# Library Database System

| | |
|---|---|
| **Course Title:** | **Database Systems I** |
| **Course Number:** | **CPS 510** |
| **Semester/Year (e.g. F2016)** | **W2023** |
| **Instructor** | **Dr. Abdoreza Abhari** |

| | |
|---|---|
| **Assignment** | **Lab Assignment 10** |
| **Assignment Title** | **FINAL REPORT** |

| | |
|---|---|
| **Submission Date:** | **Nov. 30, 2023,** |
| **Due Date:** | **Nov. 30, 2023,** |

| Student Last Name | Student First Name | Student Number | Section | Signature* |
|---|---|---|---|---|
| **Lee** | **Woojin** | **501031919** | **03** | **W. L.** |
| **Jiang** | **Talon** | **501125811** | **03** | **T.J** |
| **Minh** | **Tran** | **501132173** | **03** | **M.T** |

# Table of Contents

**ASSIGNMENT 1: Logical Database Design**

# Introduction

In this following report, we will describe our library database system, which manages and controls the essential functions of the typical library system. As mentioned, the report will give the reader a general overview of how the library data is processed, and how the library system is to be used from a user standpoint.

# General Description of the Library Database System

The purpose of the system is to emulate a library that uses a database to store information about members, staff, and other forms of media. For this system, four main entities will be considered: The librarians as administrators who manage the data of books, and the members. The media entity consists of the content and properties of all media. Furthermore, the member entity can request a viewing of the books and other information about the library. Finally, the library entity is the control system that keeps track of all the staff, media, and members. In the future, there will be some adjustments depending on the system we construct; hence, there will be some differences or some functions that we can add to our library to create an easily accessible and friendly environment for users. We will update our process along with those changes in the next reports if needed.

## Usage of the System

The system described in this report is run via Jar File with Java versions 16 and 21, and includes all necessary dependencies in an Uber Jar format. The system makes use of the OJDBC Library to read and write to an Oracle 11g database system, alongside JavaFX and Apache POI to handle data and front-end interactions.

To date, we have compiled two different versions of the application, an Alpha version described in Assignment 9, which makes use of simple console interactions, and a more complete version described in Assignment 10, which makes use of a login system and graphical user interfaces.

### Using the Console Application

The Library System in Assignment 9 consists of a terminal based selection program, which creates interactions with the database depending on the option selected by the user. OpenVPN connections to the Toronto Metropolitan network are required to use the system properly, as the interactions with the database will not execute successfully otherwise.

Options A-C involve the manipulation of tables in the database; To use the tables, they first must be created and populated. If there is no data in the table when the queries are run, then there will be an empty result.

Options from D-I correspond with queries in the database system, where each query performs a different action on the Database System. The output is formatted and printed to the console in each instance.

Inputting option J will exit the System and save the state of the database.

**Using the Graphical Application**

The graphical user interface system described in Assignment 10 is a more completed version of the project, however not all functionality has been implemented due to time constraints in the project timeline. Future implementations of these functions are planned.

The graphical user interface currently consists of a login page, a query page, and a table modification area which allows the user to add new entries to the database system. The application is configured also to be an Uber jar containing all necessary dependencies for system to function, alongside resources which it uses to interact with tables.

When first running the system, the application will create several files which contain commands necessary for running the program. These files are then read by the program, of which the data inside is used to create, drop, and populate tables. Like the version described in Assignment 9, this version of the program also requires the user to connect to the Toronto Metropolitan network via VPN in order to execute queries successfully.

To use the system, the user must first log in as an existing user to the system. Functionality is planned to allow administrators to authenticate new users and create user accounts.
When a user logs in, they are presented by a panel which allows them to make specific queries, as well as search for media by name. An output log below shows the result of each interaction.

If the user chooses to add or delete data, they will be presented with a screen which allows them to add and delete media and member accounts. It was chosen to not allow the user to modify the history of transactions due to the possibility that they may tamper with the data. Functionality is planned to allow librarians to interact and view transactions and feedback tables.

# Functions of the Database System

| Entity | Functions |
|---|---|
| Media (Books, CDs, Articles, etc.) | Holds information about the content of the media<br>Can be borrowed and reserved by members |
| Librarians | Manages the members in the library system<br>Allow users to reset their password<br>Receive feedback from users<br>Manages the media in the database system |
| Members | Send requests to view media<br>Borrow and reserve media<br>Create accounts to access the system<br>Send feedback to librarians |
| Feedback | Keep track of the librarians and member list<br>Keep track of the media in the library<br>Sorts(searches) for the media depending on the member's request |
| Transaction details | Keeps track of orders (Reservations, borrows, returns) in the system<br>Sends overdue transactions to penalties |

| | |
|---|---|
| Penalties | Keeps track of penalties for members<br>Applies penalties to members |

## Relationships between entities

| Entity | Relationships |
|---|---|
| Media | - Relationship between the librarian and itself for when new content is added to the system. It also stores the system. (Many : Many)<br>- Relationship between itself to transaction history for when it gets borrowed or reserved, the media can be tracked down. (1 : Many) |
| Librarian | - Relationship between itself and the members as the Librarian is able to View, edit the information of the members if necessary. (1:Many)<br>- Relationship between itself and the media to edit, add or sort the data. (1:Many)<br>- Relationship between itself and the feedback as it receives feedback from users, which is inputted into the system. (1 : many) |
| Members | - Relationship between itself and the librarian when a member account is created and is stored into the |

| | |
|---|---|
| | database (Many : Many)<br>- Relationship between itself and the transaction history as when the user wants to borrow or reserve the media, the information gets sent in. (1 : many)<br>- Relationship between itself and the penalty as if the member was unsuccessful in returning media on time, some sort of penalty will be applied to the member. (1 : many) |
| Transaction details | ● Relationship between itself and members as when members make order, its id gets recorded (many : 1)<br>● Relationship between itself and books as when the transaction occurs, that book is now borrowed/reserved to a member (1 to 1)<br>● Relationship between itself and the penalty system as if the member fails to return the media on time, it gets sent to the feedback system. (1 : 1) |
| Penalty | ● Relationship between itself and the Transaction details as all the overdue information gets transferred over to the feedback section. (1 : 1)<br>● Relationship between itself and the librarian as it notifies the librarians about the overdue books (1 : 1)<br>● Relationship between itself and the members to apply |

| | |
|---|---|
| | penalties if they have overdue media. (Many : 1) |
| Feedback | • Relationship between itself and the members to receive the penalty. ()<br>• Relationship between itself and the Librarians to deliver the feedback to the librarian. () |

## ASSIGNMENT 2: ER MODEL OF THE SYSTEM:

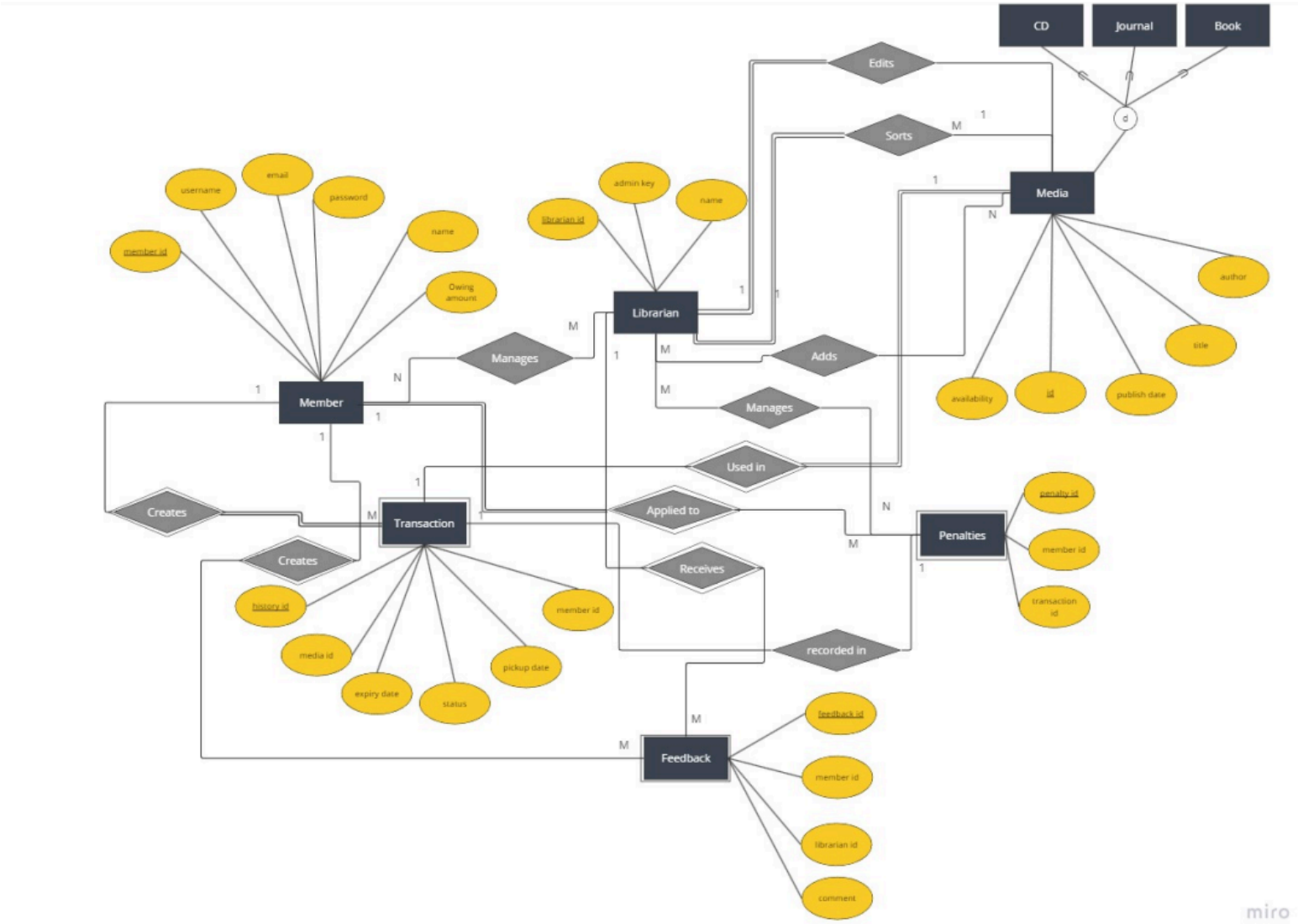



**Figure 1: ER model of library system**

## Entities in the System

**Media**

| Attribute | Attribute type |
|---|---|
| author | varchar |
| title | varchar |
| publish date | Date |
| id | Int (primary key) |
| availability | Int |

**Transactiondetails**

| Attribute | Attribute type |
|---|---|
| history id | Int (primary key) |
| media id | Int (Foreign key) |
| expiry date | Date |
| status | Int |
| pickup date | Date |
| member id | Int (Foreign key) |

**Penalties**

| Attribute | Attribute type |
|---|---|
| penalty id | Int (Primary key) |
| member id | Int (Foreign key) |
| transaction id | Int (Foreign key) |

**Members**

| Attribute | Attribute type |
|---|---|
| MemID | Int (Primary key) |
| Username | varchar |
| Pass | varchar |
| Email | varchar |
| Name | varchar |
| AmountOwed | Double |

**Feedback**

| Attribute | Attribute type |
|---|---|
| feedback id | Int (Primary key) |
| member id | Int (Foreign key) |
| librarian id | Int (Foreign key) |
| comment | String |
| Stars | Int |

**Librarian**

| Attribute | Attribute type |
|---|---|
| LibID | Int (Primary key) |
| AdminKey | Int (Primary key) |
| Name | varchar |

In Figure 1, our team has identified six main entities that play integral roles in our library system:

- **Members:** MEMBERS represent registered users who have access to library services. They are identified by a unique member ID and use usernames and passwords for authentication within the system. This entity is fundamental to user management and access control.

- **Media:** This entity serves as a representation of various types of library content, encompassing items such as books, CDs, and journal articles. It includes the author's name, title, publish date, and availability. Media plays a pivotal role in the library system, enabling users to borrow, reserve, and interact with the available resources.

- **Transaction:** The Transaction entity records the various transactions within the library system, including borrowing, reservation, and return activities. It maintains essential information such as transaction history ID, media ID, expiry date, status, pickup date, and member ID. These records help in tracking the interactions between members and library resources.

- **Penalties:** The Penalties entity manages penalties imposed on library members for overdue items. It maintains a record of penalties associated with specific members and transactions. This entity plays a crucial role in ensuring the timely return of borrowed items.

- **Feedback:** The Feedback entity facilitates communication between library users and librarians. It receives input from members, allowing them to express their thoughts and concerns. This feedback is associated with specific members and librarians, enabling effective communication and response.

- **Librarians**: Librarians are the administrative backbone of the library system. They manage member accounts, facilitate password resets, handle user feedback, and maintain the library's media inventory. They have one-to-many relationships with members, media, and feedback, allowing them to oversee these critical aspects of the system efficiently.

# ASSIGNMENT 3: RELATIONAL MODEL AND SQL STATEMENTS

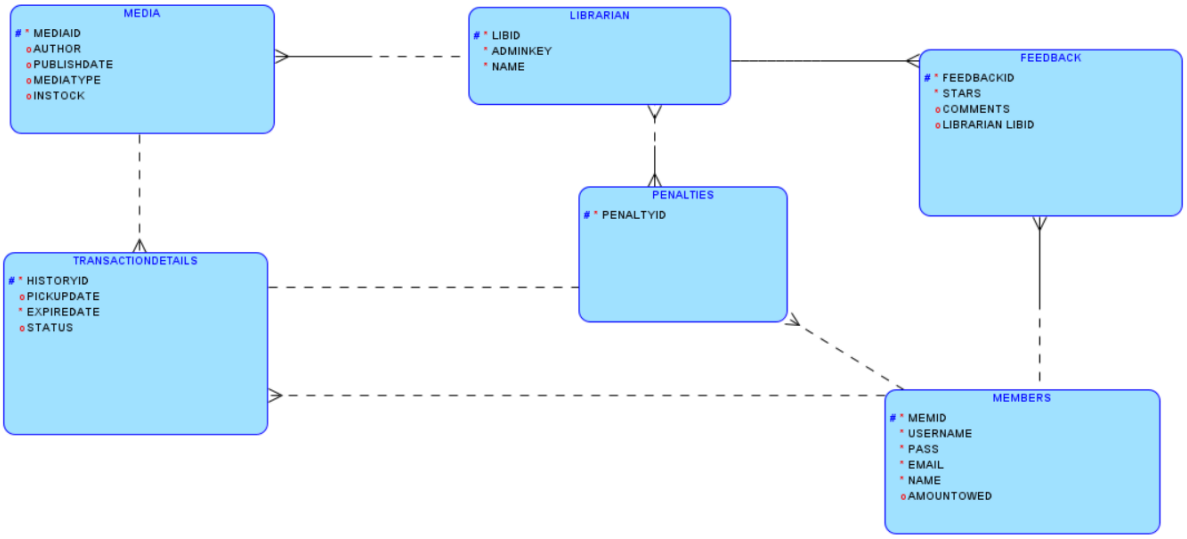



**Figure 2: Logical model of library management**

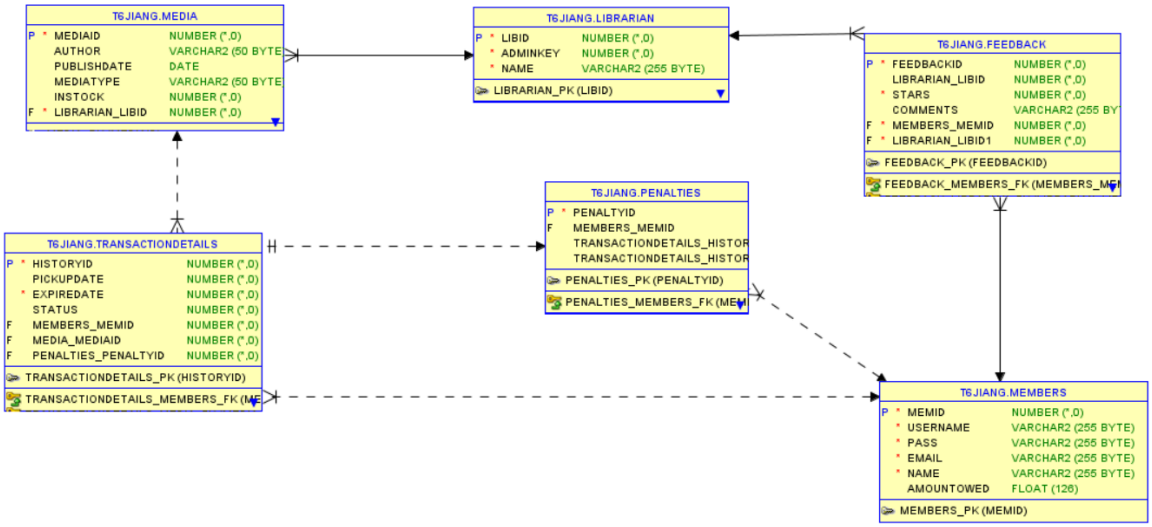



**Figure 3: Relational model of library management**

## SQL TABLE CREATION STATEMENTS

```
CREATE TABLE Media(
  MediaID INT PRIMARY KEY,
  author VARCHAR(50) DEFAULT 'Unknown',
  MediaTitle VARCHAR(50) NOT NULL,
  publishDate DATE DEFAULT '1900-01-01',
  MediaType VARCHAR(50) DEFAULT
'undefined',
inStock int DEFAULT 1);
```

```
CREATE TABLE Members (
MemID int             PRIMARY KEY,
Username varchar(255)   NOT NULL,
Pass varchar(255)       NOT NULL,
Email varchar(255)      NOT NULL,
Name varchar(255)       NOT NULL,
AmountOwed double PRECISION     Default 0.0
);
```

```
CREATE TABLE Librarian (
LibID int           PRIMARY KEY,
AdminKey int            NOT NULL,
Name varchar(255)       NOT NULL
);
```

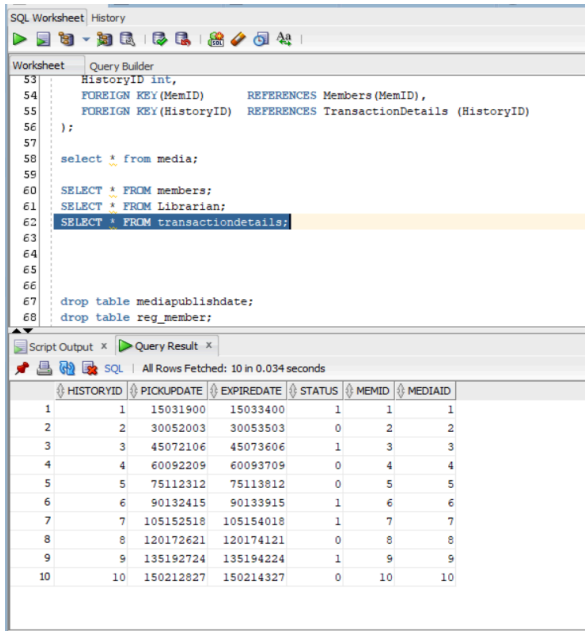
```
CREATE TABLE TransactionDetails (
HistoryID int         PRIMARY KEY,
PickupDate Date         DEFAULT '1900-01-01',
ExpireDate Date         NOT NULL,
Status int            Default 0,
MemID int,
MediaID int,
FOREIGN KEY(MemID)     REFERENCES
Members(MemID),
FOREIGN KEY(MediaID)   REFERENCES
Media (MediaID),
CHECK (Status >= 0 AND Status <= 4)
);
```

```
CREATE TABLE Penalties(
PenaltyID int           PRIMARY KEY,
MemID int,
HistoryID int,
FOREIGN KEY(MemID)      REFERENCES
Members(MemID),
FOREIGN KEY(HistoryID)  REFERENCES
TransactionDetails (HistoryID)
);
```

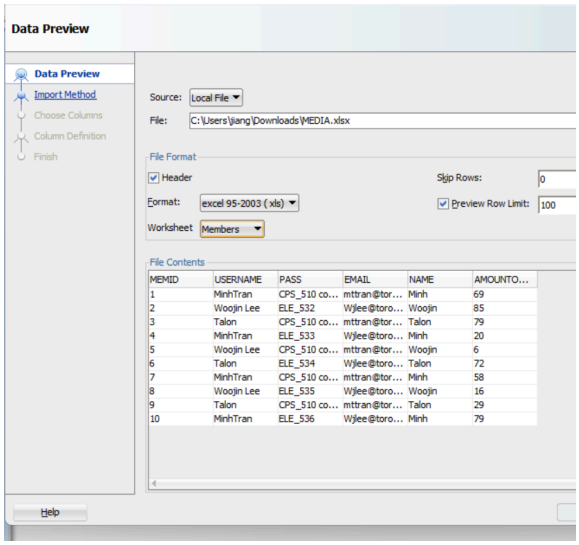
```
CREATE TABLE Feedback (
FeedbackID int          PRIMARY KEY,
MemID int,
LibID int,
Stars int               NOT Null,
Comments varChar(255)    Default 'No Feedback',
FO
REIGN KEY(MemID)      REFERENCES
Members(MemID),
FOREIGN KEY(LibID)      REFERENCES
Librarian (LibID),
CHECK (Stars >= 0 AND Stars <= 5)
);
```

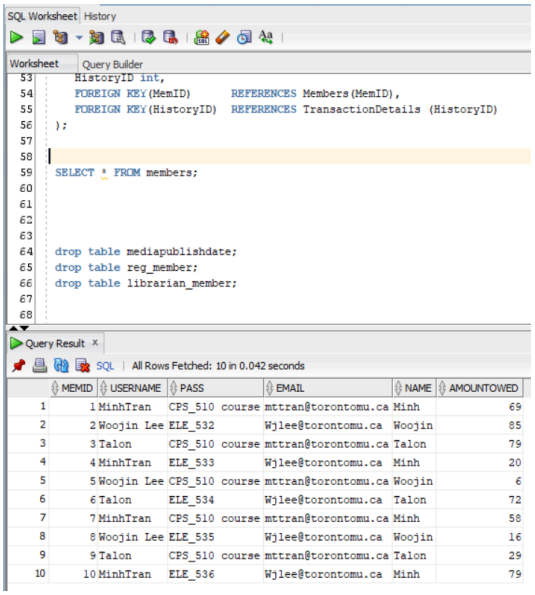
**Figure 4: SQL code for the Library System**

## TransactionDetails

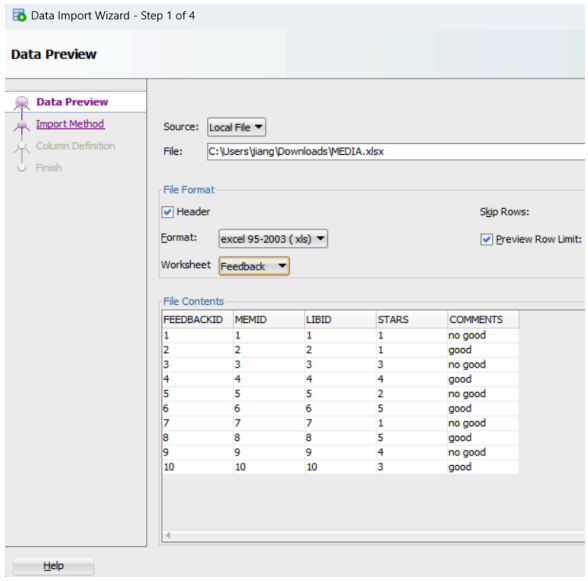


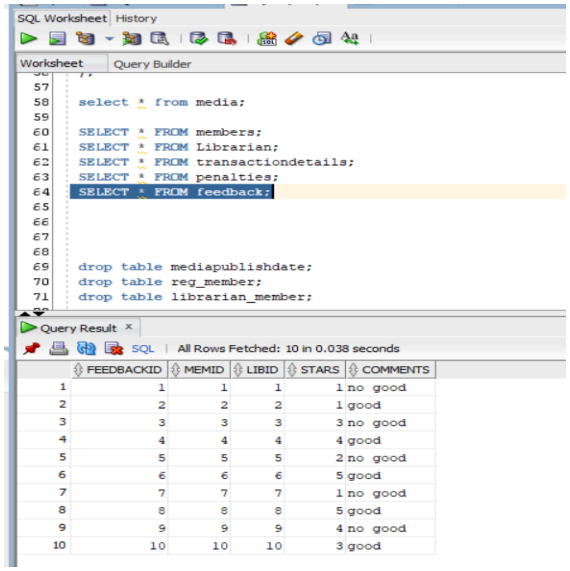

## Members



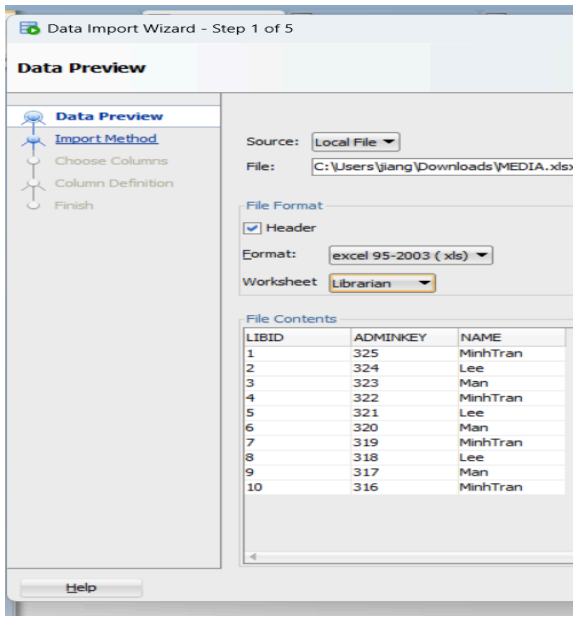



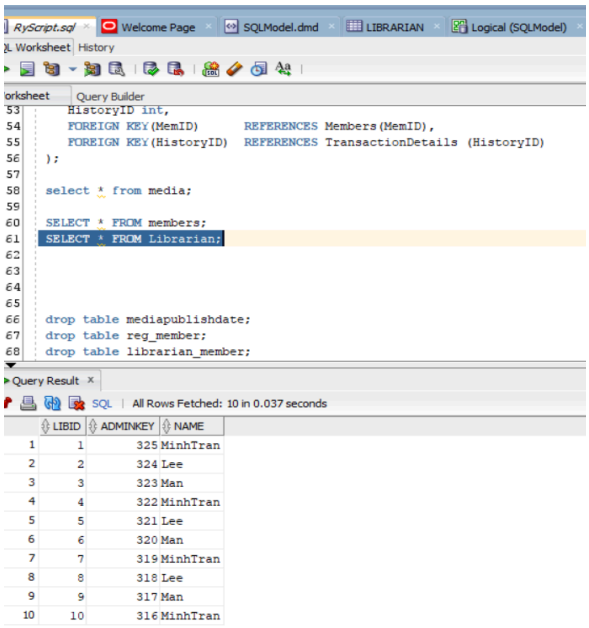
## Feedback




## Librarian



## Media

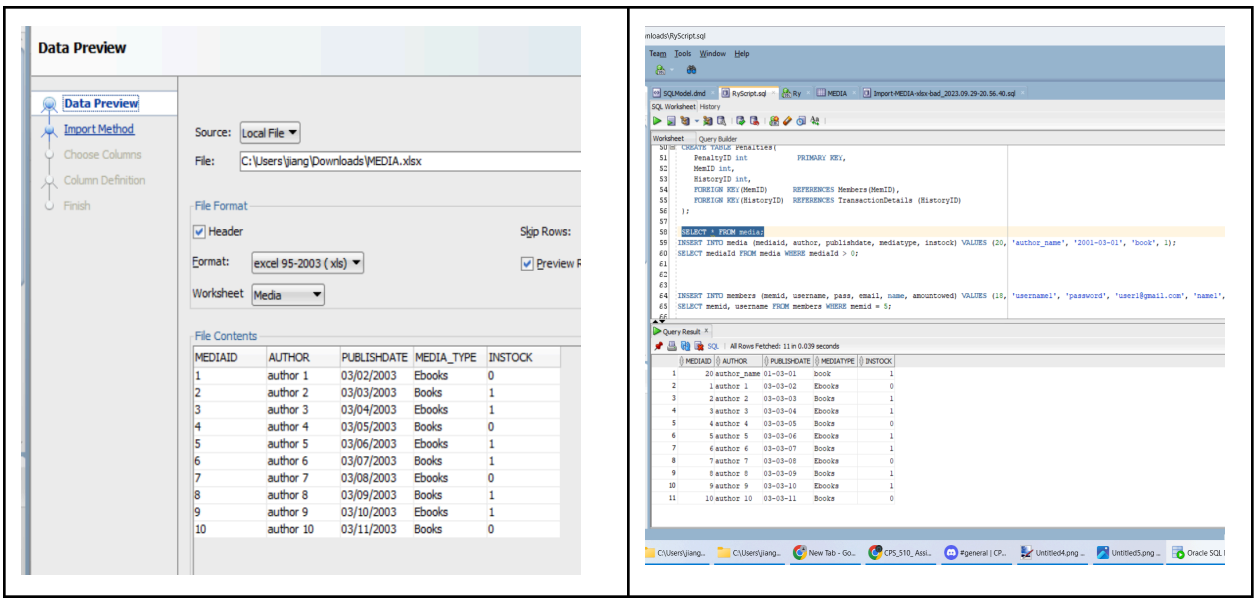



**Figure 5: Importing Data into SQL using Excel**

# ASSIGNMENT 4: SAMPLE SELECT AND QUERIES IN SQL

## Sample select queries for table Media

```
Select * FROM Media ORDER BY INSTOCK ASC;                    //If the midia is INSTOCK, it get ordered on the top.
```

