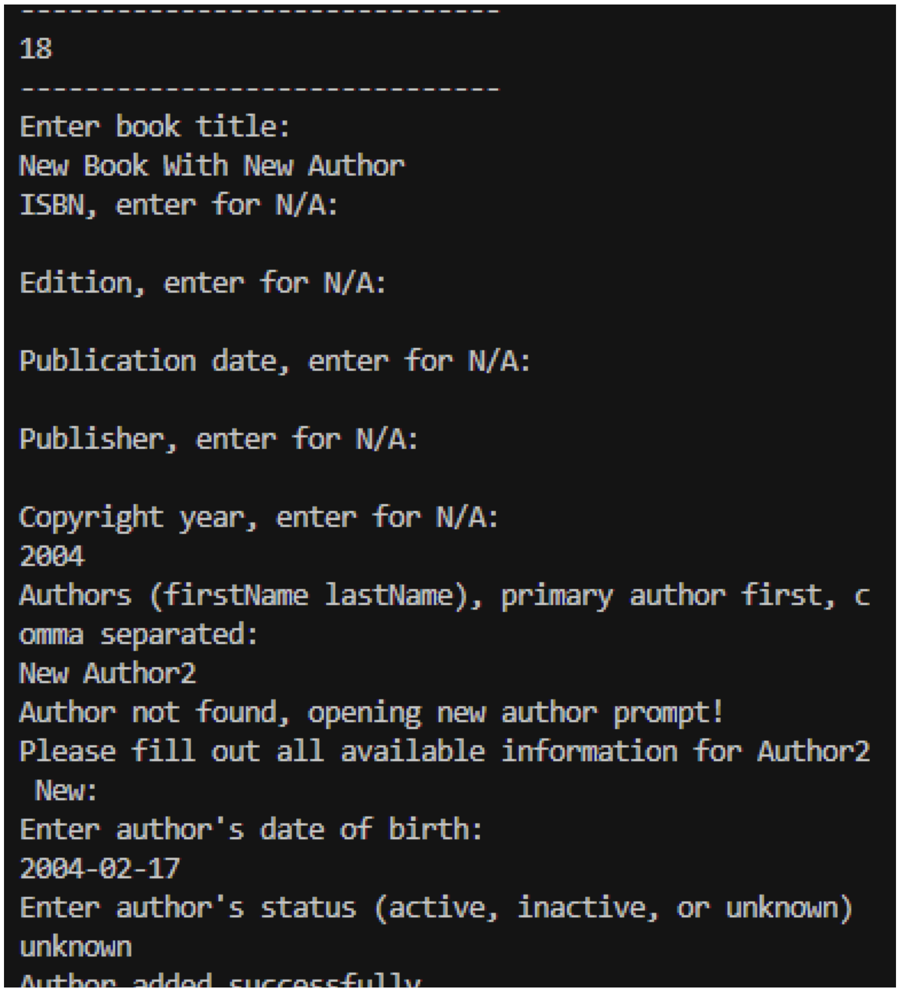
e. **Screenshots:**

Adding a new author with no associated books



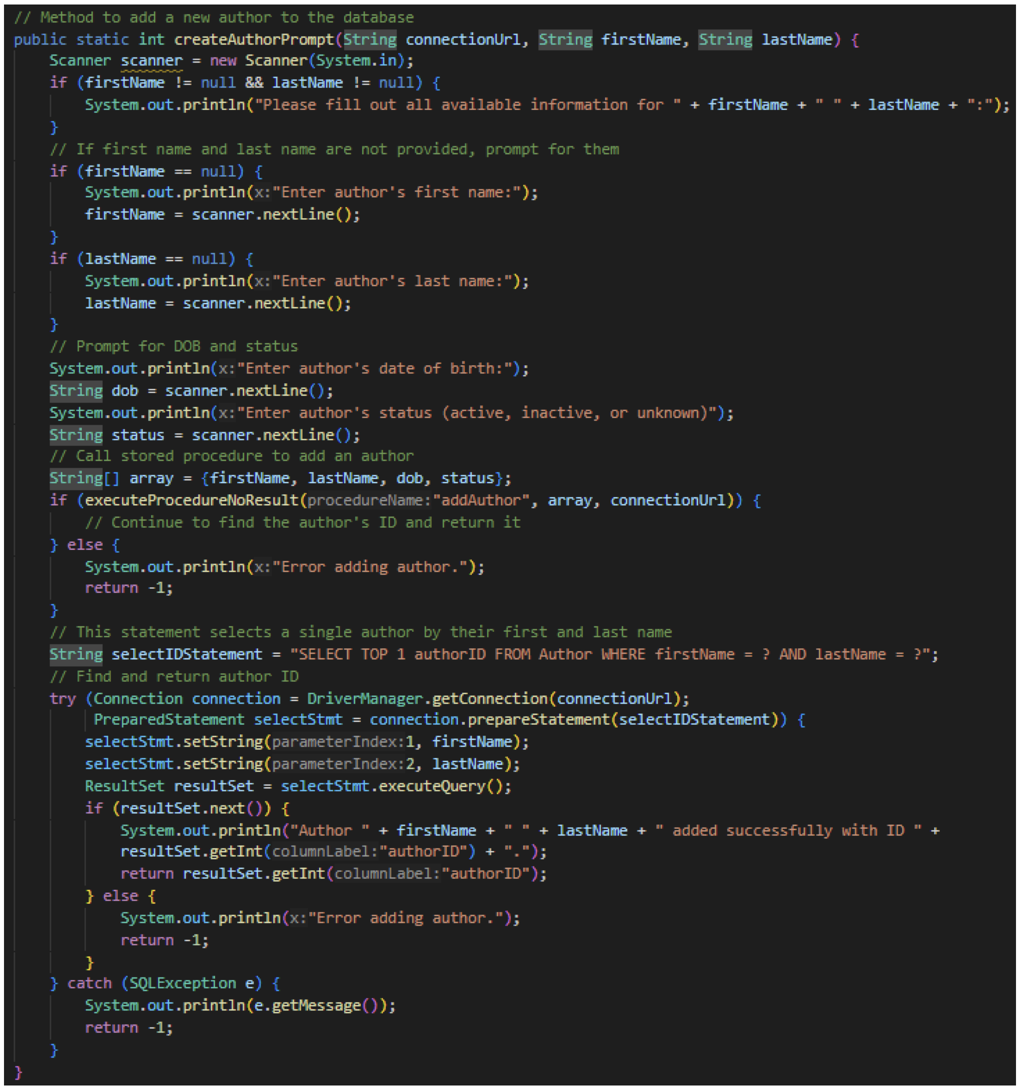
```
------------------------------
19
------------------------------
Enter author's first name:
NewAuthorFirstName
Enter author's last name:
NewAuthorLastName
Enter author's date of birth:
2004-02-17
Enter author's status (active, inactive, or unknown)
inactive
Author added successfully.
```

