COMP 3005 B — Health & Fitness Project

Zarif Khan 101224172

Arun Karki 101219923

April 12, 2024

2.1 Conceptual Design

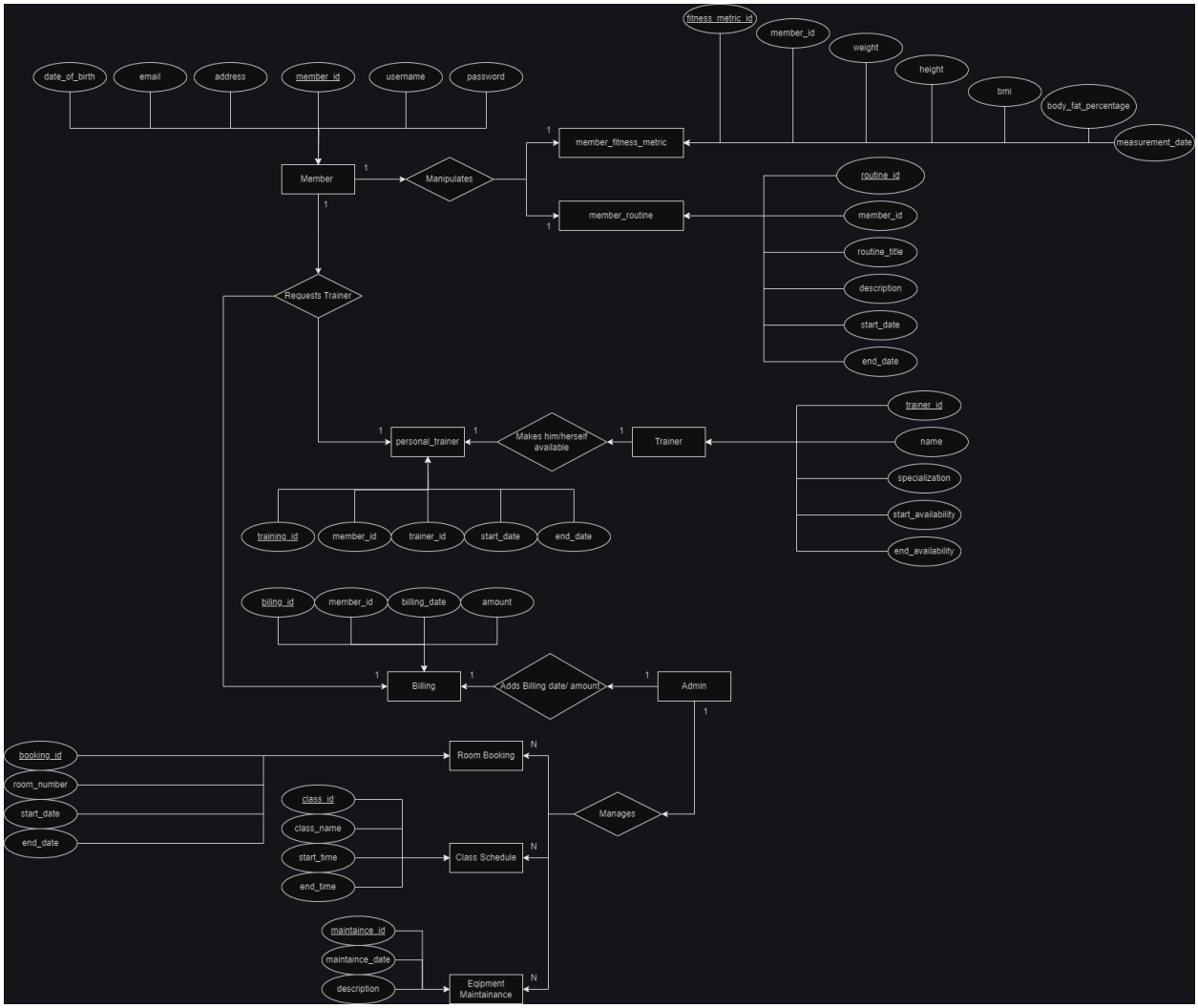
The Health and Fitness Club Management System is an all-in-one app made for club members, trainers, and the admin team. It lets members handle their accounts, set and follow fitness goals, and book time with trainers. Trainers can plan their workdays and check out member info. The admin team takes care of room reservations, keeps an eye on equipment, updates class times, and handles payments. The system's database design is shown in a diagram that maps out all the parts and how they connect. This report will go over each part, how they link up, and the rules about how they work together. Understanding the following details will show how the system works and matches the issues mentioned in Section 1:

1. **Entities**
    - *Member***:** Represents club members, with attributes like member ID, name, address, etc.
    - *Trainer*: Represents a person employed by the gym to provide fitness instruction and guidance to members
    - *Class Schedule:* Represents the planned times and details of fitness classes offered by the gym
    - *Billing:* Represents financial transactions related to gym memberships and services
    - *Maintenance:* Represents tasks or services required to keep gym equipment and facilities operational
    - *Member Fitness Metric:* Represents a specific measurement or data point related to a member's fitness progress
    - *Routine:* Represents a personalized exercise program designed for a member by a trainer
2. **Relationships**
    - *Member - Manipulates - Member Fitness Metric:* This relationship One-to-Many represents a member recording their own fitness metrics . A member can manipulate many fitness metrics, and a fitness metric belongs to one member
    - *Member - Requests Trainer - Personal Training:* This Many-to-Many relationship represents a member requesting a personal training session with a trainer. A member can request many personal training sessions, and a personal training session can be requested by one member.
    - *Trainer - Makes Himself/Herself Available - Personal Training:* This Many-to-Many relationship represents a trainer indicating their availability for personal training sessions. A trainer can make themselves available for many sessions, and a personal training session can be offered by one trainer.

- *Member - Enrolls In - Class Schedule:* This relationship Many-to-Many represents a member registering for a fitness class. A member can enroll in many classes, and a class can have many members enrolled.
- *Admin - Manages - Room Booking:* This relationship Many-to-Many represents the admin staff managing room bookings for various purposes (e.g., classes, personal training). An admin can manage many bookings, and a booking is managed by one admin.
- *Admin - Oversees - Billing:* This relationship Many-to-Many represents the admin staff overseeing the billing process for membership fees, personal training, and other services. An admin can oversee many billings, and a billing is overseen by one admin.
- *Admin - Maintains - Maintenance:* This Many-to-Many relationship represents the admin staff maintaining gym equipment and facilities. An admin can maintain many maintenance logs, and a maintenance log is maintained by one admin.

3. **Assumptions**
   - *Member Fitness Metric:* This entity assumes a one-to-many relationship with Member. This implies a member can have many fitness metrics recorded, but a metric belongs to only one member
   - *Personal Training:* The many-to-many relationship between Member and Trainer through Personal Training assumes separate request and availability entities might exist. This avoids directly assigning a trainer to a member without considering scheduling
   - *Room Booking and Maintenance:* Both these entities have many-to-many relationships with Admin. This assumes an admin can manage multiple bookings/maintenance tasks, and vice versa.

The system matches the issues we talked about in Section 1. It's built for the different things club members, trainers, and the admin team need to do. Members can sign up, control their accounts, set fitness targets, and put in health details. They can also book, change, or cancel sessions with trainers. Trainers keep track of their workdays and look at member info. The admin team handles room bookings, checks on equipment, updates class times, looks after billing, and takes care of payments for memberships, training, and other services. The diagram we have shows all these jobs and how they're connected.


fitness_metric_id
member_id
weight
height
bmi
body_fat_percentage
measurement_date
date_of_birth
email
address
member_id
username
password
member_fitness_metric
Member
Manipulates
member_routine
routine_id
member_id
routine_title
description
start_date
end_date
Requests Trainer
trainer_id
name
specialization
start_availability
end_availability
personal_trainer
Makes him/herself available
Trainer
training_id
member_id
trainer_id
start_date
end_date
biling_id
member_id
billing_date
amount
Billing
Adds Billing date/ amount
Admin
booking_id
room_number
start_date
end_date
Room Booking
class_id
class_name
start_time
end_time
Class Schedule
Manages
maintaince_id
maintaince_date
description
Eqipment Maintainance

2.2 Reduction to Relational Schemas

Turning the ER diagram into relational schemas is essential for database creation. This involves converting diagram elements into tables, columns, and keys. In the Health and Fitness Club Management System, this step has been completed successfully. The ER diagram's components have become tables, their details are now columns, and the connections are shown using keys. The next section will detail this process, illustrating how the ER diagram was transformed into relational schemas that meet the system's requirements:

1. Entities to Tables: Each entity in the ER diagram is converted into a separate table. For example, the entities *Member*, *Billing*, *Personal_Trainer*, *Trainer*, *Class_Schedule*, *Equipment_Maintenance*, *Room_Booking*, and *Member_Fitness_Metric* are all converted into separate tables.
2. Attributes to Columns: The attributes of each entity become the columns of the corresponding table. For instance, the *Member* entity's attributes *member_id*, *username*, *password*, *email*, *date_of_birth*, and *address* become columns in the *Member* table.
3. Primary Keys: Each entity's primary key is designated as the primary key in the corresponding table. For example, *member_id* is the primary key for the *Member* table.
4. Relationships to Foreign Keys: Relationships between entities are represented using foreign keys. For example, the *member_id* in the *Billing* table is a foreign key referencing the *member_id* in the *Member* table, representing the relationship that a *Member* enrolls in a *Billing*.
5. Cardinality: The cardinality of the relationships in the ER diagram is preserved in the relational schema. For example, a *Member* can have multiple *Billing* records, which is represented by having multiple *Billing* records with the same *member_id*.