스크립트 출력 | 질의 결과

SQL | 인출된 모든 행: 20(0.093초)

| | MEDIAID | MEDIATITLE | AUTHOR | PUBLISHDATE | MEDIATYPE | INSTOCK |
|---|---|---|---|---|---|---|
| 1 | 1 | The Science of Snails | Charles Hadwell | 23/02/01 | Book | 0 |
| 2 | 18 | Lisp for 1990 | Moris Lisp | 03/12/31 | Book | 0 |
| 3 | 17 | How to build a nuclear reactor | Charles Hadwell | 00/04/30 | CD | 0 |
| 4 | 7 | NEAR API | Matt Shell | 21/11/08 | Book | 0 |
| 5 | 10 | 21st Century Fashion | Sarah Crown | 13/03/26 | Book | 0 |
| 6 | 4 | Electromagnetic waves | Marty small | 11/08/19 | Journal | 0 |
| 7 | 13 | Atomic Science | Kelly Spencer | 90/02/07 | Book | 0 |
| 8 | 12 | Iceberg melting | Lawrence Walker | 90/03/07 | Journal | 1 |
| 9 | 14 | Lisp is Best | Kevin Spencer | 90/11/25 | Book | 1 |
| 10 | 15 | Functional Programming with Lisp | Mark Gotet | 00/06/17 | Book | 1 |
| 11 | 16 | C Language History | Kyle Wang | 00/05/18 | Book | 1 |
| 12 | 19 | Christmas tales | Mark Rober | 20/12/31 | CD | 1 |
| 13 | 11 | Study of Slang | Kevin Spencer | 21/05/01 | Journal | 1 |
| 14 | 9 | Studio C | Matt Meese | 18/05/01 | CD | 1 |
| 15 | 8 | Fall tunes | Matt Meese | 06/03/09 | CD | 1 |
| 16 | 6 | Anthropology | Venice Hare | 08/03/02 | Journal | 1 |
| 17 | 5 | Air currents | Marty small | 00/04/07 | Journal | 1 |
| 18 | 3 | C programming | Alex Ufkes | 10/01/04 | Book | 1 |
| 19 | 20 | Exploding cucumbers | Sarah Crown | 14/05/08 | Book | 1 |
| 20 | 2 | How to grow blueberries | Charles Hadwell | 98/07/21 | Book | 1 |

1. Select * FROM Media ORDER BY INSTOCK ASC;

Comment : //If the media is INSTOCK, it gets ordered on the top.

```
Select * FROM Media ORDER BY MEDIATITLE DESC;                // Order Alphabetically (Decesnding) According to mediaTitle.
```

스크립트 출력 | 질의 결과 2

SQL | 인출된 모든 행: 20(0.082초)

| | MEDIAID | MEDIATITLE | AUTHOR | PUBLISHDATE | MEDIATYPE | INSTOCK |
|---|---|---|---|---|---|---|
| 1 | 1 | The Science of Snails | Charles Hadwell | 23/02/01 | Book | 0 |
| 2 | 11 | Study of Slang | Kevin Spencer | 21/05/01 | Journal | 1 |
| 3 | 9 | Studio C | Matt Meese | 18/05/01 | CD | 1 |
| 4 | 7 | NEAR API | Matt Shell | 21/11/08 | Book | 0 |
| 5 | 14 | Lisp is Best | Kevin Spencer | 90/11/25 | Book | 1 |
| 6 | 18 | Lisp for 1990 | Moris Lisp | 03/12/31 | Book | 0 |
| 7 | 12 | Iceberg melting | Lawrence Walker | 90/03/07 | Journal | 1 |
| 8 | 2 | How to grow blueberries | Charles Hadwell | 98/07/21 | Book | 1 |
| 9 | 17 | How to build a nuclear reactor | Charles Hadwell | 00/04/30 | CD | 0 |
| 10 | 15 | Functional Programming with Lisp | Mark Gotet | 00/06/17 | Book | 1 |
| 11 | 8 | Fall tunes | Matt Meese | 06/03/09 | CD | 1 |
| 12 | 20 | Exploding cucumbers | Sarah Crown | 14/05/08 | Book | 1 |
| 13 | 4 | Electromagnetic waves | Marty small | 11/08/19 | Journal | 0 |
| 14 | 19 | Christmas tales | Mark Rober | 20/12/31 | CD | 1 |
| 15 | 3 | C programming | Alex Ufkes | 10/01/04 | Book | 1 |
| 16 | 16 | C Language History | Kyle Wang | 00/05/18 | Book | 1 |
| 17 | 13 | Atomic Science | Kelly Spencer | 90/02/07 | Book | 0 |
| 18 | 6 | Anthropology | Venice Hare | 08/03/02 | Journal | 1 |
| 19 | 5 | Air currents | Marty small | 00/04/07 | Journal | 1 |
| 20 | 10 | 21st Century Fashion | Sarah Crown | 13/03/26 | Book | 0 |

2. Select * FROM Media ORDER BY MEDIATITLE DESC;

Comment : // Order Alphabetically (Descending) According to mediaTitle.

```
Select * FROM MEDIA ORDER BY MEDIATYPE, MEDIATITLE;                    // Order by Type first, and within the type, order by title.
```

스크립트 출력 x | 질의 결과 3 x

SQL | 인출된 모든 행: 20(0.026초)

| | MEDIAID | MEDIATITLE | AUTHOR | PUBLISHDATE | MEDIATYPE | INSTOCK |
|---|---|---|---|---|---|---|
| 1 | 10 | 21st Century Fashion | Sarah Crown | 13/03/26 | Book | 0 |
| 2 | 13 | Atomic Science | Kelly Spencer | 90/02/07 | Book | 0 |
| 3 | 16 | C Language History | Kyle Wang | 00/05/18 | Book | 1 |
| 4 | 3 | C programming | Alex Ufkes | 10/01/04 | Book | 1 |
| 5 | 20 | Exploding cucumbers | Sarah Crown | 14/05/08 | Book | 1 |
| 6 | 15 | Functional Programming with Lisp | Mark Gotet | 00/06/17 | Book | 1 |
| 7 | 2 | How to grow blueberries | Charles Hadwell | 98/07/21 | Book | 1 |
| 8 | 18 | Lisp for 1990 | Moris Lisp | 03/12/31 | Book | 0 |
| 9 | 14 | Lisp is Best | Kevin Spencer | 90/11/25 | Book | 1 |
| 10 | 7 | NEAR API | Matt Shell | 21/11/08 | Book | 0 |
| 11 | 1 | The Science of Snails | Charles Hadwell | 23/02/01 | Book | 0 |
| 12 | 19 | Christmas tales | Mark Rober | 20/12/31 | CD | 1 |
| 13 | 8 | Fall tunes | Matt Meese | 06/03/09 | CD | 1 |
| 14 | 17 | How to build a nuclear reactor | Charles Hadwell | 00/04/30 | CD | 0 |
| 15 | 9 | Studio C | Matt Meese | 18/05/01 | CD | 1 |
| 16 | 5 | Air currents | Marty small | 00/04/07 | Journal | 1 |
| 17 | 6 | Anthropology | Venice Hare | 08/03/02 | Journal | 1 |
| 18 | 4 | Electromagnetic waves | Marty small | 11/08/19 | Journal | 0 |
| 19 | 12 | Iceberg melting | Lawrence Walker | 90/03/07 | Journal | 1 |
| 20 | 11 | Study of Slang | Kevin Spencer | 21/05/01 | Journal | 1 |

3. Select * FROM MEDIA ORDER BY MEDIATYPE, MEDIATITLE;

Comment : // Order by Type first, and within the type, order by title.

```
Select DISTINCT Author FROM MEDIA;                    // Display Unique Authors within the media database.
```

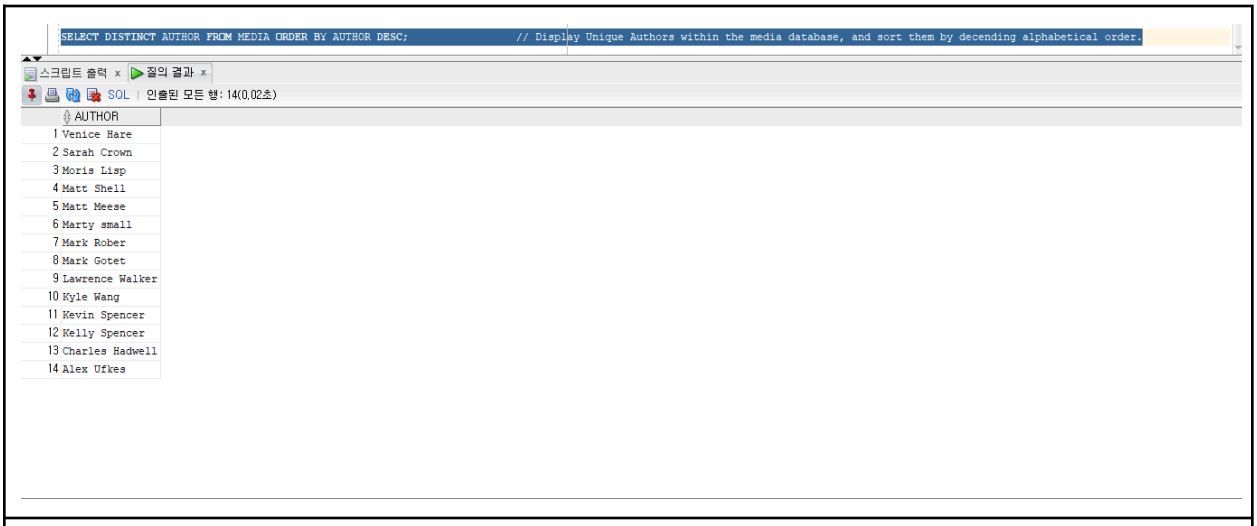
스크립트 출력 x | 질의 결과 4 x

SQL | 인출된 모든 행: 14(0.019초)

| | AUTHOR |
|---|---|
| 1 | Marty small |
| 2 | Sarah Crown |
| 3 | Mark Gotet |
| 4 | Venice Hare |
| 5 | Lawrence Walker |
| 6 | Matt Meese |
| 7 | Kevin Spencer |
| 8 | Matt Shell |
| 9 | Mark Rober |
| 10 | Alex Ufkes |
| 11 | Kyle Wang |
| 12 | Moris Lisp |
| 13 | Charles Hadwell |
| 14 | Kelly Spencer |

4. Select DISTINCT Author FROM MEDIA;

Comment // Display Unique Authors within the media database.

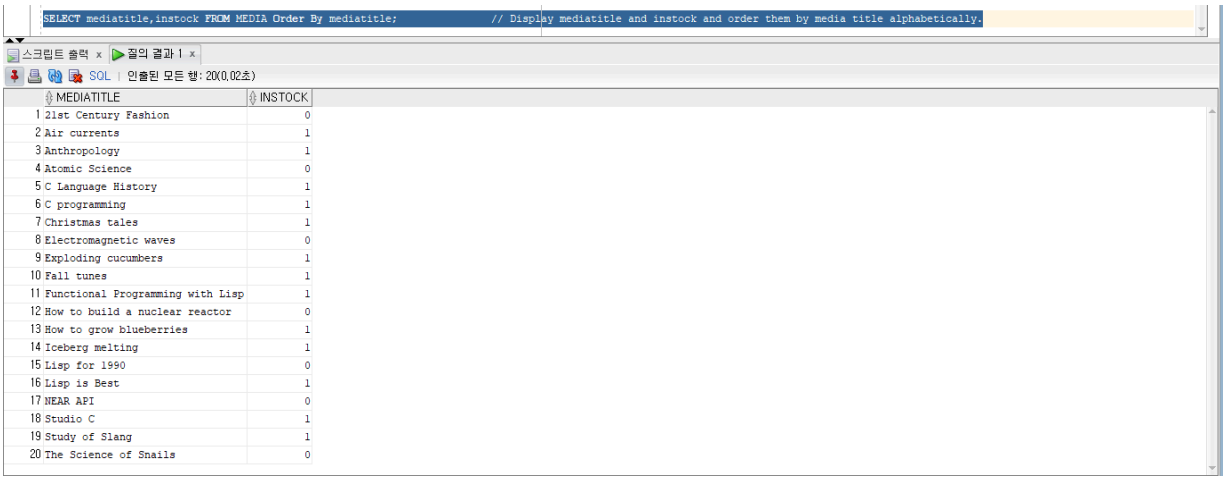



5. SELECT DISTINCT AUTHOR FROM MEDIA ORDER BY AUTHOR DESC;

Comment // Display Unique Authors within the media database, and sort them by descending alphabetical order.


6. SELECT MEDIATYPE as "Media Type" , COUNT(*) AS "Total Count" FROM MEDIA GROUP BY MEDIATYPE;

Comment // Display Only the media type, and count how many duplicated media types there are.



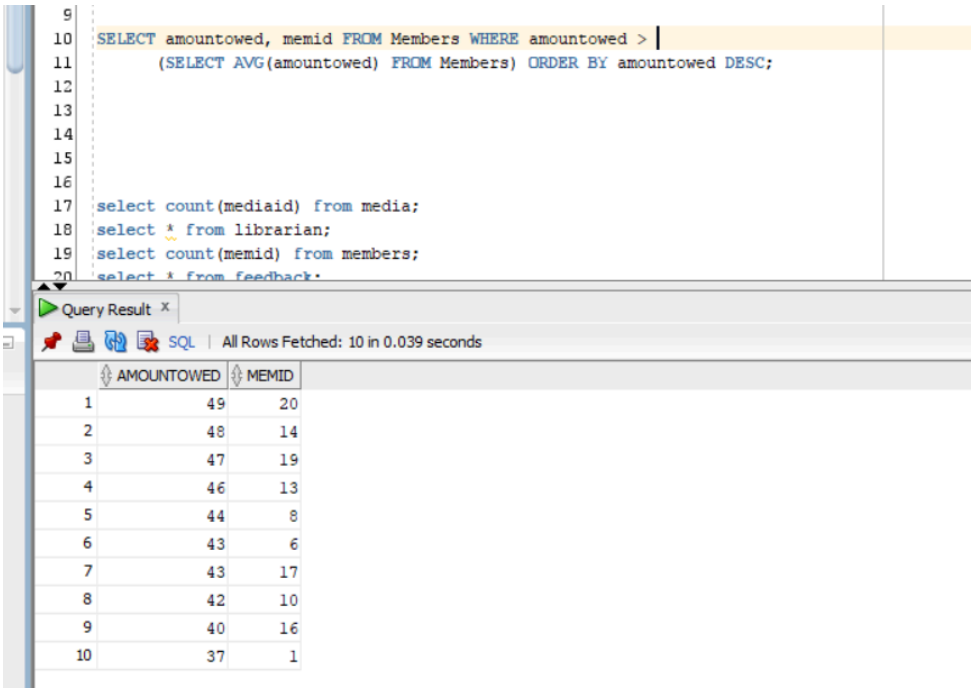

| | MEDIATITLE | INSTOCK |
|---|---|---|
| 1 | 21st Century Fashion | 0 |
| 2 | Air currents | 1 |
| 3 | Anthropology | 1 |
| 4 | Atomic Science | 0 |
| 5 | C Language History | 1 |
| 6 | C programming | 1 |
| 7 | Christmas tales | 1 |
| 8 | Electromagnetic waves | 0 |
| 9 | Exploding cucumbers | 1 |
| 10 | Fall tunes | 1 |
| 11 | Functional Programming with Lisp | 1 |
| 12 | How to build a nuclear reactor | 0 |
| 13 | How to grow blueberries | 1 |
| 14 | Iceberg melting | 1 |
| 15 | Lisp for 1990 | 0 |
| 16 | Lisp is Best | 1 |
| 17 | NEAR API | 0 |
| 18 | Studio C | 1 |
| 19 | Study of Slang | 1 |
| 20 | The Science of Snails | 0 |

7. SELECT mediatitle,instock FROM MEDIA Order By mediatitle;

Comment // Display mediatitle and instock and order them by media title alphabetically.

## Sample select statements for Members

--Selecting all users where the amount owed is greater than the average
SELECT amountowed, memid FROM Members WHERE amountowed >
(SELECT AVG(amountowed) FROM Members) ORDER BY amountowed DESC;



```
SELECT amountowed, memid FROM Members WHERE amountowed >
      (SELECT AVG(amountowed) FROM Members) ORDER BY amountowed DESC;



select count(mediaid) from media;
select * from librarian;
select count(memid) from members;
```

Query Result | SQL | All Rows Fetched: 10 in 0.039 seconds

| | AMOUNTOWED | MEMID |
|---|---|---|
| 1 | 49 | 20 |
| 2 | 48 | 14 |
| 3 | 47 | 19 |
| 4 | 46 | 13 |
| 5 | 44 | 8 |
| 6 | 43 | 6 |
| 7 | 43 | 17 |
| 8 | 42 | 10 |
| 9 | 40 | 16 |
| 10 | 37 | 1 |

-- Selecting all users using gmail for their email
SELECT username, email, name FROM members WHERE
email LIKE '%gmail.com';

```
9
10  --selecting all users using gmail
11  SELECT username, email, name FROM members WHERE
12      email LIKE '%gmail.com';
13
14
15  --finding someones password
16  SELECT distinct name, email, pass as Password FROM members WHERE email LIKE 'wwiend%'
17      AND name LIKE 'John%' ORDER BY name DESC;
18
19
20  --getting the guys with penalties
21  SELECT DISTINCT members.memid, members.username FROM members
22   INNER JOIN penalties ON penalties.memid = members.memid ORDER BY
23      members.username ASC;
```

Script Output | Query Result

SQL | All Rows Fetched: 18 in 0.043 seconds

| | USERNAME | EMAIL | NAME |
|---|---|---|---|
| 1 | wwiegand | wwiend@gmail.com | John |
| 2 | schinner.gladyce | schice@gmail.com | Schinner |
| 3 | orie.armstrong | orieng@gmail.com | Orie |
| 4 | jwalsh | jwalsh@gmail.com | Josh |
| 5 | ljacobson | ljacon@gmail.com | Jacob |
| 6 | andy.gerlach | andych@gmail.com | Andy |
| 7 | rdubuque | rdubue@gmail.com | Ruby |
| 8 | brekke.robb | brekbb@gmail.com | Brekke |
| 9 | bbartoletti | imbaroque@gmail.com | Bart |
| 10 | little.noel | noel21@gmail.com | Noel |

– Finding the password for a member

SELECT distinct name, email, pass FROM members WHERE email LIKE 'wwiend%'
AND name LIKE 'John%' ORDER BY name DESC;

```
13
14
15  --finding someones password
16  SELECT distinct name, email, pass as Password FROM members WHERE email LIKE 'wwiend%'
17      AND name LIKE 'John%' ORDER BY name DESC;
18
19
20  --getting the guys with penalties
21  SELECT DISTINCT members.memid, members.username FROM members
22   INNER JOIN penalties ON penalties.memid = members.memid ORDER BY
23      members.username ASC;
24
25
26  --all members who have borrowed something
```

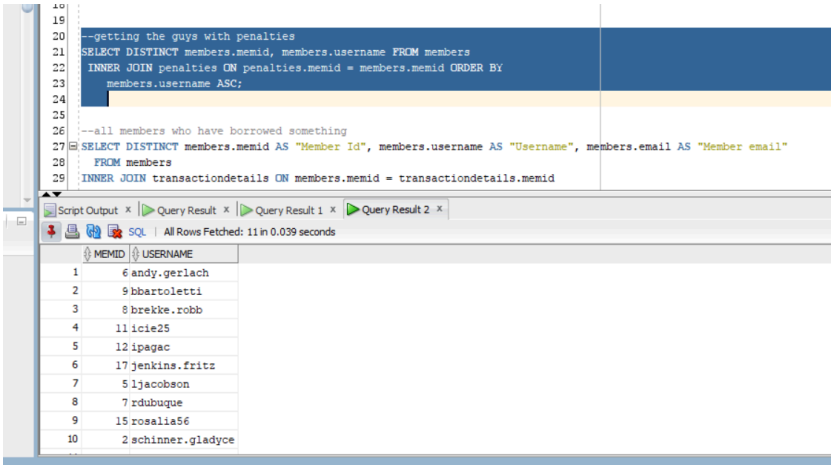
Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.123 seconds

| | NAME | EMAIL | PASSWORD |
|---|---|---|---|
| 1 | John | wwiend@gmail.com | FCFLRKUI |

- - Finding all members who have an entry in the penalties table.

```
SELECT DISTINCT members.memid, members.username FROM members
 INNER JOIN penalties ON penalties.memid = members.memid ORDER BY
    members.username ASC;
```



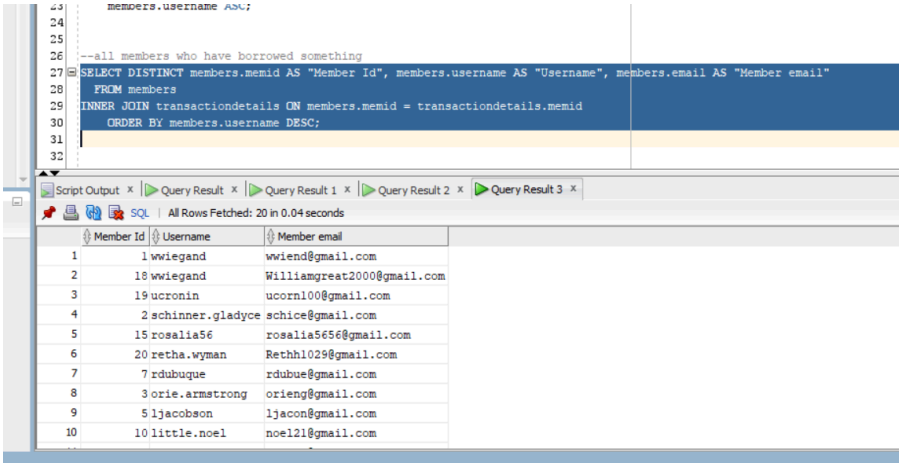
```
--getting the guys with penalties
SELECT DISTINCT members.memid, members.username FROM members
 INNER JOIN penalties ON penalties.memid = members.memid ORDER BY
    members.username ASC;

--all members who have borrowed something
SELECT DISTINCT members.memid AS "Member Id", members.username AS "Username", members.email AS "Member email"
  FROM members
INNER JOIN transactiondetails ON members.memid = transactiondetails.memid
```

Script Output | Query Result | Query Result 1 | Query Result 2 | All Rows Fetched: 11 in 0.039 seconds

| | MEMID | USERNAME |
|---|---|---|
| 1 | 6 | andy.gerlach |
| 2 | 9 | bbartoletti |
| 3 | 8 | brekke.robb |
| 4 | 11 | icie25 |
| 5 | 12 | ipagac |
| 6 | 17 | jenkins.fritz |
| 7 | 5 | ljacobson |
| 8 | 7 | rdubuque |
| 9 | 15 | rosalia56 |
| 10 | 2 | schinner.gladyce |

– Finding all members who have created a transaction

```
SELECT DISTINCT members.memid AS "Member Id", members.username AS "Username", members.email AS
"Member email"
  FROM members
INNER JOIN transactiondetails ON members.memid = transactiondetails.memid
   ORDER BY members.username DESC;
```



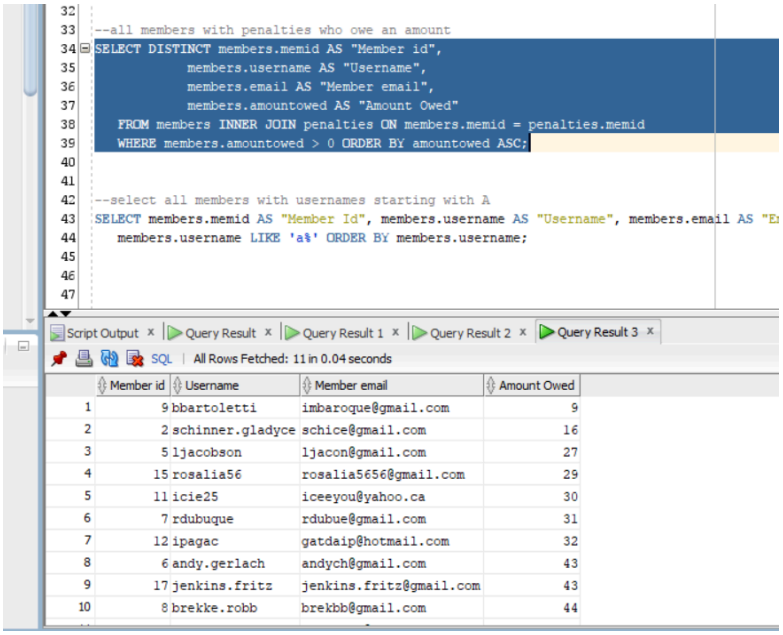
```
    members.username ASC;

--all members who have borrowed something
SELECT DISTINCT members.memid AS "Member Id", members.username AS "Username", members.email AS "Member email"
  FROM members
INNER JOIN transactiondetails ON members.memid = transactiondetails.memid
   ORDER BY members.username DESC;
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | All Rows Fetched: 20 in 0.04 seconds

| | Member Id | Username | Member email |
|---|---|---|---|
| 1 | 1 | wwiegand | wwiend@gmail.com |
| 2 | 18 | wwiegand | Williamgreat2000@gmail.com |
| 3 | 19 | ucronin | ucornl00@gmail.com |
| 4 | 2 | schinner.gladyce | schice@gmail.com |
| 5 | 15 | rosalia56 | rosalia5656@gmail.com |
| 6 | 20 | retha.wyman | Rethhl029@gmail.com |
| 7 | 7 | rdubuque | rdubue@gmail.com |
| 8 | 3 | orie.armstrong | orieng@gmail.com |
| 9 | 5 | ljacobson | ljacon@gmail.com |
| 10 | 10 | little.noel | noel21@gmail.com |

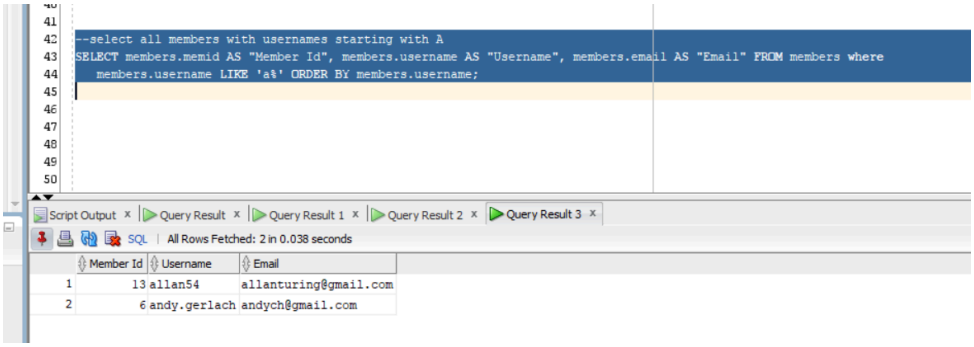
– Finding all members with penalties who owe an amount

```
SELECT DISTINCT members.memid AS "Member id",
        members.username AS "Username",
        members.email AS "Member email",
        members.amountowed AS "Amount Owed"
  FROM members INNER JOIN penalties ON members.memid = penalties.memid
  WHERE members.amountowed > 0 ORDER BY amountowed ASC;
```




– Finding all members whose username starts with A.