Adding a new book with an unknown author, which prompts the user for information about the author



```
------------------------------
18
------------------------------
Enter book title:
New Book With New Author
ISBN, enter for N/A:

Edition, enter for N/A:

Publication date, enter for N/A:

Publisher, enter for N/A:

Copyright year, enter for N/A:
2004
Authors (firstName lastName), primary author first, c
omma separated:
New Author2
Author not found, opening new author prompt!
Please fill out all available information for Author2
 New:
Enter author's date of birth:
2004-02-17
Enter author's status (active, inactive, or unknown)
unknown
Author added successfully.
```

f. **Java Implementation:**



```java
// Method to add a new author to the database
public static int createAuthorPrompt(String connectionUrl, String firstName, String lastName) {
    Scanner scanner = new Scanner(System.in);
    if (firstName != null && lastName != null) {
        System.out.println("Please fill out all available information for " + firstName + " " + lastName + ":");
    }
    // If first name and last name are not provided, prompt for them
    if (firstName == null) {
        System.out.println(x:"Enter author's first name:");
        firstName = scanner.nextLine();
    }
    if (lastName == null) {
        System.out.println(x:"Enter author's last name:");
        lastName = scanner.nextLine();
    }
    // Prompt for DOB and status
    System.out.println(x:"Enter author's date of birth:");
    String dob = scanner.nextLine();
    System.out.println(x:"Enter author's status (active, inactive, or unknown)");
    String status = scanner.nextLine();
    // Call stored procedure to add an author
    String[] array = {firstName, lastName, dob, status};
    if (executeProcedureNoResult(procedureName:"addAuthor", array, connectionUrl)) {
        // Continue to find the author's ID and return it
    } else {
        System.out.println(x:"Error adding author.");
        return -1;
    }
    // This statement selects a single author by their first and last name
    String selectIDStatement = "SELECT TOP 1 authorID FROM Author WHERE firstName = ? AND lastName = ?";
    // Find and return author ID
    try (Connection connection = DriverManager.getConnection(connectionUrl);
         PreparedStatement selectStmt = connection.prepareStatement(selectIDStatement)) {
        selectStmt.setString(parameterIndex:1, firstName);
        selectStmt.setString(parameterIndex:2, lastName);
        ResultSet resultSet = selectStmt.executeQuery();
        if (resultSet.next()) {
            System.out.println("Author " + firstName + " " + lastName + " added successfully with ID " +
            resultSet.getInt(columnLabel:"authorID") + ".");
            return resultSet.getInt(columnLabel:"authorID");
        } else {
            System.out.println(x:"Error adding author.");
            return -1;
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return -1;
    }
}
```

Fei Triolo’s Use Cases

F1.

12.

a. **Title:** Searching for a book based on multiple keywords or genres

b. **Description:** When a patron is browsing the library they may be interested in books about multiple subjects. The user can query the database for books that share multiple keywords or genres by searching for a list of those keywords or genres separated by commas and will be returned book objects that have each of the keywords in their list

c.

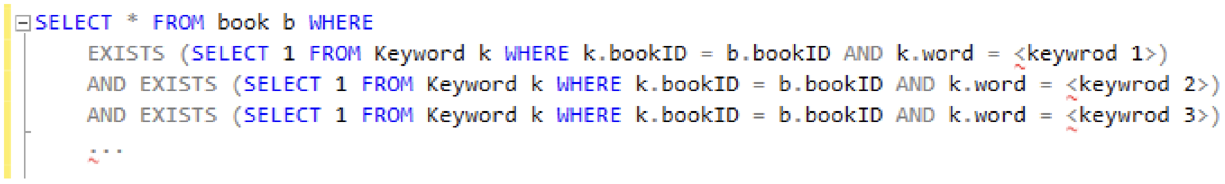
User Requirements:

i. The user must connect to the database from the application with any permissions

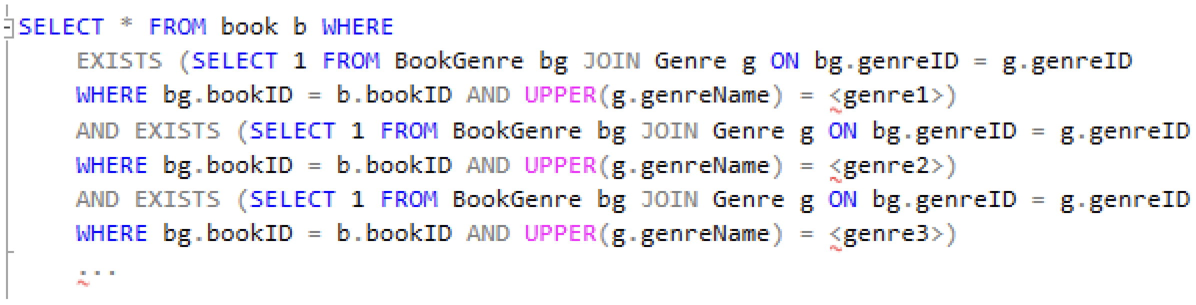
ii. The user must select the menu option to search for books with all keywords or search for books with all genres

iii. The user will be prompted to enter a comma-separated list of keywords/genres they would like to search for

d. **SQL Queries:** [NOTE: SQL Code dynamically generated in JDBC]



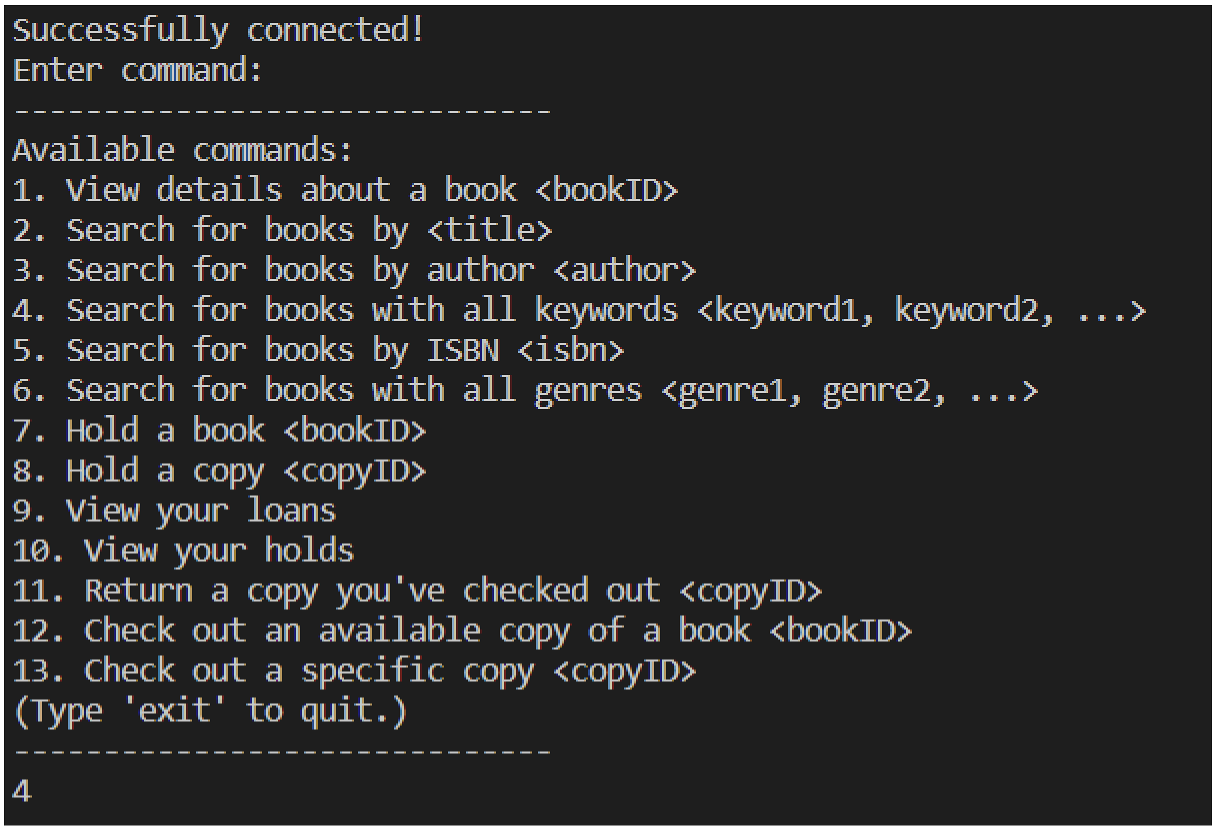
```sql
SELECT * FROM book b WHERE
    EXISTS (SELECT 1 FROM Keyword k WHERE k.bookID = b.bookID AND k.word = <keywrod 1>)
    AND EXISTS (SELECT 1 FROM Keyword k WHERE k.bookID = b.bookID AND k.word = <keywrod 2>)
    AND EXISTS (SELECT 1 FROM Keyword k WHERE k.bookID = b.bookID AND k.word = <keywrod 3>)
    ...
```



```sql
SELECT * FROM book b WHERE
    EXISTS (SELECT 1 FROM BookGenre bg JOIN Genre g ON bg.genreID = g.genreID
    WHERE bg.bookID = b.bookID AND UPPER(g.genreName) = <genre1>)
    AND EXISTS (SELECT 1 FROM BookGenre bg JOIN Genre g ON bg.genreID = g.genreID
    WHERE bg.bookID = b.bookID AND UPPER(g.genreName) = <genre2>)
    AND EXISTS (SELECT 1 FROM BookGenre bg JOIN Genre g ON bg.genreID = g.genreID
    WHERE bg.bookID = b.bookID AND UPPER(g.genreName) = <genre3>)
    ...
```

