# Final Powerlifting DB Report Assignment

Youssef Cherrat, Samay Jamakayala, Jaimin Thakkar, Claire Kim

University of Virginia

CS 4750: Database Systems

Dr. Mary L. Smith

June 14, 2024

## Table of Contents

Names and Responsibilities ……………………………………………………………………2

Overview…………………………………………………………………………………..3

Description………………………………………………………………………………...3

Connection………………………………………………………………………………...4

Architecture………………………………………………………………………………..4

Business Rules……………………………………………………………………………….5

ERD………………………………………………………………………………………..7

Screens/UI…………………………………………………………………………………8

Executive Member Login Screen……………………………………………………….9

Full Member Log Screen ……………………………………………………….......10

Detailed Member Information Screen ………………………………………………...11

Add Member Screen………………………………………………………………..12

New Screens/UI…………………………………………………………………………13

Change Password Screen………………………………………………………...13

Alumni Screen……………………………………………………………………...14

SQL Script and Outputs…………………………………………………………………15

SQL Script Table Creation…………………………………………………..…...15

SQL Insert Statements………………………………………………………...……18

SQL Select Statements for Each Table……………………………….……………22

SQL Select Statements for Female Members……………………………………....22

SQL Select Statements that Summarize Data……………………………………....23

SQL Select Statements that Join Tables for a Master-Detail Report……………...…....24

SQL Select Statements for Associative Entities…………………………………………....27
Updated SQL Queries for Implementation……………………………………………….....29
Example Member Logins……………………………………………………………………...35
Example Executive Logins……………………………………………………………………36
References……………………………………………………………………………………......37

## Names and Responsibilities

- **Youssef Cherrat**
  - Handling the organization of business rules to ensure project structure matches Client's needs. Spearhead database design for proper matching of Client needs including the ERD Diagram, Database selection, and design choices in the front end.
- **Samay J**
  - Handling the frontend development, spearheading the proper design aspects and UI for the application. Will help with backend development regarding database aspects and ensure proper connection between frontend and backend.
- **Claire**
  - Handling the backend development of the database using SQLite and Java with Jaimin.
- **Jaimin**
  - Helping Claire with the backend development of the database using SQLite and Java\.

## Overview

This project aims to build a database for the Powerlifting Club at UVA. There are roughly three requirements needed for this database. One, it needs to be able to keep track of member attendance monthly. Second, the database should have the ability so that only admins are able to modify data and not members. Lastly, the database should be able to track competition requirements per academic year. The goal of this project is to create a database that streamlines administrative tasks, provides accurate and accessible records, and enhances member engagement to improve the overall management and operation of the club.

## Description

For context, Powerlifting is a strength sport that includes 3 events – the squat, bench, and the deadlift. The total weight in KG determines the placings in competitions. In the Powerlifting Club at UVA, there are a couple of requirements that need to be met in order to have a valid membership and a selection process that takes place to decide who will be selected for the annual Collegiate Nationals. For one, the club has a 2x per month attendance requirement for practices and a 1x per calendar year sanctioned competition requirement anywhere in the USAPL federation.

Additionally, the annual Collegiate National Meet requires each team to send only 2 lifters per weight class (male and female). As a result, the team has a policy that includes sending only the top 2 lifters in terms of quantitative amount (Lifting Total in KG). We hope to create an application that is effective and efficient in allowing executive members of the team to track attendance monthly for each member. To do so, we wish to create a database that would hold all the necessary data instead of the data being compiled in a spreadsheet. In addition to other information such as member name, age, and weight class, the database should be able to record the quantitative best total of each lifter to keep track of roster spots in each weight class for the Collegiate National Meet.

## Connection

The biggest connection to the organization is Youssef's role in the executive team of the UVAPL organization as a Membership Coordinator. Part of the role includes keeping track of member attendance, information, and other membership to uphold the rules and standards of the club. As a result, the Powerlifting Database was proposed to streamline the process for better data management, and a full stack solution to connect the issues at hand including the 2x per month attendance requirement and the 1x per academic year competition requirement.

## Architecture

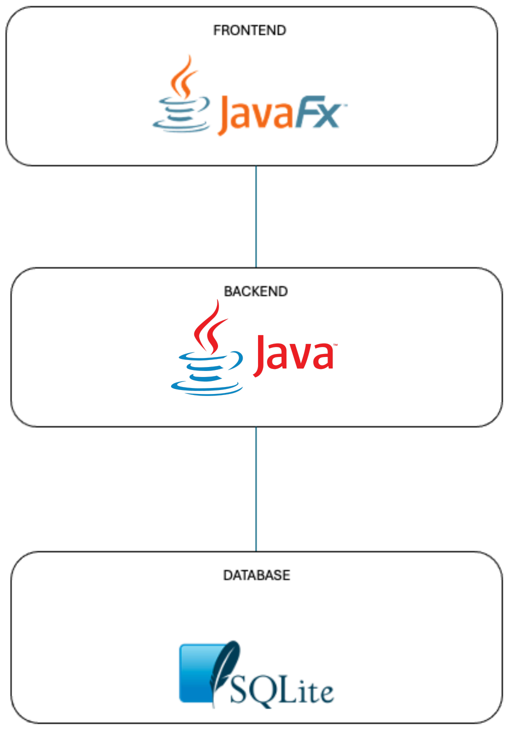


**Figure 1: Architecture Diagram**

The diagram shown in Figure 1 represents the project's structure. Displayed at the top is the planned frontend framework, JavaFX, for the user interface along with an appropriate Java controller. Python and Flask are the planned environments for the backend coding, as well as connecting the front end with our backend services. At the bottom is the SQLite database, which stores all relevant data. This setup ensures smooth communication between the user interface and the database, making the application efficient and effective.

## Business Rules

- A **year** will have a one-to-many relationship with **Semester**, as there can be multiple semesters in a year.
- A **year** will have a one-to-many relationship with **Competition**, as there can be multiple competitions in a year
- An updated **Member** list is maintained each **semester,** one semester to many members
    - Before a semester starts, members are added individually if they have **paid dues** for that semester before the deadline
    - If the semester end date matches a member's grad date at the end of the semester, the member can choose to be permanently added to an **Alumni** entity
    - **Member** information includes the ID, the link to that semester, the first name, last name, date of birth, Grad Date, Weight Class, Best Total KG, Gender, Email, and Password Hash
    - The Email and Password will be used for log-in for a member to only view their specific information, the hash is computed at execution and will be compared with the hash in the database.

- An updated **Alumni** list is maintained each semester, one semester for many alumni
    - Alumni can not log in, they're information can be viewed as a record for current Members and Executives for contact information
    - Alumni includes First Name, Last Name, Class Year, and Email
        - Only Alumni contact information is stored, they do not participate in practices or competitions
- **Practice** will have a many-to-many relationship with **Member** through **Attendance** with its own data
    - **Attendance** will bind a member to practice with a corresponding status for presence/absence
    - For each month, a member must have 2 **attendances** at practices within that month
- An updated **Executive** list is maintained each semester, one semester to many executives
    - An **Executive** is can not be a **Member** of the same person
    - The **Executive** with the role of **President** can add other Executives
    - Executive information includes First Name, Last Name, Role, Email, and Password Hash
    - The Email and Password will be used for log-in for a member to only view their specific information, the hash is computed at execution and will be compared with the hash in the database.
- **Members** have a many-to-many relationship with **Competitions** through **Competition_Member**

- Multiple members may compete in many competitions, and many competitions can hold many members
- **Competition_Member** links **Member** and **Competition** by junction table with the many-to-many relationship using Competition ID and Member ID
- **Competition** information includes Competition ID, Location, Year ID, and Competition Date

## ERD

The ERD diagram displayed in Figure 2 considers the business rules of the organization that include attendance requirement and competition requirement. As a result, a semester entity was needed, that would have a relationship to the member entity in order to keep track of all members for the given semester. Furthermore, members are attached to other many-to-many joint entities within the diagram in accordance with the business rules. Additionally, the Semester entity is connected to the Executive entity which should be able to control the information of Members. There is no need for establishing the redundancy of Membership between Semesters, as the attributes of Members can change, such as weight, gender, total lifted, and even a name change, therefore the member information can completely change for each individual.