```
SELECT members.memid AS "Member Id", members.username AS "Username", members.email AS "Email" FROM
members where
  members.username LIKE 'a%' ORDER BY members.username;
```



## Penalties

### /* 1. Search Penalties with a specific Member ID*/

select *from Penalties Where MemID = 5;

```
102  select * from Penalties;
103  /* 1. Search Penalties with a specific Member ID*/
104  select *from Penalties Where MemID = 5;
105
106  /* 2. */
```

Query Result | Script Output | Query Result 1

SQL | All Rows Fetched: 2 in 0.02 seconds

| | PENALTYID | MEMID | HISTORYID |
|---|---|---|---|
| 1 | 6 | 5 | 6 |
| 2 | 7 | 5 | 11 |

---

**2. List members with Penalties by Join statement**

select Members.*, Penalties.*
From Members
Join Penalties ON Members.MemID = Penalties.MemID;

```
/* 2. List member with Penalties by Join statement*/
select Members.*, Penalties.*
From Members
Join Penalties ON Members.MemID = Penalties.MemID;

/* 3. */
/* 4. */
/* 5. */
```

ery Result × | Script Output × | Query Result 1 ×

SQL | All Rows Fetched: 20 in 0.092 seconds

| | MEMID | USERNAME | PASS | EMAIL | NAME | AMOUNTOWED | PENALTYID | MEMID_1 | HISTORYID |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | orie.armstrong | KYROYOZJ | orieng@gmail.com | Orie | 16 | 5 | 3 | 15 |
| 2 | 4 | jwalsh | SIQMKMSK | jwalsh@gmail.com | Josh | 26 | 17 | 4 | 7 |
| 3 | 4 | jwalsh | SIQMKMSK | jwalsh@gmail.com | Josh | 26 | 1 | 4 | 2 |
| 4 | 5 | ljacobson | OBCUDFZN | ljacon@gmail.com | Jacob | 35 | 7 | 5 | 11 |
| 5 | 5 | ljacobson | OBCUDFZN | ljacon@gmail.com | Jacob | 35 | 6 | 5 | 6 |
| 6 | 7 | rdubuque | LGJCVQBK | rdubue@gmail.com | Ruby | 11 | 13 | 7 | 19 |
| 7 | 8 | brekke.robb | DXKZJZAX | brekbb@gmail.com | Brekke | 0 | 20 | 8 | 20 |
| 8 | 9 | bbartoletti | LFERCCLJ | imbaroque@gmail.com | Bart | 9 | 12 | 9 | 17 |
| 9 | 9 | bbartoletti | LFERCCLJ | imbaroque@gmail.com | Bart | 9 | 9 | 9 | 8 |
| 0 | 9 | bbartoletti | LFERCCLJ | imbaroque@gmail.com | Bart | 9 | 2 | 9 | 4 |
| 1 | 10 | little.noel | SJHLJWOT | noel21@gmail.com | Noel | 0 | 18 | 10 | 18 |
| 2 | 13 | allan54 | ZAEXWAGN | allanturing@gmail.com | Allan | 17 | 15 | 13 | 13 |

**3. retrieving distinct HistoryID values associated with penalties**

SELECT DISTINCT HistoryID FROM Penalties;

```
111  /*retrieving distinct HistoryID values associated with penalties */
112  SELECT DISTINCT HistoryID FROM Penalties;
113
114
115  /* 4. Penalties for Transaction according to the date range*/
116  SELECT Penalties.*
```

Query Result × | Script Output × | Query Result 1 ×

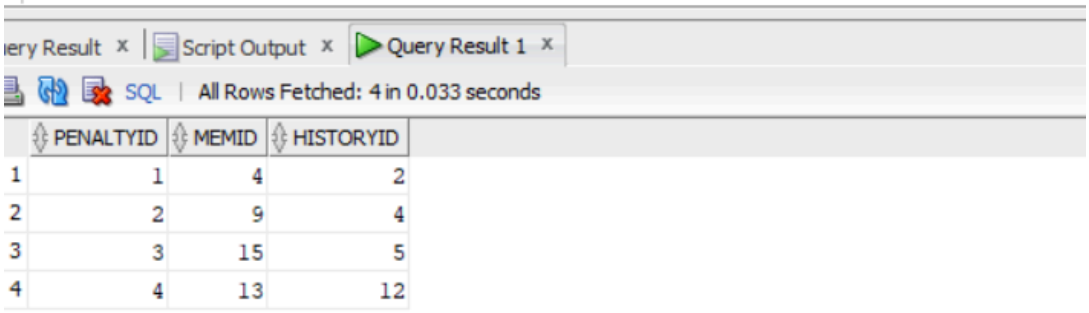
SQL | All Rows Fetched: 20 in 0.041 seconds

| | HISTORYID |
|---|---|
| 1 | 1 |
| 2 | 6 |
| 3 | 11 |
| 4 | 13 |
| 5 | 2 |
| 6 | 14 |
| 7 | 20 |
| 8 | 4 |
| 9 | 5 |
| 10 | 8 |
| 11 | 17 |
| 12 | 3 |

**4. Penalties for Transaction according to the date range**

SELECT Penalties.*
FROM Penalties
JOIN TransactionDetails ON Penalties.HistoryID = TransactionDetails.HistoryID
WHERE TransactionDetails.PickupDate BETWEEN '01-JAN-23' AND '05-JAN-23';

```
/* 4. Penalties for Transaction according to the date range*/

SELECT Penalties.*
FROM Penalties
JOIN TransactionDetails ON Penalties.HistoryID = TransactionDetails.HistoryID
WHERE TransactionDetails.PickupDate BETWEEN '01-JAN-23' AND '05-JAN-23';


/* 5. */
/* 6. */
/* 7. */
```




| | PENALTYID | MEMID | HISTORYID |
|---|---|---|---|
| 1 | 1 | 4 | 2 |
| 2 | 2 | 9 | 4 |
| 3 | 3 | 15 | 5 |
| 4 | 4 | 13 | 12 |

---

**5 Count the Number of Penalties per Member:**

SELECT MemID, COUNT(*) AS PenaltyCount
FROM Penalties
GROUP BY MemID;

```
121  /* 5 Count the Number of Penalties per Member:
122
123  SELECT MemID, COUNT(*) AS PenaltyCount
124  FROM Penalties
125  GROUP BY MemID;
126
127  /* 6. */
128  /* 7. */
```

Query Result | Script Output | Query Result 1

SQL | All Rows Fetched: 12 in 0.022 seconds

| | MEMID | PENALTYCOUNT |
|---|---|---|
| 1 | 13 | 2 |
| 2 | 4 | 2 |
| 3 | 5 | 2 |
| 4 | 17 | 1 |
| 5 | 8 | 1 |
| 6 | 3 | 1 |
| 7 | 18 | 2 |
| 8 | 7 | 1 |
| 9 | 9 | 3 |
| 10 | 15 | 1 |
| 11 | 19 | 3 |
| 12 | 10 | 1 |

**6. retrieving all penalties from the "Penalties" table and order them by the "PenaltyID" in ascending order.**

SELECT * FROM Penalties
ORDER BY PenaltyID ASC;

```
126 /* 6.  retrieveing all penalties from the "Penalties" table and orders them by the "PenaltyID" in ascending order.*/
127 SELECT * FROM Penalties
128 ORDER BY PenaltyID ASC;
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 20 in 0.028 seconds

| | PENALTYID | MEMID | HISTORYID |
|---|---|---|---|
| 1 | 1 | 4 | 2 |
| 2 | 2 | 9 | 4 |
| 3 | 3 | 15 | 5 |
| 4 | 4 | 13 | 12 |
| 5 | 5 | 3 | 15 |
| 6 | 6 | 5 | 6 |
| 7 | 7 | 5 | 11 |
| 8 | 8 | 18 | 10 |
| 9 | 9 | 9 | 8 |
| 10 | 10 | 19 | 9 |
| 11 | 11 | 18 | 1 |
| 12 | 12 | 9 | 17 |

---

**7. Retrieving penalties for a specific transaction (specified by HistoryID).**

SELECT * FROM Penalties WHERE HistoryID = 12;

```
128 /* 7. Retrieving penalties for a specific transaction (specified by HistoryID)./
129 SELECT * FROM Penalties WHERE HistoryID = 12;
130
131
132
```

Query Result | Script Output | Query Result 1

SQL | All Rows Fetched: 1 in 0.029 seconds

| | PENALTYID | MEMID | HISTORYID |
|---|---|---|---|
| 1 | 4 | 13 | 12 |

## Librarian

**1. Select librarians with names starting with a specific letter**
SELECT * FROM Librarian WHERE Name LIKE 'M%';

```
/* 1 Select librarians with names starting with a specific letter*/
 SELECT * FROM Librarian WHERE Name LIKE 'M%';

/* 2 Count the total number of librarians */
```

uery Result x | Query Result 1 x | Query Result 2 x | Query Result 3 x | Query Result

SQL | All Rows Fetched: 2 in 0.023 seconds

| | LIBID | ADMINKEY | NAME |
|---|---|---|---|
| 1 | 1 | 801 | Minh Tran |
| 2 | 5 | 321 | Morgan Freeman |

**2. Count the total number of librarians**
SELECT COUNT(*) FROM Librarian;

```
76
77  /* 2 Count the total number of librarians */
78   SELECT COUNT(*) FROM Librarian;
79
80  /*3 Select librarians with distinct names ord
```

Query Result x | Query Result 1 x | Query Result 2 x | Q

SQL | All Rows Fetched: 1 in 0.035 seconds

| | COUNT(*) |
|---|---|
| 1 | 5 |

**3. Select librarians with distinct names ordered by AdminKey**
select COUNT(*) FROM ( select distinct AdminKey FROM Librarian);

```
80  /*3 Select librarians with distinct names ordered by AdminKey*/
81   select COUNT(*) FROM ( select distinct AdminKey FROM Librarian);
82
83  /* 4 Select all librarians sorted by name in ascending order*/
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 1 in 0.038 seconds

| | COUNT(*) |
|---|---|
| 1 | 5 |

**4. Select all librarians sorted by name in ascending order**
select * from Librarian order by name ASC;

```
83  /* 4 Select all librarians sorted by name in ascending order*/
84   select * from Librarian order by name ASC;
85
86  /* 5 Count the number of librarians with lowest AdminKeys*/
```

Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 5 in 0.018 seconds

| | LIBID | ADMINKEY | NAME |
|---|---|---|---|
| 1 | 3 | 323 | Ho Kim Lee |
| 2 | 4 | 322 | Kevin Jones |
| 3 | 2 | 324 | Kimberly Delamere |
| 4 | 1 | 801 | Minh Tran |
| 5 | 5 | 321 | Morgan Freeman |

**5. Count the number of librarians with lowest AdminKeys**

select * from Librarian where Adminkey = (select min(Adminkey) FROM Librarian);

```
86  /* 5 Count the number of librarians with lowest AdminKeys*/
87   select * from Librarian where Adminkey = (select min(Adminkey) FROM Librarian);
88
89  /*6  Show information of Librarian according to LibID */
```

Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.024 seconds

| | LIBID | ADMINKEY | NAME |
|---|---|---|---|
| 1 | 5 | 321 | Morgan Freeman |

**6. Show information of Librarian according to LibID**

select LibID, Name From Librarian;

```
89  /*6  Show information of Librarian according to LibID */
90    select  LibID, Name From Librarian;
91
92  /* 7 Searching for specific Librarian name */
```

Query Result

SQL | All Rows Fetched: 5 in 0.043 seconds

| | LIBID | NAME |
|---|---|---|
| 1 | 1 | Minh Tran |
| 2 | 2 | Kimberly Delamere |
| 3 | 3 | Ho Kim Lee |
| 4 | 4 | Kevin Jones |
| 5 | 5 | Morgan Freeman |

**7. Searching for specific Librarian name.**

select * from Librarian Where NAME LIKE 'Minh Tran' ;

```
/* 7 Searching for specific Librarian name */
select * from Librarian Where NAME LIKE 'Minh Tran' ;
```

y Result

SQL | All Rows Fetched: 1 in 0.013 seconds

| LIBID | ADMINKEY | NAME |
|---|---|---|
| 1 | 801 | Minh Tran |

## Transaction details

```
SELECT * FROM TransactionDetails Order By PickupDate Asc; // sort by earliest pickup dates
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 20(0.018초)

| | HISTORYID | PICKUPDATE | EXPIREDATE | STATUS | MEMID | MEDIAID |
|---|---|---|---|---|---|---|
| 1 | 2 | 23/01/02 | 23/01/09 | 1 | 1 | 13 |
| 2 | 4 | 23/01/03 | 23/01/10 | 0 | 19 | 14 |
| 3 | 5 | 23/01/04 | 23/01/11 | 1 | 12 | 15 |
| 4 | 12 | 23/01/05 | 23/01/09 | 0 | 16 | 20 |
| 5 | 15 | 23/01/06 | 23/01/14 | 0 | 6 | 1 |
| 6 | 6 | 23/01/07 | 23/01/19 | 1 | 2 | 16 |
| 7 | 11 | 23/01/08 | 23/01/09 | 1 | 15 | 17 |
| 8 | 10 | 23/01/09 | 23/01/19 | 0 | 13 | 5 |
| 9 | 8 | 23/01/10 | 23/01/22 | 1 | 5 | 6 |
| 10 | 9 | 23/01/11 | 23/01/19 | 0 | 17 | 12 |
| 11 | 1 | 23/01/12 | 23/01/20 | 0 | 11 | 19 |
| 12 | 17 | 23/01/13 | 23/01/21 | 1 | 20 | 10 |
| 13 | 19 | 23/01/14 | 23/01/22 | 1 | 9 | 11 |
| 14 | 3 | 23/01/15 | 23/01/23 | 0 | 4 | 7 |
| 15 | 13 | 23/01/16 | 23/01/24 | 1 | 14 | 8 |
| 16 | 14 | 23/01/17 | 23/01/25 | 1 | 3 | 9 |
| 17 | 7 | 23/01/18 | 23/01/26 | 1 | 8 | 18 |
| 18 | 18 | 23/01/19 | 23/01/27 | 0 | 10 | 2 |
| 19 | 16 | 23/01/20 | 23/01/28 | 1 | 18 | 3 |
| 20 | 20 | 23/01/21 | 23/01/29 | 1 | 7 | 4 |

1. SELECT * FROM TransactionDetails Order By PickupDate Asc;

Comment// sort by earliest pickup dates.

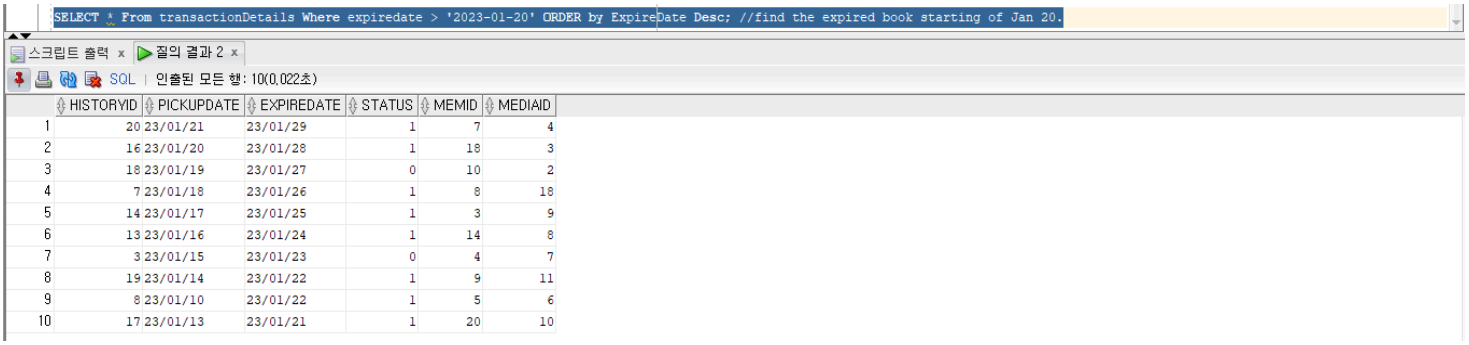
```
SELECT Status, COUNT(*) AS TotalReturned FROM TransactionDetails GROUP BY Status; // Count how many people have returned the book (1 is returned, and 0 is not)
```

스크립트 출력 x | 질의 결과 1 x

SQL | 인출된 모든 행: 2(0.024초)

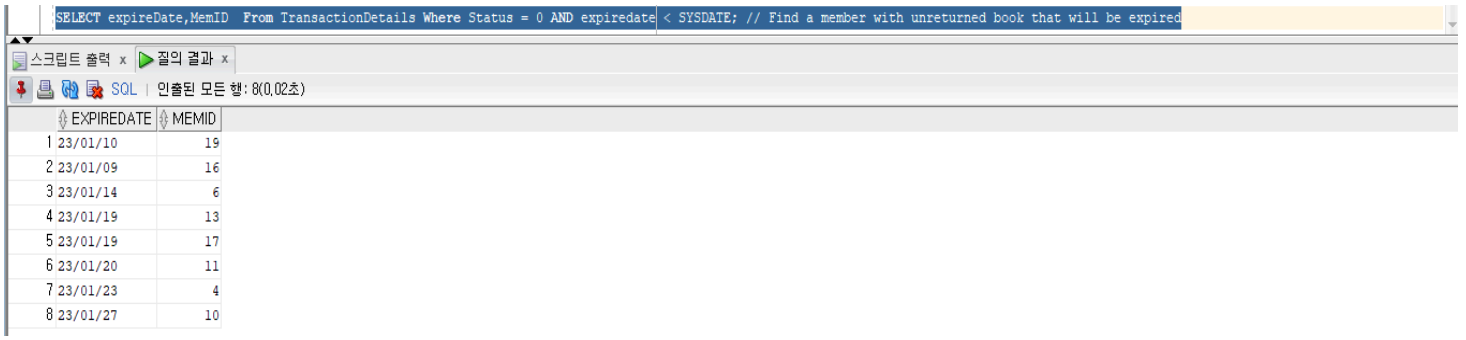
| | STATUS | TOTALRETURNED |
|---|---|---|
| 1 | 1 | 12 |
| 2 | 0 | 8 |

2. SELECT Status, COUNT(*) AS TotalReturned FROM TransactionDetails GROUP BY Status; Comment // Count how many people have returned the book (1 is returned, and 0 is not)




| | HISTORYID | PICKUPDATE | EXPIREDATE | STATUS | MEMID | MEDIAID |
|---|---|---|---|---|---|---|
| 1 | 20 | 23/01/21 | 23/01/29 | 1 | 7 | 4 |
| 2 | 16 | 23/01/20 | 23/01/28 | 1 | 18 | 3 |
| 3 | 18 | 23/01/19 | 23/01/27 | 0 | 10 | 2 |
| 4 | 7 | 23/01/18 | 23/01/26 | 1 | 8 | 18 |
| 5 | 14 | 23/01/17 | 23/01/25 | 1 | 3 | 9 |
| 6 | 13 | 23/01/16 | 23/01/24 | 1 | 14 | 8 |
| 7 | 3 | 23/01/15 | 23/01/23 | 0 | 4 | 7 |
| 8 | 19 | 23/01/14 | 23/01/22 | 1 | 9 | 11 |
| 9 | 8 | 23/01/10 | 23/01/22 | 1 | 5 | 6 |
| 10 | 17 | 23/01/13 | 23/01/21 | 1 | 20 | 10 |

3. SELECT * From transactionDetails Where expiredate > '2023-01-20' ORDER by ExpireDate Desc;
Comment //find the expired book starting Jan 20.




| | EXPIREDATE | MEMID |
|---|---|---|
| 1 | 23/01/10 | 19 |
| 2 | 23/01/09 | 16 |
| 3 | 23/01/14 | 6 |
| 4 | 23/01/19 | 13 |
| 5 | 23/01/19 | 17 |
| 6 | 23/01/20 | 11 |
| 7 | 23/01/23 | 4 |
| 8 | 23/01/27 | 10 |

4. SELECT expireDate,MemID From TransactionDetails Where Status = 0 AND expiredate < SYSDATE;
Comment // Find a member with unreturned book that will be expired

```
SELECT DISTINCT pickupdate From transactionDetails; // Find unique pickupdate (ALL of the variable have distinct values)
```

스크립트 출력 | 질의 결과 5

SQL | 인출된 모든 행: 20(0.018초)

| | PICKUPDATE |
|---|---|
| 1 | 23/01/10 |
| 2 | 23/01/12 |
| 3 | 23/01/13 |
| 4 | 23/01/15 |
| 5 | 23/01/20 |
| 6 | 23/01/03 |
| 7 | 23/01/21 |
| 8 | 23/01/05 |
| 9 | 23/01/16 |
| 10 | 23/01/02 |
| 11 | 23/01/08 |
| 12 | 23/01/09 |
| 13 | 23/01/07 |
| 14 | 23/01/11 |
| 15 | 23/01/14 |
| 16 | 23/01/17 |
| 17 | 23/01/18 |
| 18 | 23/01/19 |
| 19 | 23/01/04 |
| 20 | 23/01/06 |

5. SELECT DISTINCT pickupdate From transactionDetails;

Comment // Find unique pickupdate (ALL of the variable have distinct values)

```
SELECT * From Transactiondetails WHERE Pickupdate BETWEEN '2023-01-10' AND '2023-01-15'; //Pickupdate between jan 10 to jan 15.
```

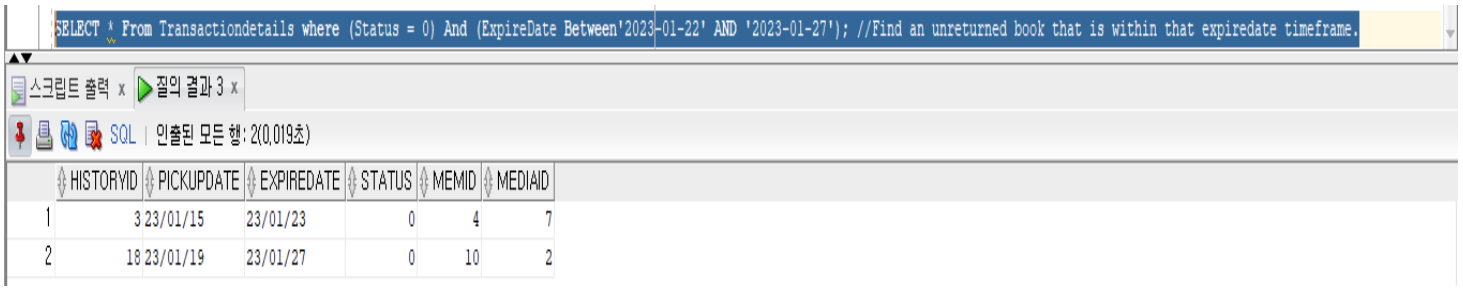
스크립트 출력 | 질의 결과 2

SQL | 인출된 모든 행: 6(0.018초)

| | HISTORYID | PICKUPDATE | EXPIREDATE | STATUS | MEMID | MEDIAID |
|---|---|---|---|---|---|---|
| 1 | 8 | 23/01/10 | 23/01/22 | 1 | 5 | 6 |
| 2 | 9 | 23/01/11 | 23/01/19 | 0 | 17 | 12 |
| 3 | 1 | 23/01/12 | 23/01/20 | 0 | 11 | 19 |
| 4 | 17 | 23/01/13 | 23/01/21 | 1 | 20 | 10 |
| 5 | 19 | 23/01/14 | 23/01/22 | 1 | 9 | 11 |
| 6 | 3 | 23/01/15 | 23/01/23 | 0 | 4 | 7 |

6. SELECT * From Transactiondetails WHERE Pickupdate BETWEEN '2023-01-10' AND '2023-01-15';

Comment //Pickupdate between Jan 10 to Jan 15.




7. SELECT * From Transaction details where (Status = 0) And (ExpireDate Between 2023-01-22' AND '2023-01-27');

Comment //Find an unreturned book that is within that expiredate time frame.

## Feedback

**1. number of comments per librarian for each member**

SELECT COUNT(libid), memid FROM feedback
GROUP BY memid;

```
--number of comments per librarian for each member
SELECT COUNT(libid), memid FROM feedback
   GROUP BY memid;

--getting reviews better than the average
SELECT libid, stars, comments FROM feedback WHERE
   stars >= (SELECT AVG(stars) FROM feedback) ORDER BY
     stars DESC;


--get all reviews for a librarian
SELECT DISTINCT librarian.libid, librarian.name, feedback.stars,
    librarian ON librarian.libid = feedback.libid WHERE librarian
```

Script Output x | Query Result x

SQL | All Rows Fetched: 13 in 0.042 seconds

| | COUNT(LIBID) | MEMID |
|---|---|---|
| 1 | 1 | 13 |
| 2 | 1 | 11 |
| 3 | 1 | 6 |

**2. getting reviews better than the average**

SELECT libid, stars, comments FROM feedback WHERE
stars >= (SELECT AVG(stars) FROM feedback) ORDER BY
stars DESC;

```
57  --getting reviews better than the average
58  SELECT libid, stars, comments FROM feedback WHERE
59     stars >= (SELECT AVG(stars) FROM feedback) ORDER BY
60       stars DESC;
61
62
63  --get all reviews for a librarian
64  SELECT DISTINCT librarian.libid, librarian.name, feedback.stars, feedback.com
65      librarian ON librarian.libid = feedback.libid WHERE librarian.name = 'Ho
66
67
68  -- getting all reviews for each member with name
69  SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JO
70      members.memid = feedback.memid ORDER BY members.memid ASC;
71
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0.037 seconds

| | LIBID | STARS | COMMENTS |
|---|---|---|---|
| 1 | 2 | 5 | They were my highschool teacher |
| 2 | 4 | 5 | They're ok |
| 3 | 2 | 5 | Best librarian ever but they only come on tuesdays |
| 4 | 4 | 4 | Siuuu that's nice library for student |

**3. get all reviews for a librarian**

SELECT DISTINCT librarian.libid, librarian.name, feedback.stars, feedback.comments
FROM feedback INNER JOIN
librarian ON librarian.libid = feedback.libid WHERE librarian.name = 'Ho Kim Lee';

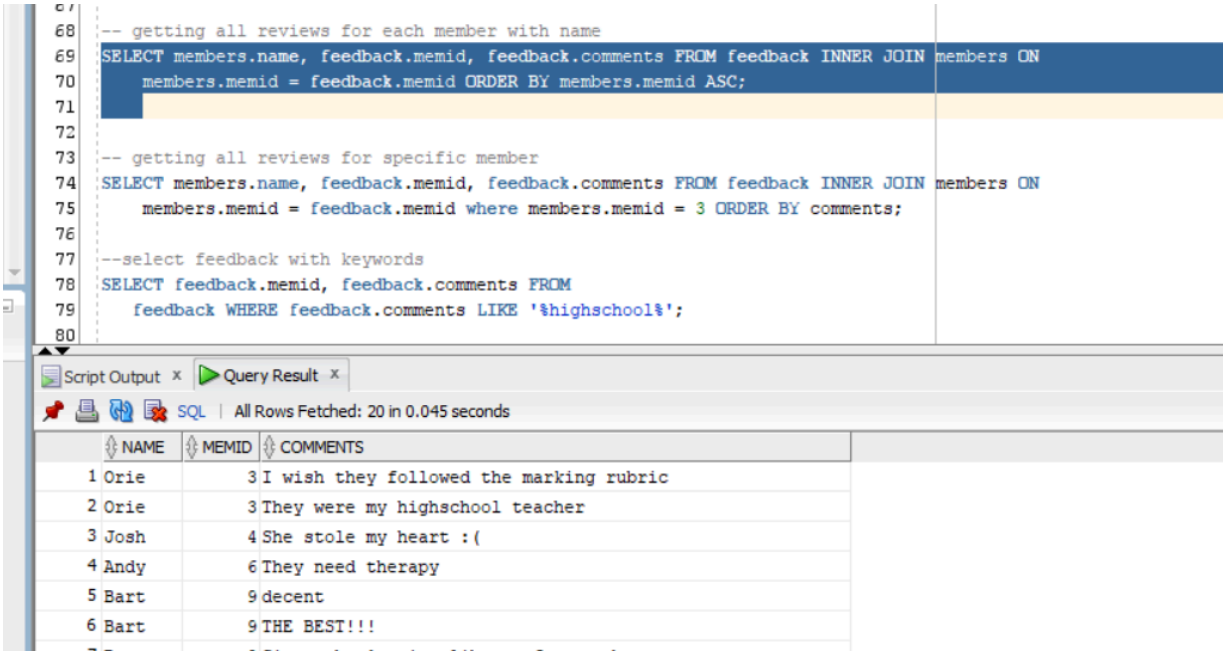
```
62
63  --get all reviews for a librarian
64  SELECT DISTINCT librarian.libid, librarian.name, feedback.stars, feedback.comments FROM feedback INNER JOIN
65      librarian ON librarian.libid = feedback.libid WHERE librarian.name = 'Ho Kim Lee';
66
67
68  -- getting all reviews for each member with name
69  SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON
70      members.memid = feedback.memid ORDER BY members.memid ASC;
71
72
73  -- getting all reviews for specific member
74  SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.041 seconds

| | LIBID | NAME | STARS | COMMENTS |
|---|---|---|---|---|
| 1 | 3 | Ho Kim Lee | 2 | They are very quiet but otherwise good |
| 2 | 3 | Ho Kim Lee | 1 | Need update of reading offline |
| 3 | 3 | Ho Kim Lee | 1 | They should consider a career change. |

**4. getting all reviews for each member with name**
SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON
members.memid = feedback.memid ORDER BY members.memid ASC;

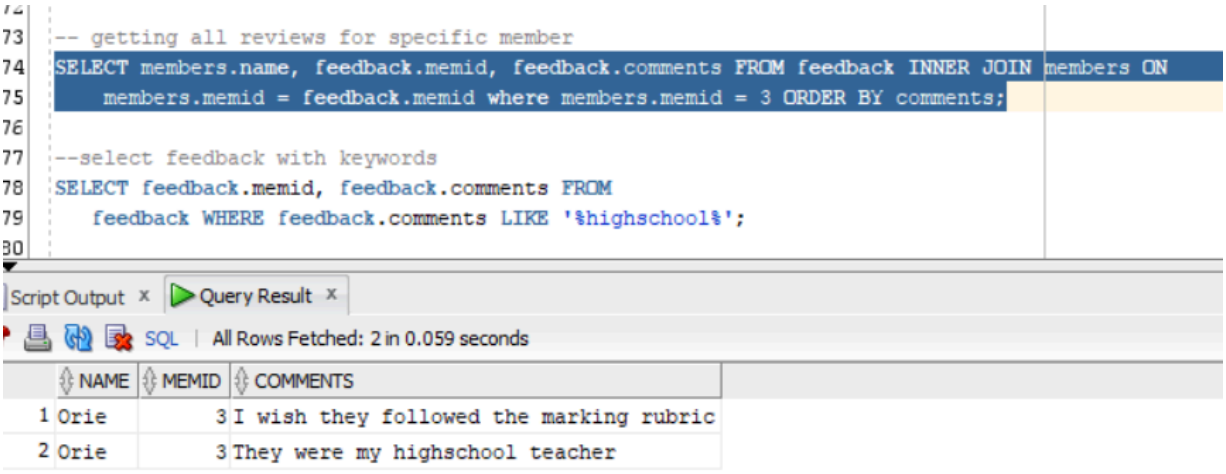


```
-- getting all reviews for each member with name
SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON
    members.memid = feedback.memid ORDER BY members.memid ASC;

-- getting all reviews for specific member
SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON
    members.memid = feedback.memid where members.memid = 3 ORDER BY comments;

--select feedback with keywords
SELECT feedback.memid, feedback.comments FROM
   feedback WHERE feedback.comments LIKE '%highschool%';
```

Script Output | Query Result | SQL | All Rows Fetched: 20 in 0.045 seconds

| | NAME | MEMID | COMMENTS |
|---|---|---|---|
| 1 | Orie | 3 | I wish they followed the marking rubric |
| 2 | Orie | 3 | They were my highschool teacher |
| 3 | Josh | 4 | She stole my heart :( |
| 4 | Andy | 6 | They need therapy |
| 5 | Bart | 9 | decent |
| 6 | Bart | 9 | THE BEST!!! |

**5. -- getting all reviews for specific member SELECT members.name**, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON members.memid = feedback.memid where members.memid = 3 ORDER BY comments;



```
-- getting all reviews for specific member
SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER JOIN members ON
    members.memid = feedback.memid where members.memid = 3 ORDER BY comments;

--select feedback with keywords
SELECT feedback.memid, feedback.comments FROM
   feedback WHERE feedback.comments LIKE '%highschool%';
```

