COMP3005 - V2 Project Report

Health and Fitness Club Management System

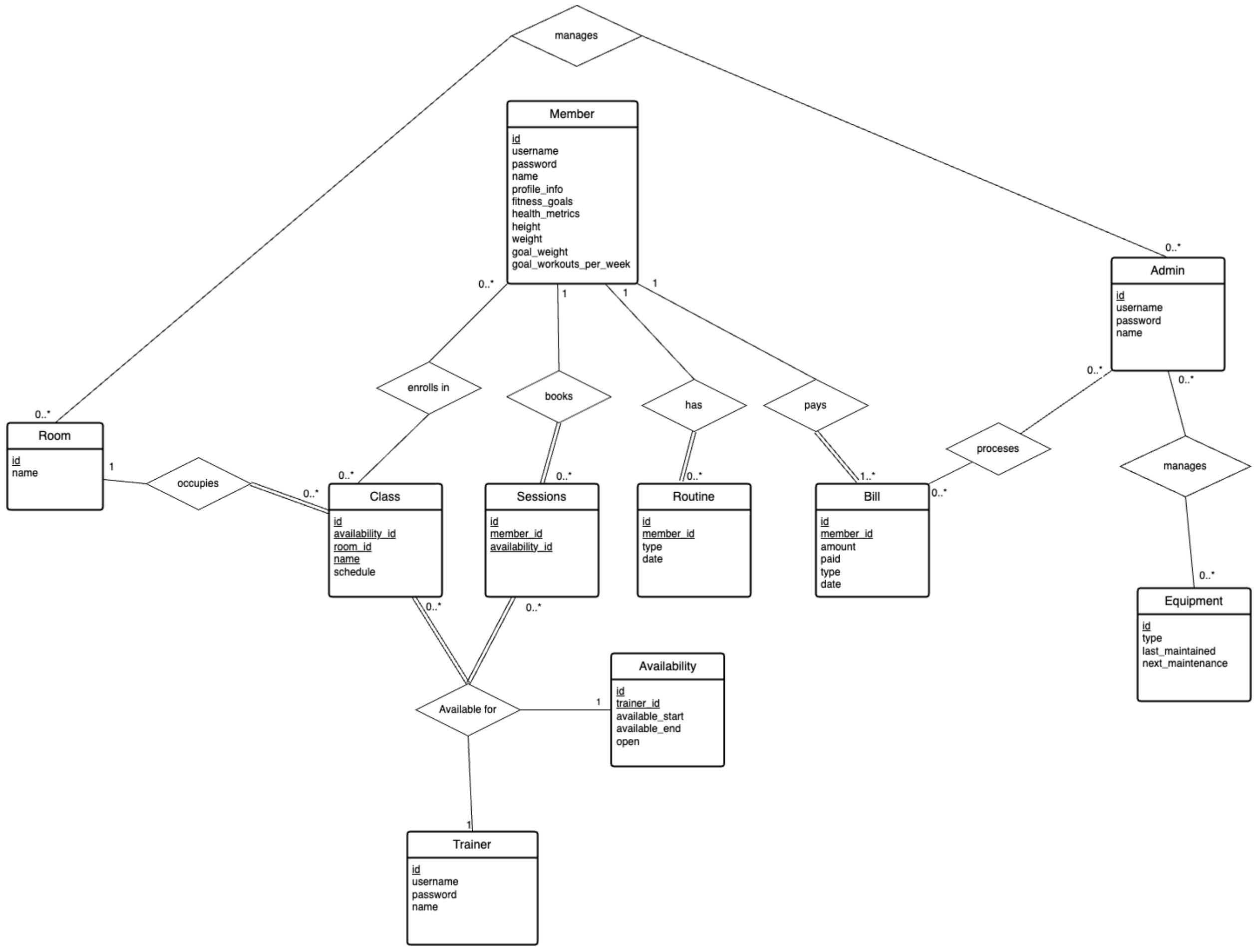
Nathan El-Khoury
101 265 875

Links and Information

Github Repository: https://github.com/elkhourynathan/comp3005_project_v2
Video Link: https://youtu.be/HEGNmxlj7iQ

SQL DDL and DML files are found within the repository.