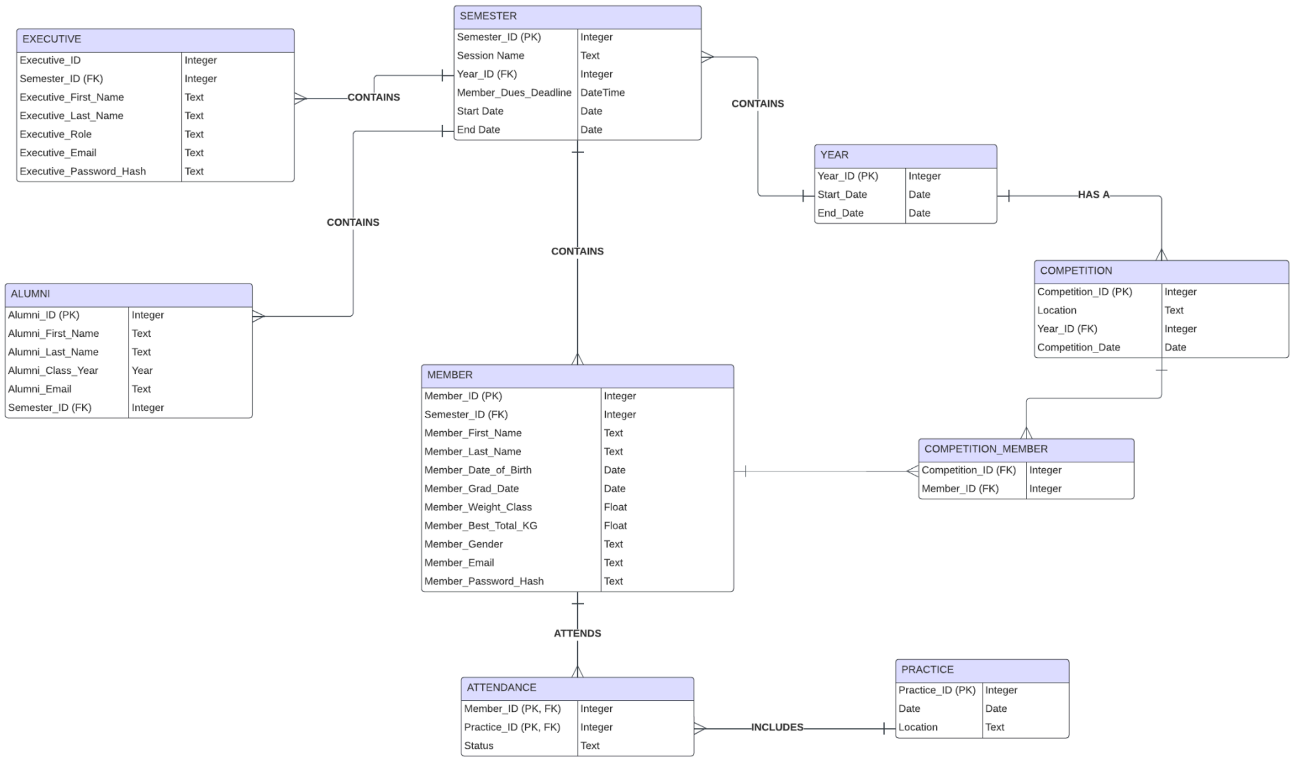


**Figure 2:** ERD of DB Design

## Screens/UI

The desktop application will include various screens for executive members to view and edit member logs to keep in the UVA Powerlifting Club's records. There will be a login screen where executives are able to login into the club's application where they are able to view member details, add members, and more. Additionally, there is a change password screen where users will have the option to change their password.

**Executive/Member Login Screen**

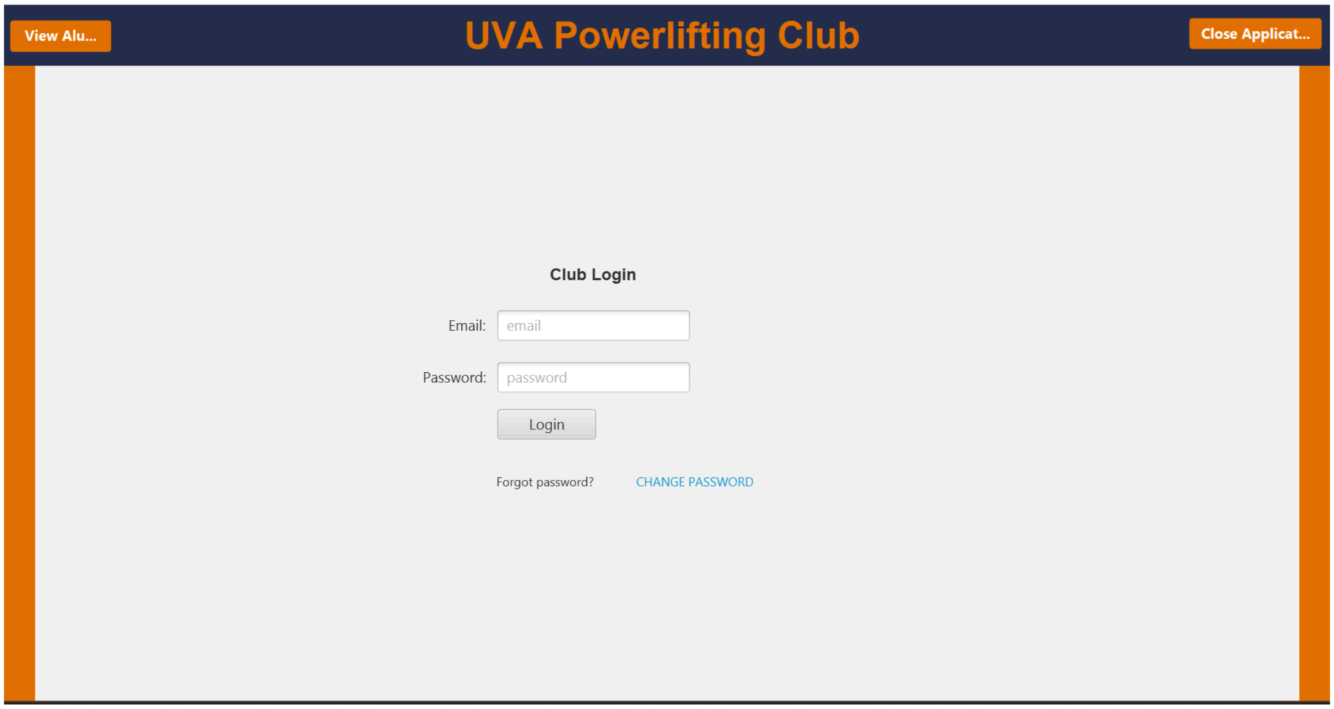
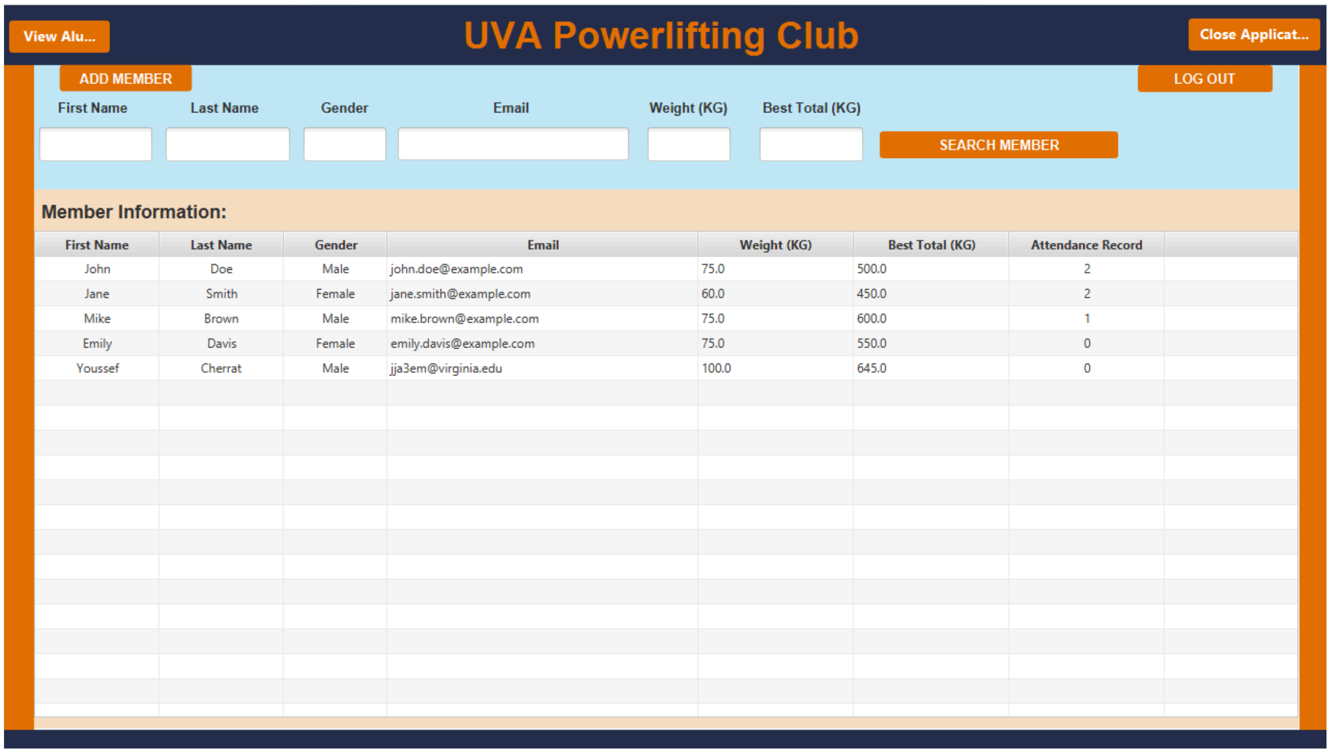