Script Output | Query Result | SQL | All Rows Fetched: 2 in 0.059 seconds

| | NAME | MEMID | COMMENTS |
|---|---|---|---|
| 1 | Orie | 3 | I wish they followed the marking rubric |
| 2 | Orie | 3 | They were my highschool teacher |

**6. select feedback with keywords**

SELECT feedback.memid, feedback.comments FROM
feedback WHERE feedback.comments LIKE '%highschool%';

```
77  --select feedback with keywords
78  SELECT feedback.memid, feedback.comments FROM
79     feedback WHERE feedback.comments LIKE '%highschool%';
80
81
82  --select all comments with a given rating
83  SELECT feedback.memid, feedback.stars,
84     feedback.comments FROM feedback WHERE feedback.stars = 5;
85
86
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.038 seconds

| | MEMID | COMMENTS |
|---|---|---|
| 1 | 3 | They were my highschool teacher |

**7. select all comments with a given rating**

SELECT feedback.memid, feedback.stars,

feedback.comments FROM feedback WHERE feedback.stars = 5;

```
81
82  --select all comments with a given rating
83  SELECT feedback.memid, feedback.stars,
84     feedback.comments FROM feedback WHERE feedback.stars = 5;
85
86
87
88
89  insert into testing (id, country) values (1,'a');
90
91  select * from testing order by id asc;
92
93  select distinct id, country from testing where
94     id > 0 order by id asc;
95
```

Script Output | Query Result | SQL | All Rows Fetched: 3 in 0.037 seconds

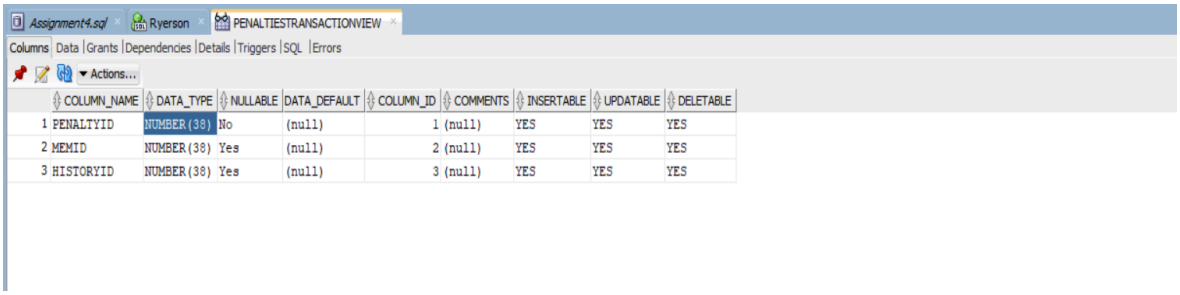
| | MEMID | STARS | COMMENTS |
|---|---|---|---|
| 1 | 3 | 5 | They were my highschool teacher |
| 2 | 19 | 5 | They're ok |
| 3 | 20 | 5 | Best librarian ever but they only come on tuesdays |

## ASSIGNMENT 4 PART II:

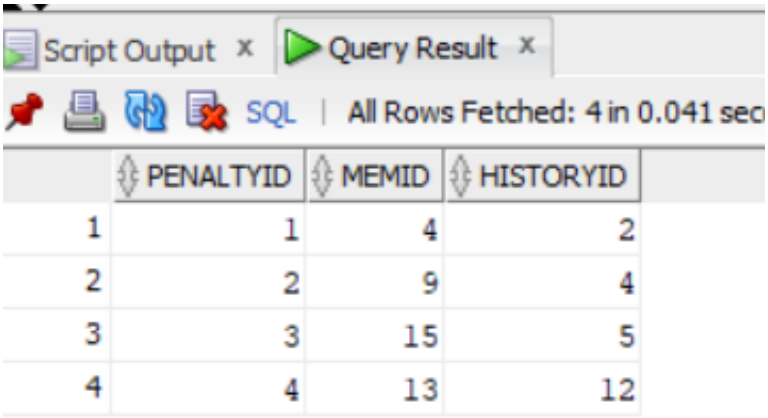
---

Create view PenaltiesTransactionView AS
SELECT Penalties.*
FROM Penalties
JOIN TransactionDetails ON Penalties.HistoryID = TransactionDetails.HistoryID
WHERE TransactionDetails.PickupDate BETWEEN '01-JAN-23' AND '05-JAN-23';

**select * from PenaltiesTransactionView;**




| | COLUMN_NAME | DATA_TYPE | NULLABLE | DATA_DEFAULT | COLUMN_ID | COMMENTS | INSERTABLE | UPDATABLE | DELETABLE |
|---|---|---|---|---|---|---|---|---|---|
| 1 | PENALTYID | NUMBER(38) | No | (null) | 1 | (null) | YES | YES | YES |
| 2 | MEMID | NUMBER(38) | Yes | (null) | 2 | (null) | YES | YES | YES |
| 3 | HISTORYID | NUMBER(38) | Yes | (null) | 3 | (null) | YES | YES | YES |




| | PENALTYID | MEMID | HISTORYID |
|---|---|---|---|
| 1 | 1 | 4 | 2 |
| 2 | 2 | 9 | 4 |
| 3 | 3 | 15 | 5 |
| 4 | 4 | 13 | 12 |

```
CREATE VIEW mediaInStock AS
  SELECT mediaid AS "Id",
   mediatitle AS "Title",
   author AS "Author",
   mediatype AS "Type"
   FROM media WHERE media.instock = 1 ORDER BY mediatitle ASC;
```

| | Id | Title | Author | Type |
|---|---|---|---|---|
| 1 | 5 | Air currents | Marty small | Journal |
| 2 | 6 | Anthropology | Venice Hare | Journal |
| 3 | 16 | C Language History | Kyle Wang | Book |
| 4 | 3 | C programming | Alex Ufkes | Book |
| 5 | 19 | Christmas tales | Mark Rober | CD |
| 6 | 20 | Exploding cucumbers | Sarah Crown | Book |
| 7 | 8 | Fall tunes | Matt Meese | CD |
| 8 | 15 | Functional Programming with Lisp | Mark Gotet | Book |
| 9 | 2 | How to grow blueberries | Charles Hadwell | Book |
| 10 | 12 | Iceberg melting | Lawrence Walker | Journal |
| 11 | 14 | Lisp is Best | Kevin Spencer | Book |
| 12 | 9 | Studio C | Matt Meese | CD |
| 13 | 11 | Study of Slang | Kevin Spencer | Journal |

```
create or replace view MEMBER_Credibility as
select members.memid as "ID”,
members.name as "Name",
members.Amountowed as "Charges " from members
where members.amountowed < 20.0
Order by memID asc;
```

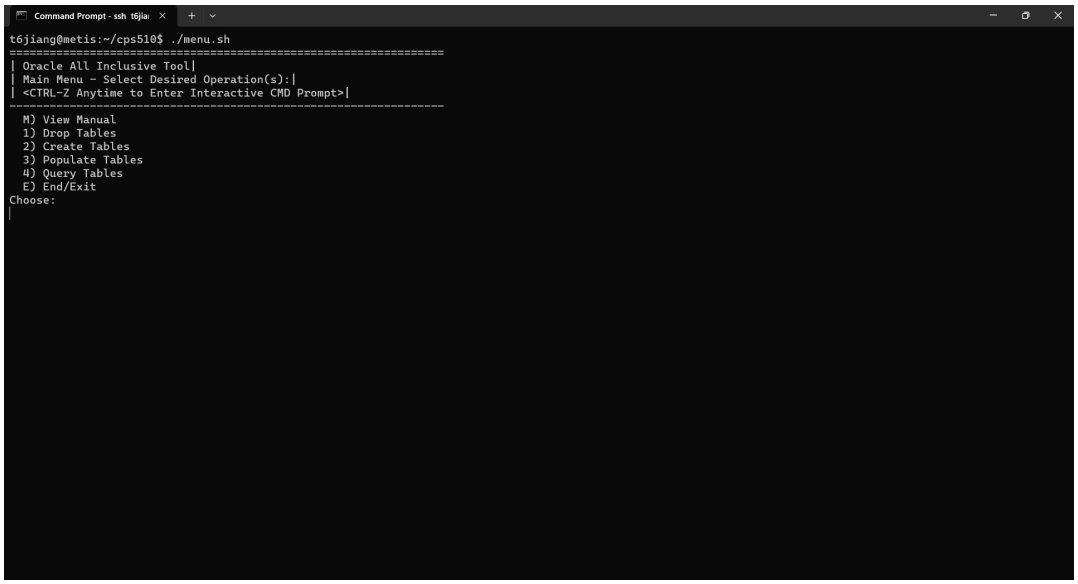
**select * from MEMBER_Credibility;**

| | ID | Name | Charges |
|---|---|---|---|
| 1 | 2 | Schinner | 8 |
| 2 | 4 | Josh | 6 |
| 3 | 5 | Jacob | 13 |
| 4 | 7 | Ruby | 5 |
| 5 | 9 | Bart | 1 |
| 6 | 10 | Noel | 14 |
| 7 | 13 | Allan | 11 |
| 8 | 15 | Rose | 7 |
| 9 | 18 | William | 6 |
| 10 | 19 | Allan | 12 |

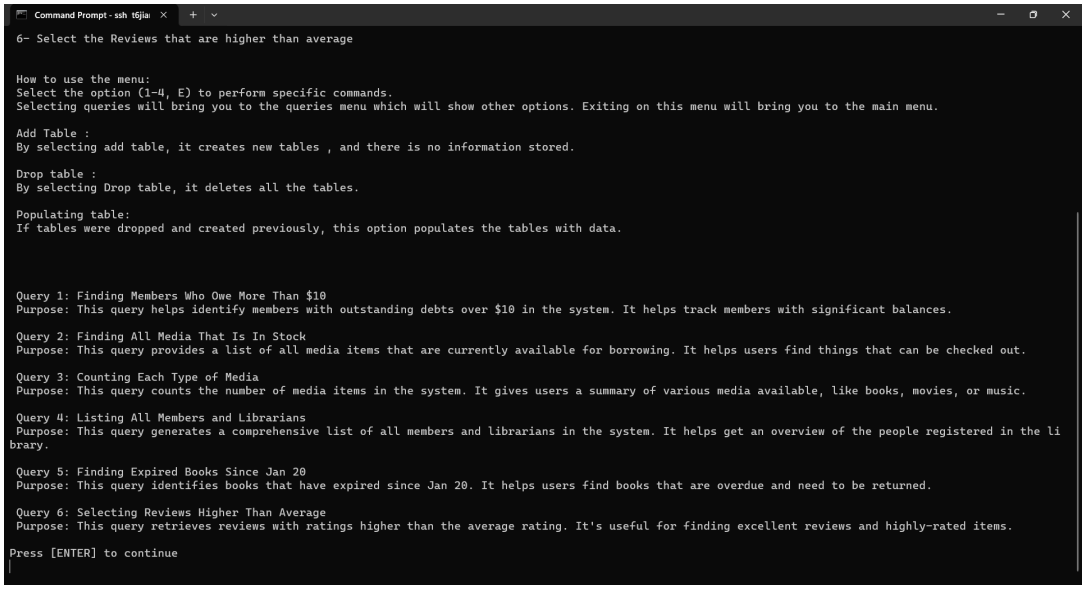
# ASSIGNMENT 5

---

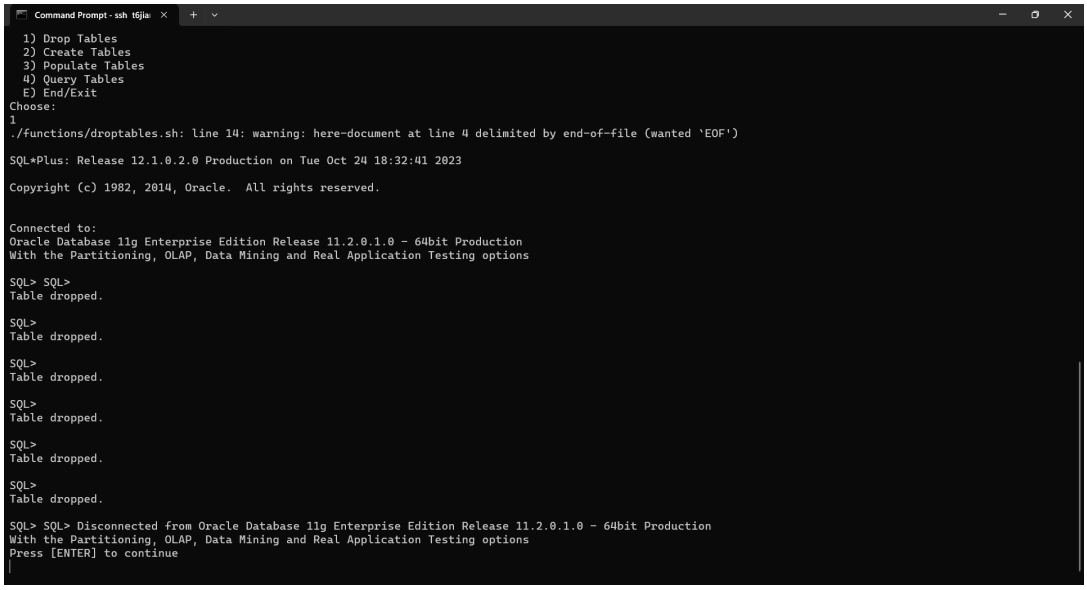
Menu program:



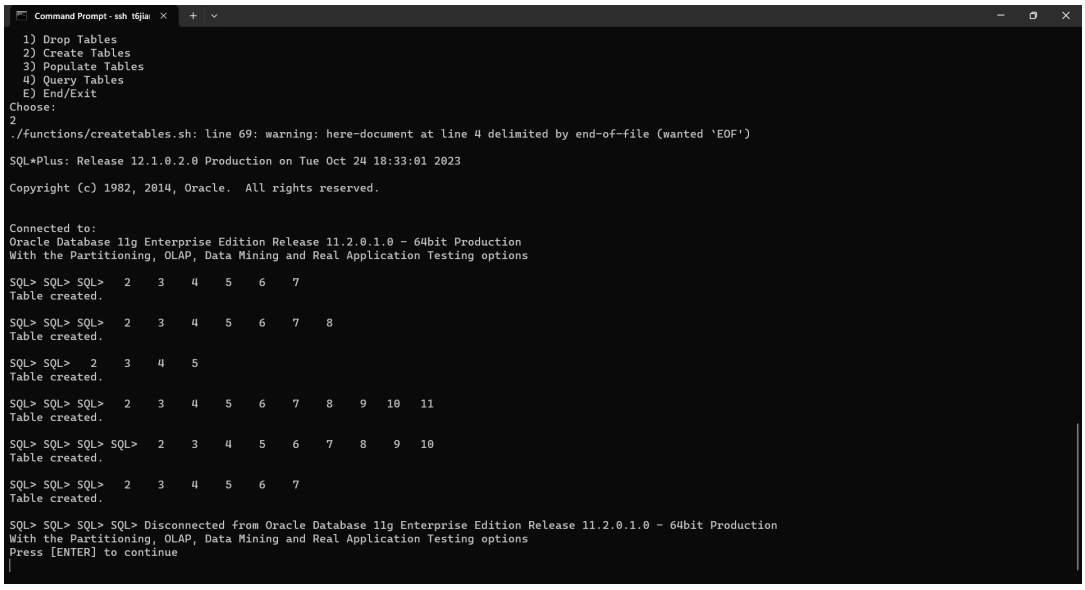

Manual:




Drop tables:

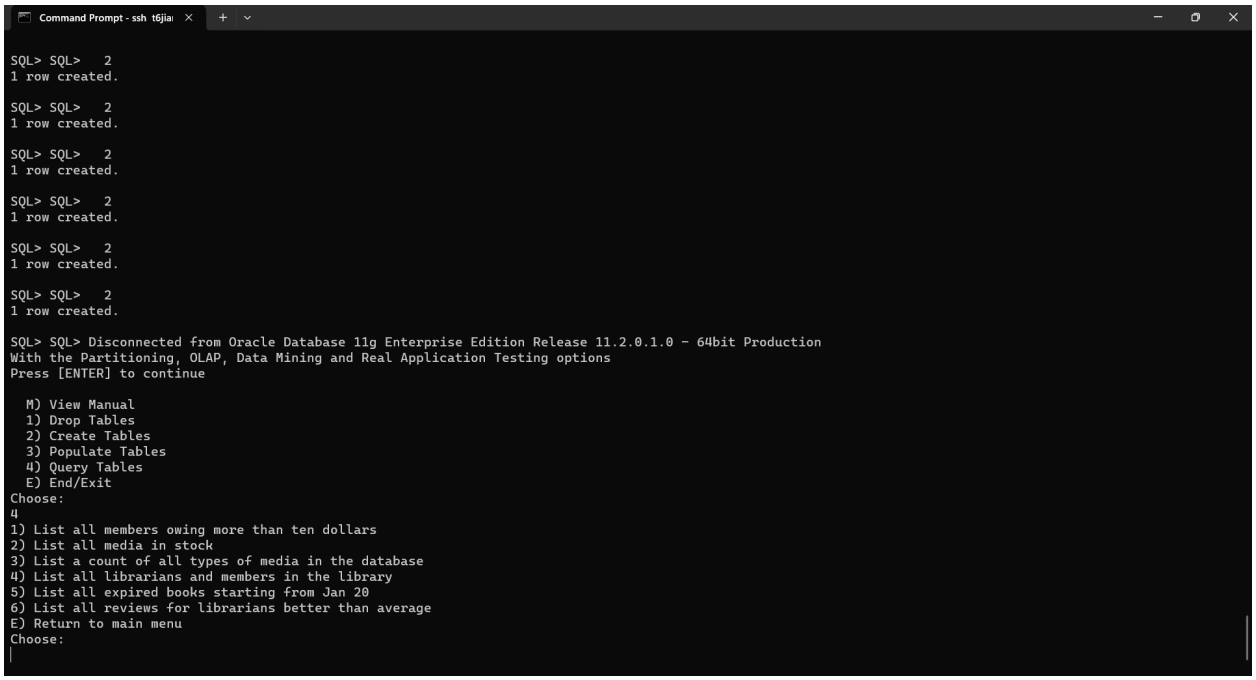



Create tables:



Populate tables:

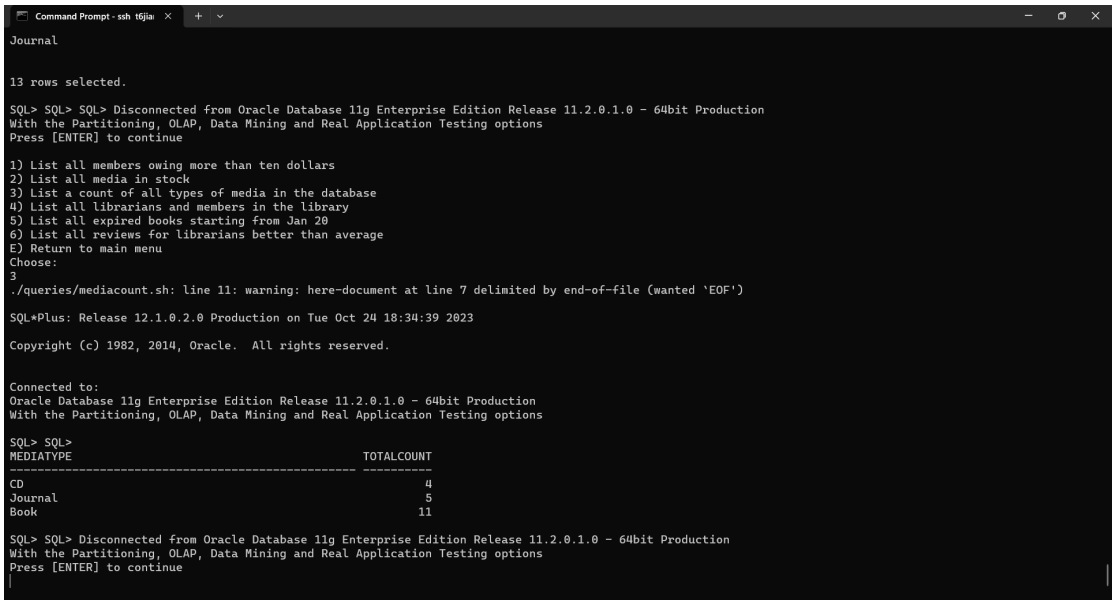

Query menu:



List all members owing more than 10 dollars:

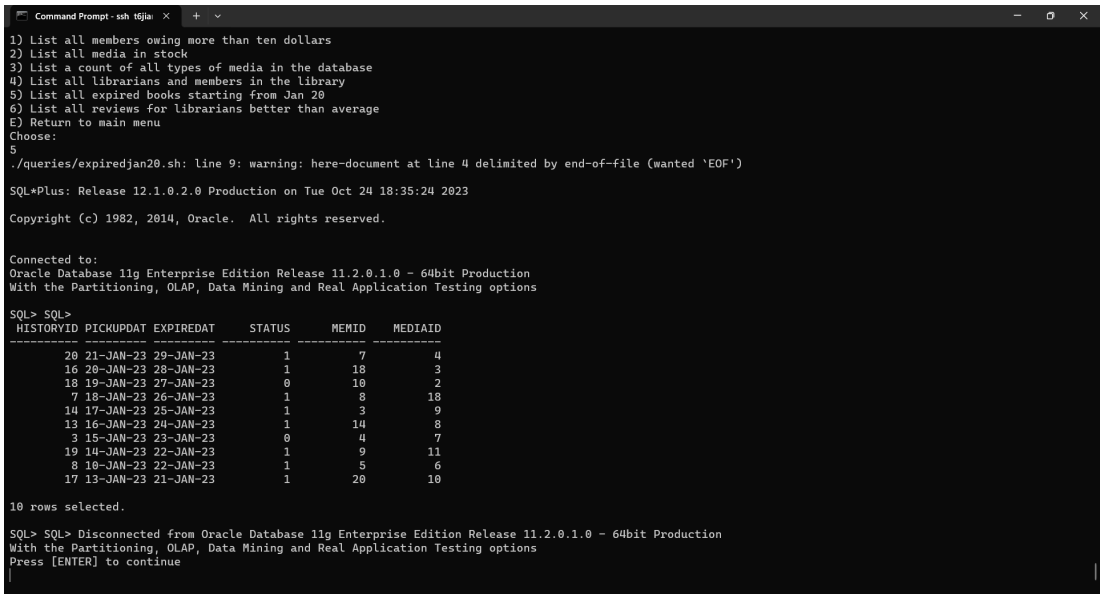

List all media in stock:

List a count of all media in the library:

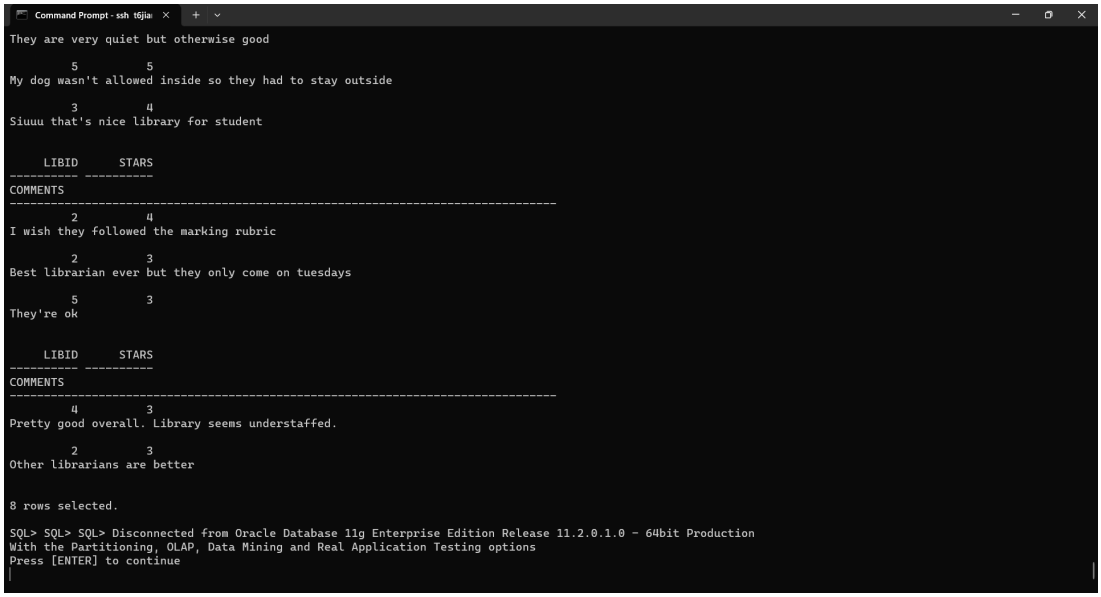



List all members and librarians in the library:

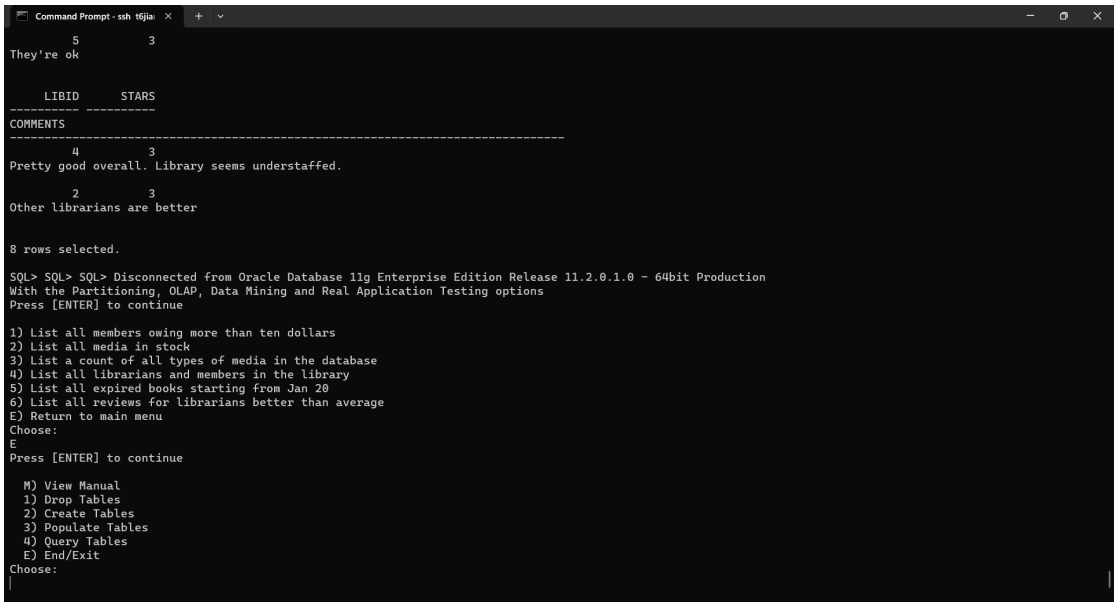
List all books expired starting from Jan 20.



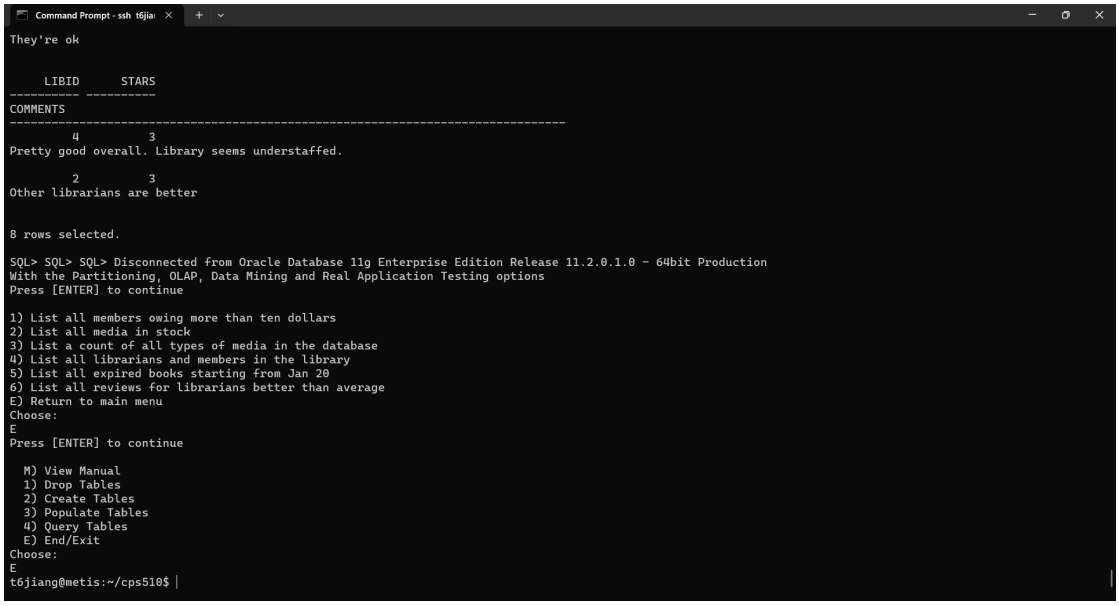

List all librarians with higher than average reviews:



Returning to main menu:




Quitting the menu program:



## ASSIGNMENT 6

**Media**

| **Attribute** | **Attribute type** |
|---|---|
| author | varchar |
| title | varchar |
| publish date | Date |
| **ID** | Int (primary key) |
| availability | Int |

```
id ⟶ {author, title, publish date, availability}

id ⟶ {author}         id ⟶ {title}
id ⟶ {publish date}   id ⟶ {availability}


====

Id -> {title, availability}
title -> {publish date, author}
```

**Penalties**

| Attribute | Attribute type |
|---|---|
| **penalty id** | Int (Primary key) |
| member id | Int (Foreign key) |
| transaction id | Int (Foreign key) |

**penalty id ⟶ { member id, transaction id }**

penalty id ⟶ { member id }

penalty id ⟶ {transaction id }

---

**Members**

| Attribute | Attribute type |
|---|---|
| **MemID** | Int (Primary key) |
| Username | varchar |
| Pass | varchar |
| Email | varchar |
| Name | varchar |
| AmountOwed | Double |