Conceptual Design



Overview

The conceptual design of the database for the Health and FItness Club is illustrated through the ER-diagram above. The diagram showcase the interrelations among the various entities needed to accomplish the club’s operations

Entities

1. **Member**: The core member entity represents the clients of the Health and Fitness Club, this entity contains personal information, fitness goals, health goals and health metrics.
2. **Trainer**: The trainer entity represents the certified trainers which provide personal training sessions and lead group fitness classes
3. **Admin**: The admin entity defines the administrator who manages backend club operations such as overseeing billing, processing payments, and managing/monitoring fitness equipment and room bookings.
4. **Availability**: Slots indicating times a trainer is available for sessions or classes.
5. **Classes**: Represent the group fitness class scheduling and are linked to rooms where they take place and availability slots for when they are run.
6. **Room**: Represents the physical locations within the club and where classes occur.
7. **Sessions**: A scheduling entity to record training sessions between a Member entity and Trainer entity.
8. **Routine**: Allows members to independently track and save routines they've completed or aim to complete.
9. **Bill**: Represent financial transactions, recording dues and payments of members for classes,sessions or membership fees.
10. **Equipment**: Represents the equipment available at the club along with maintenance schedules.

Relationships

- Members can enroll in classes, book sessions, have routines, and pay bills.
- Classes utilize a room and align with trainer availability.
- Sessions are reserved through available time slots.
- Trainers have multiple availabilities and can conduct various sessions and classes.
- Admins oversee bill transactions, equipment upkeep, and room scheduling.

Cardinalities and Participations

- A Member can enroll in zero or many classes and a class can contain zero or many members, indicating a many-to-many relationship.
- A Member can book many sessions but a session has exactly 1 member.
- A Member can create many routines but a routine has exactly 1 member.
- A Member will be attached to 1 or more bills (Members get a “membership” bill when they register), but a bill will be attached to exactly 1 member.
- A class occupies 1 room, but a room may be attached to zero or many rooms (Given they do not have conflicting times)
- A Trainer can teach zero or many Classes, but a class has 1 teacher.
- A Trainer can have zero or many Sessions, but a session has 1 trainer.

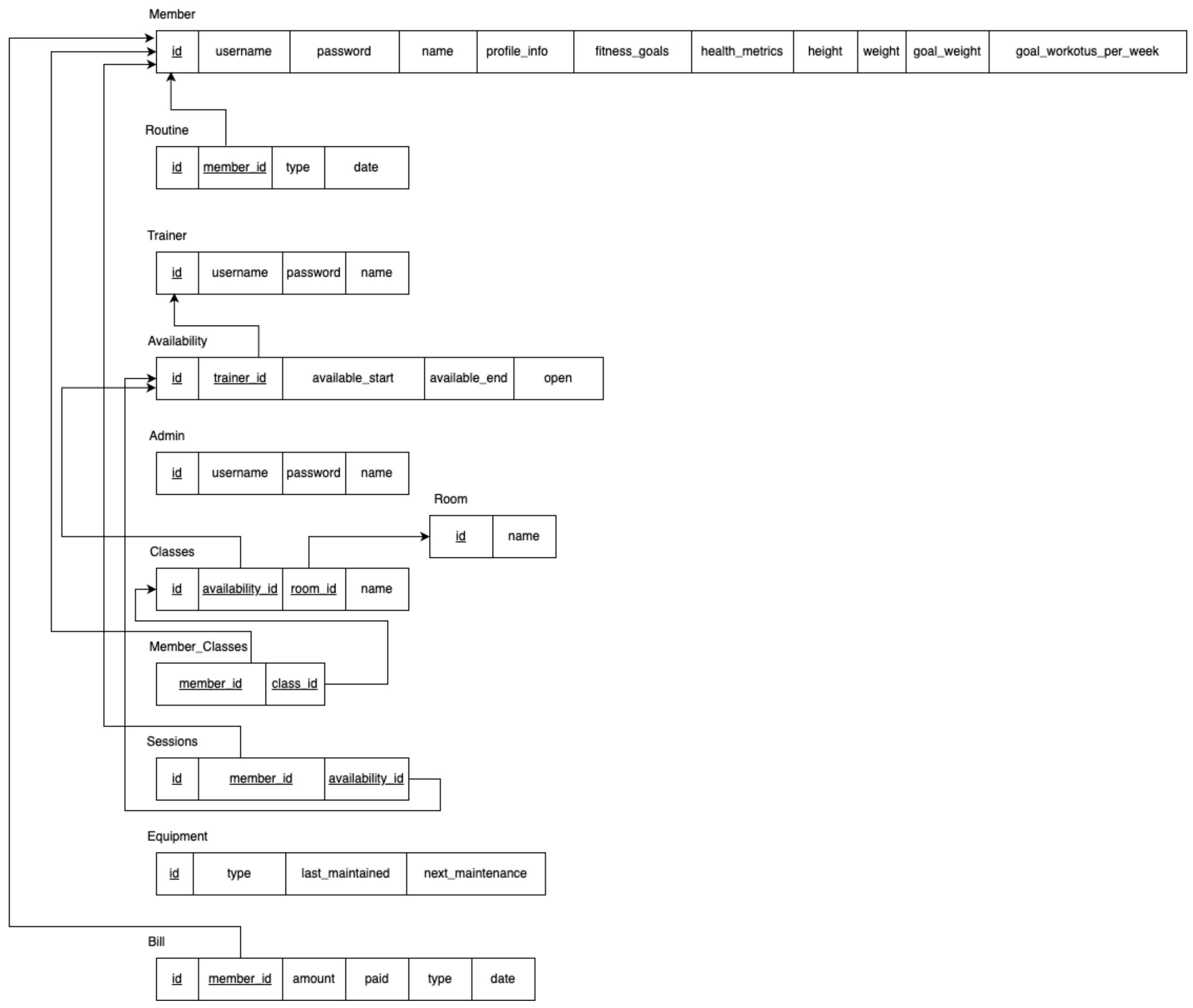
- An Admin can manage zero or many equipment, and equipment can be managed by zero or many admins.
- An Admin can manage zero or many rooms, and a room can be managed by zero or many admins.
- An Admin can process zero or many bills, and a Bill can be processed by zero or many admins.