**Figure 3:** Executive/Member Login Screen

As shown in Figure 3, club users are presented with the initial login screen upon opening the application which requires an email and a password. New user accounts are created by an executive with a temporary password. These users have the option to change their password by clicking on the 'CHANGE PASSWORD' button.

## Full Member Log  Screen

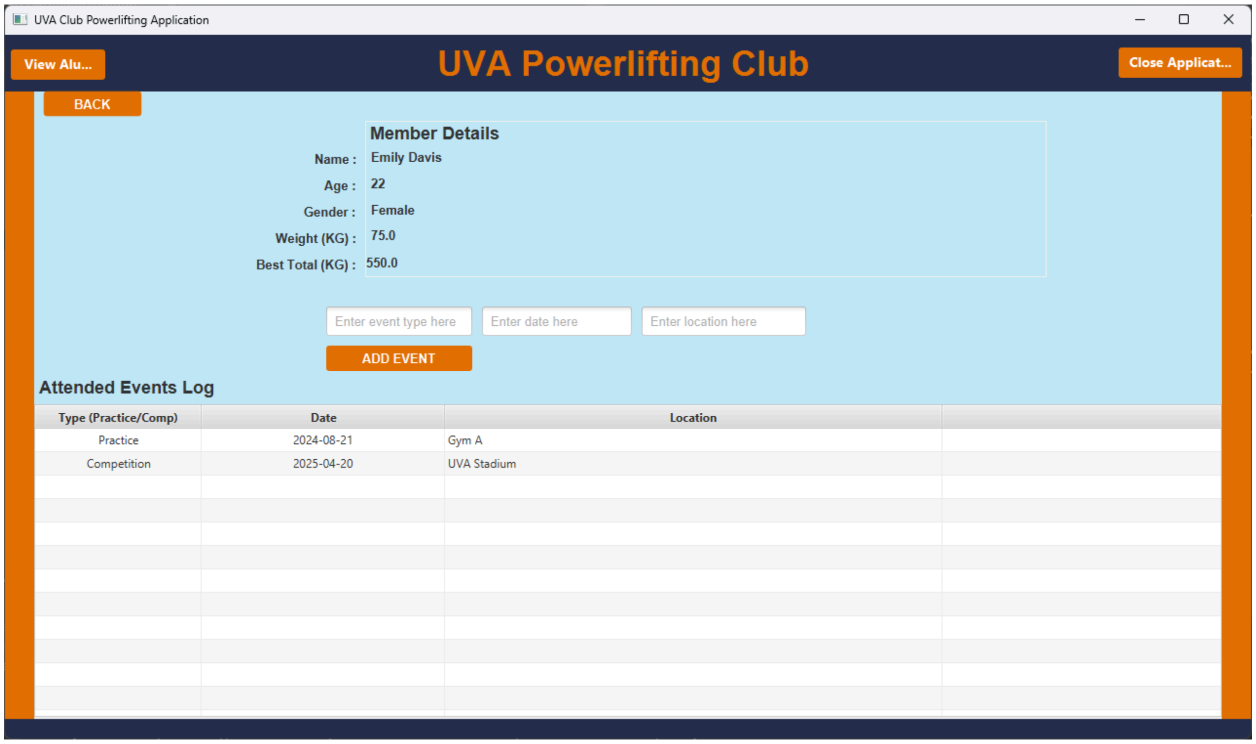


| First Name | Last Name | Gender | Email | Weight (KG) | Best Total (KG) | Attendance Record |
|---|---|---|---|---|---|---|
| John | Doe | Male | john.doe@example.com | 75.0 | 500.0 | 2 |
| Jane | Smith | Female | jane.smith@example.com | 60.0 | 450.0 | 2 |
| Mike | Brown | Male | mike.brown@example.com | 75.0 | 600.0 | 1 |
| Emily | Davis | Female | emily.davis@example.com | 75.0 | 550.0 | 0 |
| Youssef | Cherrat | Male | jja3em@virginia.edu | 100.0 | 645.0 | 0 |

**Figure 4:** Full Member Log Screen

Executives will be shown the screen displayed in Figure 4 upon logging in. Regular members will be taken straight to their own member information page. The large table will be filled with all member records, with each record having the basic fields: first name, last name, gender, email, weight (KG), best total (KG), and attendance record. This way, executives can view member logs to make sure each member has been meeting the minimum attendance requirements. Additionally, there is an option for executives to add members, provided they hit “add member” as opposed to the “search member” button. This aspect will be ensured by making sure that only logged in executives are able to utilize the add member button. The search functionality allows for easy navigation if executives are searching for a particular member.

## Detailed Member Information Screen



Member Details

| | |
|---|---|
| Name : | Emily Davis |
| Age : | 22 |
| Gender : | Female |
| Weight (KG) : | 75.0 |
| Best Total (KG) : | 550.0 |

Attended Events Log

| Type (Practice/Comp) | Date | Location | |
|---|---|---|---|
| Practice | 2024-08-21 | Gym A | |
| Competition | 2025-04-20 | UVA Stadium | |

**Figure 5:** Detailed Member Information Screen

If a particular member is selected from the previous screen's table (Figure 4), more detailed information regarding each member will be displayed. This is also the screen that members will see when they login. Figure 5 above shows the details for member Jane Smith as an example. In addition to their name, their age, gender, weight (KG), and best total lift (KG) can be seen. The table at the bottom of the image displays their attended events log so that executives can view what practices/competitions that members have attended. Executives are also able to edit and add new events as needed. Furthermore, if needed, executives can have their own personal notes in the title/description field for certain members if they feel that they did particularly well in a given session/event. This screen is the only access regular members will have regarding information viewing, as they will not be able to view other members' information.