```
memID ⟶ {username, pass, email, name, amount owed}

memID ⟶ {username}
memID ⟶ {pass}
memID ⟶ {name}
memID ⟶ {amount owed}
```

---

**Transaction Details**

| **Attribute** | **Attribute type** |
|---|---|
| **history id** | Int (primary key) |
| media id | Int (Foreign key) |
| expiry date | Date |
| status | Int |
| pickup date | Date |
| member id | Int (Foreign key) |

**history id** ⟶ { media id, expiry date, status, pickup date, member id }
history id ⟶ { media id }
history id ⟶ { expiry date }
history id ⟶ { status }
history id ⟶ { pickup date}
history id ⟶ { member id }

---

**Librarian**

| Attribute | Attribute type |
|---|---|
| **LibID** | Int (Primary key) |
| AdminKey | Int (Primary key) |
| Name | varchar |

{LibID, AdminKey} → Name

---

**Feedback**

| **Attribute** | **Attribute type** |
|---|---|
| **feedback id** | Int (Primary key) |
| member id | Int (Foreign key) |
| librarian id | Int (Foreign key) |
| comment | String |
| Stars | Int |

**Feedback ID** → {memberId, librarianId, comment , Stars}

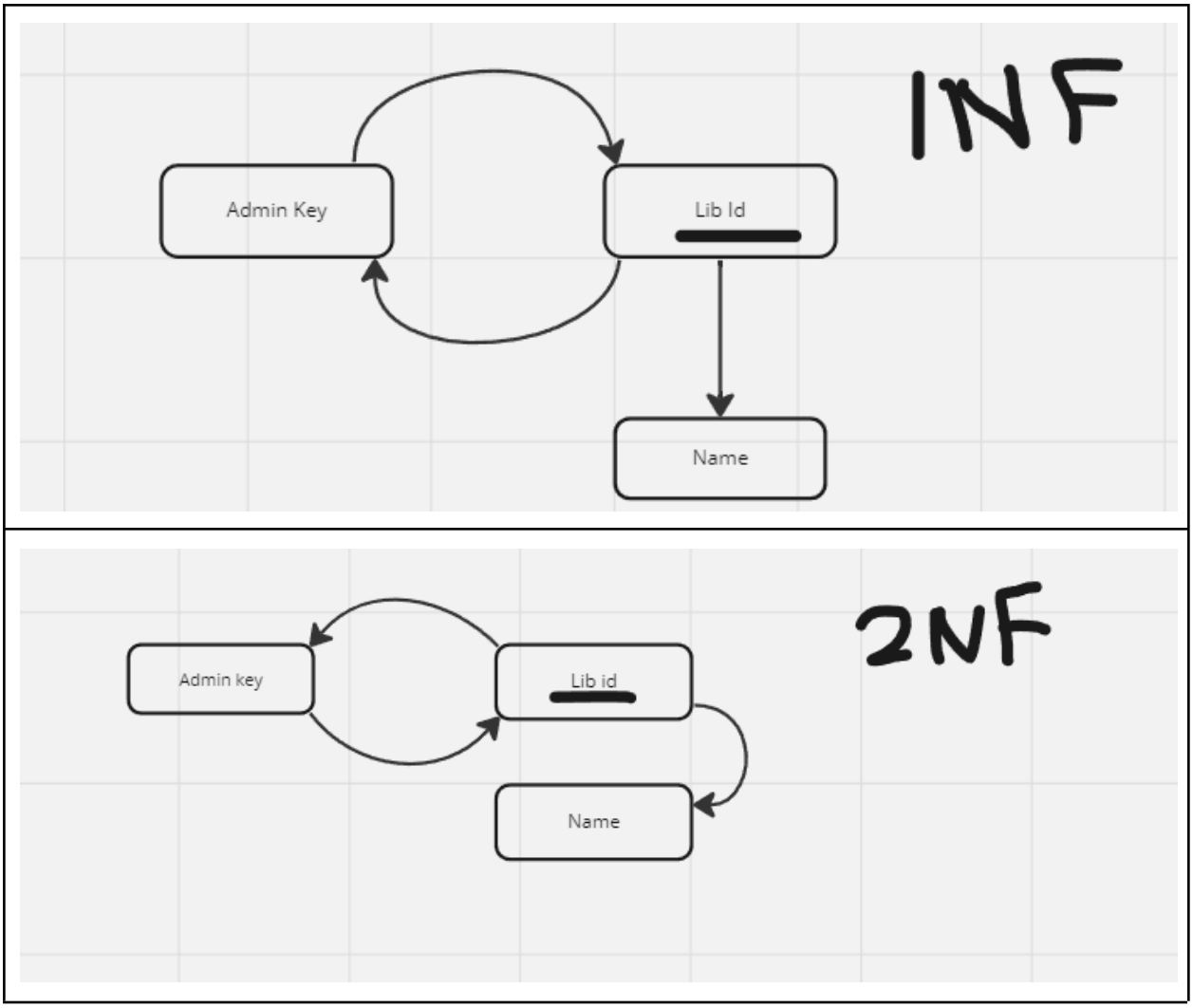
Feedback ID → {memberID}
Feedback ID → {librarianID}
Feedback ID → {comment}
Feedback ID → {Stars}

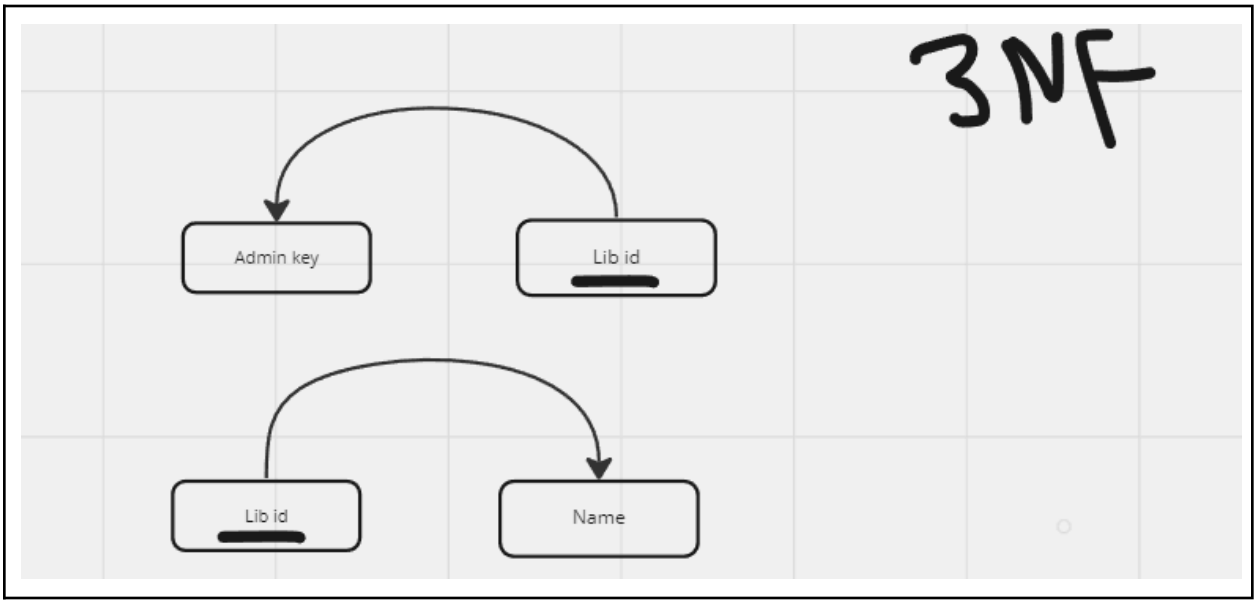
From observing all of the tables we have currently, we have been able to determine that the attributes are atomic, which means that the tables are in 1NF. By further observation, We also observe many of these attributes are functionally dependent on their key values and that some do not have any transitive dependencies on one another. Due to this, we can conclude that the tables some of the tables may already be in 3NF, while the others may be in a combination of 1 and 2

## ASSIGNMENT 7:

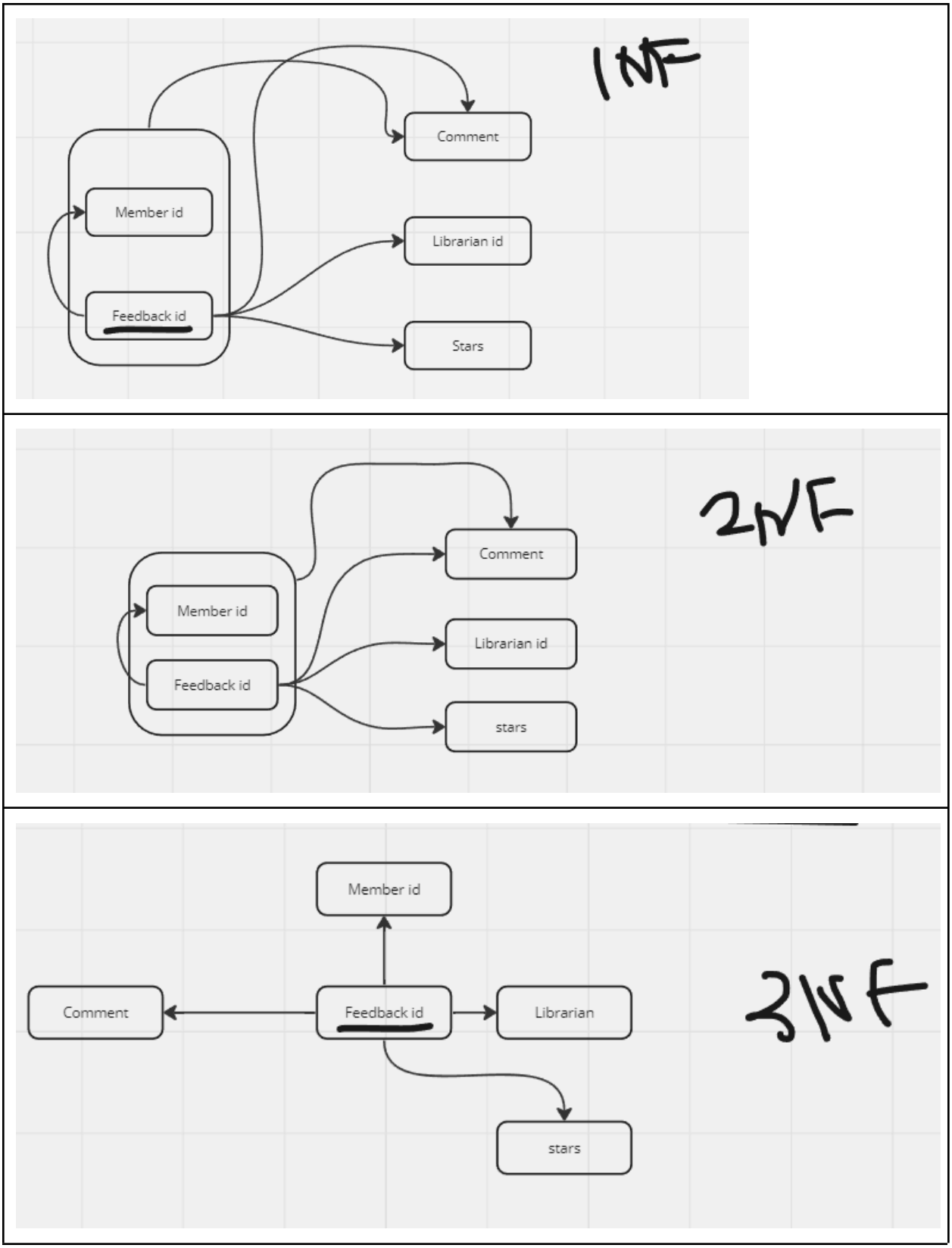
---

Librarian

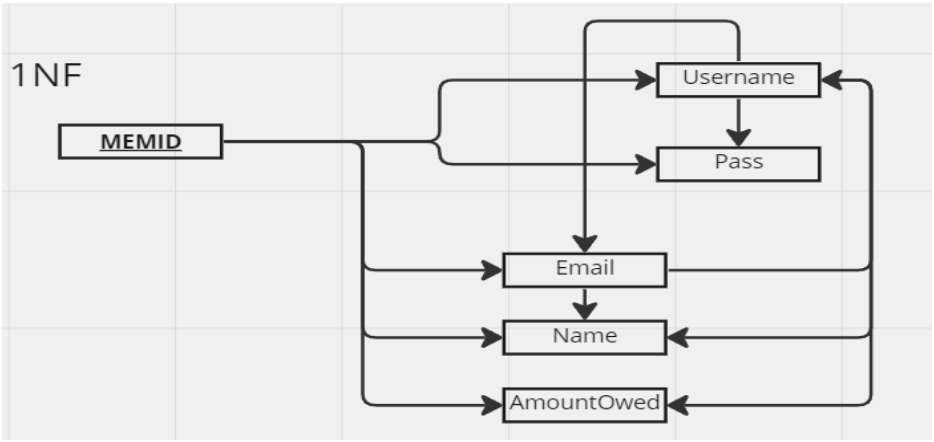


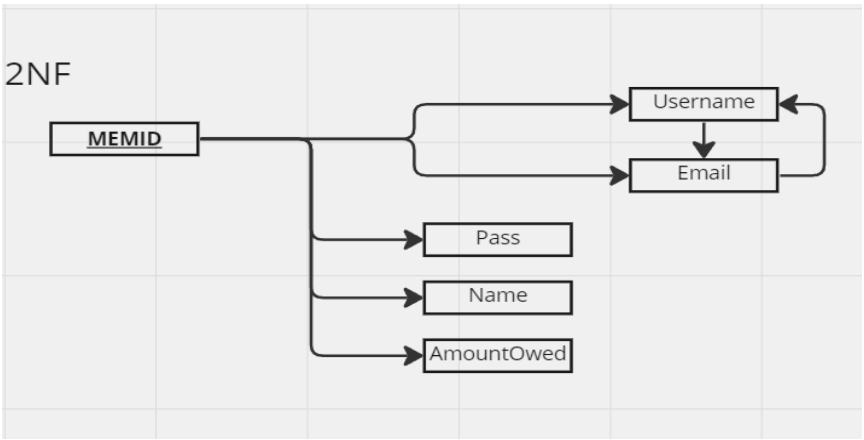

3NF
Admin key
Lib id
Lib id
Name

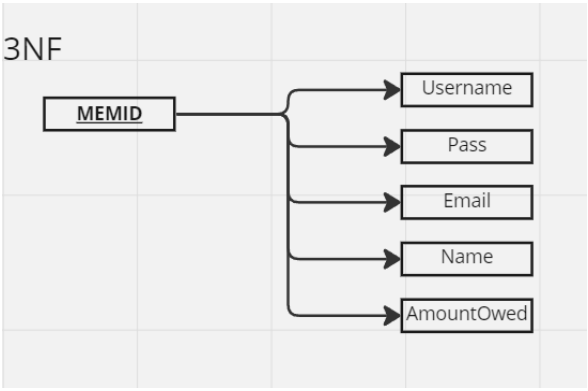
Feedback


1NF
Comment
Member id
Librarian id
Feedback id
Stars
2NF
Comment
Member id
Librarian id
Feedback id
stars
Member id
Comment
Feedback id
Librarian
stars
3NF

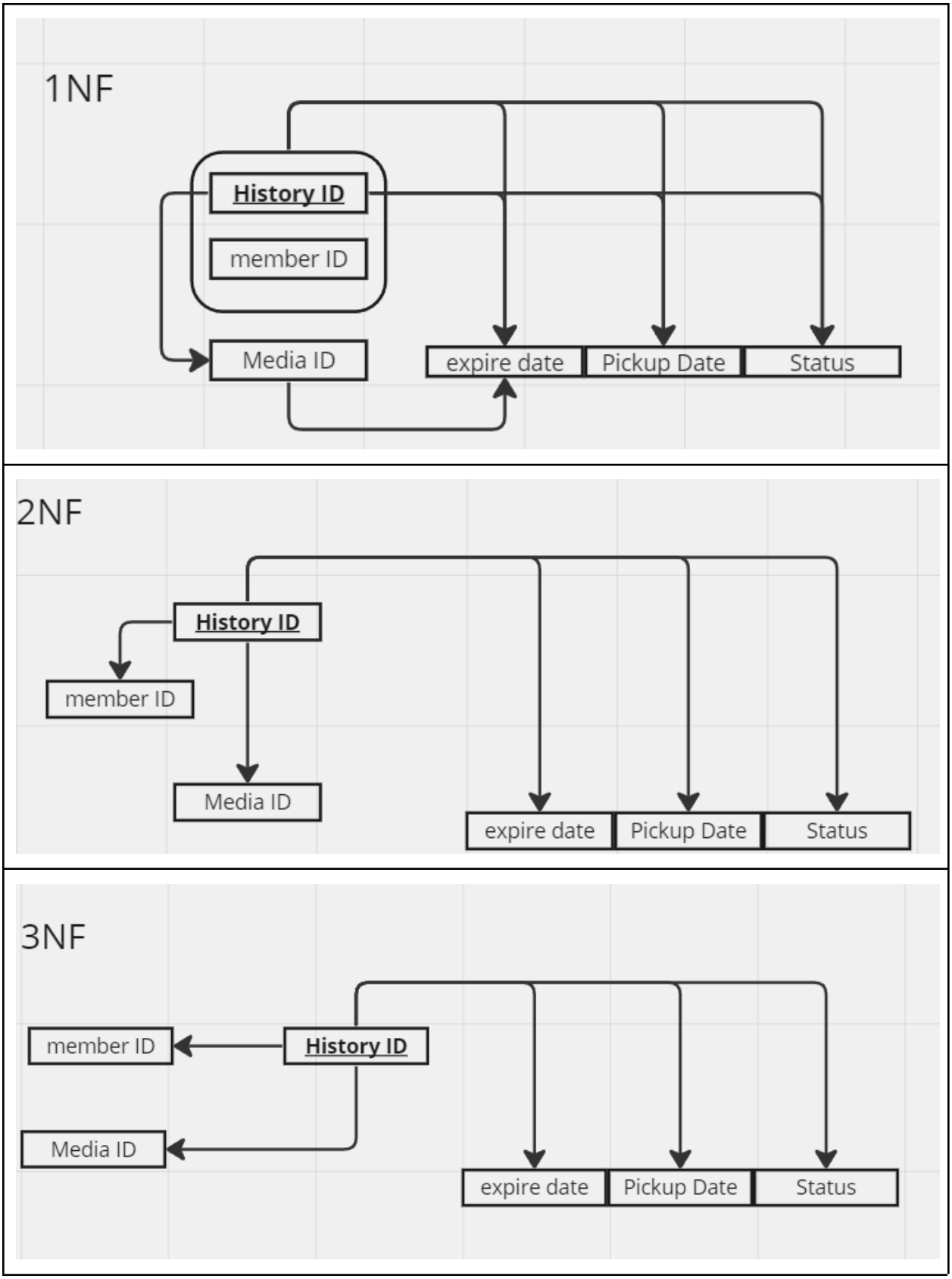
Members


1NF
MEMID
Username
Pass
Email
Name
AmountOwed

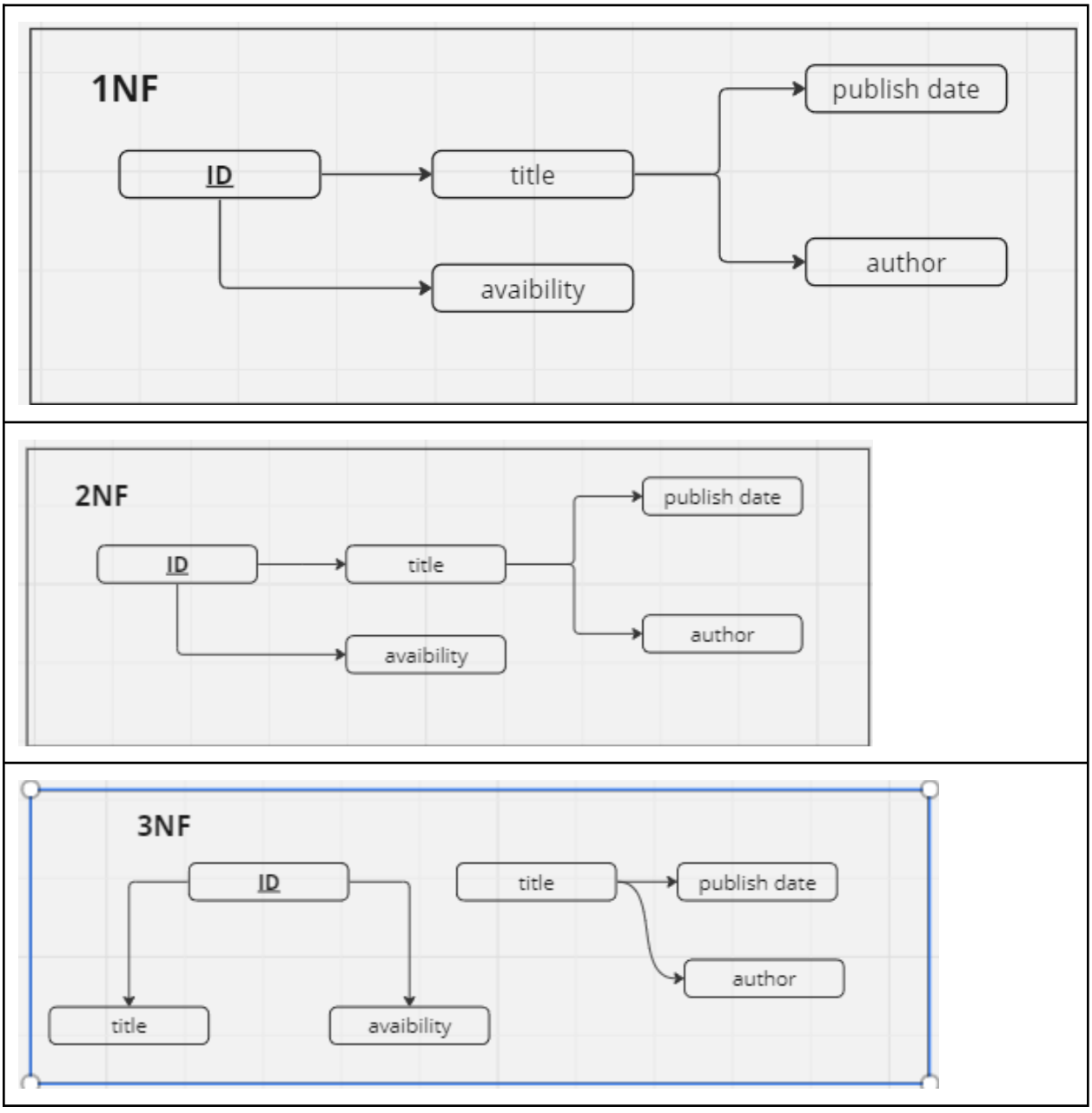

2NF
MEMID
Username
Email
Pass
Name
AmountOwed


3NF
MEMID
Username
Pass
Email
Name
AmountOwed

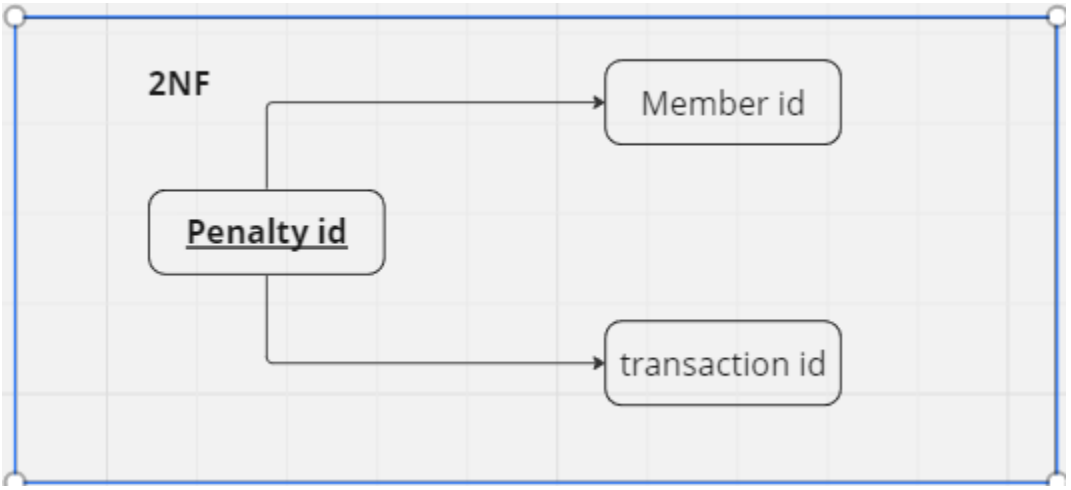
Transaction Details

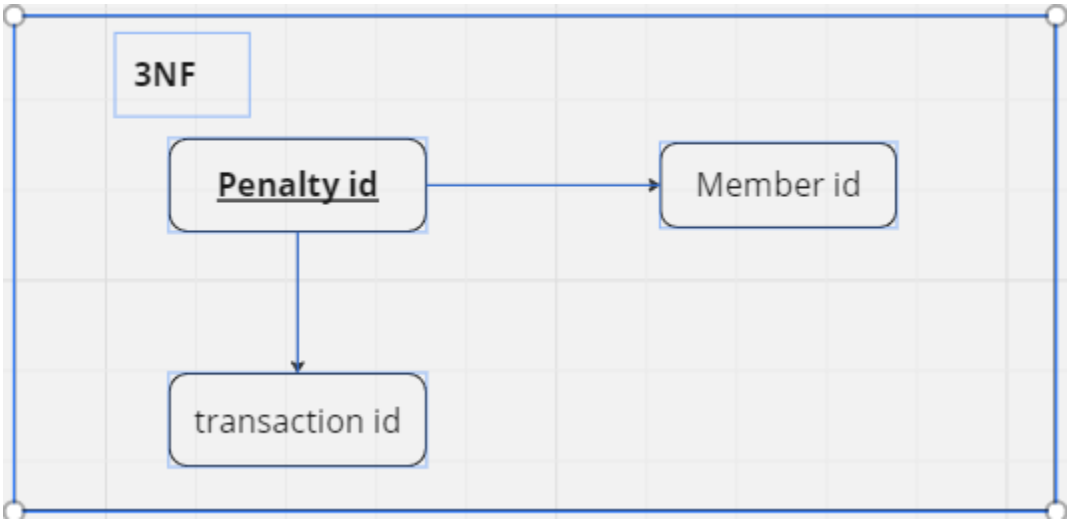


Media


1NF
ID
title
publish date
author
availability
2NF
ID
title
publish date
author
availability
3NF
ID
title
publish date
author
title
availability

Penalties


2NF
Member id
Penalty id
transaction id

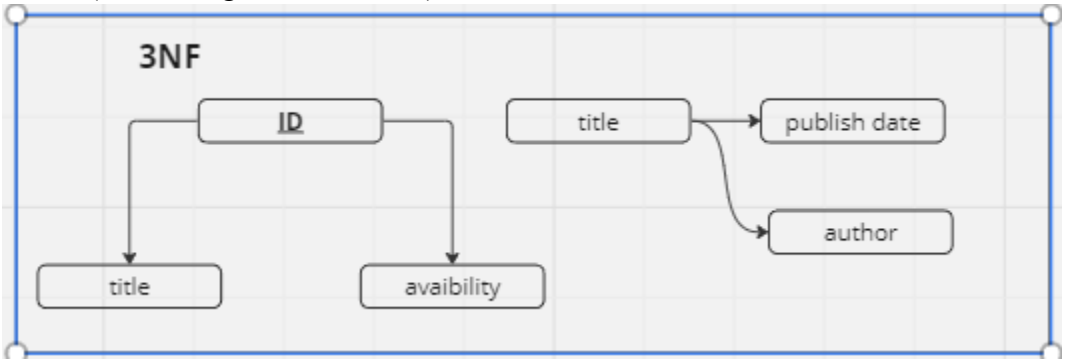

3NF
Penalty id
Member id
transaction id

## ASSIGNMENT 8:

***DISCUSSION:***

*The tables above present the BCNF form. In the context of Boyce-Codd Normal Form (BCNF) in database normalization, they must be in their 3NF form, and they ensure that for every non-trivial functional dependency in a relation, the set of attributes on the left side of the dependency constitutes a superkey. In simpler terms, if a non-key determines a key or a key determines a key, it is necessary to break down the table into smaller tables to meet the BCNF criteria, thereby avoiding anomalies and maintaining data integrity. Our tables are in BCNF form based on the criteria.*

Media (BCNF Algorithm is Used)

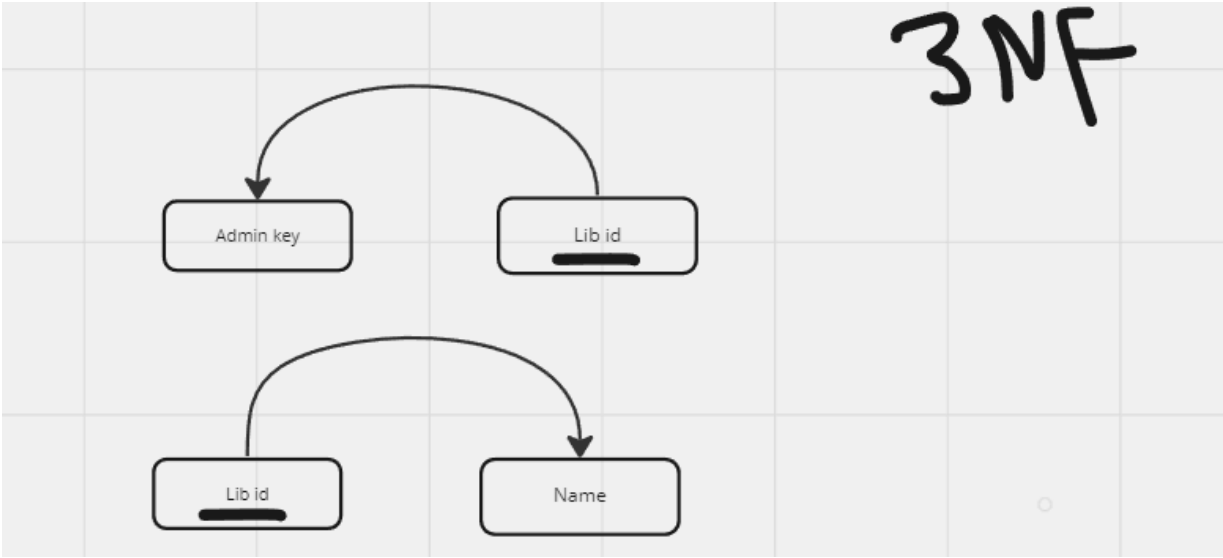



Functional dependencies
id ⟶ {author, title, publish date, availability}

id ⟶ {author}  id ⟶ {title}
id ⟶ {publish date}  id ⟶ {availability}

===
id ⟶ {title, availability}
title ⟶ {publish date, author}

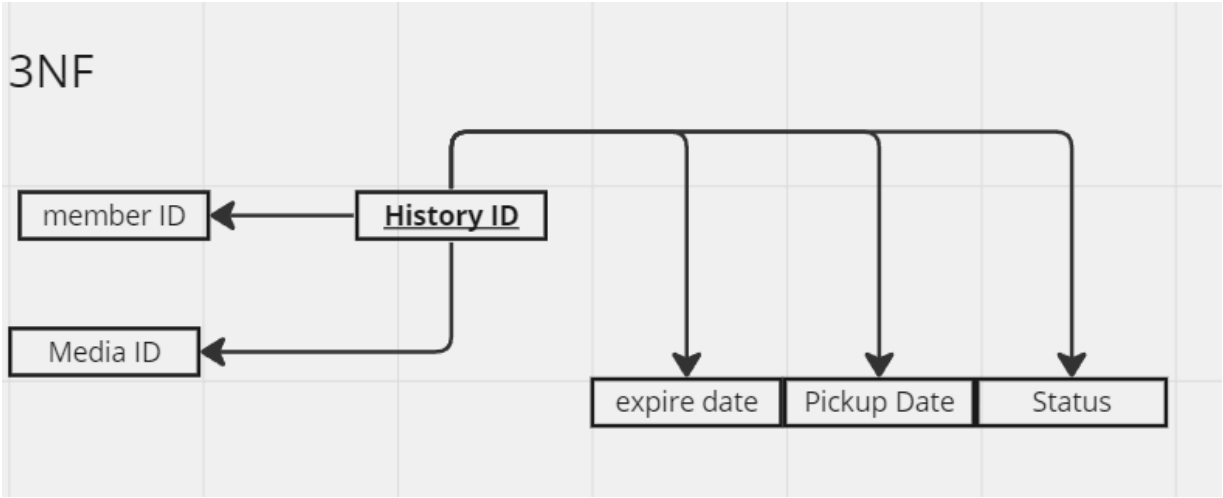
Librarian




Functional dependencies

LibId -> Admin key
LibId -> Name

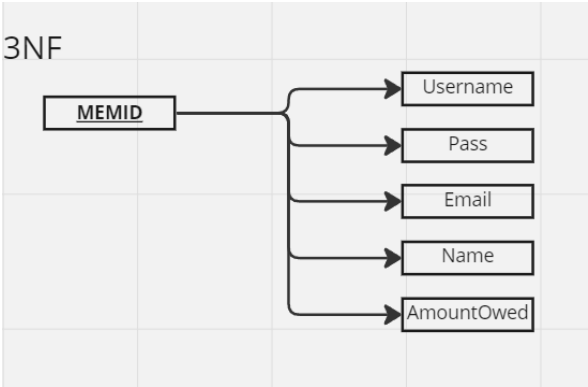
Transaction Details




Functional Dependencies:

```
HistoryId -> memberId
HistoryId -> MediaId
HistoryId -> expireDate
HistoryId -> pickupDate
HistoryId -> status
```

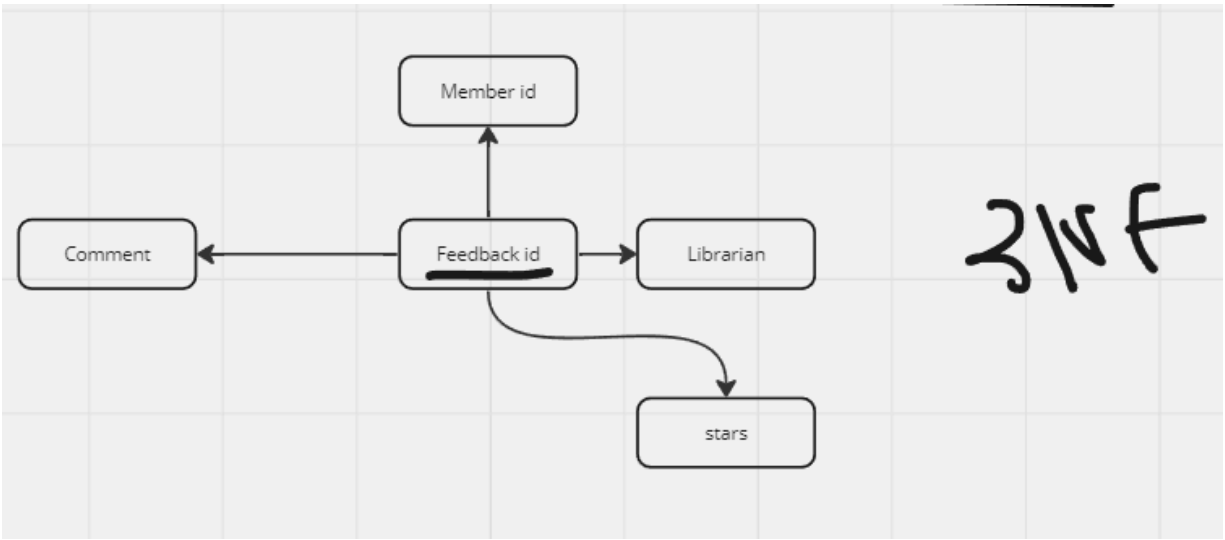
Members



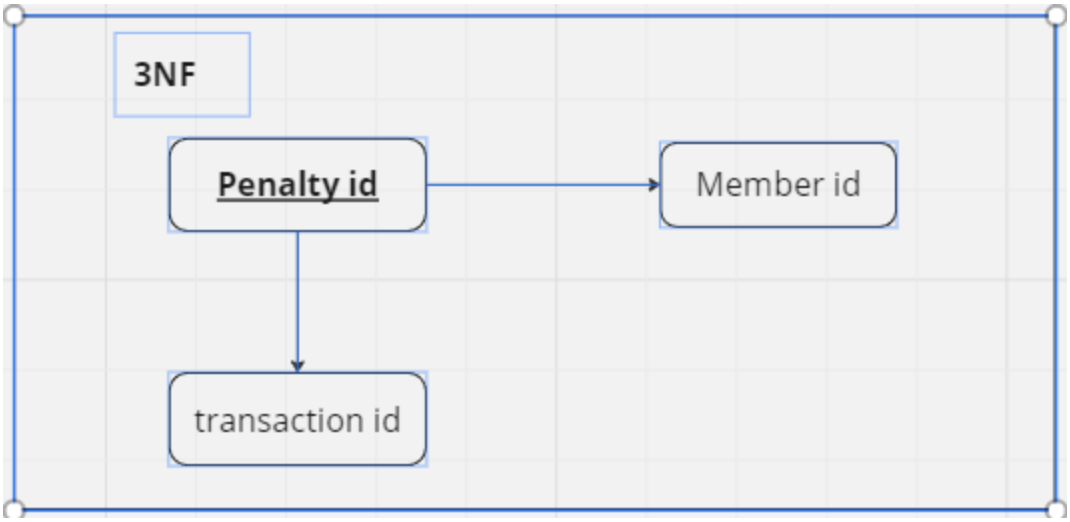

Functional Dependencies

MEMID -> Username
MEMID -> Pass
MEMID -> Email
MEMID -> Name
MEMID -> AmountOwed

Feedback




FeedbackId -> MemberId
FeedbackId -> Librarian
FeedbackId -> Stars
FeedbackId -> Comment

| Penalty |
| --- |
|  |
| Function Dependencies:<br><br>Penalty ID - > Member ID<br>Penalty ID -> Transaction ID |

Berstein's Algorithm

Step 1:
Determine attributes, FDs about the real world.
From our application, we determined that we require the following attributes for Table Media.

Type refers to the media being digital, paperback, etc
Category refers to Novel, manga, journal, video, etc

***TAKE MEDIA AS A MODEL EXAMPLE:***

***ID, Type, Title, Author, Publisher, Category, Language, Reviews,OnSale***

**ID** ⟶ {Title, Publisher, Author, Reviews}
Title ⟶ {Language, Category,Reviews}
Publisher ⟶ {onSale, Type}
Author ⟶ {Language}

Step 2a:
2a : Remove the redundant dependencies. From our table, we determined that we could get the following functional dependencies after removal of the redundancies:

**Attribute closures of the attributes**
$ID^+$ = {ID, Title, Language, Publisher, onSale, Type, Author, Reviews, Category}

$Title^+$ = {Language, Category}

Publisher* = {Publisher, onSale, Type}

**Functional Dependencies after removing redundancies:**
ID -> {Title, Publisher, Author, Reviews}
Title -> {Language, Category}
Publisher -> {OnSaile, Type}

Step 2b:
2b : Find and remove partial dependencies. From our table, these following partial dependencies will be removed to complete this step.

This process will make this table into a 2NF.

**1NF:**
Since all of the attributes in our table are already atomic, the table is already in 1NF.

**2NF:**
After looking at the table, we have determined that decomposition is needed to bring the relation to 2NF.

***ID, Type, Title, Author, Publisher, Category, Language, Reviews,OnSale***
Original table:

R(ID, Title, Language, Publisher, onSale, Type, Author, Reviews, Category )

Decomposed tables:
R1(ID, Title, Publisher, Author, Reviews)

R2(Publisher, Onsale, Type)

R3(Title, Language, Category)

Step 3:

*Now that your diagram is in 2nf, you can:*
*find the keys of your diagram.*
*This requires you to find closures of all your attributes and combine them with all other attributes to determine if they're keys.*
*This is computationally expensive.*

Functional dependencies:

ID -> {Title, Publisher, Author, Reviews}
Title -> {Language, Category}
Publisher -> {OnSale, Type}

**Decomposed Tables:**
R1(ID, Title, Publisher, Author, Reviews)

R2(Publisher, Onsale, Type)

R3(Title, Language, Category)

**Attribute closures:**

$ID^+$ = {ID, Title, Publisher, Author, Reviews, Language, Category, Onsale, Type}

$Publisher^+$ = {Publisher, OnSale, Type}

$Title^+$ = {Title, Language, Category}

$Type^+$ = {Type}

$Author^+$ = {Author}

$Reviews^+$ = {Reviews}

$OnSale^+$ = {OnSale}

Category = {Category}

From the attribute closure analysis, We have determined that the key should be ID, Publisher, Title, because combining all the tables, it will enclose all the attributes that are part of media.

Step 4:

From step 3, we have determined that the relational schema from above should satisfy the requirements for the keys.

R1(ID, Title, Publisher, Author, Reviews)

R2(Publisher, Onsale, Type)

R3(Title, Language, Category)

**Key Relations:**

Reviewing the three decomposed tables, all of the tables have key relations, therefore step 4 is complete and the table is in 3NF that is lossless and dependency preserving.

---

**Counter Example of a BCNF:**

An Example of a table that is not in BCNF is an example such as:

R(MediaId, ISBN, Title, Category, PubishDate)
FD:

MediaID -> {ISBN}

MediaID -> {Title, Category, PublishDate}

This table is not in BCNF because there is a key attribute (MediaId), which determines another key attribute (ISBN). By definition of BCNF, all keys should be on the left side of the title:relation, however in this case, ISBN is not.

---

# ASSIGNMENT 9:

Screenshots of the system:


Dropping tables:

Creating tables:

```
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

A
Creating tables...
Created tables.

======================
Option A: Create table
Option B: Delete table
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:
```

Populating Tables:

```
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

C
Populating tables...
Populated table Media...
Populated table Members...
Populated table Librarian...
Populated table Transaction Details...
Populated table Feedback...
Populated table Penalties...
Populated all tables.

======================
Option A: Create table
Option B: Delete table
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
```

Query 1 : Query “Expired Jan 20th”

```
======================
Option A: Create table
Option B: Delete table
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

D
History id: 20 |  Media Id: 4 |  status: 1 | Member ID: 7 | Pick up date: 2023-01-21|Expired Day: 2023-01-29
History id: 16 |  Media Id: 3 |  status: 1 | Member ID: 18 | Pick up date: 2023-01-20|Expired Day: 2023-01-28
History id: 18 |  Media Id: 2 |  status: 0 | Member ID: 10 | Pick up date: 2023-01-19|Expired Day: 2023-01-27
History id: 7 |  Media Id: 18 |  status: 1 | Member ID: 8 | Pick up date: 2023-01-18|Expired Day: 2023-01-26
History id: 14 |  Media Id: 9 |  status: 1 | Member ID: 3 | Pick up date: 2023-01-17|Expired Day: 2023-01-25
History id: 13 |  Media Id: 8 |  status: 1 | Member ID: 14 | Pick up date: 2023-01-16|Expired Day: 2023-01-24
History id: 3 |  Media Id: 7 |  status: 0 | Member ID: 4 | Pick up date: 2023-01-15|Expired Day: 2023-01-23
History id: 8 |  Media Id: 6 |  status: 1 | Member ID: 5 | Pick up date: 2023-01-10|Expired Day: 2023-01-22
History id: 19 |  Media Id: 11 |  status: 1 | Member ID: 9 | Pick up date: 2023-01-14|Expired Day: 2023-01-22
History id: 17 |  Media Id: 10 |  status: 1 | Member ID: 20 | Pick up date: 2023-01-13|Expired Day: 2023-01-21
```

Query 2 : Query “Good Reviews”

```
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

E
5 5 My dog wasn't allowed inside so they had to stay outside
1 5 They are very quiet but otherwise good
3 4 Siuuu that's nice library for student
2 4 I wish they followed the marking rubric
5 3 They're ok
4 3 Pretty good overall. Library seems understaffed.
2 3 Other librarians are better
2 3 Best librarian ever but they only come on tuesdays

======================
Option A: Create table
```

Query 3 : Query “Query List Everyone”

```
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

F
Allan allanturing@gmail.com Member
Allan ucorn100@gmail.com Member
Andy andych@gmail.com Member
Bart imbaroque@gmail.com Member
Brekke brekbb@gmail.com Member
Ellen ehu0098@gmail.com Member
Emerson Emerson.watsica@gmail.com Member
Ho Kim Lee null Librarian
Jacob ljacon@gmail.com Member
Jenkins jenkins.fritz@gmail.com Member
John wwiend@gmail.com Member
Josh jwalsh@gmail.com Member
Kevin Jones null Librarian
Kimberly Delamere null Librarian
Minh Tran null Librarian
Morgan Freeman null Librarian
Noel noel21@gmail.com Member
Orie orieng@gmail.com Member
Pam gatdaip@hotmail.com Member
Retha Rethh1029@gmail.com Member
Rose rosalia5656@gmail.com Member
Ruby rdubue@gmail.com Member
Schinner schice@gmail.com Member
```

Query 4 : Query “Media Count”

```
Retha Rethh1029@gmail.com Member
Rose rosalia5656@gmail.com Member
Ruby rdubue@gmail.com Member
Schinner schice@gmail.com Member
Susan iceeyou@yahoo.ca Member
William Williamgreat2000@gmail.com Member

======================
Option A: Create table
Option B: Delete table
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

G
Type: CD Amount:4
Type: Book Amount:11
Type: Journal Amount:5
```

Query 5 : Query Media InStock

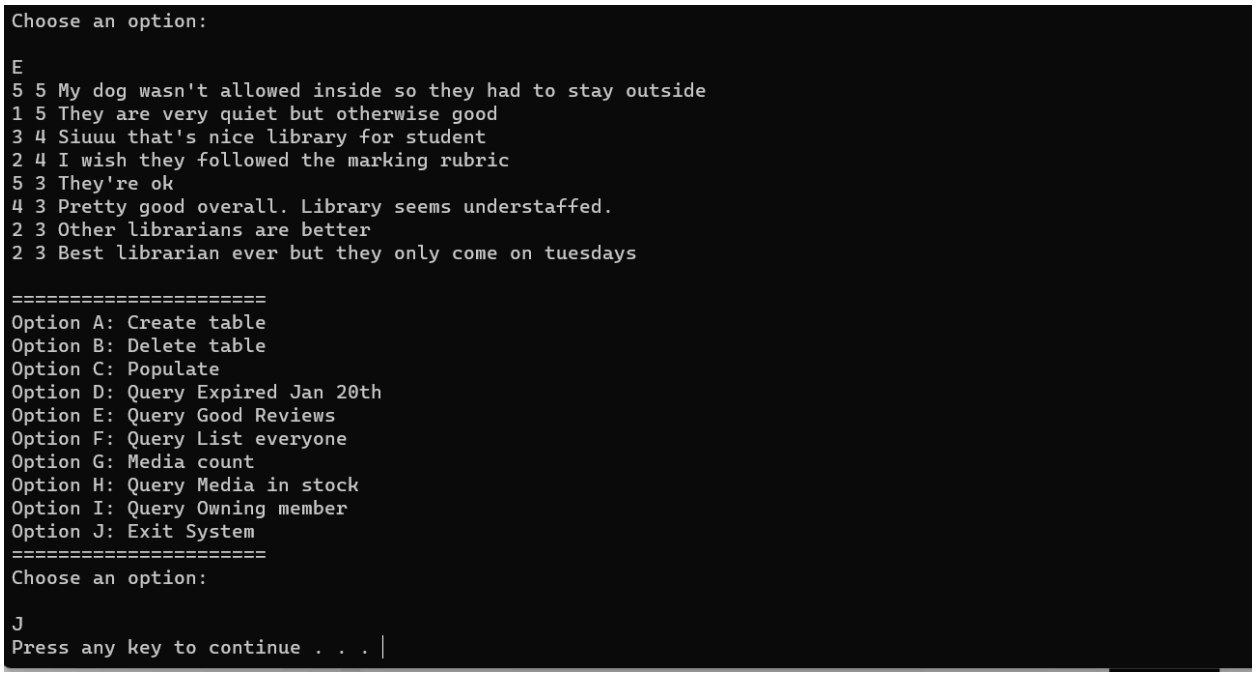
```
======================
Choose an option:

H
Id: 2 Title: How to grow blueberries Author: Charles Hadwell Type: Book
Id: 3 Title: C programming Author: Alex Ufkes Type: Book
Id: 5 Title: Air currents Author: Marty small Type: Journal
Id: 6 Title: Anthropology Author: Venice Hare Type: Journal
Id: 8 Title: Fall tunes Author: Matt Meese Type: CD
Id: 9 Title: Studio C Author: Matt Meese Type: CD
Id: 11 Title: Study of Slang Author: Kevin Spencer Type: Journal
Id: 12 Title: Iceberg melting Author: Lawrence Walker Type: Journal
Id: 14 Title: Lisp is Best Author: Kevin Spencer Type: Book
Id: 15 Title: Functional Programming with Lisp Author: Mark Gotet Type: Book
Id: 16 Title: C Language History Author: Kyle Wang Type: Book
Id: 19 Title: Christmas tales Author: Mark Rober Type: CD
Id: 20 Title: Exploding cucumbers Author: Sarah Crown Type: Book

======================
Option A: Create table
Option B: Delete table
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
```

\

Query 6 : Query Owning Member

```
Option I: Query Owning member
Option J: Exit System
======================
Choose an option:

I
Member id: 11 username: icie25 email:iceeyou@yahoo.ca amount owed: 13.0
Member id: 2 username: schinner.gladyce email:schice@gmail.com amount owed: 15.0
Member id: 20 username: retha.wyman email:Rethh1029@gmail.com amount owed: 20.0
Member id: 8 username: brekke.robb email:brekbb@gmail.com amount owed: 21.0
Member id: 6 username: andy.gerlach email:andych@gmail.com amount owed: 22.0
Member id: 14 username: emerson.watsica email:Emerson.watsica@gmail.com amount owed: 22.0
Member id: 9 username: bbartoletti email:imbaroque@gmail.com amount owed: 23.0
Member id: 19 username: ucronin email:ucorn100@gmail.com amount owed: 38.0
Member id: 16 username: ehuel email:ehu0098@gmail.com amount owed: 42.0
Member id: 4 username: jwalsh email:jwalsh@gmail.com amount owed: 43.0

======================
Option A: Create table
Option B: Delete table
Option C: Populate
Option D: Query Expired Jan 20th
Option E: Query Good Reviews
Option F: Query List everyone
Option G: Media count
Option H: Query Media in stock
Option I: Query Owning member
Option J: Exit System
======================
```

Exiting the System:




---

# ASSIGNMENT 10:

### SQL to Relational Algebra:

"SELECT mediaid AS \"Id\"," + "  mediatitle AS \"Title\"," + "  author AS \"Author\"," + " mediatype AS \"Type\"" + "  FROM media WHERE media.instock = 1 ORDER BY mediatitle ASC

Relational Algebra:
π ← (σ_(expiredate > TO_DATE('2023-01-20','RRRR-MM-DD')) (transactionDetails))

String query = "SELECT libid, stars, comments FROM feedback WHERE stars >= (SELECT AVG(stars) FROM feedback) ORDER BY stars DESC" ;

Relational Algebra:
π ← (libid, stars, comments)
σ ←(stars >= ρ_(AVG(stars) AVG_stars)  (feedback) ⋈ true))

SELECT DISTINCT Name AS PersonnelName, Email AS ContactInfo, 'Member' AS PersonnelType FROM Members" + " UNION " + "SELECT DISTINCT Name AS PersonnelName, NULL AS ContactInfo, 'Librarian' AS PersonnelType FROM Librarian

Relational Algebra:
$\pi \leftarrow$(Name, Email, 'Member' as PersonnelType) (Members) $\cup$ $\pi \leftarrow$(Name, null as ContactInfo, 'Librarian' as PersonnelType) (Librarian)

SELECT MEDIATYPE, COUNT(*) AS TotalCount FROM MEDIA GROUP BY MEDIATYPE

Relational Algebra:
$\pi \leftarrow$(MEDIATYPE, COUNT(*) as TotalCount) (MEDIA) $\div$ MEDIATYPE

SELECT mediaid AS \"Id\"," + "  mediatitle AS \"Title\"," + "  author AS \"Author\"," + " mediatype AS \"Type\"" + "  FROM media WHERE media.instock = 1 ORDER BY mediatitle ASC

Relational Algebra:
$\pi \leftarrow$ (mediaid as Id, mediatitle as Title, author as Author, mediatype as Type)
($\sigma \leftarrow$ (instock = 1) (media))

```
"SELECT DISTINCT members.memid AS \"Member id\", " +
            "members.username AS \"Username\", " +
            "members.email AS \"Member email\", " +
```

```
            "members.amountowed AS \"Amount Owed\" " +
            "FROM members INNER JOIN penalties ON members.memid =
penalties.memid" +
            "WHERE members.amountowed > 10 ORDER BY amountowed ASC";
```

$\pi \leftarrow$ (memid as Member_id, username as Username, email as Member_email, amountowed as Amount_Owed)
($\sigma \leftarrow$ (amountowed > 10)
(Members $\bowtie$ (memid = memid) penalties))

---

```
Select * FROM Media ORDER BY INSTOCK ASC;
```

Sort $\leftarrow \sigma$(Media)
Result $\leftarrow \tau_{\text{INSTOCK ASC}}$ (Sort)

```
Select * FROM Media ORDER BY MEDIATITLE DESC;
```

Sort $\leftarrow \sigma$ (Media)
Result $\leftarrow \tau_{\text{MEDIATITLE DESC}}$ (Sort)

```
Select * FROM MEDIA ORDER BY MEDIATYPE, MEDIATITLE;
```

Sort $\leftarrow \sigma$ (Media)
Result $\leftarrow \tau_{\text{MEDIATYPE , MEDIATITLE DESC}}$ (Sort)

```
Select DISTINCT Author FROM MEDIA;

///The tuples returned from RA operations are ALWAYS distinct, so No Distinct keyword
needed
```

Proj ← $\pi_{Author}$(MEDIA)
Result ← σ(Proj)

SELECT DISTINCT AUTHOR FROM MEDIA ORDER BY AUTHOR DESC;

Proj ← $\pi_{Author}$(MEDIA)
Sort ← σ(Proj)
Result ← $\tau_{MEDIATYPE\ ,\ MEDIATITLE\ DESC}$ (Proj)

SELECT MEDIATYPE as "Media Type" , COUNT(*) AS "Total Count" FROM MEDIA
GROUP BY MEDIATYPE;

Temp ← $\pi_{MediaType,\ COUNT(*)}$(Media)
Temp2 ← $\rho_{(Media\ Type,\ Total\ Count)}$Temp
Result ← $\gamma_{MediaType,\ Total\ Count}$(Temp2)

SELECT mediatitle,instock FROM MEDIA Order By mediatitle;

Proj ← $\pi_{Mediatitle,instock}$ (Media)
Result ← $\tau_{\ mediatitle}$ (Proj)

SELECT amountowed, memid FROM Members WHERE amountowed >
(SELECT AVG(amountowed) FROM Members) ORDER BY amountowed DESC;

//γ <, grouping / aggregation
//δ duplicate elimination
//τ sorting

Mems ← $\pi_{MEMID,\ AMOUNTOWED}$ ( Members )
Ave ← AVG(AmountOwed)
Filter ← $\sigma_{\ AmountOwed\ >\ Ave}$ (Mems)
Result ←$\tau_{\ amountowed\ DESC}$ (Filter)

SELECT username, email, name FROM members WHERE

```
   email LIKE '%gmail.com';
```

$Temp = \pi_{USERNAME\ ,\ EMAIL\ ,\ NAME}$
$Result \leftarrow \sigma_{email\ Like\ (\%gmail.com)}\ (Temp)$

```
SELECT distinct name, email, pass FROM members WHERE email LIKE 'wwiend%'
   AND name LIKE 'John%' ORDER BY name DESC;
```

$Temp \leftarrow \pi_{NAME\ ,\ EMAIL\ ,\ PASS}$
$Filter \leftarrow \sigma_{(email\ Like\ (wwiend\%)\ \cap\ Name\ LIKE\ (John\ \%))}\ (Temp)$
$Result \leftarrow \tau_{name\ DESC}\ (Filter)$

```
SELECT DISTINCT members.memid, members.username FROM members
 INNER JOIN penalties ON penalties.memid = members.memid ORDER BY
   members.username ASC;
```

$Cross \leftarrow Penalties\ x\ Members$
$Filter \leftarrow \delta_{Penalties.memId\ =\ Members.memId}(Cross)$
$Ordered \leftarrow \tau_{Members.Username\ ASC}(Filter)$

```
SELECT DISTINCT members.memid AS "Member Id", members.username AS "Username",
members.email AS "Member email"
  FROM members
INNER JOIN transactiondetails ON members.memid = transactiondetails.memid
   ORDER BY members.username DESC;
```

$Crossed \leftarrow TransactionDetails\ x\ Members$
$Filtered \leftarrow \delta_{Members.MemId\ =\ TransactionDetails.memId}(Crossed)$
$Renamed \leftarrow p_{(Member\ Id,\ Username,\ Member\ Email)}(\pi_{Members.MemId,\ Members.Username,\ Members.email}(Filtered))$
$Ordered \leftarrow \tau_{Members.Username}\ DESC\ (Filtered)$

```
SELECT DISTINCT members.memid AS "Member id",
        members.username AS "Username",
        members.email AS "Member email",
        members.amountowed AS "Amount Owed"
  FROM members INNER JOIN penalties ON members.memid = penalties.memid
  WHERE members.amountowed > 0 ORDER BY amountowed ASC;
```

Join $\leftarrow$ Members $\bowtie_{\text{Members.memId = Penalties.MemId}}$ (Penalties)

Distinct $\leftarrow$ $P_{\text{(Member Id, Username, Member Email, Amount Owed))}}\pi_{\text{Members.memId, members.username, members.email, members.amountowed}}$(Join)

$\tau_{\text{Members.amountOwed ASC}}$ ($\delta_{\text{Amount owed > 0}}$ (Distinct))

```
SELECT members.memid AS "Member Id", members.username AS "Username",
members.email AS "Email" FROM members where
  members.username LIKE 'a%' ORDER BY members.username;
```

$\delta_{\text{members.username LIKE 'a%'}}$(Members)

filtered $\leftarrow$ $\pi_{\text{members.memId, members.username, members.email}}$(Members)

$\tau_{\text{members.username}}$($p_{\text{(Member Id, Username, Email)}}$(filtered))

```
select * from Penalties Where MemID = 5;
```

$\delta_{\text{MemId = 5}}$(Penalties)