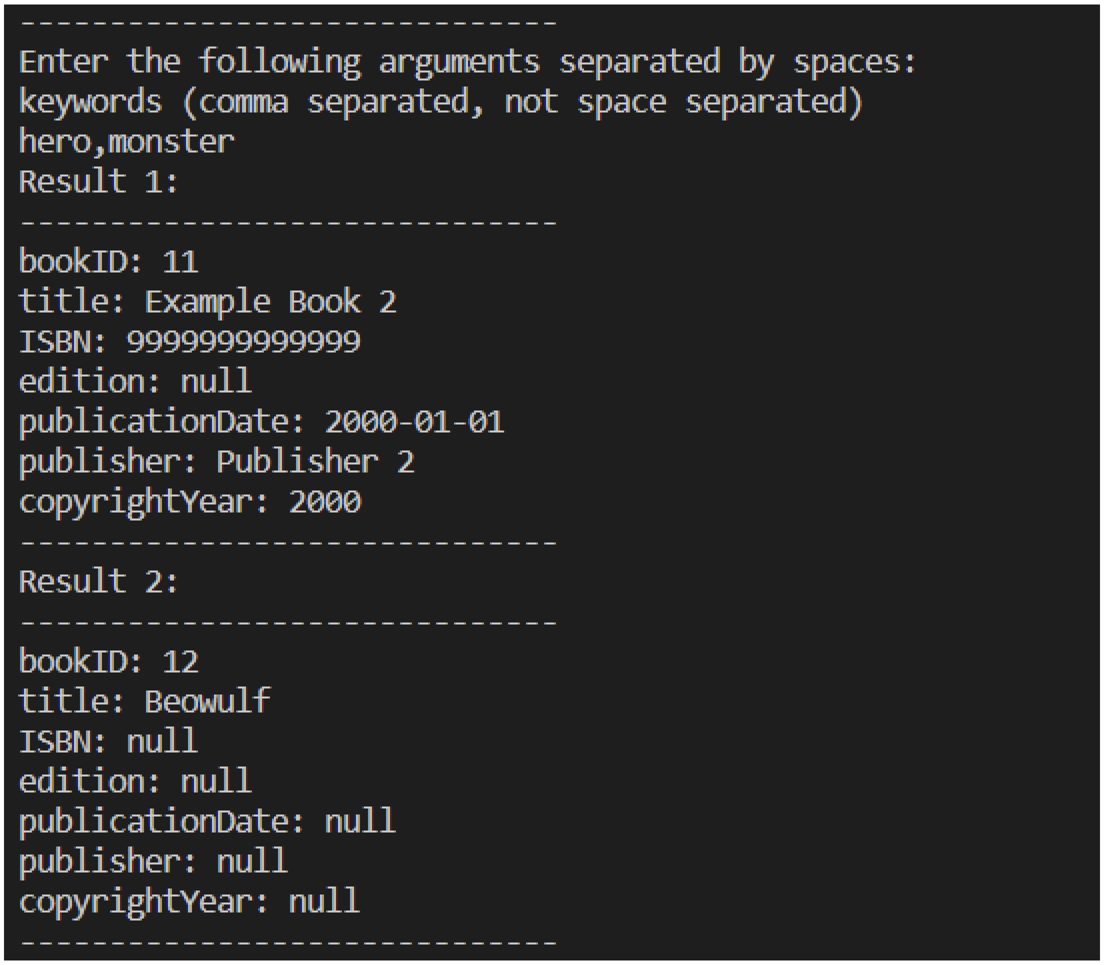
e. **Screenshots:**

Connecting to the database and selecting the keyword search option:



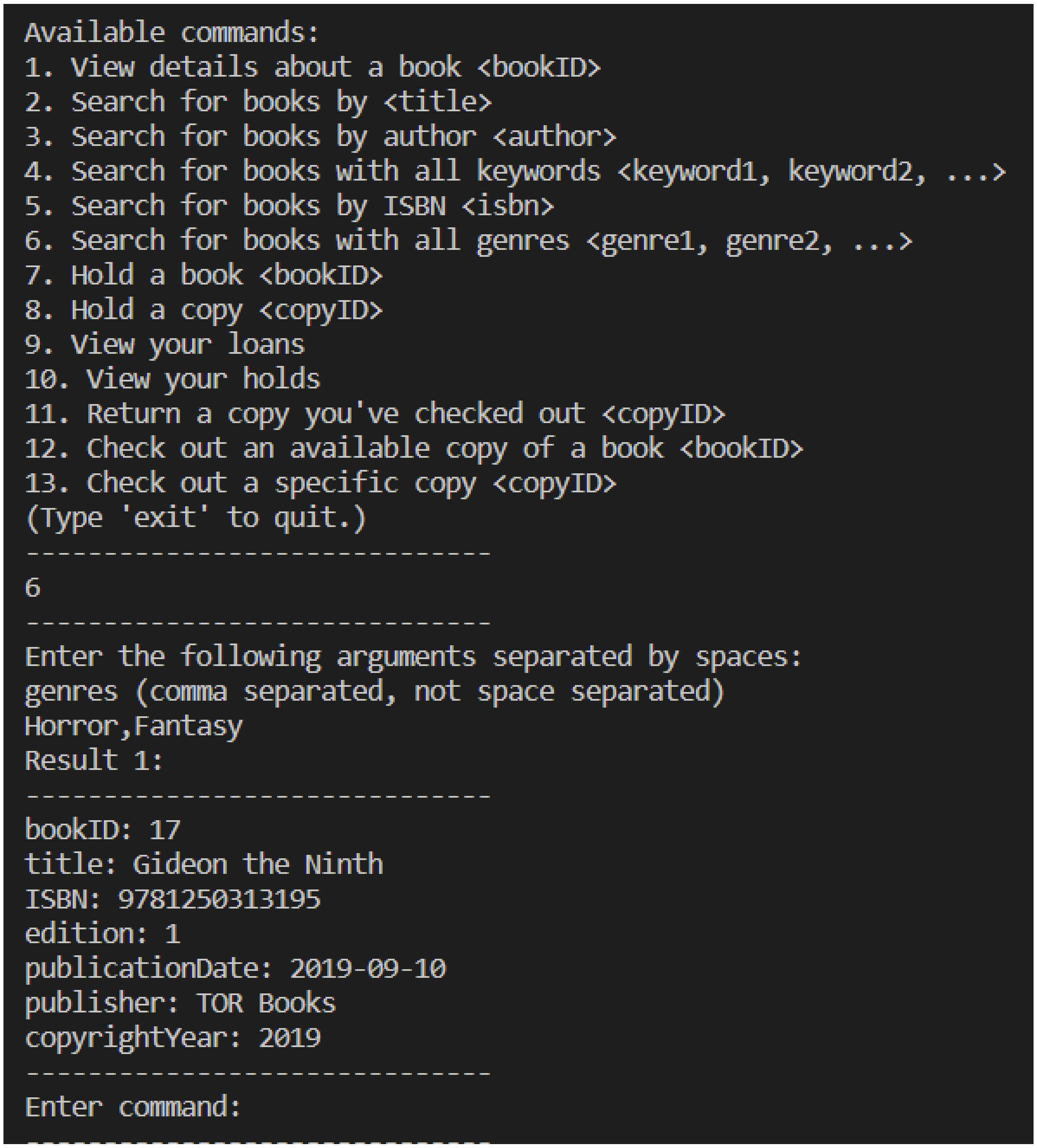
```
Successfully connected!
Enter command:
------------------------------
Available commands:
1. View details about a book <bookID>
2. Search for books by <title>
3. Search for books by author <author>
4. Search for books with all keywords <keyword1, keyword2, ...>
5. Search for books by ISBN <isbn>
6. Search for books with all genres <genre1, genre2, ...>
7. Hold a book <bookID>
8. Hold a copy <copyID>
9. View your loans
10. View your holds
11. Return a copy you've checked out <copyID>
12. Check out an available copy of a book <bookID>
13. Check out a specific copy <copyID>
(Type 'exit' to quit.)
------------------------------
4
```