## Add Member Screen

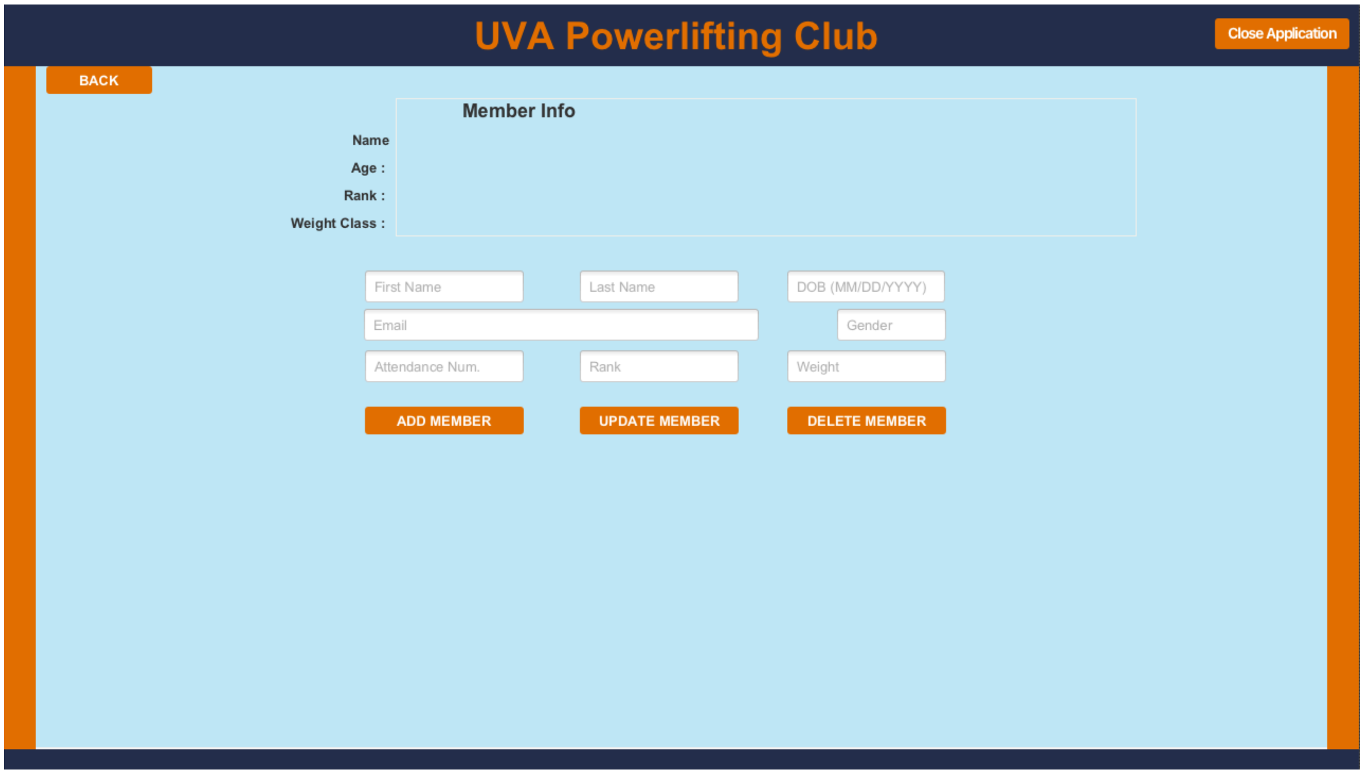


**Figure 6:** Add Member Screen

Upon clicking the add member button, executives will reach the screen displayed in Figure 6. This screen is only accessible by executive members, while members are able to view only their own information.

Overall, these primary screens represent the overall UI for the application. The database will be presented in the form of the tables mentioned, and additional information may be added based on further testing and feedback.

## New Screens/UI

### Change Password Screen

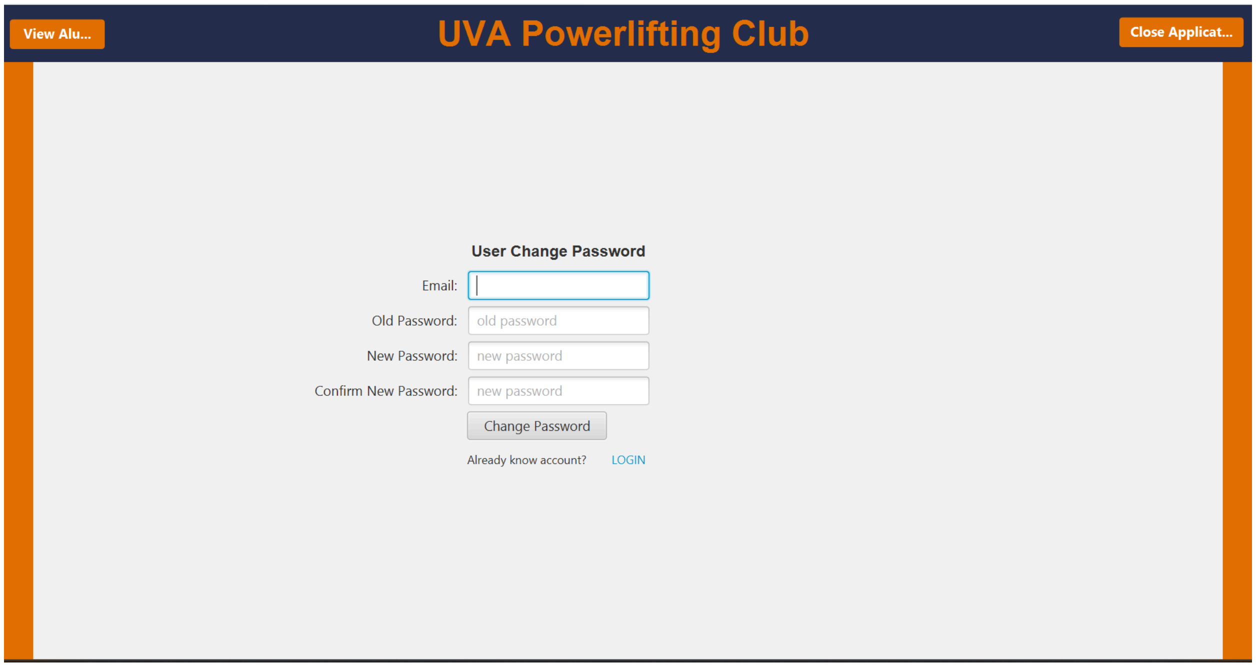
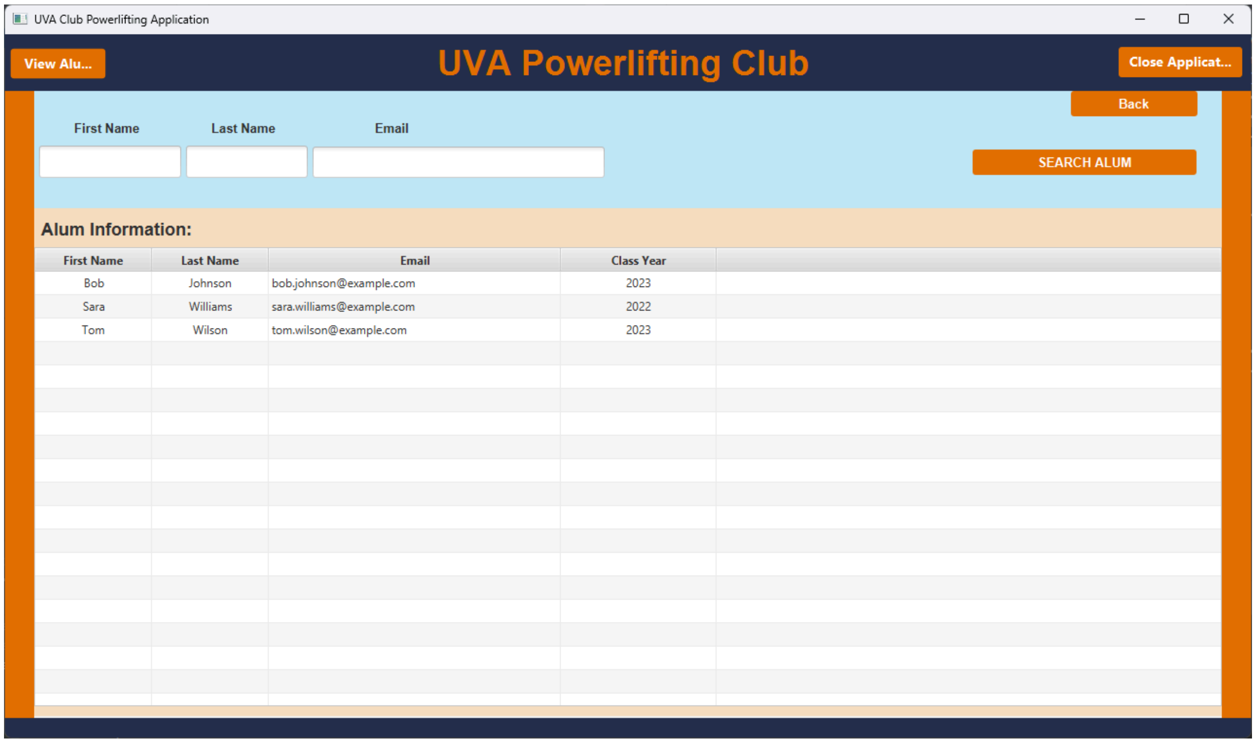