```
select Members.*, Penalties.*
From Members
Join Penalties ON Members.MemID = Penalties.MemID;
```

$\sigma$(Members $\bowtie_{\text{members.memId = Penalties.memId}}$(Penalties))

SELECT DISTINCT HistoryID FROM Penalties;

$\delta_{HistoriyId}(Penalties)$

SELECT Penalties.*
FROM Penalties
JOIN TransactionDetails ON Penalties.HistoryID = TransactionDetails.HistoryID
WHERE TransactionDetails.PickupDate BETWEEN '01-JAN-23' AND '05-JAN-23';

join $\leftarrow$ TransactionDetails $\bowtie_{Penalties.HistoryId = TransactionDetails.HistoryId}$ (Penalties)
$\delta_{TransactionDetails.PickupDate\ BETWEEN\ '01\text{-}JAN\text{-}23'\ AND\ '05\text{-}JAN\text{-}23'}$ (join)

SELECT MemID, COUNT(*) AS PenaltyCount
FROM Penalties
GROUP BY MemID;

filtered $\leftarrow \pi_{MemId,\ Count(*)}(Penalties)$
$\gamma_{MemId}(filtered)$

SELECT * FROM Penalties
ORDER BY PenaltyID ASC;

$\tau_{PenaltyId\ ASC}(\sigma_{*}(Penalties))$

SELECT * FROM Penalties WHERE HistoryID = 12;

$\sigma_{HistoryId = 12}$ (Penalties)

SELECT * FROM Librarian WHERE Name LIKE 'M%';

$\sigma_{\text{Name LIKE 'M\%}}(\text{Librarian})$,

SELECT COUNT(*) FROM Librarian;

$\sigma_{\text{COUNT(*)}}(\text{Librarian})$

select COUNT(*) FROM ( select distinct AdminKey FROM Librarian);

$\sigma_{\text{COUNT(*)}}\ (\delta_{\text{AdminKey}}\ (\text{Librarian}))$

select * from Librarian order by name ASC;

$\tau_{\text{name ASC}}(\sigma_{*}(\text{Librarian}))$

select * from Librarian where Adminkey = (select min(Adminkey) FROM Librarian);

$\sigma_{\text{AdminKey} = (}\sigma_{\text{min(AdminKey)}}(\text{Librarian})_{)}$

select LibID, Name From Librarian;

$\sigma_{\text{LibId, Name}}\ (\text{Librarian})$

select * from Librarian Where NAME LIKE 'Minh Tran' ;
$\pi \leftarrow$ (ID, NAME, ...)
($\sigma \leftarrow$ (NAME='Minh Tran') (Librarian))

SELECT * FROM TransactionDetails Order By PickupDate Asc;

$\tau_{\text{PickupDate ASC}}\ (\sigma_{*}(\text{TransactionDetails}))$

SELECT Status, COUNT(*) AS TotalReturned FROM TransactionDetails GROUP BY Status;

Rename ← p(TotalReturned)($_{Count(*)}$(TransactionDetails))
Proj ← $\pi_{Status, Rename}$ (TransactionDetails)
Final ← $\gamma_{Sttus}$ (Proj)

SELECT * From transactionDetails Where expiredate > '2023-01-20' ORDER by ExpireDate Desc;

Filt ← $\sigma_{expiredate > '2023 - 01 - 20'}$ (TransactionDetails)
Result ← $\delta_{Expiredate\ DESC}$ (Filt)

SELECT expireDate,MemID From TransactionDetails Where Status = 0 AND expiredate < SYSDATE;

Filt 1 ← $\sigma_{status = 0}$ (TransactionDetails)
Filt 2 ← $\sigma_{expiredate < SYSDATE}$ (TransactionDetails)
Filt ← Filt1 ∩ Filt2
Result ← $\pi_{expireDate, MemID}$ (Filt)

SELECT DISTINCT pickupdate From transactionDetails;

Result ← $\delta_{Pickupdate}$ (transactionDetails)

SELECT * From Transactiondetails WHERE Pickupdate BETWEEN '2023-01-10' AND '2023-01-15';

Filt 1 ← $\sigma_{Expiredate > 2023 - 01 - 10}$ (TransactionDetails)
Filt 2 ← $\sigma_{Expiredate < '2023 - 01 - 15'}$ (TransactionDetails)
Result ← Filt1 ∩ Filt2

SELECT * From TransactionDetails where (Status = 0) And (ExpireDate Between 2023-01-22' AND '2023-01-27');

Filt1 ← $\sigma_{Status = 0}$ (TransactionDetails)

Filt2 ← $\sigma_{\text{Expiredate‘ > 2023 - 01 - 22}}$ (TransactionDetails)
Filt 3 ← $\sigma_{\text{Expiredate < ‘2023 - 01 - 27’}}$ (TransactionDetails)
Filt ← Filt2 ∩ Filt3
Result ← Filt1 ∩ Filt

SELECT COUNT(libid), memid FROM feedback GROUP BY memid;

Proj ← $\pi_{\text{COUNT(libid), memid}}$ (feedback)
Result ← $\gamma_{\text{memid}}$ (Proj)

SELECT libid, stars, comments FROM feedback WHERE
stars >= (SELECT AVG(stars) FROM feedback) ORDER BY
stars DESC;

Proj ← $\pi_{\text{libid, stars, comments}}$ (feedback)
Aver ← $AVE_{\text{stars}}$ (Proj)
Filt ← $\sigma_{\text{stars >= Aver}}$ (Proj)
Result ← $\tau_{\text{members.memid ASC}}$ (Filt)

SELECT DISTINCT librarian.libid, librarian.name, feedback.stars, feedback.comments
FROM feedback INNER JOIN
librarian ON librarian.libid = feedback.libid WHERE librarian.name = 'Ho Kim Lee';

Joinedmembers ← librarian.libid ⋈ feedback.libid (Feedback)
Distinct ← $\delta_{\text{Joinedmembers, librarian.name, feedback.stars, feedback.comments}}$
Result ← $\sigma_{\text{librarian.namte = ‘Ho Kim Lee’}}$ (Distinct)

SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER
JOIN members ON members.memid = feedback.memid ORDER BY members.memid ASC;

Joinedmembers ← members.memid ⋈ feedback.memid (Feedback)
Proj ← $\pi_{\text{members.name, feedback.memid, feedback. Comments}}$ (feedback)
Result ← $\tau_{\text{members.memid ASC}}$ (Proj)

SELECT members.name, feedback.memid, feedback.comments FROM feedback INNER
JOIN members ON members.memid = feedback.memid where members.memid = 3 ORDER

BY comments;

JoinedMembers ← members.memid $\bowtie_{\text{members.memid = 3}}$ feedback.memid (feedback)
Proj ← $\pi_{\text{name, memid, comments}}$ (JoinedMem)
Result ← $\tau_{\text{comments}}$ (Joined)

SELECT feedback.memid, feedback.comments FROM
feedback WHERE feedback.comments LIKE '%highschool%';

Proj ← $\pi_{\text{memid,comments}}$ (feedback)
Result ← $\sigma_{\text{comments Like '%highschool'}}$ (Proj)

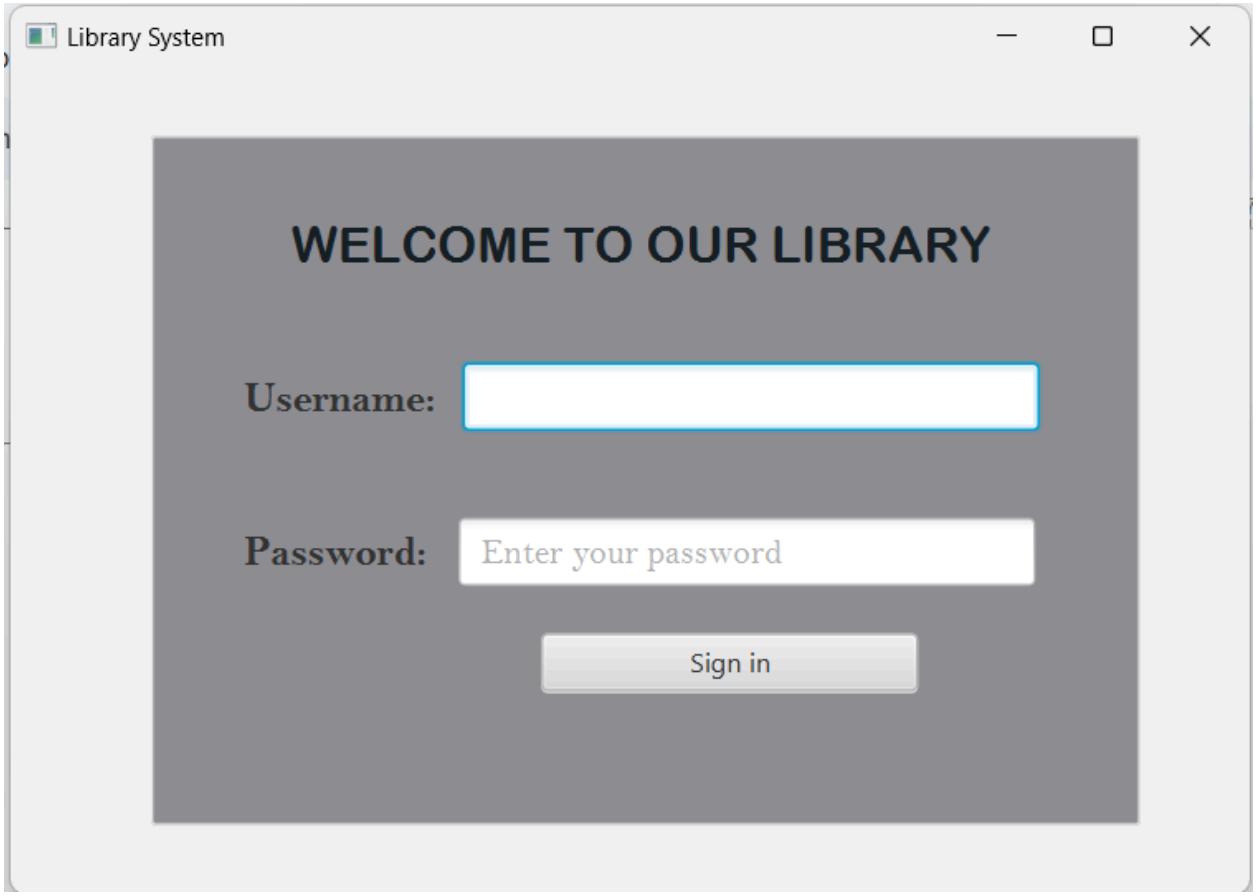
SELECT feedback.memid, feedback.stars,
feedback.comments FROM feedback WHERE feedback.stars = 5;

Proj ← $\pi_{\text{memid,stars,comments}}$ (feedback)
Result ← $\sigma_{\text{Stars = 5}}$ (Proj)

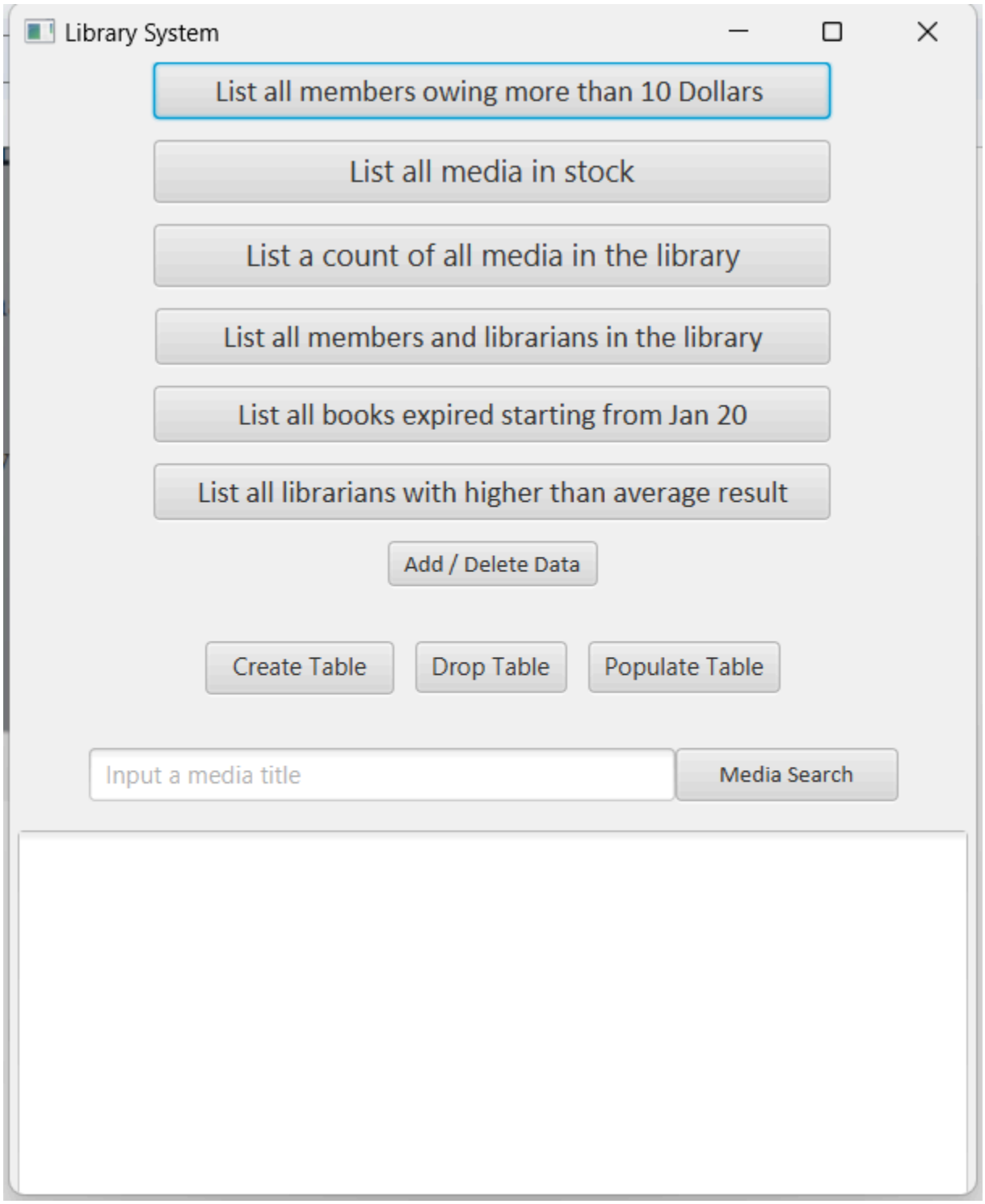
## Library System Graphical User Interface

---

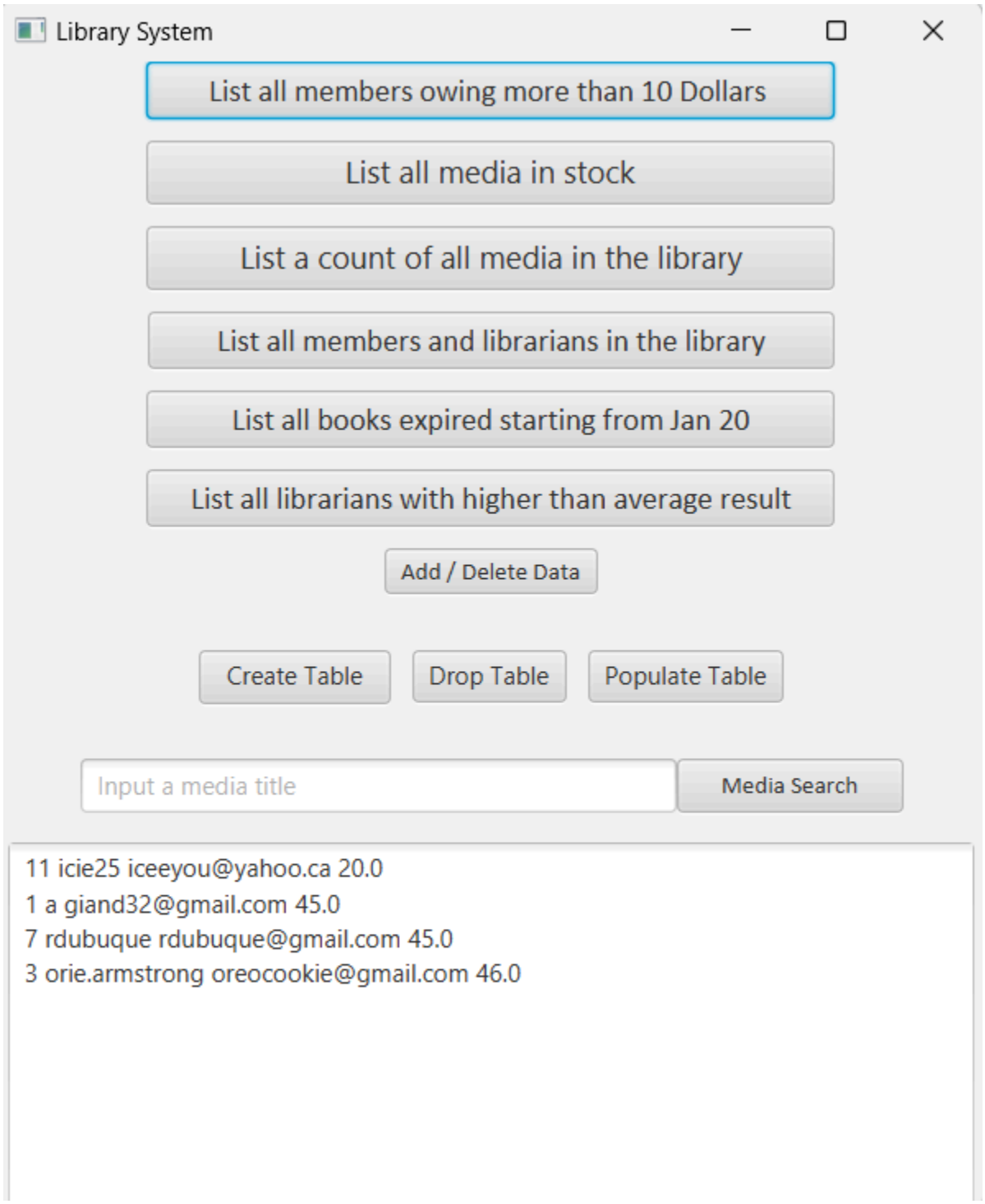
Login page for Members:



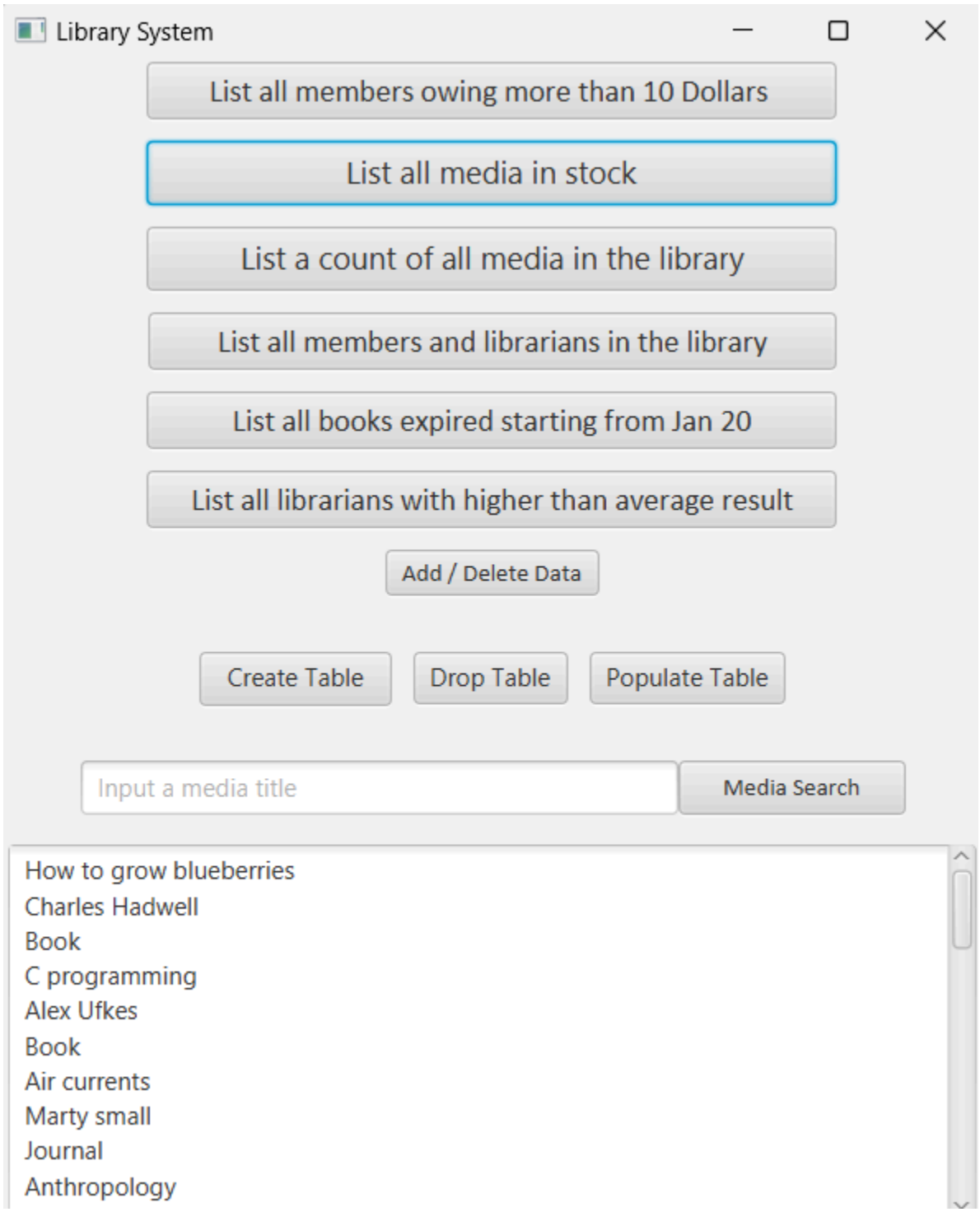
Main Library system after login for librarian:



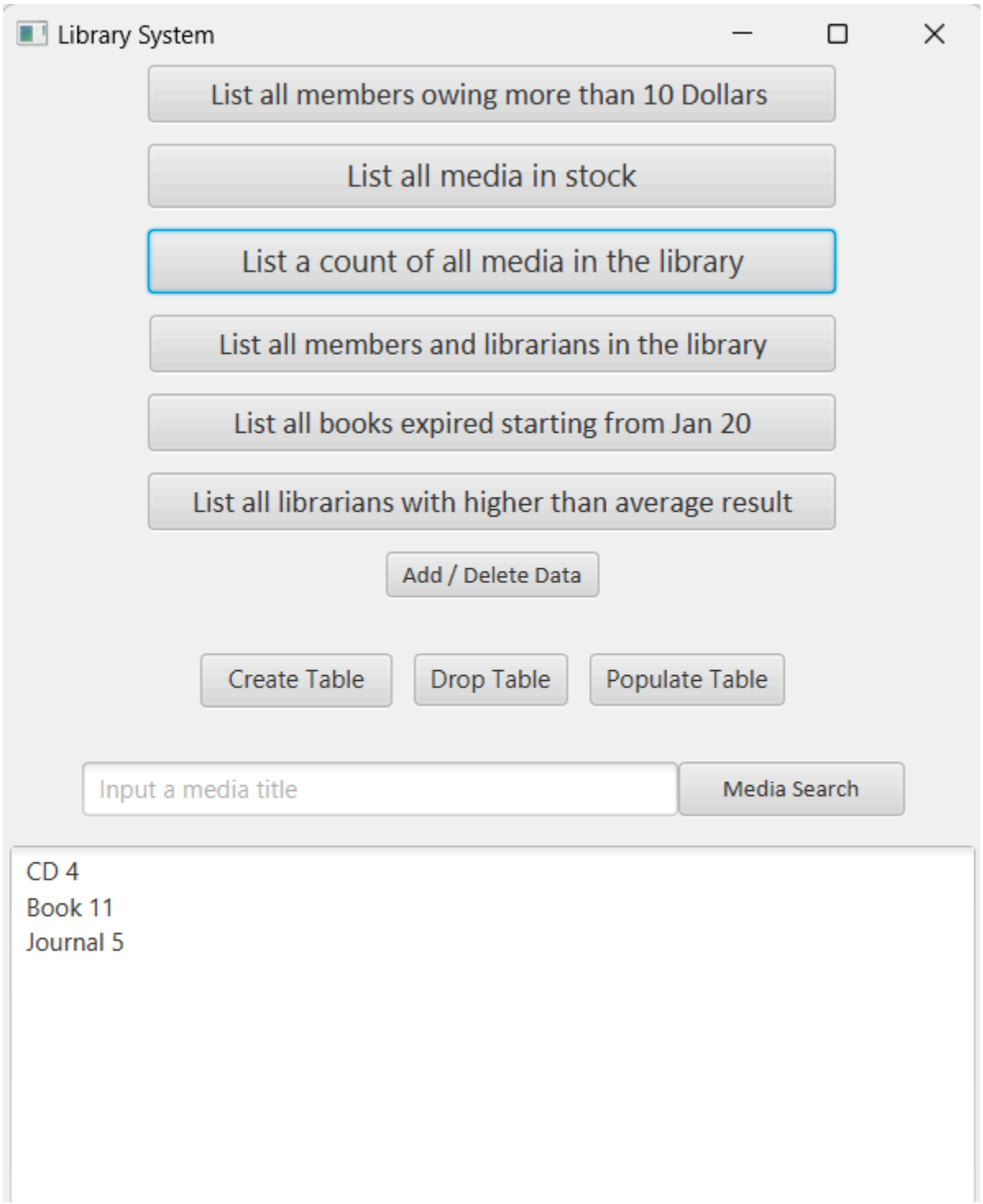
Query 1 : List all members Owing more than 10 dollars on javaFX graphical user interface



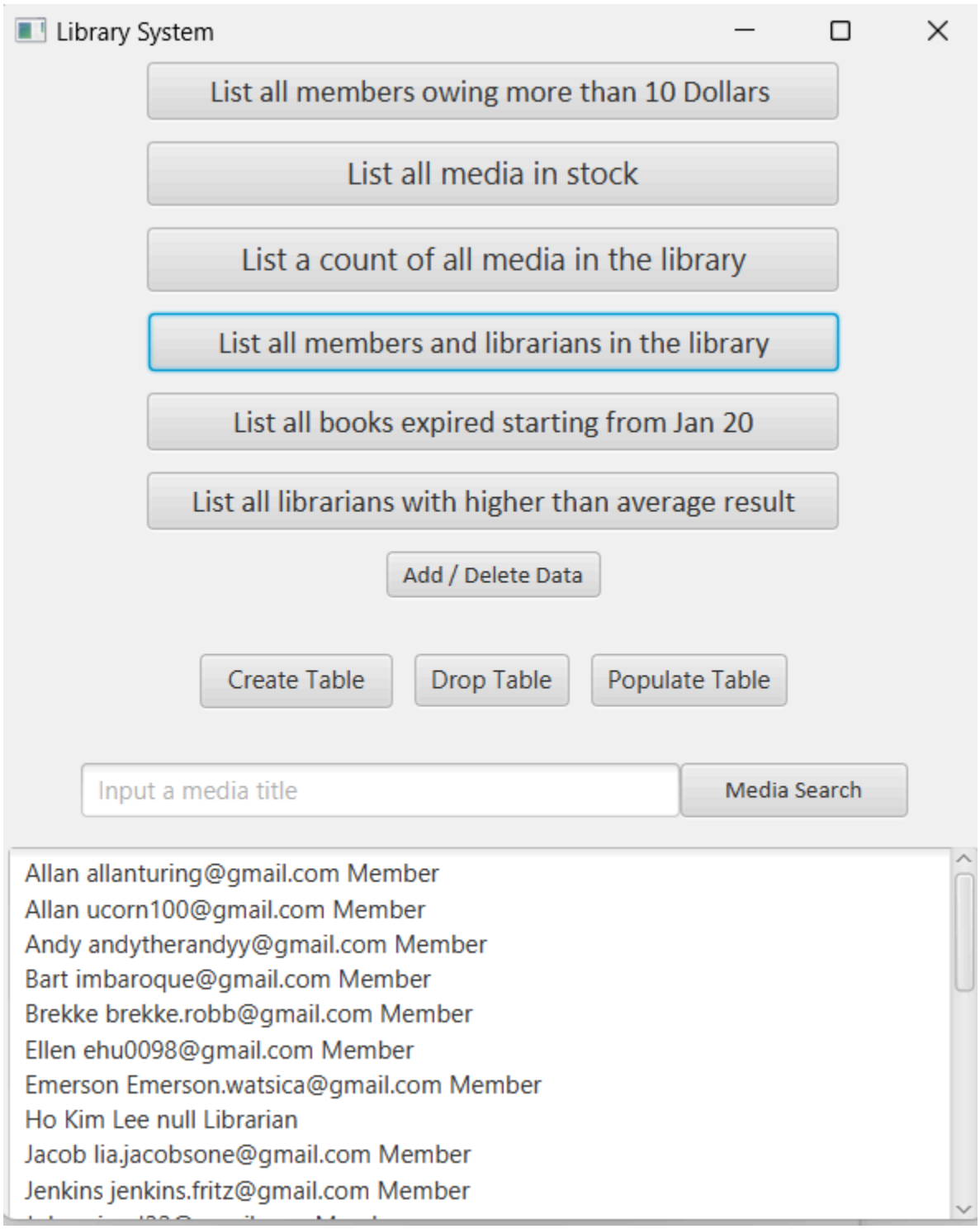
Query 2 : List all media in stock on javaFX graphical user interface



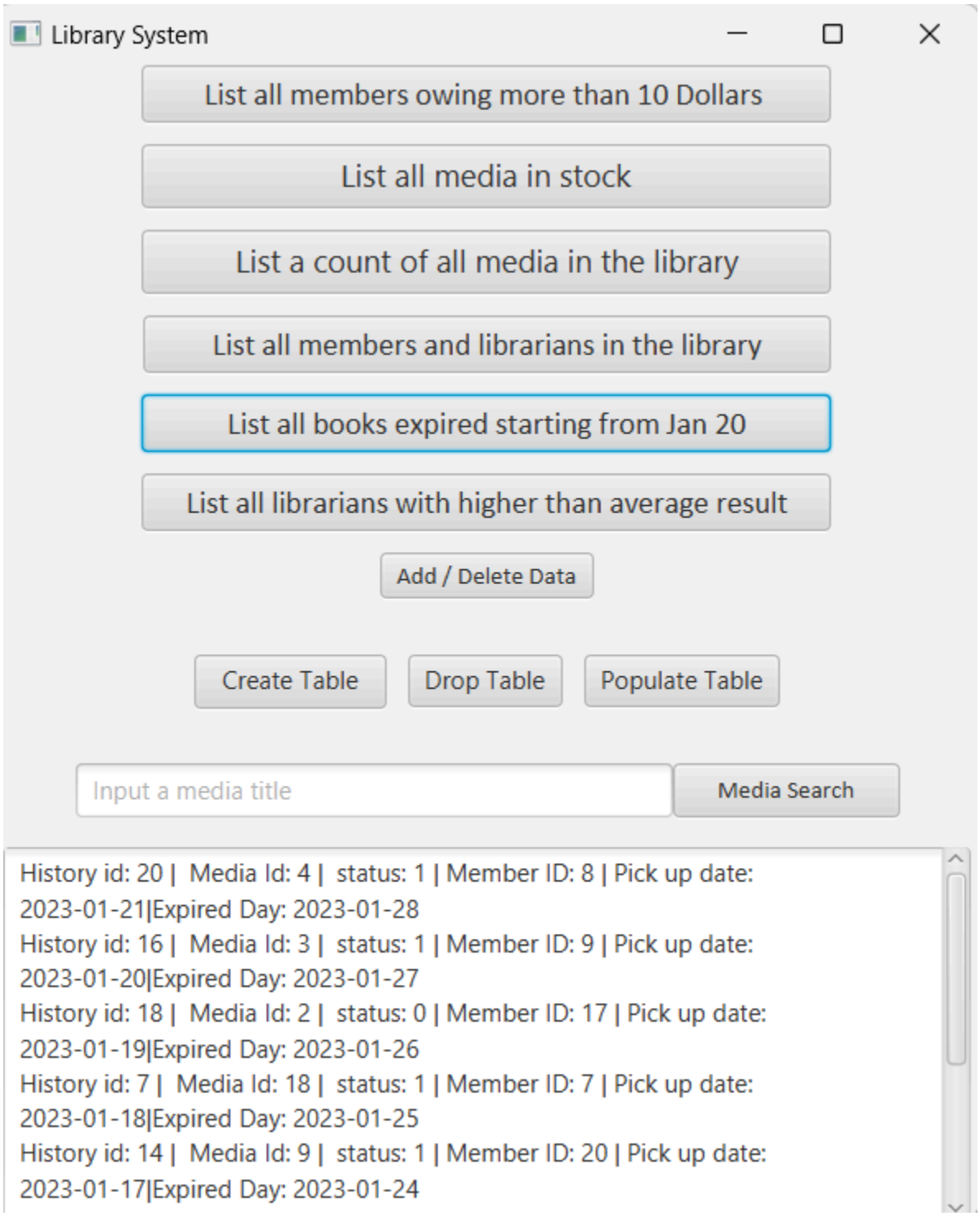
Query 3 : List a count of all media in the library using FX graphical user interface



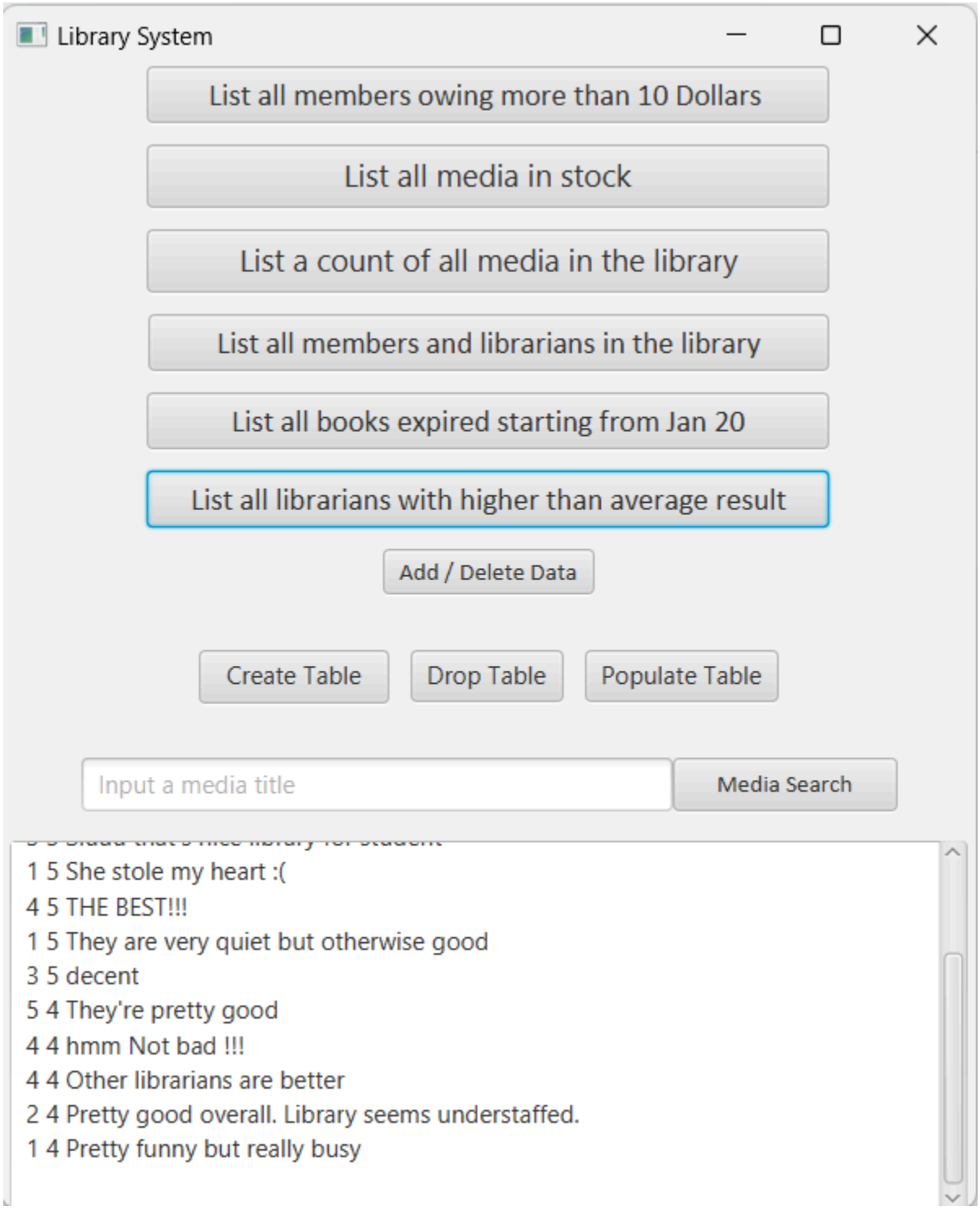
Query 4 : List all members and librarians on javaFX graphical user interface



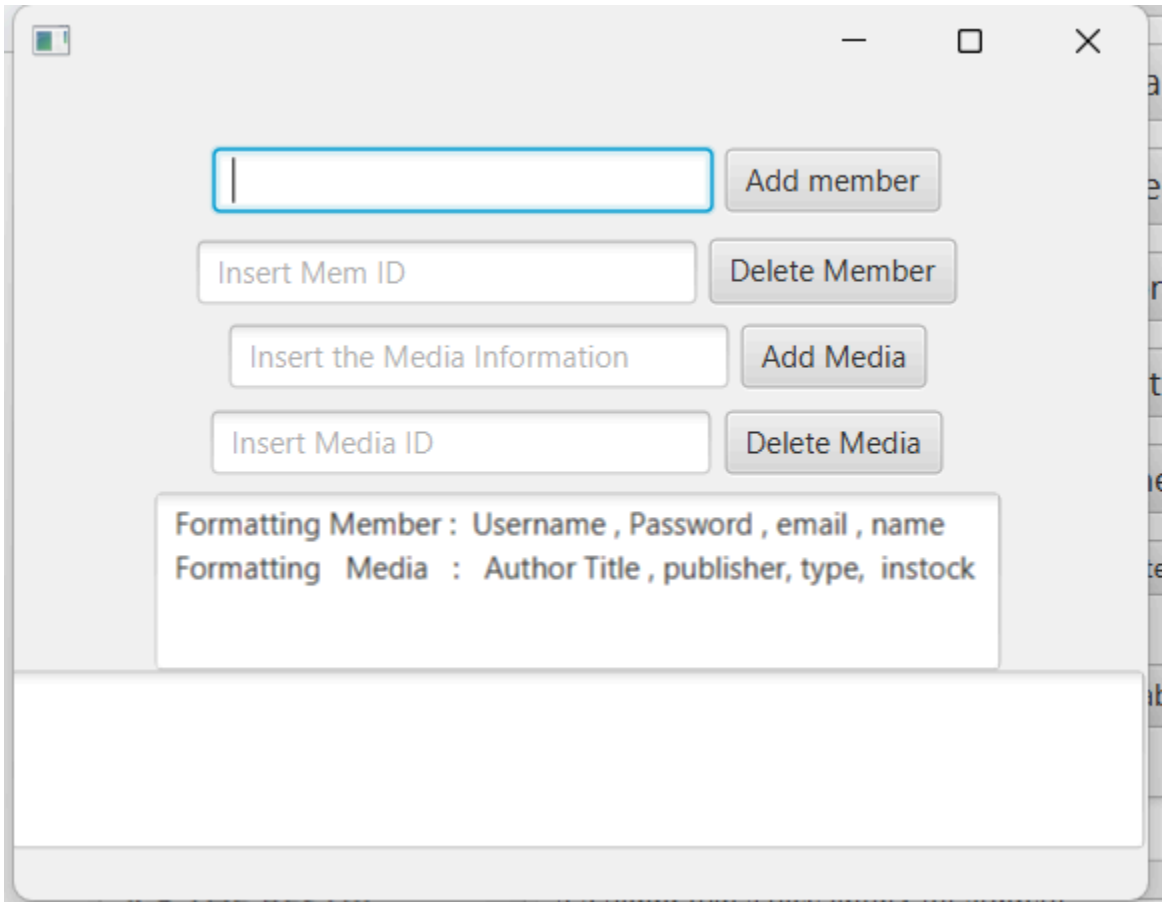
Query 5 : List all books expired starting from Jan 20 on javaFX graphical user interface



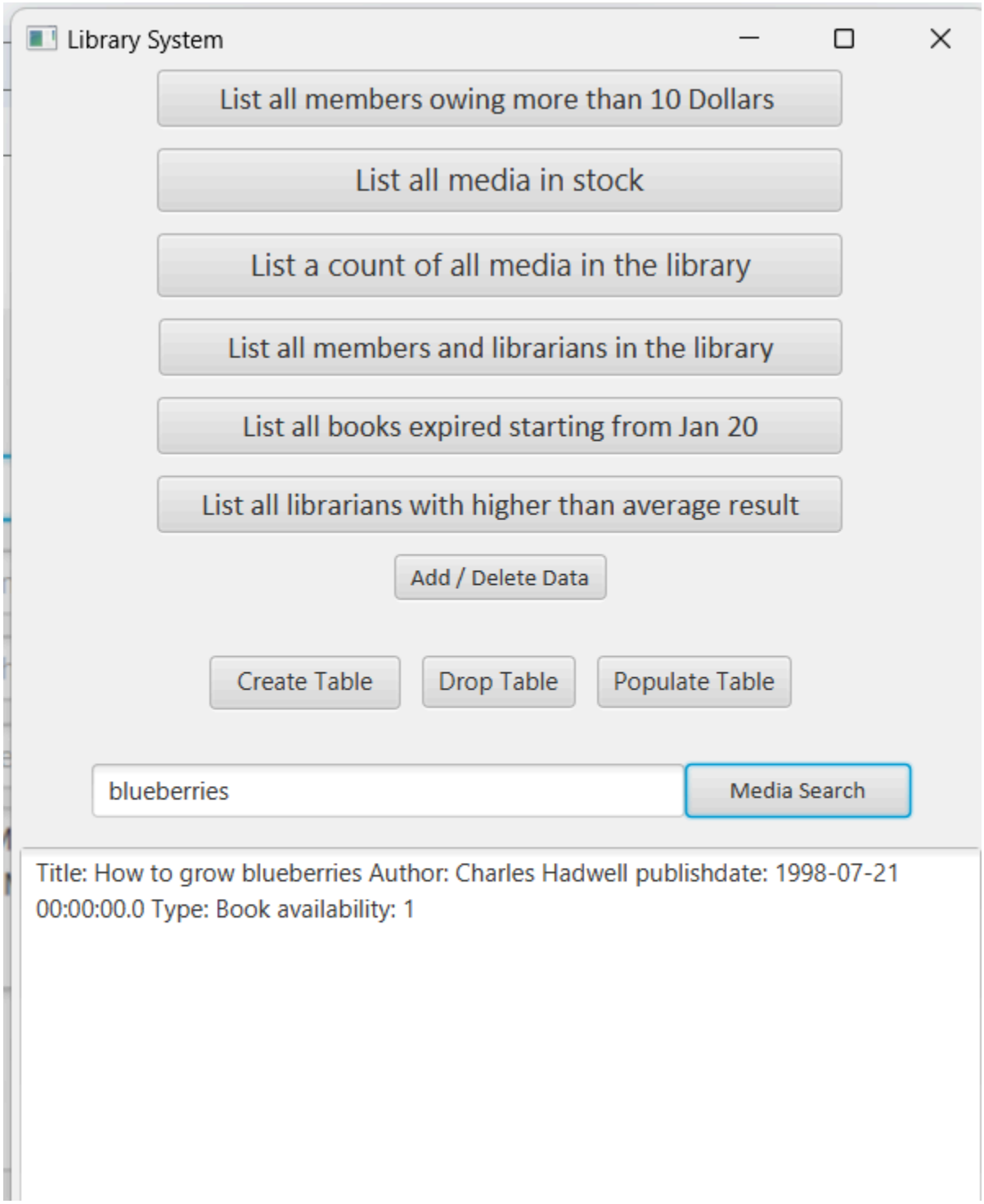
Query 6 : List all List all librarians with higher than average result on javaFX graphical user interface



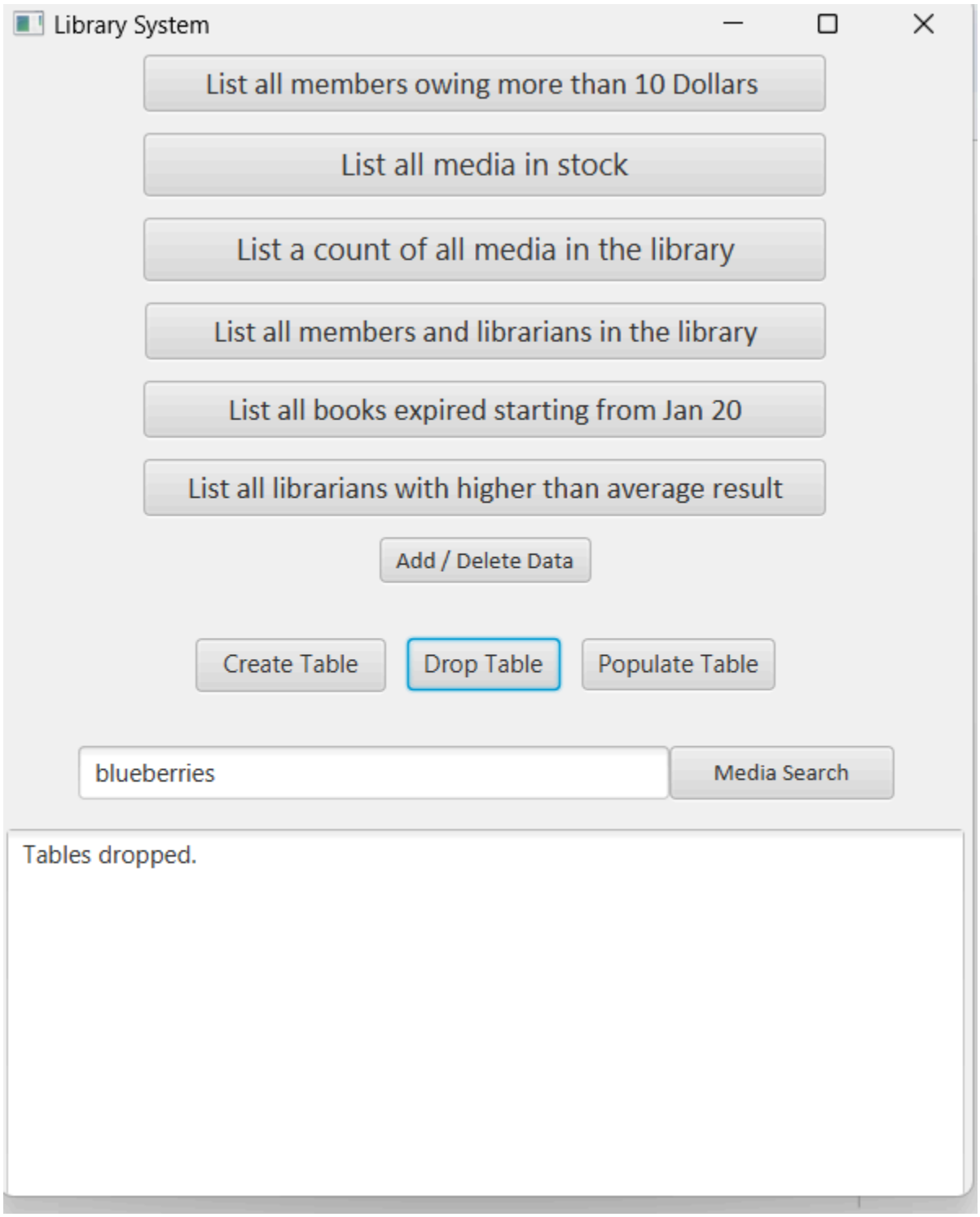