Searching for keywords 'hero' and 'monster' and receiving books in the library database with those keywords:



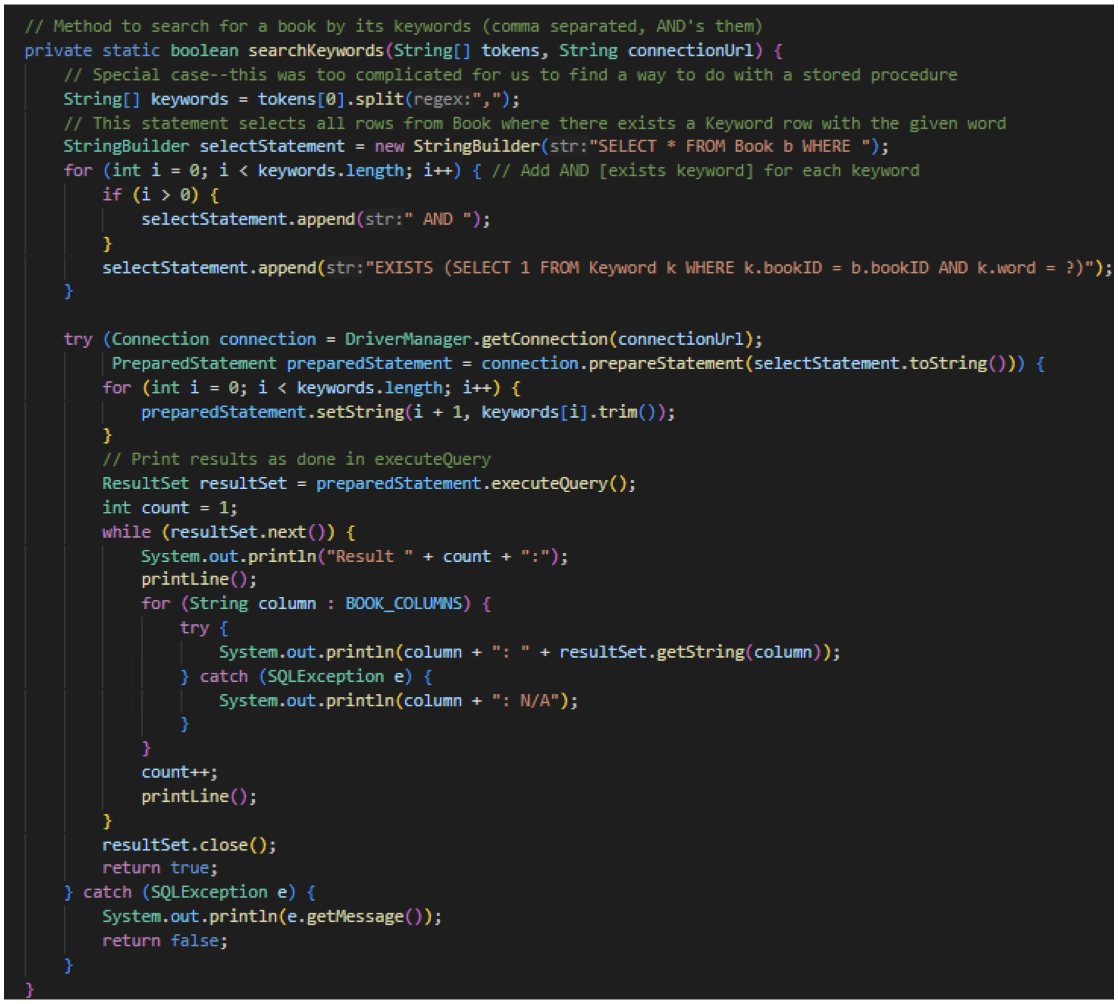
```
------------------------------
Enter the following arguments separated by spaces:
keywords (comma separated, not space separated)
hero,monster
Result 1:
------------------------------
bookID: 11
title: Example Book 2
ISBN: 9999999999999
edition: null
publicationDate: 2000-01-01
publisher: Publisher 2
copyrightYear: 2000
------------------------------
Result 2:
------------------------------
bookID: 12
title: Beowulf
ISBN: null
edition: null
publicationDate: null
publisher: null
copyrightYear: null
------------------------------
```

Same process for searching by genres:

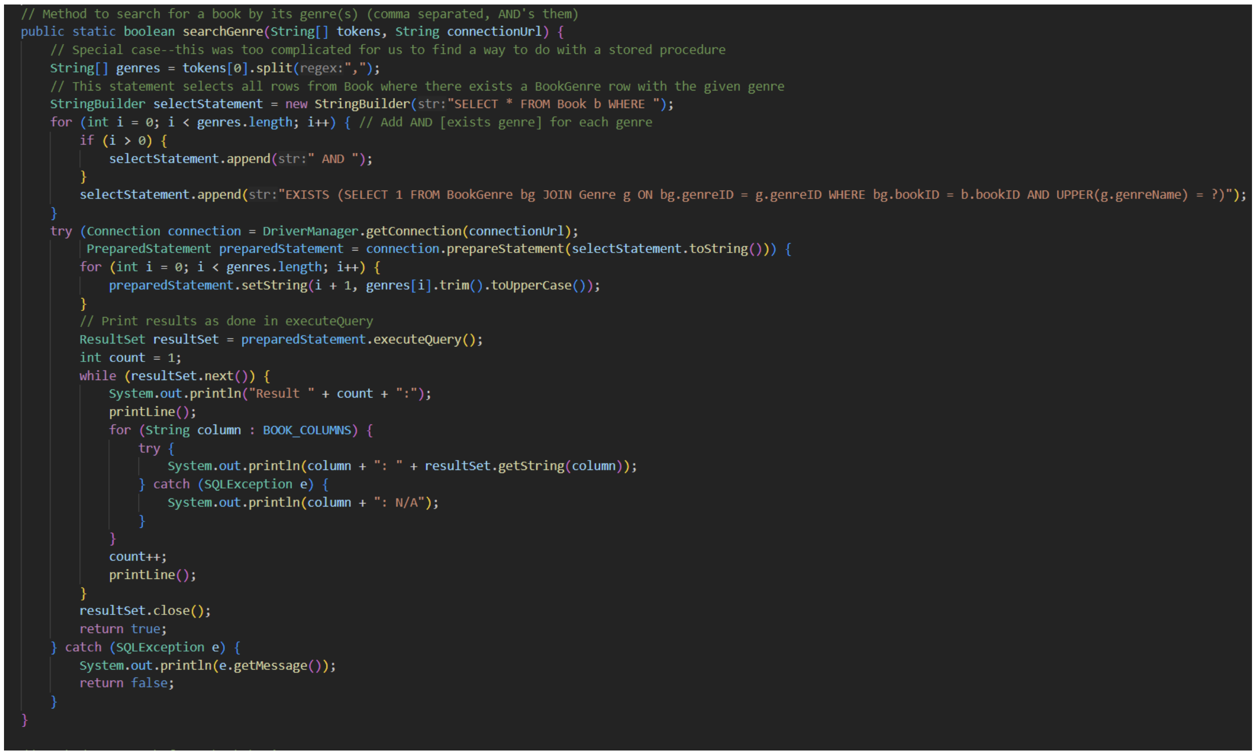


```
Available commands:
1. View details about a book <bookID>
2. Search for books by <title>
3. Search for books by author <author>
4. Search for books with all keywords <keyword1, keyword2, ...>
5. Search for books by ISBN <isbn>
6. Search for books with all genres <genre1, genre2, ...>
7. Hold a book <bookID>
8. Hold a copy <copyID>
9. View your loans
10. View your holds
11. Return a copy you've checked out <copyID>
12. Check out an available copy of a book <bookID>
13. Check out a specific copy <copyID>
(Type 'exit' to quit.)
------------------------------
6
------------------------------
Enter the following arguments separated by spaces:
genres (comma separated, not space separated)
Horror,Fantasy
Result 1:
------------------------------
bookID: 17
title: Gideon the Ninth
ISBN: 9781250313195
edition: 1
publicationDate: 2019-09-10
publisher: TOR Books
copyrightYear: 2019
------------------------------
Enter command:
```

f. **Java Implementation:**



```java
// Method to search for a book by its keywords (comma separated, AND's them)
private static boolean searchKeywords(String[] tokens, String connectionUrl) {
    // Special case--this was too complicated for us to find a way to do with a stored procedure
    String[] keywords = tokens[0].split(regex:",");
    // This statement selects all rows from Book where there exists a Keyword row with the given word
    StringBuilder selectStatement = new StringBuilder(str:"SELECT * FROM Book b WHERE ");
    for (int i = 0; i < keywords.length; i++) { // Add AND [exists keyword] for each keyword
        if (i > 0) {
            selectStatement.append(str:" AND ");
        }
        selectStatement.append(str:"EXISTS (SELECT 1 FROM Keyword k WHERE k.bookID = b.bookID AND k.word = ?)");
    }

    try (Connection connection = DriverManager.getConnection(connectionUrl);
         PreparedStatement preparedStatement = connection.prepareStatement(selectStatement.toString())) {
        for (int i = 0; i < keywords.length; i++) {
            preparedStatement.setString(i + 1, keywords[i].trim());
        }
        // Print results as done in executeQuery
        ResultSet resultSet = preparedStatement.executeQuery();
        int count = 1;
        while (resultSet.next()) {
            System.out.println("Result " + count + ":");
            printLine();
            for (String column : BOOK_COLUMNS) {
                try {
                    System.out.println(column + ": " + resultSet.getString(column));
                } catch (SQLException e) {
                    System.out.println(column + ": N/A");
                }
            }
            count++;
            printLine();
        }
        resultSet.close();
        return true;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```



```
// Method to search for a book by its genre(s) (comma separated, AND's them)
public static boolean searchGenre(String[] tokens, String connectionUrl) {
    // Special case--this was too complicated for us to find a way to do with a stored procedure
    String[] genres = tokens[0].split(regex:",");
    // This statement selects all rows from Book where there exists a BookGenre row with the given genre
    StringBuilder selectStatement = new StringBuilder(str:"SELECT * FROM Book b WHERE ");
    for (int i = 0; i < genres.length; i++) { // Add AND [exists genre] for each genre
        if (i > 0) {
            selectStatement.append(str:" AND ");
        }
        selectStatement.append(str:"EXISTS (SELECT 1 FROM BookGenre bg JOIN Genre g ON bg.genreID = g.genreID WHERE bg.bookID = b.bookID AND UPPER(g.genreName) = ?)");
    }
    try (Connection connection = DriverManager.getConnection(connectionUrl);
         PreparedStatement preparedStatement = connection.prepareStatement(selectStatement.toString())) {
        for (int i = 0; i < genres.length; i++) {
            preparedStatement.setString(i + 1, genres[i].trim().toUpperCase());
        }
        // Print results as done in executeQuery
        ResultSet resultSet = preparedStatement.executeQuery();
        int count = 1;
        while (resultSet.next()) {
            System.out.println("Result " + count + ":");
            printLine();
            for (String column : BOOK_COLUMNS) {
                try {
                    System.out.println(column + ": " + resultSet.getString(column));
                } catch (SQLException e) {
                    System.out.println(column + ": N/A");
                }
            }
            count++;
            printLine();
        }
        resultSet.close();
        return true;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```

F2.

13.

a. **Title:** Logging in as member, employee, or curator

b. **Description:** In order to execute certain commands in the app, a user must log in as one of three roles. All users have permission to search for books in the database. Members have permission to check and hold books for themselves. Employees have permission to check out or hold books for any member, look up details about library patrons, and add or remove members to the library. Curators are able to do anything employees do in addition to adding and updating books, copies, authors, genres, and keywords.

c.

