2.1

Member:

Members can register/manage profiles, set personal fitness goals, and input health metrics. They can also access exercise routines, fitness achievements, and health statistics. They can also schedule, reschedule, and cancel personal training sessions with certified trainers. Members can also register for group fitness classes, workshops, and other events. They also can track there loyalty points which increases with every purchase.

Trainer

Trainers are selected/reserved by members through the app, which provides their name and specialization. Their schedule will be accessed by the member in order for them to book. They will also provide progress notes after each training session.

Extra classes:

Members are also able to register for distinct group fitness classes, workshops, and other events which are updated weekly.

Administrative Staff:

Staff can manage room bookings, monitor equipment maintenance, and update class schedules.

Payment:

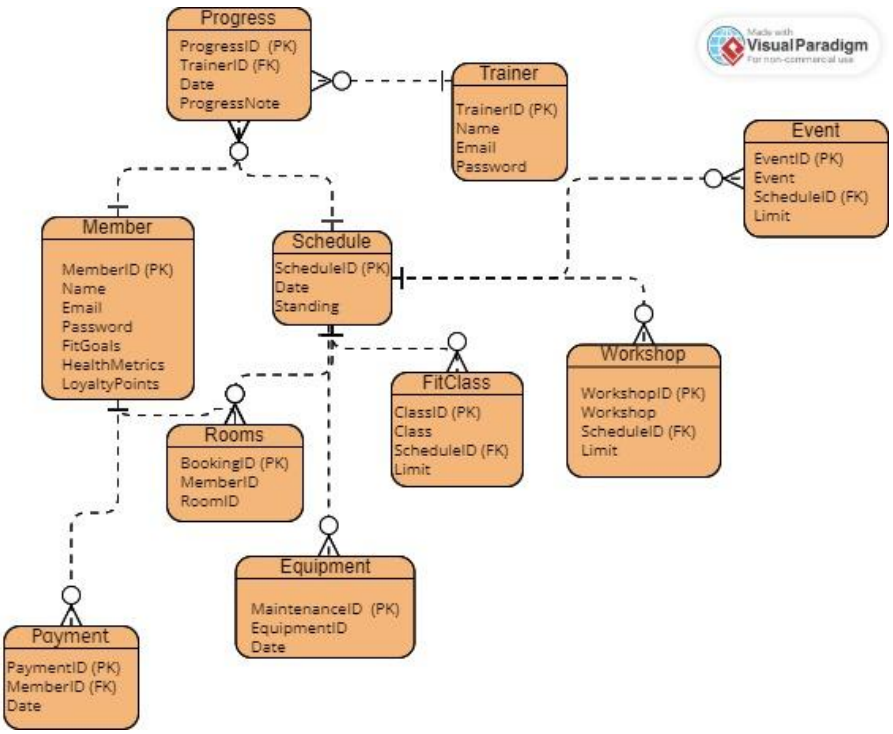
After each purchase by the members, loyalty points will increase

Conceptual design:

The design that I decided to go with was a very simplistic outline that focused on a very user friendly experience. I wanted it to be simplistic because if this were a business model, I would prefer very simple designs that is easy to understand. With that being said, I had many one to many relationships, which allowed for more simplicity, and less confusion, despite having a couple many to many relationships. The administrative staff

also has direct access to all the information and can reliably access any of it with leisure. With that being said, the simplistic design allows for more room for growth, creativity, and bonus structure.

ER Diagram:



Assumptions regarding cardinalities/participation types:

Due to there being so many, I highlighted the rarer, most important ones that relate to the problem statement.

Many to many relationships:

Members -> Fitclass

- One member can have multiple classes. One Fitclass can have multiple members.

Members -> Event

- One member can go to multiple events. One event can have multiple members.

Members -> Workshop

- One member can have multiple workshops. One workshop can have multiple members.

Trainer -> Fitclass

- One trainer can have multiple classes. One Fitclass can have multiple trainers.