**Figure 7:** New Member Password Change Screen

Clicking on the 'CHANGE PASSWORD' button will display the User Change Password screen as shown in Figure 7. The user will be prompted to put in their email, old password (the temporary password), and the new password they wish to have. After clicking 'Change Password', the password will be updated, and the user will be able to Login using their new password. When a new password is created,

### Alumni Screen



Alum Information:

| First Name | Last Name | Email | Class Year |
|---|---|---|---|
| Bob | Johnson | bob.johnson@example.com | 2023 |
| Sara | Williams | sara.williams@example.com | 2022 |
| Tom | Wilson | tom.wilson@example.com | 2023 |

**Figure 8:** Alumni Information Screen

Clicking on the View Alumni button leads to the Alumni Information page as seen in Figure 8. The table displays an alum's first name, last name, email address, and the year they graduated. Additionally, an alum can conveniently be looked up by entering their name and/or email and clicking the Search Alum button on the right hand side.

## SQL Script and Outputs

**SQL Script Table Creation**

A SQL script that creates all your tables. Primary key and foreign key constraints must be included. Check constraints must be included as well, where applicable.

```
USE powerlifting;
CREATE TABLE IF NOT EXISTS Year (
   Year_ID INTEGER PRIMARY KEY,
   Start_Date DATE,
   End_Date DATE
);

CREATE TABLE IF NOT EXISTS Semester (
   Semester_ID INTEGER PRIMARY KEY,
   Year_ID INTEGER,
   Session_Name TEXT,
   Member_Dues_Deadline DATETIME,
   Start_Date DATE,
   End_Date DATE,
   FOREIGN KEY (Year_ID) REFERENCES Year(Year_ID)
);

CREATE TABLE IF NOT EXISTS Member (
   Member_ID INTEGER PRIMARY KEY,
   Semester_ID INTEGER,
   Member_First_Name TEXT,
   Member_Last_Name TEXT,
   Member_Date_of_Birth DATE,
   Member_Grad_Date DATE,
   Member_Weight_Class FLOAT,
   Member_Best_Total_KG FLOAT,
   Member_Gender TEXT,
   Member_Email TEXT,
   Member_Password_Hash TEXT,
   FOREIGN KEY (Semester_ID) REFERENCES Semester(Semester_ID)
);

CREATE TABLE IF NOT EXISTS Executive (
   Executive_ID INTEGER PRIMARY KEY,
   Semester_ID INTEGER,
   Executive_First_Name TEXT,
   Executive_Last_Name TEXT,
   Executive_Role TEXT,
   Executive_Email TEXT,
   Executive_Password_Hash TEXT,
   FOREIGN KEY (Semester_ID) REFERENCES Semester(Semester_ID)
);
```

```
CREATE TABLE IF NOT EXISTS Alumni (
    Alumni_ID INTEGER PRIMARY KEY,
    Alumni_First_Name TEXT,
    Alumni_Last_Name TEXT,
    Alumni_Class_Year INTEGER,
    Alumni_Email TEXT,
    Semester_ID INTEGER,
    FOREIGN KEY (Semester_ID) REFERENCES Semester(Semester_ID)
);

CREATE TABLE IF NOT EXISTS Practice (
    Practice_ID INTEGER PRIMARY KEY,
    Date DATE,
    Location TEXT
);

CREATE TABLE IF NOT EXISTS Attendance (
    Member_ID INTEGER,
    Practice_ID INTEGER,
    Status TEXT,
    PRIMARY KEY (Member_ID, Practice_ID),
    FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID),
    FOREIGN KEY (Practice_ID) REFERENCES Practice(Practice_ID)
);

CREATE TABLE IF NOT EXISTS Competition (
    Competition_ID INTEGER PRIMARY KEY,
    Location TEXT,
    Year_ID INTEGER,
    Competition_Date DATE,
    FOREIGN KEY (Year_ID) REFERENCES Year(Year_ID)
);

CREATE TABLE IF NOT EXISTS Competition_Member (
    Competition_ID INTEGER,
    Member_ID INTEGER,
    PRIMARY KEY (Competition_ID, Member_ID),
    FOREIGN KEY (Competition_ID) REFERENCES Competition(Competition_ID),
    FOREIGN KEY (Member_ID) REFERENCES Member(Member_ID)
);
```

**SQL Insert Statements**

SQL insert statements to fill your tables with initial data. Include enough data to show proper testing of your SQL select statements below.

```
-- Insert into Year
```

```
INSERT INTO Year (Year_ID, Start_Date, End_Date) VALUES
(1, '2024-01-01', '2024-12-31'),
(2, '2025-01-01', '2025-12-31');

-- Insert into Semester
INSERT INTO Semester (Semester_ID, Year_ID, Session_Name,
Member_Dues_Deadline, Start_Date, End_Date)
VALUES (1, 1, 'Spring 2024', '2024-01-15 23:59:59', '2024-01-10', '2024-05-20'),
    (2, 1, 'Fall 2024', '2024-08-15 23:59:59', '2024-08-20', '2024-12-15'),
    (3, 2, 'Spring 2025', '2025-01-15 23:59:59', '2025-01-10', '2025-05-20');

-- Insert into Member
INSERT INTO Member (Member_ID, Semester_ID, Member_First_Name,
Member_Last_Name, Member_Date_of_Birth, Member_Grad_Date,
Member_Weight_Class, Member_Best_Total_KG, Member_Gender, Member_Email,
Member_Password_Hash)
VALUES (1, 1, 'John', 'Doe', '2000-01-01', '2024-05-20', 75.0, 500.0, 'Male',
'john.doe@example.com', 'hash1'),
    (2, 1, 'Jane', 'Smith', '2001-02-01', '2025-05-20', 60.0, 450.0, 'Female',
'jane.smith@example.com', 'hash3'),
    (3, 2, 'Mike', 'Brown', '1999-03-03', '2024-12-15', 82.5, 600.0, 'Male',
'mike.brown@example.com', 'hash4'),
    (4, 3, 'Emily', 'Davis', '2002-04-04', '2025-12-15', 70.0, 550.0, 'Female',
'emily.davis@example.com', 'hash5');

-- Insert into Executive
INSERT INTO Executive (Executive_ID, Semester_ID, Executive_First_Name,
Executive_Last_Name, Executive_Role, Executive_Email, Executive_Password_Hash)
VALUES (1, 1, 'Alice', 'Smith', 'President', 'alice.smith@example.com', 'hash2'),
    (2, 2, 'Robert', 'Lee', 'Vice President', 'robert.lee@example.com', 'hash6'),
    (3, 3, 'Linda', 'White', 'Secretary', 'linda.white@example.com', 'hash7');

-- Insert into Alumni
INSERT INTO Alumni (Alumni_ID, Alumni_First_Name, Alumni_Last_Name,
Alumni_Class_Year, Alumni_Email, Semester_ID)
VALUES (1, 'Bob', 'Johnson', 2023, 'bob.johnson@example.com', 1),
    (2, 'Sara', 'Williams', 2022, 'sara.williams@example.com', 1),
    (3, 'Tom', 'Wilson', 2023, 'tom.wilson@example.com', 2);

-- Insert into Practice
INSERT INTO Practice (Practice_ID, Date, Location)
VALUES (1, '2024-02-01', 'Gym A'),
    (2, '2024-02-03', 'Gym B'),
    (3, '2024-02-05', 'Gym C'),
    (4, '2024-08-21', 'Gym A');
```

```
-- Insert into Attendance
INSERT INTO Attendance (Member_ID, Practice_ID, Status)
VALUES (1, 1, 'Present'),
    (1, 2, 'Present'),
    (1, 3, 'Absent'),
    (2, 2, 'Present'),
    (2, 4, 'Present'),
    (3, 3, 'Present'),
    (4, 4, 'Absent');
-- Insert into Competition
INSERT INTO Competition (Competition_ID, Location, Year_ID, Competition_Date)
VALUES (1, 'City Arena', 1, '2024-03-15'),
    (2, 'Downtown Gym', 1, '2024-10-10'),
    (3, 'UVA Stadium', 2, '2025-04-20');

-- Insert into Competition_Member
INSERT INTO Competition_Member (Competition_ID, Member_ID)
VALUES (1, 1),
    (1, 2),
    (2, 1),
    (2, 3),
    (3, 4);
```

**SQL Select Statements for Each Table**

A SQL select statement for each table that shows all rows.