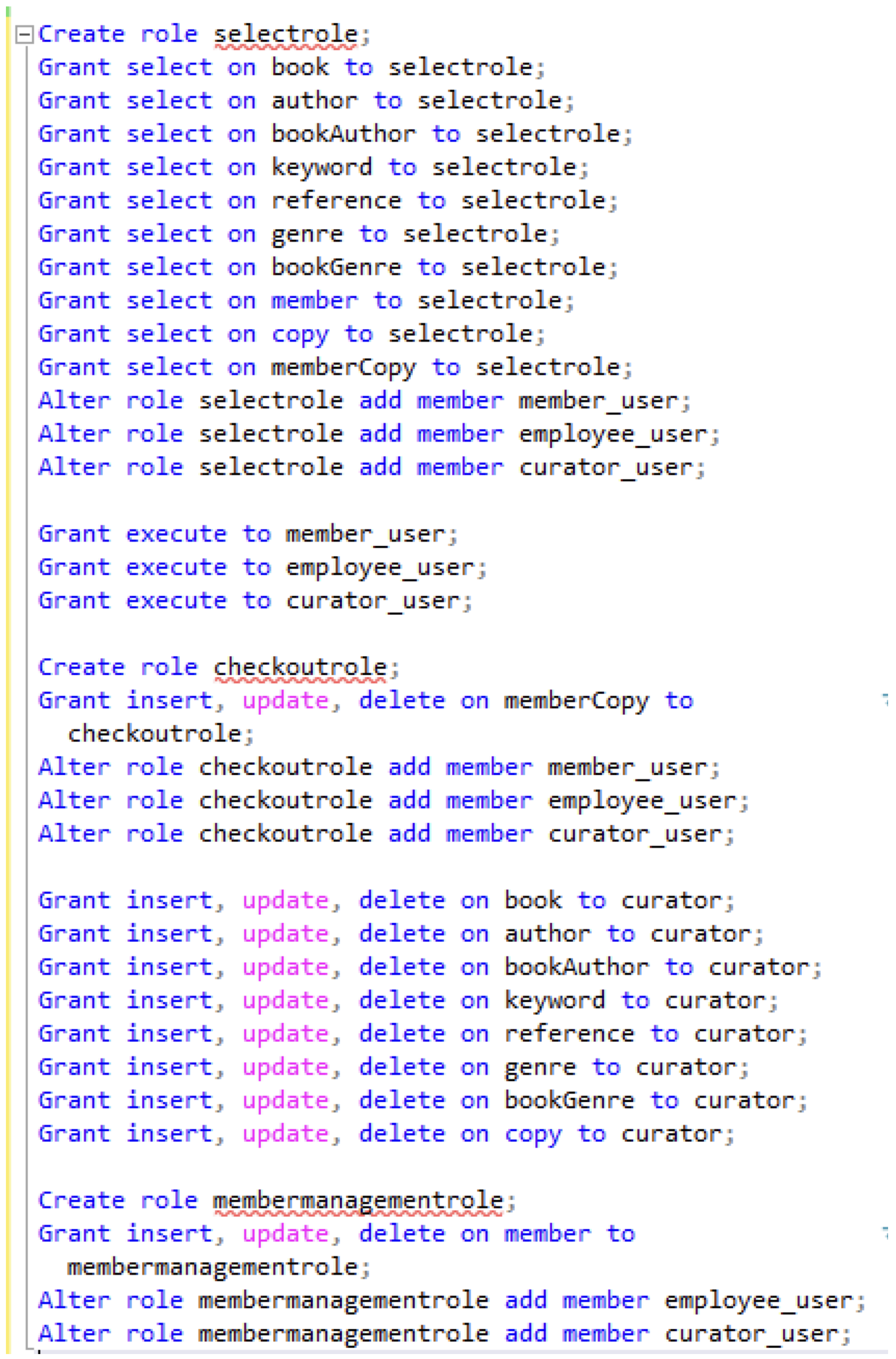
User Requirements:

i. The user wishing to log in must know the credentials to connect to the database (logins: member_login, employee_login, curator_login; passwords: MemberPassword!, EmployeePassword!, CuratorPassword!)

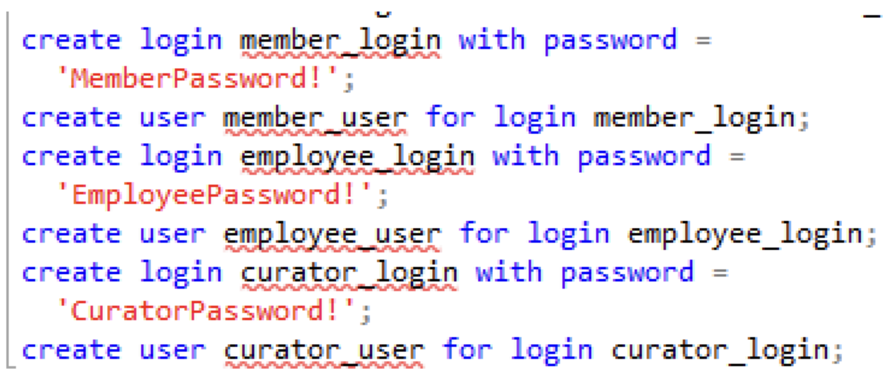
ii. If the user logging in is merely a library member, they must additionally know and enter their library ID (which is returned when an employee creates their account for them)

d. **SQL Queries:**

(The underlines indicating errors are because these roles and logins already exist in the database)

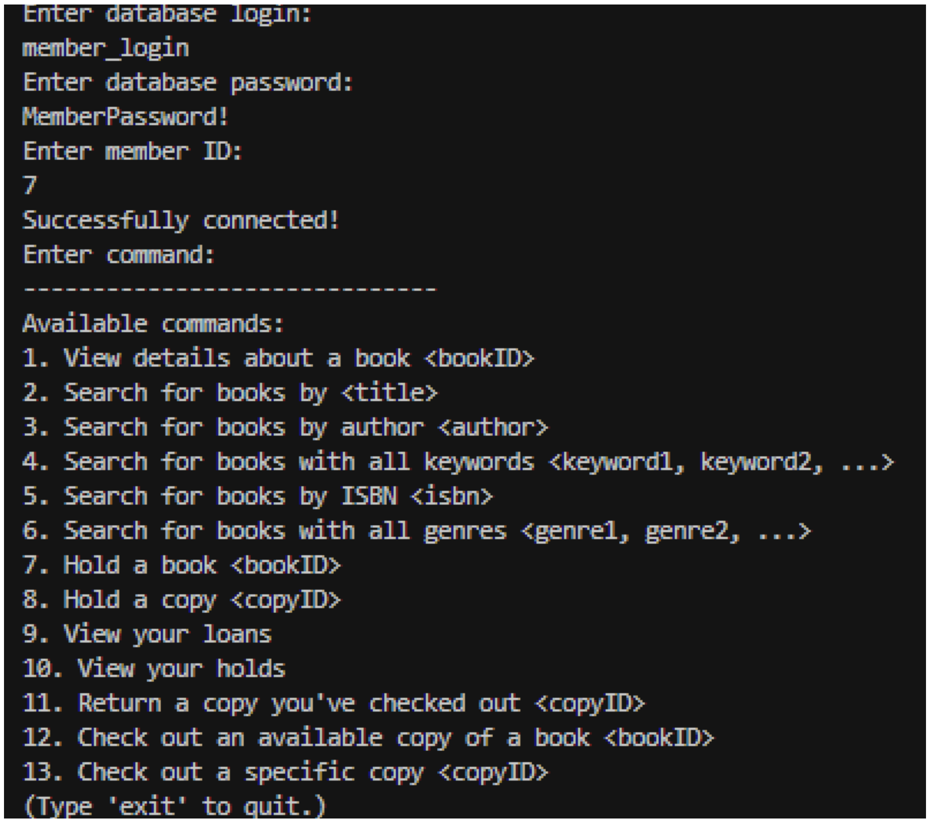


```
Create role selectrole;
Grant select on book to selectrole;
Grant select on author to selectrole;
Grant select on bookAuthor to selectrole;
Grant select on keyword to selectrole;
Grant select on reference to selectrole;
Grant select on genre to selectrole;
Grant select on bookGenre to selectrole;
Grant select on member to selectrole;
Grant select on copy to selectrole;
Grant select on memberCopy to selectrole;
Alter role selectrole add member member_user;
Alter role selectrole add member employee_user;
Alter role selectrole add member curator_user;

Grant execute to member_user;
Grant execute to employee_user;
Grant execute to curator_user;

Create role checkoutrole;
Grant insert, update, delete on memberCopy to
  checkoutrole;
Alter role checkoutrole add member member_user;
Alter role checkoutrole add member employee_user;
Alter role checkoutrole add member curator_user;

Grant insert, update, delete on book to curator;
Grant insert, update, delete on author to curator;
Grant insert, update, delete on bookAuthor to curator;
Grant insert, update, delete on keyword to curator;
Grant insert, update, delete on reference to curator;
Grant insert, update, delete on genre to curator;
Grant insert, update, delete on bookGenre to curator;
Grant insert, update, delete on copy to curator;

Create role membermanagementrole;
Grant insert, update, delete on member to
  membermanagementrole;
Alter role membermanagementrole add member employee_user;
Alter role membermanagementrole add member curator_user;
```