Trainer -> Event

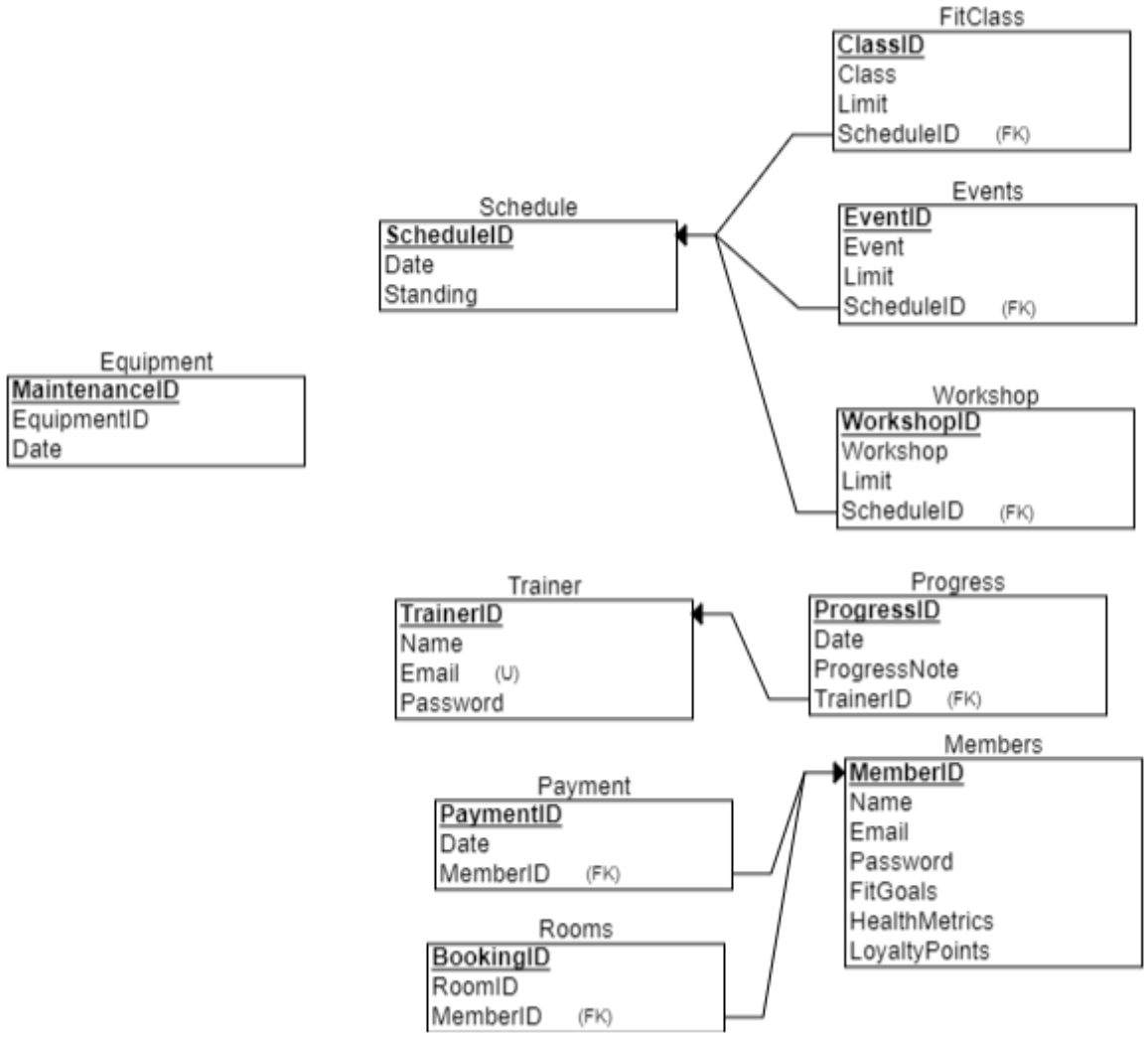
- One trainer can have multiple events. One event can have multiple trainers.

Trainer -> Workshop

- One trainer can have multiple workshops. One workshop can have multiple trainers.