2.3 DDL File

```
-- Member table
CREATE TABLE "member" (
    member_id SERIAL PRIMARY KEY NOT NULL,
    username VARCHAR(255),
    "password" VARCHAR(255),
    email VARCHAR(255) UNIQUE,
    date_of_birth DATE,
    "address" VARCHAR(255)
);

-- Trainer table
CREATE TABLE Trainer (
    trainer_id SERIAL PRIMARY KEY NOT NULL,
    "name" VARCHAR(255),
    specialization VARCHAR(255),
    start_availability DATE,
    end_availability DATE
);

-- Personal Trainer table
CREATE TABLE Personal_Training (
    training_id SERIAL PRIMARY KEY NOT NULL,
    member_id INT REFERENCES "member"(member_id),
    trainer_id INT REFERENCES Trainer(trainer_id),
    "start_date" DATE,
    end_date DATE
);

-- Billing table
CREATE TABLE Billing (
    billing_id SERIAL PRIMARY KEY NOT NULL,
    member_id INT REFERENCES "member"(member_id),
    billing_date DATE,
    amount FLOAT
);

-- Class Schedule table
```

```sql
CREATE TABLE Class_Schedule (
    class_id SERIAL PRIMARY KEY NOT NULL,
    class_name VARCHAR(255),
    start_time TIME,
    end_time TIME
);

-- Room Booking table
CREATE TABLE Room_Booking (
    booking_no SERIAL PRIMARY KEY NOT NULL,
    room_number INT,
    "start_date" DATE,
    end_date DATE
);

-- Equipment Maintenance table
CREATE TABLE Equipment_Maintenance (
    maintenance_id SERIAL PRIMARY KEY NOT NULL,
    maintenance_date DATE,
    "description" VARCHAR(255)
);

-- Member Fitness Metric table
CREATE TABLE Member_Fitness_Metric (
    fitness_metric_id SERIAL PRIMARY KEY NOT NULL,
    member_id INT REFERENCES "member"(member_id),
    "weight" FLOAT,
    height FLOAT,
    bmi FLOAT,
    body_fat_percentage FLOAT,
    measurement_date DATE
);
```

2.4 DML File

```
-- Sample data for Member table
INSERT INTO "member" (username, "password", email, date_of_birth,
"address")
VALUES
    ('arun_karki', 'password123', 'karki@example.com', '2003-02-04',
'123 Main St'),
    ('zarif_khan', 'abc123', 'khan@example.com', '2003-02-23', '456
Elm St');

-- Sample data for Trainer table
INSERT INTO Trainer ("name", specialization, start_availability,
end_availability)
VALUES
    ('Mike Johnson', 'Weightlifting', '2024-01-01', '2024-12-31'),
    ('Emily White', 'Yoga', '2024-01-01', '2024-12-31');

-- Sample data for Personal_Training table
INSERT INTO Personal_Training (member_id, trainer_id, "start_date",
end_date)
VALUES
    (1, 1, '2024-04-01', '2024-04-30'),
    (2, 2, '2024-04-05', '2024-04-25');

-- Sample data for Billing table
INSERT INTO Billing (member_id, billing_date, amount)
VALUES
    (1, '2024-04-10', 50.00),
    (2, '2024-04-15', 75.00);

-- Sample data for Class_Schedule table
INSERT INTO Class_Schedule (class_name, start_time, end_time)
VALUES
    ('Yoga Class', '10:00:00', '11:00:00'),
    ('Weightlifting Class', '14:00:00', '15:00:00');

-- Sample data for Room_Booking table
```

```
INSERT INTO Room_Booking (room_number, "start_date", end_date)
VALUES
    (101, '2024-04-10', '2024-04-12'),
    (102, '2024-04-15', '2024-04-17');

-- Sample data for Equipment_Maintenance table
INSERT INTO Equipment_Maintenance (maintenance_date, "description")
VALUES
    ('2024-04-10', 'Treadmill maintenance'),
    ('2024-04-15', 'Elliptical machine repair');

-- Sample data for Member_Fitness_Metric table
INSERT INTO Member_Fitness_Metric (member_id, "weight", height, bmi,
body_fat_percentage, measurement_date)
VALUES
    (1, 180.5, 72, 24.5, 18.5, '2024-04-10'),
    (2, 150.0, 65, 22.3, 22.0, '2024-04-15');
```

2.5 Implementation

This software design uses common patterns and rules.

It has a special part called JDBCconnect that follows the Singleton Pattern, meaning it's made to have only one instance that sets up the database connection. This helps avoid mistakes and problems since there's just one connection. The JDBCconnect also works like a Factory Pattern, creating a user interface class based on the connection. This lets the system make new objects as needed for different situations.

The design is similar to the Model-View-Controller (MVC) setup. The JDBCconnect is the Model, handling data. The user interface classes are the View, showing data to users. The Member, Trainer, and Admin roles are the Controller, guiding user actions and connecting the Model and View.

The design also uses Dependency Injection, giving the JDBC connection to the Member, Trainer, and Admin from outside. This makes the design flexible, easier to test, and ready for changes. Overall, the design focuses on making the system modular, adaptable, and easy to maintain.

2.7 GitHub Repository

Please refer to the **GitHub Repository**