SELECT * FROM Year;

SELECT * FROM Semester;

SELECT * FROM Member;

SELECT * FROM Executive;

SELECT * FROM Alumni;

SELECT * FROM Practice;

SELECT * FROM Attendance;

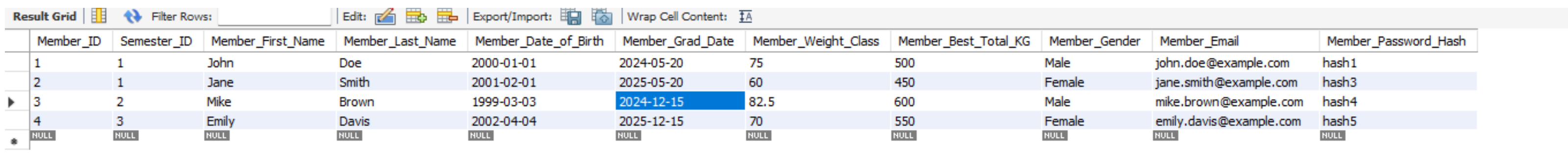
SELECT * FROM Competition;

SELECT * FROM Competition_Member;

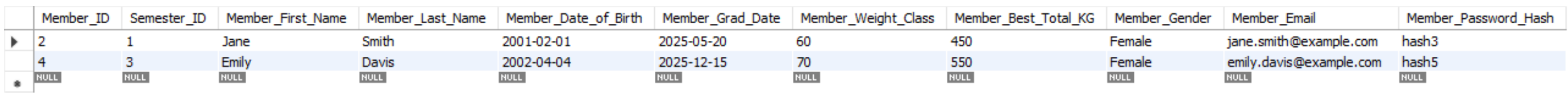
**SQL Select Statements for Female Members**

SQL select statements that shows only the Female Member List

Before:



| Member_ID | Semester_ID | Member_First_Name | Member_Last_Name | Member_Date_of_Birth | Member_Grad_Date | Member_Weight_Class | Member_Best_Total_KG | Member_Gender | Member_Email | Member_Password_Hash |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | John | Doe | 2000-01-01 | 2024-05-20 | 75 | 500 | Male | john.doe@example.com | hash1 |
| 2 | 1 | Jane | Smith | 2001-02-01 | 2025-05-20 | 60 | 450 | Female | jane.smith@example.com | hash3 |
| 3 | 2 | Mike | Brown | 1999-03-03 | 2024-12-15 | 82.5 | 600 | Male | mike.brown@example.com | hash4 |
| 4 | 3 | Emily | Davis | 2002-04-04 | 2025-12-15 | 70 | 550 | Female | emily.davis@example.com | hash5 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |



| Member_ID | Semester_ID | Member_First_Name | Member_Last_Name | Member_Date_of_Birth | Member_Grad_Date | Member_Weight_Class | Member_Best_Total_KG | Member_Gender | Member_Email | Member_Password_Hash |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | Jane | Smith | 2001-02-01 | 2025-05-20 | 60 | 450 | Female | jane.smith@example.com | hash3 |
| 4 | 3 | Emily | Davis | 2002-04-04 | 2025-12-15 | 70 | 550 | Female | emily.davis@example.com | hash5 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

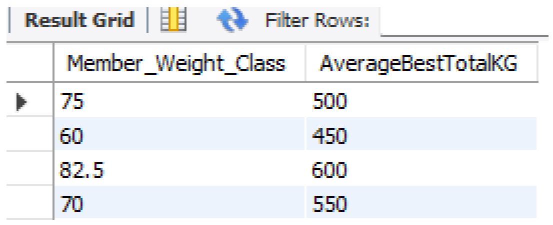
After:

SELECT * FROM Member

WHERE Member_Gender = 'Female';

**SQL Select Statements that Summarize Data**

a. This query groups the members by their weight class and calculates the average best total weight lifted for each class. The result will show the weight class and the average best total KG. This is an average across all members in that weight class to showcase the overall strength of the team for each weight class.



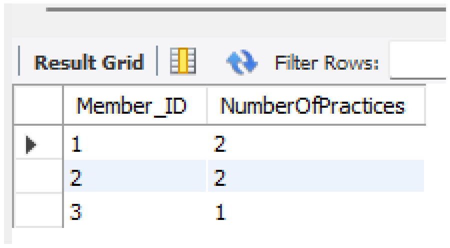
| Member_Weight_Class | AverageBestTotalKG |
|---|---|
| 75 | 500 |
| 60 | 450 |
| 82.5 | 600 |
| 70 | 550 |

SELECT Member_Weight_Class, AVG(Member_Best_Total_KG) AS AverageBestTotalKG

FROM Member

GROUP BY Member_Weight_Class;

b. This query groups the attendance records by Member_ID and counts the number of practices attended (where the status is 'Present') by each member. The result will show the member ID and the number of practices they attended.


Result Grid Filter Rows:

| Member_ID | NumberOfPractices |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 3 | 1 |

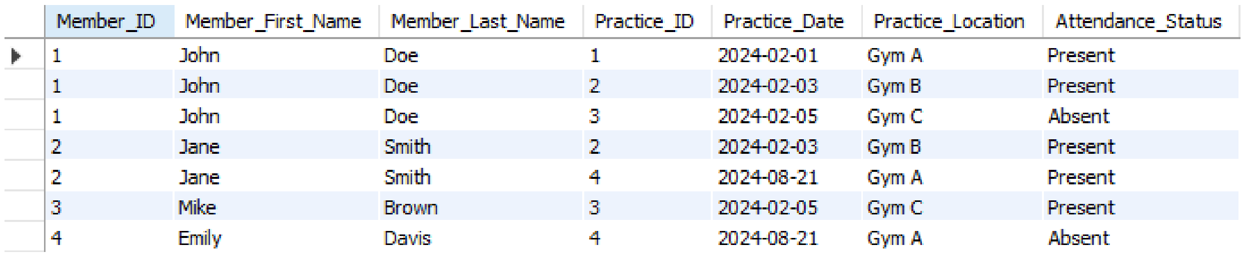
```
SELECT Member_ID, COUNT(Practice_ID) AS NumberOfPractices
FROM Attendance
WHERE Status = 'Present'
GROUP BY Member_ID;
```

**SQL Select Statements that Join Tables for a Master-Detail Report**

c. This query joins the Member, Attendance, and Practice tables to show each member's attendance records, including practice date and location. The results are ordered by member ID and practice date.

```
SELECT
    Member.Member_ID,
    Member.Member_First_Name,
    Member.Member_Last_Name,
    Practice.Practice_ID,
    Practice.Date AS Practice_Date,
    Practice.Location AS Practice_Location,
    Attendance.Status AS Attendance_Status
FROM
    Member
JOIN
```

```
    Attendance ON Member.Member_ID = Attendance.Member_ID
JOIN
    Practice ON Attendance.Practice_ID = Practice.Practice_ID
ORDER BY
    Member.Member_ID, Practice.Date;
```