Assumptions

1. Each member can enroll in multiple classes and a class can have many enrolled members. Illustrated how it was resolved in the next section via a Member_Classes associative entity.
2. Members can independently create, update and remove their routines.
3. Members receive a membership bill after registering, if registered in person then an Admin will have to manually assign said bill. If a bill is then paid for in person, an Admin will have to manually process it as paid.
4. Members pay for classes and sessions when registering, creating a new bill. If a member decides to cancel the session or class they will be refunded creating a new bill with equal negative balance.
5. Trainers can have many non-overlapping Availabilities. Each availability slot will be marked with open equal to true or assigned to a single session or class.
    a. The open attribute signifies whether the slot is currently booked/previously booked ensuring no double bookings and displays only available bookings to members.
6. The availability trainers create will be taken to be used in classes, if an availability is not there and an Admin is assigning a trainer to a class, an availability is added to accommodate as long as the class schedule does not conflict with any of the trainers prior obligations. The assumption here is the Admin is the trainers boss and they are assigning them work to be done.
7. Availabilities are created with 1 hour intervals ie from 7 AM to 8 AM, 8 AM to 9AM…
8. The bill processing, equipment management and room management by admins is considered atomic; the entities can be handled independently by any admin. An admin is not linked to any of the entities.
9. If an Admin creates a new piece of equipment it is assumed it was last maintained the time it was added to the system.

These assumptions were created to align with the problem statement in section 1.

Reduction to Relational Schemas



Entities to Tables

1. Member
    a. Converted into a Member table
    b. Attributes: id (PK), username, password, name, profile_info, fitness_goals, health_metrics, height, weight, goal_weight, goal_workouts_per_week
2. Trainer
    a. Converted into a Trainer table
    b. Attributes: id (PK), username, password, name
3. Admin
    a. Converted into a Admin table
    b. Attributes: id (PK), username, password, name
4. Routine

    a. Attributes: id (PK), member_id (FK) references Member table, type, date
5. Availability
    a. Attributes: id (PK), trainer_id (FK) references Trainer table, available_start, available_end, open
6. Room
    a. Attributes: id (PK), name
7. Classes
    a. Attributes: id (PK), availability_id (FK) references Availability table, room_id (FK) reference Room table, name
8. Member to Classes relationship : Member_Classes
    a. Created to facilitate the many-to-many relationship
    b. Attributes: (member_id, class_id) (PK)
9. Sessions
    a. Attributes: id (PK), member_id (FK) reference the Member table, availability_id (FK) references the Availability table
10. Equipment:
    a. Attributes: id (PK), type, last_maintained, next_maintenance
11. Bill:
    a. Attributes: id (PK), member_id (FK) references the Member table, amount, paid, type, date