Adding and Deleting members and media using javaFX graphical user interface.



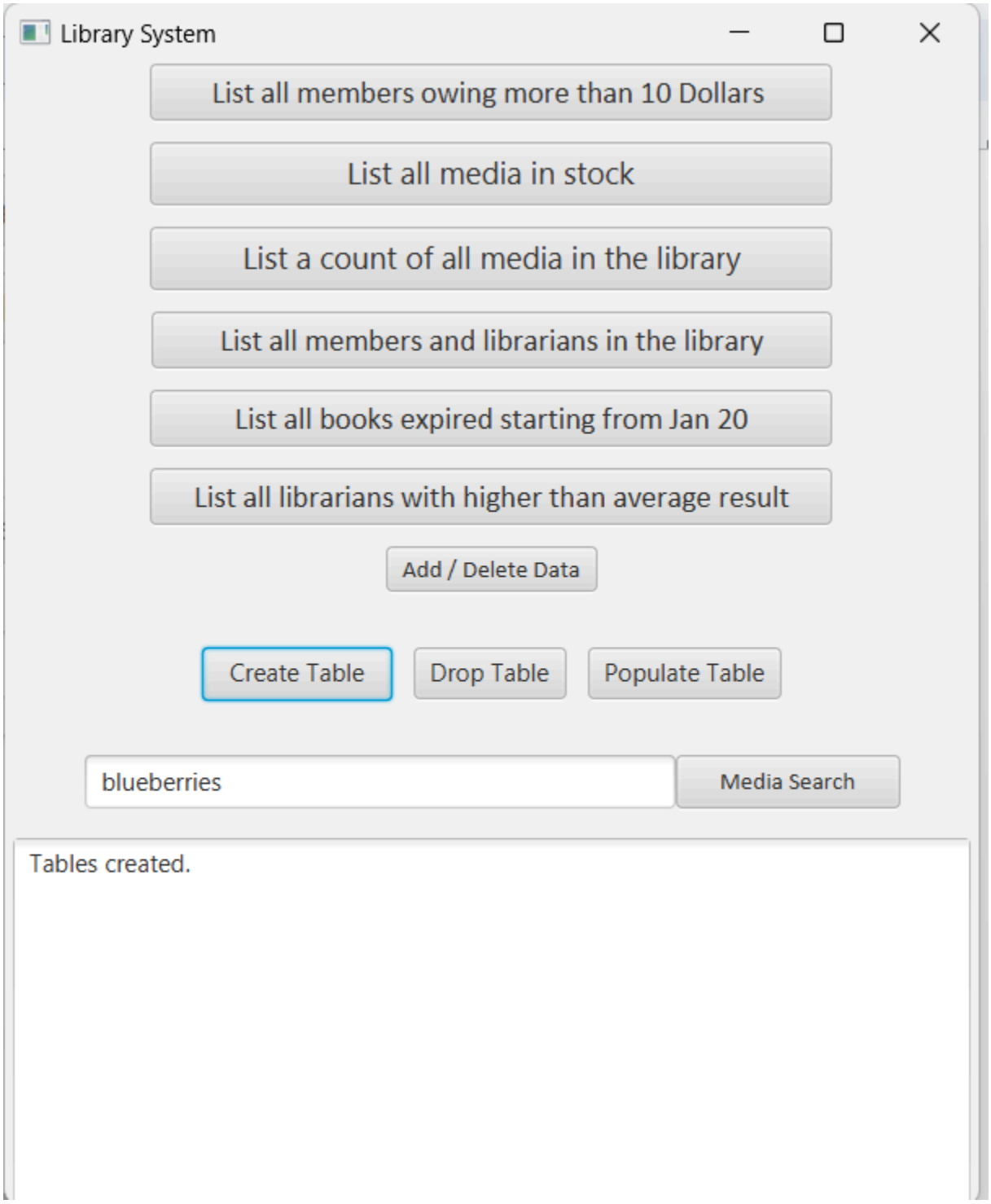
Query 7 : Entering Title keyword to search media on javaFX graphical user interface



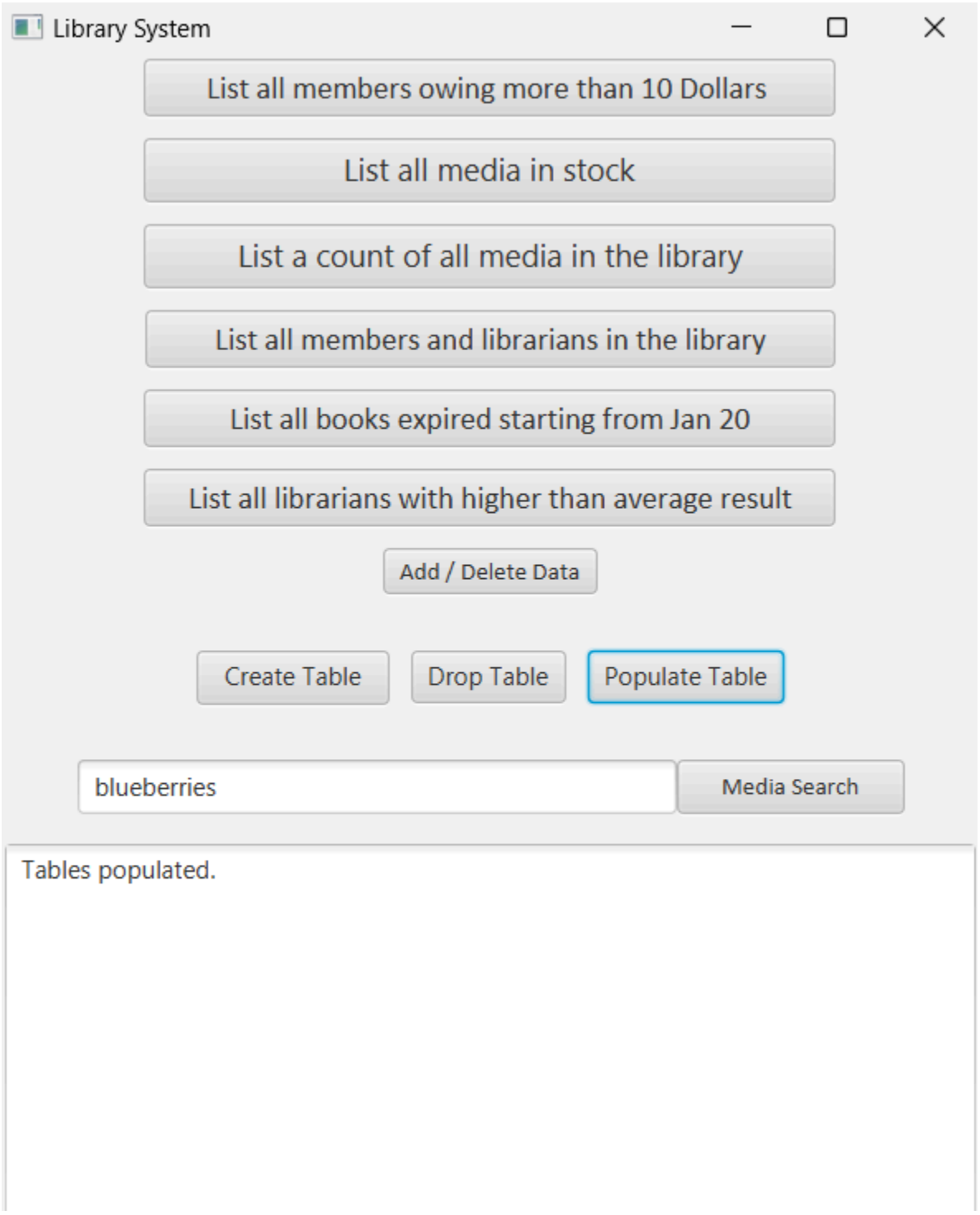
Dropping tables on JavaFX graphical user interface.



Creating tables on JavaFX graphical user interface.



Populate table on JavaFX graphical user interface.



# CONCLUSION

**Conclusion: Minh Tran**
Throughout our final project, we undertook ten assignments that guided us from conceptualization to developing a fully functional program capable of user interaction. These assignments deepened our comprehension of the SQL language and the Oracle database management system, particularly in working with data pertinent to our library system, including media, members, and transaction details.

Initiating assignments 1 and 2, we gained familiarity with entity concepts and ER diagrams, providing invaluable insights into translating our library system concept into a practical application. Assignment 3 marked the initial phase of constructing our SQL system by creating tables corresponding to our predefined entities—subsequently, assignment four involved formulating queries based on these entities crafting meaningful statements for our dataset.

While assignment 6 posed challenges in exporting queries via the terminal, teaching us to connect our SQL system with the school database and present results on the command line, assignments 7 and 8 delved into the intricacies of the third standard form and Bernstein algorithm. Assignment 9 proved the most complex, requiring us to grapple with formatting issues in integrating our work into the JavaFX interface.

Concluding with assignment 10, where we translated our SQL queries into relational algebra, the task demanded substantial time investment, given the approximately 40 queries developed in assignment 4. Despite the time consuming  -, this process significantly augmented our understanding of relational model operations and the SQL standard query language for relational database management systems. In retrospect, this project has been a valuable learning experience, and we express sincere gratitude for the guidance and support provided by our professor and TA.

**Conclusion: Woojin Lee**

Throughout the whole lab, 10 labs were conducted and were able to understand and demonstrate the real process of database management and programming. Although all the assignments were handed in without extensions or late submissions, there were a lot of conflicts and resolutions throughout working with the assignments.

As for example, more tables were needed to be added and the modification of the system had to be done during the first few weeks of the semester, resulting in a brainstorming until late night to think of a new design and attribute for the database. This has taught us that even in the real work environment, these stressful situations occur on a daily basis, and it was a preparation, or a tutorial of how to come to a resolution.

Additionally, having a great schema, and preparation was a key aspect to finish this project successfully. By creating and maintaining a great basis and schematics has helped our group to create an optimized plan that will help everyone to manage their time. However, the last few assignments have not been the case, where having multiple different projects from different courses at the same time has made time management difficult. However, it was done and the graphic component was also finished on time to submit.

Overall, this lab has encouraged the students to get a grasp of how the data management system works and how the DBMS operates. Making a mock database system has enhanced the understanding of the database system, as well as the coding aspect of the project. A lot of independent studying had to be conducted as well, but has become a new skill for myself.

**Conclusion: Talon Jiang**

The purpose of this project was to create a complex Library System using Oracle 11g and the Java programming language. Throughout the duration of this project, we underwent a series of assignments to take the Library System from a conceptual level to an interactive user interface complete with graphics, user interactions and complex queries to retrieve and write data to a centralized Oracle database.

Through the first few assignments, we laid the groundwork for the conceptual level of the database system through the extensive use of SQL Developer. By determining the functions and entities in the system, we derived ER and relational diagrams for the system. Due to some oversights, we recognized that our system did not initially include the amount of functionality required, thus it was necessary to redesign the system before proceeding with further development.

In later assignments, our team took the time to develop a prototype of the system based on the centralized server provided by the institution using Bash, which proved useful in both finalizing the initial designs of our tables, and ensuring that the queries given to the system were meaningful and functioned as expected. We encountered several issues with database connections due to unfamiliarity with the server initially, however we were soon able to launch the system on time with the expected functionality as required.

When designing the tables for the System, we were initially unaware about whether our tables were optimized into normal forms. Further analysis revealed that most of the tables, fortunately, were in 3rd Normal form, although there were instances where some tables required further work to normalize them to an acceptable level.

By far, the most challenging part of this project has been the implementation phase, where copious amounts of research and testing were required to bring the project to an acceptable level of functionality. Due to time constraints leading into the project, our team was unable to fully implement all of the functions outlined in our planning phases, however we were able to bring out functionality for the majority of functions within the database system, and plan to further refine and grow the service of the system in the future.

Overall, this project has brought into light important lessons when building complex systems in a team, alongside the importance of planning and communication while implementing similar projects in the future.