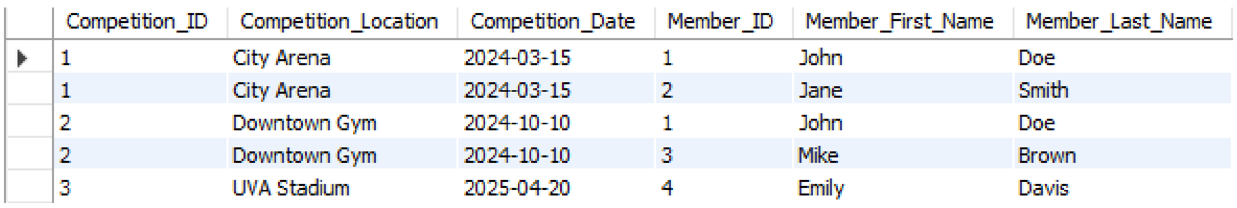


| Member_ID | Member_First_Name | Member_Last_Name | Practice_ID | Practice_Date | Practice_Location | Attendance_Status |
|---|---|---|---|---|---|---|
| 1 | John | Doe | 1 | 2024-02-01 | Gym A | Present |
| 1 | John | Doe | 2 | 2024-02-03 | Gym B | Present |
| 1 | John | Doe | 3 | 2024-02-05 | Gym C | Absent |
| 2 | Jane | Smith | 2 | 2024-02-03 | Gym B | Present |
| 2 | Jane | Smith | 4 | 2024-08-21 | Gym A | Present |
| 3 | Mike | Brown | 3 | 2024-02-05 | Gym C | Present |
| 4 | Emily | Davis | 4 | 2024-08-21 | Gym A | Absent |

d. This query joins the Competition, Competition_Member, and Member tables to show each competition and the members who participated. The results are ordered by competition date and ID.

```
SELECT
    Competition.Competition_ID,
    Competition.Location AS Competition_Location,
    Competition.Competition_Date,
    Member.Member_ID,
    Member.Member_First_Name,
    Member.Member_Last_Name
FROM
    Competition
JOIN
```

```
    Competition_Member ON Competition.Competition_ID =
Competition_Member.Competition_ID
JOIN
    Member ON Competition_Member.Member_ID = Member.Member_ID
ORDER BY
    Competition.Competition_Date, Competition.Competition_ID;
```

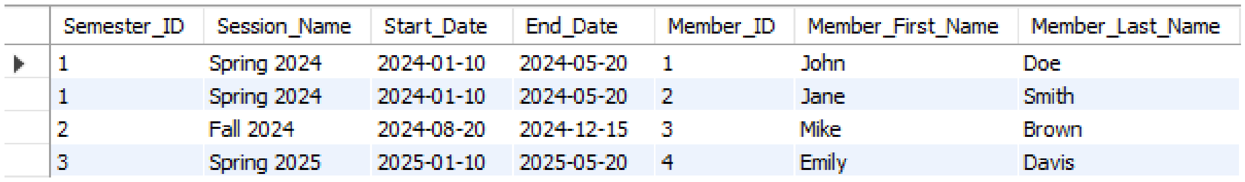


| Competition_ID | Competition_Location | Competition_Date | Member_ID | Member_First_Name | Member_Last_Name |
|---|---|---|---|---|---|
| 1 | City Arena | 2024-03-15 | 1 | John | Doe |
| 1 | City Arena | 2024-03-15 | 2 | Jane | Smith |
| 2 | Downtown Gym | 2024-10-10 | 1 | John | Doe |
| 2 | Downtown Gym | 2024-10-10 | 3 | Mike | Brown |
| 3 | UVA Stadium | 2025-04-20 | 4 | Emily | Davis |

e. This query joins the Semester and Member tables to show each semester and the members enrolled in it. The results are ordered by semester start date and semester ID.

```
SELECT
    Semester.Semester_ID,
    Semester.Session_Name,
    Semester.Start_Date,
    Semester.End_Date,
    Member.Member_ID,
    Member.Member_First_Name,
    Member.Member_Last_Name
FROM
    Semester
```

```
JOIN
    Member ON Semester.Semester_ID = Member.Semester_ID
ORDER BY
    Semester.Start_Date, Semester.Semester_ID;
```



| Semester_ID | Session_Name | Start_Date | End_Date | Member_ID | Member_First_Name | Member_Last_Name |
|---|---|---|---|---|---|---|
| 1 | Spring 2024 | 2024-01-10 | 2024-05-20 | 1 | John | Doe |
| 1 | Spring 2024 | 2024-01-10 | 2024-05-20 | 2 | Jane | Smith |
| 2 | Fall 2024 | 2024-08-20 | 2024-12-15 | 3 | Mike | Brown |
| 3 | Spring 2025 | 2025-01-10 | 2025-05-20 | 4 | Emily | Davis |

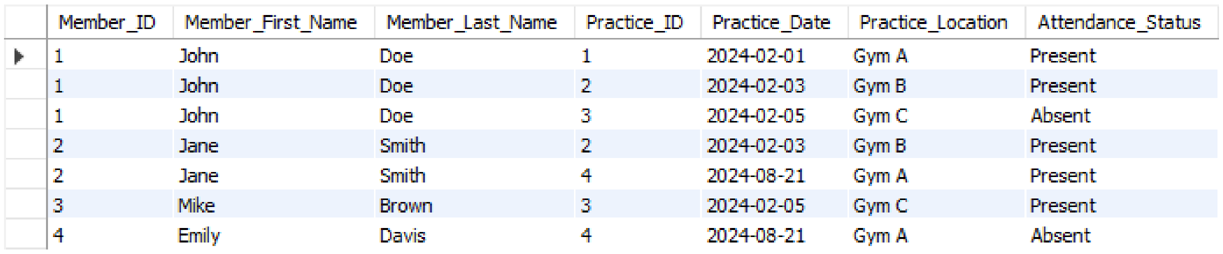
## SQL Select Statements For Associative Entities

A SQL select statement for each of your associative entities that shows the contents of your associative entities. These SQL statements must join the related tables and include some information from each table directly related to the associative entity.

a. This query joins the Attendance, Member, and Practice tables to show each attendance record along with the member's first and last name, practice date, and location, as well as the attendance status. The results are ordered by member ID and practice ID.

```
SELECT
    Attendance.Member_ID,
    Member.Member_First_Name,
    Member.Member_Last_Name,
    Practice.Practice_ID,
    Practice.Date AS Practice_Date,
    Practice.Location AS Practice_Location,
    Attendance.Status AS Attendance_Status
FROM
```

```
    Attendance
JOIN
    Member ON Attendance.Member_ID = Member.Member_ID
JOIN
    Practice ON Attendance.Practice_ID = Practice.Practice_ID
ORDER BY
    Attendance.Member_ID, Practice.Practice_ID;
```

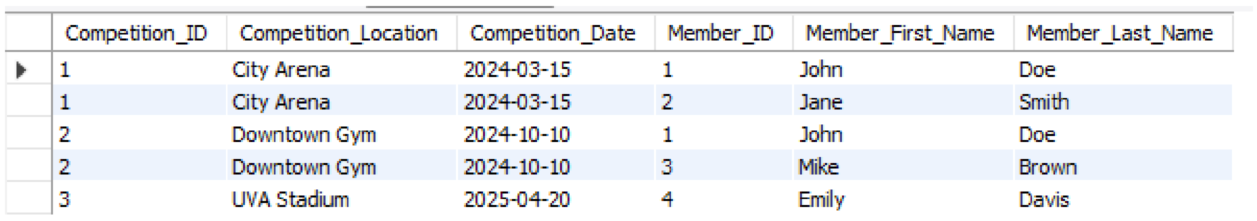


| Member_ID | Member_First_Name | Member_Last_Name | Practice_ID | Practice_Date | Practice_Location | Attendance_Status |
|---|---|---|---|---|---|---|
| 1 | John | Doe | 1 | 2024-02-01 | Gym A | Present |
| 1 | John | Doe | 2 | 2024-02-03 | Gym B | Present |
| 1 | John | Doe | 3 | 2024-02-05 | Gym C | Absent |
| 2 | Jane | Smith | 2 | 2024-02-03 | Gym B | Present |
| 2 | Jane | Smith | 4 | 2024-08-21 | Gym A | Present |
| 3 | Mike | Brown | 3 | 2024-02-05 | Gym C | Present |
| 4 | Emily | Davis | 4 | 2024-08-21 | Gym A | Absent |