DDL File

```
-- Create the Member table
DROP TABLE IF EXISTS Bill, Sessions, Classes, Admin,Member_Classes, Equipment,
Availability, Trainer, Routine, Member, Room CASCADE;
CREATE TABLE Member (
   id SERIAL PRIMARY KEY,
   username VARCHAR(255) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL,
   name VARCHAR(255) NOT NULL,
   profile_info TEXT,
   fitness_goals TEXT,
   health_metrics TEXT,
   height FLOAT,
   weight FLOAT,
   goal_weight FLOAT,
   goal_workouts_per_week INT
);

-- Create the Routine table
CREATE TABLE Routine (
```

```sql
   id SERIAL PRIMARY KEY,
   member_id INT REFERENCES Member(id),
   type VARCHAR(255) NOT NULL,
   date DATE NOT NULL
);

-- Create the Trainer table
CREATE TABLE Trainer (
   id SERIAL PRIMARY KEY,
   username VARCHAR(255) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL,
   name VARCHAR(255) NOT NULL
);

-- Create the Availability table
CREATE TABLE Availability (
   id SERIAL PRIMARY KEY,
   trainer_id INT REFERENCES Trainer(id),
   available_start TIMESTAMP NOT NULL,
   available_end TIMESTAMP NOT NULL,
   open BOOLEAN DEFAULT TRUE
);

-- Create the Admin table
CREATE TABLE Admin (
   id SERIAL PRIMARY KEY,
   username VARCHAR(255) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL,
   name VARCHAR(255) NOT NULL
);

-- Create the Room table
CREATE TABLE Room (
   id SERIAL PRIMARY KEY,
   name VARCHAR(255)
);

-- Create the Class table
CREATE TABLE Classes (
   id SERIAL PRIMARY KEY,
   availability_id INT REFERENCES Availability(id),
   room_id INT REFERENCES Room(id),
   name VARCHAR(255)
);


-- Create the Member_Classes table
CREATE TABLE Member_Classes (
   member_id INT REFERENCES Member(id),
   class_id INT REFERENCES Classes(id),
   PRIMARY KEY (member_id, class_id)
);

-- Create the Sessions table
CREATE TABLE Sessions (
   id SERIAL PRIMARY KEY,
   member_id INT REFERENCES Member(id),
   availability_id INT REFERENCES Availability(id)
);

-- Create the Equipment table
CREATE TABLE Equipment (
   id SERIAL PRIMARY KEY,
   type VARCHAR(255),
   last_maintained TIMESTAMP,
   next_maintenance TIMESTAMP
);

-- Create the Bill table
CREATE TABLE Bill (
   id SERIAL PRIMARY KEY,
   member_id INT REFERENCES Member(id),
   amount FLOAT,
   paid BOOLEAN,
   type VARCHAR(255),
   date TIMESTAMP
);


-- Indexes for member id related tables
CREATE INDEX idx_member_name on Member(name);
CREATE INDEX idx_availability_trainer_id on Availability(trainer_id);
CREATE INDEX idx_member_classes_id on Member_Classes(member_id);
```

The raw DDL.sql file can be found in the /SQL directory. Below is information regarding the file including assumptions and constraints. Please use/copy from the raw DML.sql file.

Assumptions:

1. Members, Admins, Trainers usernames are unique identifiers within the database.

Bonuses:

1. Indexes were created on the name attribute in Member, trainer_id attribute on Availability and member_id attribute on Member_Classes. Reasoning is found in the Bonuses section.