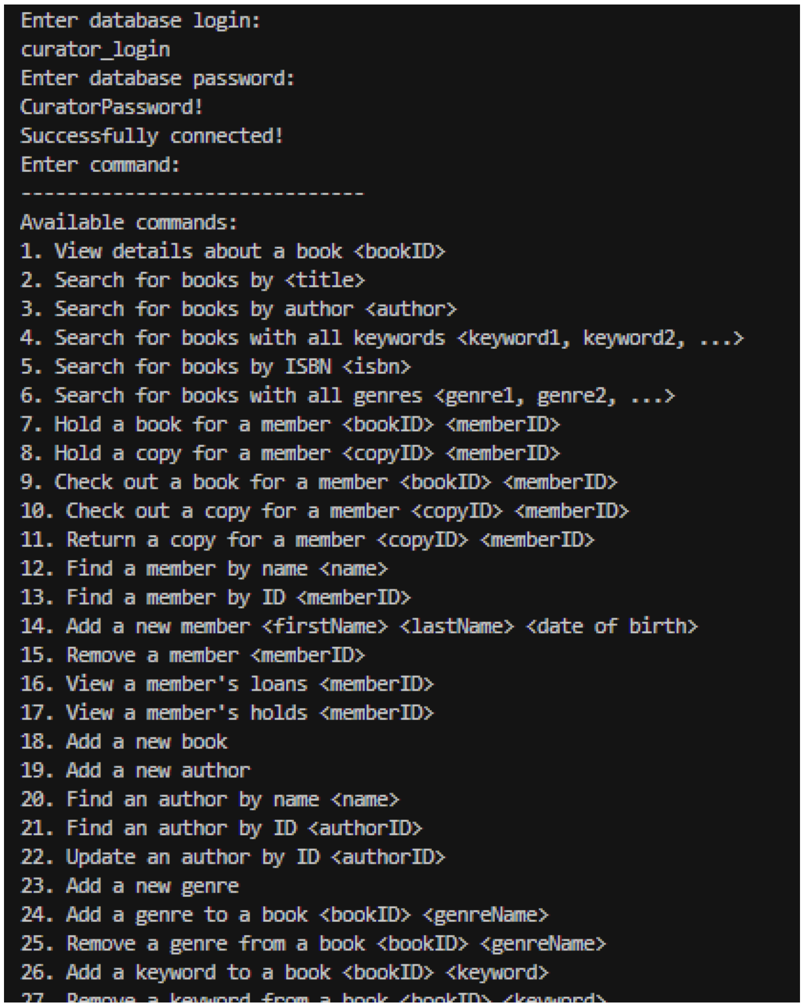
```
create login member_login with password =
  'MemberPassword!';
create user member_user for login member_login;
create login employee_login with password =
  'EmployeePassword!';
create user employee_user for login employee_login;
create login curator_login with password =
  'CuratorPassword!';
create user curator_user for login curator_login;
```

e. **Screenshots:**
Logging into the database as a member



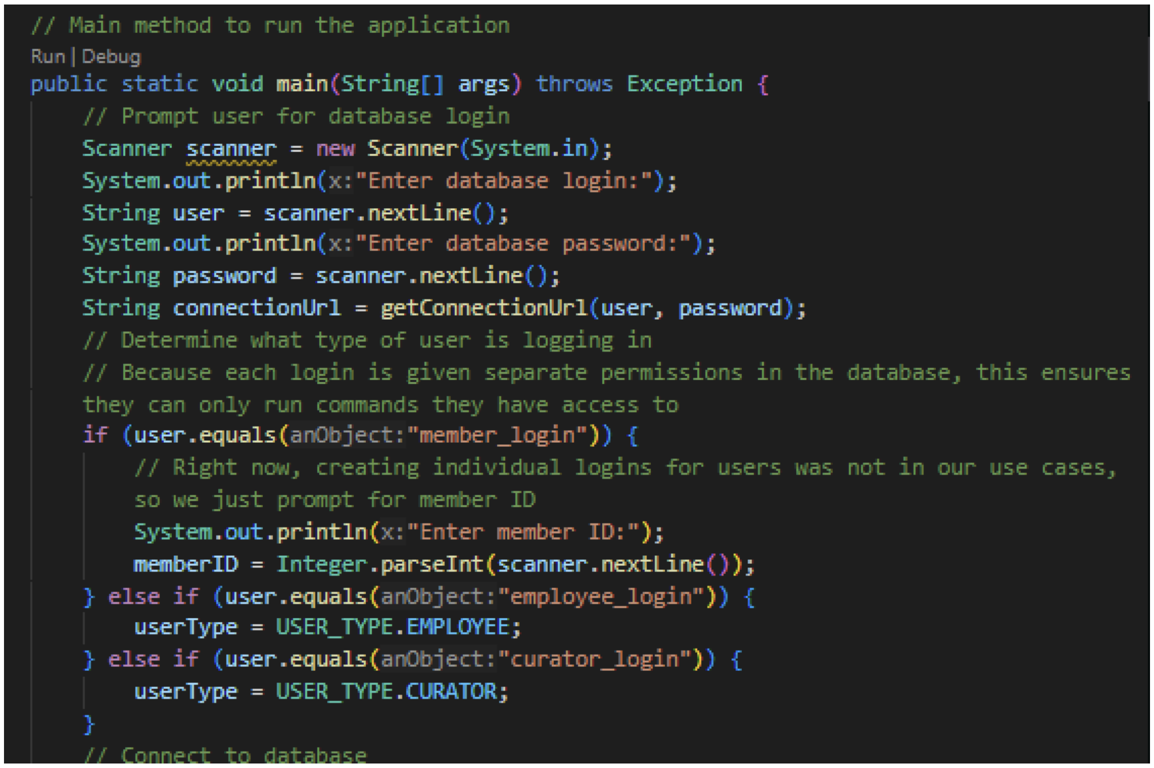
```
Enter database login:
member_login
Enter database password:
MemberPassword!
Enter member ID:
7
Successfully connected!
Enter command:
------------------------------
Available commands:
1. View details about a book <bookID>
2. Search for books by <title>
3. Search for books by author <author>
4. Search for books with all keywords <keyword1, keyword2, ...>
5. Search for books by ISBN <isbn>
6. Search for books with all genres <genre1, genre2, ...>
7. Hold a book <bookID>
8. Hold a copy <copyID>
9. View your loans
10. View your holds
11. Return a copy you've checked out <copyID>
12. Check out an available copy of a book <bookID>
13. Check out a specific copy <copyID>
(Type 'exit' to quit.)
```