b. This query joins the Competition_Member, Member, and Competition tables to show each competition participation record along with the member's first and last name, competition location, and competition date. The results are ordered by competition date and competition ID.

```
SELECT
    Competition_Member.Competition_ID,
    Competition.Location AS Competition_Location,
    Competition.Competition_Date,
    Member.Member_ID,
    Member.Member_First_Name,
    Member.Member_Last_Name
```

```
FROM
    Competition_Member
JOIN
    Member ON Competition_Member.Member_ID = Member.Member_ID
JOIN
    Competition ON Competition_Member.Competition_ID =
Competition.Competition_ID
ORDER BY
    Competition.Competition_Date, Competition.Competition_ID;
```



| Competition_ID | Competition_Location | Competition_Date | Member_ID | Member_First_Name | Member_Last_Name |
|---|---|---|---|---|---|
| 1 | City Arena | 2024-03-15 | 1 | John | Doe |
| 1 | City Arena | 2024-03-15 | 2 | Jane | Smith |
| 2 | Downtown Gym | 2024-10-10 | 1 | John | Doe |
| 2 | Downtown Gym | 2024-10-10 | 3 | Mike | Brown |
| 3 | UVA Stadium | 2025-04-20 | 4 | Emily | Davis |

**Updated SQL Queries**

**SELECT Member_ID, Member_First_Name, Member_Last_Name, Member_Gender, Member_Email FROM Member**

This select statement was used to extract only the member information that is displayed on the member search screen for table view.

**SELECT COUNT(*) AS TotalPracticesAttended FROM Attendance WHERE Member_ID = " + memberId + " AND Status = 'Present'**

This updated statement was used to calculate the number of practices attended based on the Attendance and Member tables.

**Final Implementation SQL Queries:**

1. Search for a Specific Member

**SELECT * FROM Member WHERE Member_First_Name LIKE '%{searchTerm}%' OR Member_Last_Name LIKE '%{searchTerm}%';**

This query searches for members whose first or last names contain the provided search term. The `LIKE` operator, combined with the `%` wildcards, allows for partial matching of the search term. The results include all columns for members that match the criteria.

2. Check Member Credentials

**SELECT * FROM Member WHERE Member_Email = '{email}' AND Member_Password_Hash = '{passwordHash}';**

This query checks if a member with the specified email and password hash exists in the `Member` table. If a matching record is found, the credentials are valid.

3. Get Member Data for Search

**SELECT * FROM Member;**

This query retrieves all records from the `Member` table. The result set includes various member attributes, which are used to populate a list of `Member` objects.

4. Get All Member Data

**SELECT * FROM Member;**

Similar to the previous query, this retrieves all records from the `Member` table. Additional member attributes, such as `Semester_ID`, `Member_Date_of_Birth`, and `Member_Grad_Date`, are also included in the result set.

5. Search Members with Multiple Criteria

**SELECT * FROM Member JOIN Semester ON Member.Semester_ID = Semester.Semester_ID WHERE 1=1**

**AND Member_First_Name LIKE '{firstName}' AND Member_Last_Name LIKE '{lastName}' AND Member_Email LIKE '{email}' AND Member_Gender = '{gender}' AND Member_Weight_Class = {weight} AND Member_Best_Total_KG = {result} AND Semester.Session_Name = '{semester}';**

This query searches for members based on various criteria, including first name, last name, email, gender, weight class, best total kg, and semester session name. The `JOIN` clause combines data from the `Member` and `Semester` tables.

6. Change Member Password:

**UPDATE Member SET Member_Password_Hash = '{newPassword}' WHERE Member_Email = '{email}';**

This query updates the password hash for a member identified by their email address. The new password hash replaces the old one in the `Member` table.

7. Add New Member

**INSERT INTO Member (Semester_ID, Member_First_Name, Member_Last_Name, Member_Date_of_Birth, Member_Grad_Date, Member_Weight_Class, Member_Best_Total_KG, Member_Gender, Member_Email, Member_Password_Hash) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);**

This query inserts a new member record into the `Member` table. It includes attributes such as semester ID, first name, last name, date of birth, graduation date, weight class, best total kg, gender, email, and password hash.

8. Search Member Events

**SELECT 'Practice' AS Event_Type, p.Date AS Event_Date, p.Location AS Event_Location FROM Attendance a JOIN Practice p ON a.Practice_ID = p.Practice_ID WHERE a.Member_ID = {memberId} UNION ALL SELECT 'Competition' AS Event_Type, c.Competition_Date AS Event_Date, c.Location AS Event_Location FROM Competition_Member cm JOIN Competition c ON cm.Competition_ID = c.Competition_ID WHERE cm.Member_ID = {memberId};**

This query retrieves events (both practices and competitions) that a member has participated in. It uses `UNION ALL` to combine results from two different joins: one for practices and one for competitions.

9. Fetch All Alumni Data

**SELECT * FROM Alumni;**

This query retrieves all records from the `Alumni` table, including attributes such as alumni ID, first name, last name, class year, email, and semester ID.

10. Search Alumni with Multiple Criteria

**SELECT * FROM Alumni WHERE 1=1 AND Alumni_First_Name LIKE '{firstName}' AND Alumni_Last_Name LIKE '{lastName}' AND Alumni_Email LIKE '{email}' AND Alumni_Class_Year = {classYear};**

This query searches for alumni based on various criteria, including first name, last name, email, and class year. The `LIKE` operator and exact matches are used as appropriate for each attribute.

11. Delete Member

**DELETE FROM Member WHERE Member_ID = ?;**

**DELETE FROM Competition_Member WHERE Member_ID = ?;**

**DELETE FROM Attendance WHERE Member_ID = ?;**

This query deletes a member record from the `Member` table, identified by the member ID, as well as according ‘Attendance’ and ‘Competition_Member’ records, ensuring data is consistently deleted.

12. Add Practice Event

**INSERT INTO Practice (Date, Location) VALUES (?, ?);**

**INSERT INTO Attendance (Member_ID, Practice_ID, Status) VALUES (?, ?, 'Present');**

The first query inserts a new practice event into the `Practice` table. The second query records the member's attendance at the practice event in the `Attendance` table, marking the status as 'Present'.

13. Add Competition Event

**INSERT INTO Competition (Location, Competition_Date) VALUES (?, ?);**

**INSERT INTO Competition_Member (Competition_ID, Member_ID) VALUES (?, ?);**

The first query inserts a new competition event into the `Competition` table. The second query associates a member with the competition in the `Competition_Member` table.

14. Convert Existing Member to Alumni

**SELECT * FROM Member WHERE Member_Grad_Date < DATE('now');**

**INSERT INTO Alumni (Alumni_First_Name, Alumni_Last_Name, Alumni_Class_Year, Alumni_Email, Semester_ID) VALUES (?, ?, ?, ?, ?);**

**DELETE FROM Member WHERE Member_ID = ?;**

**DELETE FROM Competition_Member WHERE Member_ID = ?;**

**DELETE FROM Attendance WHERE Member_ID = ?;**

This series of statements was used to convert members to alumni based on graduation date, effectively removing members from the member table and into the alumni table. Similar to deleting a member, other linked records like practice and competition joint table records are deleted as well.

## Member Log-ins:

**For John Doe:**

john.doe@example.com

john1

**For Jane Smith**:

jane.smith@example.com

jane123

**For Mike Brown:**

mike.brown@example.com

mike1

**For Emily:**

emily.davis@example.com

Emily1

## Executive Log-ins

**For Alice Smith:**

alice.smith@example.com

alice1

**For Robert Lee:**

robert.lee@example.com

robert1

**For Linda White:**

linda.white@example.com

linda1

## References

JavaFX. (n.d.). JavaFX. https://openjfx.io/

Lucid Software. (n.d.). Lucidchart. https://www.lucidchart.com/

MySQL. (n.d.). MySQL. https://www.mysql.com/

Python Software Foundation. (n.d.). Python. https://www.python.org/