DML File

```sql
-- Inserting Members
INSERT INTO Member (username, password, name, profile_info, fitness_goals,
health_metrics, height, weight, goal_weight, goal_workouts_per_week) VALUES
('test', 'test', 'Test User', 'Test Profile', 'Test Goals', 'Test Metrics', 180, 85,
75, 3),
('johnsmith', 'password', 'John Smith', 'I love gym', 'Lose weight, Build muscle', 'No
known issues', 180, 85, 75, 3),
('janedoe', 'password', 'Jane Doe', 'Me love run', 'Increase stamina, Run a marathon',
'Asthma', 165, 60, 58, 5);

-- Inserting Trainers
INSERT INTO Trainer (username, password, name) VALUES
('trainer_tom', 'tompas', 'Tom'),
('trainer_tina', 'tompass', 'Tina');

-- Inserting Admins
INSERT INTO Admin (username, password, name) VALUES
('admin_alice', 'alicepass', 'Alice'),
('admin_bob', 'bobspass', 'Bob');



-- Insert Availability for tom and tina
INSERT INTO Availability (trainer_id, available_start, available_end, open) VALUES
(1, '2024-04-01 09:00:00', '2024-04-01 10:00:00', False),
(1, '2024-04-02 14:00:00', '2024-04-02 15:00:00', False),
(2, '2024-04-01 08:00:00', '2024-04-01 09:00:00', True),
(1, '2024-04-01 06:00:00', '2024-04-01 07:00:00', False);
```

```sql
-- Inserting Sessions
INSERT INTO Sessions (member_id, availability_id) VALUES
(1,4);

INSERT INTO Room (name) VALUES
('Room A'),
('Room B');

-- Inserting Classes
INSERT INTO Classes (availability_id, room_id, name) VALUES
(1, 1,'Yoga Basics'),
(2, 2,'Cycling Class');

-- Insert a Member_Classes
INSERT INTO Member_Classes (member_id, class_id) VALUES
(1, 1);

-- Inserting Equipment
INSERT INTO Equipment (type, last_maintained, next_maintenance) VALUES
('Treadmill', '2024-05-01', '2024-06-01'),
('Rowing Machine', '2024-05-15', '2024-06-15'),
('Dumbbells', '2024-06-01', '2024-07-01'),
('Barbell', '2024-06-01', '2024-07-01'),
('Kettlebell', '2024-06-01', '2024-07-01'),
('Bench Press', '2024-06-01', '2024-07-01'),
('Squat Rack', '2024-06-01', '2024-07-01');


-- Inserting Routines
INSERT INTO Routine (member_id, type, date) VALUES
(1, 'Cardio', '2024-03-31'),
(1, 'Strength', '2024-04-01'),
(2, 'Cardio', '2024-04-01'),
(2, 'Strength', '2024-04-02'),
(3, 'Cardio', '2024-04-01'),
(3, 'Strength', '2024-04-02');

-- Some unprocessed bills
INSERT INTO Bill (member_id, amount, type, paid, date) VALUES
(1, 100, 'Membership', False, '2024-04-01'),
(2, 100, 'Membership', False, '2024-04-01'),
(3, 100, 'Membership', False, '2024-04-01');
```

The DML.sql file can be found in the /SQL directory. The DML file includes sample data for each table. Please use/copy from the raw DML.sql file.

Implementation

For this project, a web application was the choice of implementation, leveraging Flask and Jinja2 as the core technologies. A PostgreSQL relational database was set up using DDL and DML scripts located in the `/SQL` directory, enabling structured data storage and manipulation. The Flask application facilitates a wide range of CRUD operations, effectively addressing the problem statement outlined in Section 1.

Bonus Features

Web Application

The project was developed as a web application, utilizing Flask and Jinja2 for backend and templating functionalities, respectively. For detailed information on the code structure, installation of necessary requirements, and instructions for running the application, please refer to the README file within the project repository.

Indexes

Indexes were used to enhance performance:

- **Member Name Index**: An index on the `name` attribute in the `Member` table was created to speed up queries, allowing for efficient retrieval of member information by name. Relevant to improving the trainer's member search functionality.
- **Trainer ID Index**: In the `Availability` table, an index on the `trainer_id` attribute was established to quickly identify trainer availability, increasing the query speed for identifying colliding schedules.
- **Member ID Index**: The `member_id` attribute in the `Member_Classes` table was indexed to support optimized querying in scenarios involving multiple members enrolled in various classes.