Logging into the database as a curator



```
Enter database login:
curator_login
Enter database password:
CuratorPassword!
Successfully connected!
Enter command:
------------------------------
Available commands:
1. View details about a book <bookID>
2. Search for books by <title>
3. Search for books by author <author>
4. Search for books with all keywords <keyword1, keyword2, ...>
5. Search for books by ISBN <isbn>
6. Search for books with all genres <genre1, genre2, ...>
7. Hold a book for a member <bookID> <memberID>
8. Hold a copy for a member <copyID> <memberID>
9. Check out a book for a member <bookID> <memberID>
10. Check out a copy for a member <copyID> <memberID>
11. Return a copy for a member <copyID> <memberID>
12. Find a member by name <name>
13. Find a member by ID <memberID>
14. Add a new member <firstName> <lastName> <date of birth>
15. Remove a member <memberID>
16. View a member's loans <memberID>
17. View a member's holds <memberID>
18. Add a new book
19. Add a new author
20. Find an author by name <name>
21. Find an author by ID <authorID>
22. Update an author by ID <authorID>
23. Add a new genre
24. Add a genre to a book <bookID> <genreName>
25. Remove a genre from a book <bookID> <genreName>
26. Add a keyword to a book <bookID> <keyword>
27. Remove a keyword from a book <bookID> <keyword>
```

f. **Java Implementation:**



```
// Main method to run the application
Run | Debug
public static void main(String[] args) throws Exception {
    // Prompt user for database login
    Scanner scanner = new Scanner(System.in);
    System.out.println(x:"Enter database login:");
    String user = scanner.nextLine();
    System.out.println(x:"Enter database password:");
    String password = scanner.nextLine();
    String connectionUrl = getConnectionUrl(user, password);
    // Determine what type of user is logging in
    // Because each login is given separate permissions in the database, this ensures
    they can only run commands they have access to
    if (user.equals(anObject:"member_login")) {
        // Right now, creating individual logins for users was not in our use cases,
        so we just prompt for member ID
        System.out.println(x:"Enter member ID:");
        memberID = Integer.parseInt(scanner.nextLine());
    } else if (user.equals(anObject:"employee_login")) {
        userType = USER_TYPE.EMPLOYEE;
    } else if (user.equals(anObject:"curator_login")) {
        userType = USER_TYPE.CURATOR;
    }
    // Connect to database
```

Functional Dependencies, Physical Database Design, And Normalization

Functional Dependencies

- Dependant relations:
    - BookAuthor table is functionally dependent on some existing Book entity and Author entity in the database
    - BookGenre is functionally dependent on some existing Book entity and existing Genre details stored in the database because as per our business rules, any Book entity must have at least one related Genre but if a book is added with a genre that is not yet in the database a new one should be added
    - The Copy table is dependent on the Book entity because as per our business rules, if the library owns a copy of a book it must store details about that book in the database.
    - Keyword is functionally dependent on the Book table because the table relates keywords to existing Book entities and only stores each word if it has one or more books in the database that relate to said keyword
    - MemberCopy is dependent on Member and book. A MemberCopy entity is only created when an existing member checks out or places a hold on an existing book in the database

- Independent relations:
    - Book is an independent entity in the database because the library is allowed to store information on books they do not have a physical copy of, or know the author to
    - Author and genre are independent entities because the library is allowed to store information on authors and genres even if they don't currently store books from those authors or in those genres
    - Members are independent entities because a person can become a library member regardless of the state or existence of any other entities in the database

Physical Database Design

- Indexes
    - It will be advantageous to create an index for the Keyword table
    - This is because the Keyword table will be frequently searched against, which could cause performance issues given the vast amount of data that will be present in it
    - Each keyword of each book is represented with a separate relation, so creating an index on the bookID and word columns will speed up the process of locating the keywords for a given book or finding all books with a specific keyword
- Triggers
    - As mentioned in our previous report, we want each book to be able to have one or more authors, but only one of these authors will can designated as the primary author
    - To ensure that this is the case, we will need to create a trigger that is activated when an entry is inserted into the BookAuthor table
    - It begins by checking if the entry being inserted has the isPrimaryAuthor flag set to 1
    - If this is the case, it will need to count every relation in BookAuthor where the bookID matches the bookID of the new entry and isPrimaryAuthor is equal to one
    - If this sum is >= 1, an error will be reported indicating that there is already a primary author for the given book, and the relation will not be inserted
    - Otherwise, insert the relation like normal
- Derived Attributes and Views
    - We handled some of the functions we had planned to include in the view with our java code.

Normalization

Our database is currently at Normalization Form 3 since every extraneous dependency is split between tables. It does not qualify for BCNF because some transient functional dependencies still exist, such as word in the keyword table, which exists because the word dependency is atomic and splitting word into its own table with a unique word identity would not reduce redundancy as no new information needs to be stored about each keyword except the word itself. There is another transient dependency in the memberCopy relation because the expiry date and overdue attributes are derived attributes from createdDate and are dependent on createdDate as well as memberID and bookID which also determine createdDate. These attributes are kept in the memberCopy relation because they are needed to alert members and employees when a given member's holds or loans are overdue.

User Manual

Navigating the Interface

When using the interface, your accessible commands will depend on your given role. A user can be a member, employee, or a curator. When the program is run, you will be prompted for your database login and password. The available logins are member_login, employee_login, and curator_login. The passwords for these logins are MemberPassword!, EmployeePassword!, and CuratorPassword!. If you are a new member, you will need to contact a library employee, who will create an account with your information and give you your library ID number. If you log in with the member_login account, you will be asked for this number. This will allow you to directly check out/hold books from your account, view your account status, and other member-specific functionality. You will be given a list of applicable commands, each with a corresponding number. You will enter one of these numbers to run the command, and, if necessary, you will be prompted to enter more information. When you are finished, you can either enter `exit` or `logout` to log out of the library database.

Available Functionality

All users can:

- View details about a book
- Search for books by title
- Search for books by author
- Search for books by one or more keywords
- Search for books by ISBN
- Search for books by one or more genres

Only members can:

- Directly place a hold on a book from their account (finds an available copy of the book)
- Directly place a hold on a specific copy of a book from their account
- View all books loaned to them with their overdue status
- View all books on hold by them with their overdue status
- Return a copy of a book they have checked out
- Directly check out a book from their account (finds an available copy of the book)
- Directly check out a specific copy of a book from their account

Only employees and curators can:

- Hold a book for a specific member (finds an available copy of the book)
- Hold a specific copy of a book for a specific member

- Check out a book for a specific member (finds an available copy of the book)
- Check out a specific copy of a book for a specific member
- Return a copy of a book checked out by a specific member
- Find a library member’s information by name
- Find a library member’s information by their library ID
- Add a new member to the library database
- Remove an existing member from the library database
- View all books loaned to a specific member with their overdue status
- View all books on hold by a specific member with their overdue status

Only curators can:

- Add a new book to the library database
- Add a new author to the library database
- Search for an author’s information by name
- Search for an author’s information by author ID
- Update an author’s information
- Add a new genre to the library database
- Add a new or existing genre to a specific book
- Remove a genre from a book
- Add a keyword to a book
- Remove a keyword from a book
- Add a physical copy of a book to the library database
- Remove a physical copy of a book from the library database

Reflection

Our library database gave us an interesting opportunity to explore our learning from this class in a novel way. What we discovered with this final step in the project was the difficulty scale added to a database. Although adding new attributes was very straightforward when making or altering tables, they added greater challenges with then having to update the Java interface. Managing different levels of access meant that we would have to update these users if this new attribute impacted them in any way. It became important for us to prioritize which attributes were most crucial for our project and scale down those which were not high priority. Our focused efforts allowed us to complete the database in time with all of our most crucial features