2.2

Relational Schema Diagram:



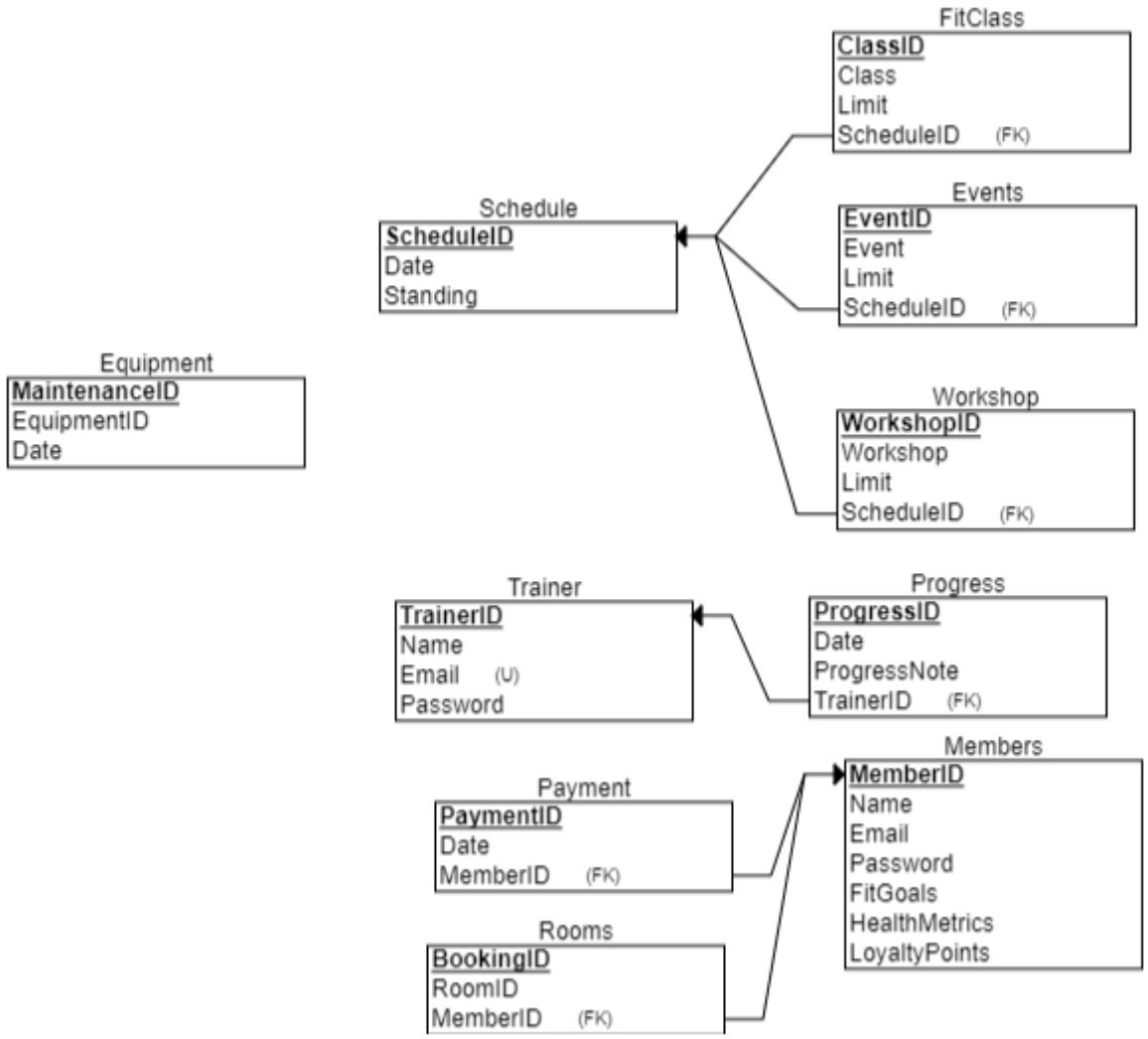
2.3

As shown, my current relational database schema is in both 2NF and 3NF. Reason being, is that there are no partial dependencies and every attribute that is non prime - meaning that it is not apart of the primary key

The relational schema is also in 3NF since there are no transitive dependencies. An example of this would be the Progress entity

where Date, ProgressNote and TrainerID are all dependent on ProgressID. With that being said each table has a primary key and every attribute is non prime and dependent on the primary key making this both 2NF and 3NF.

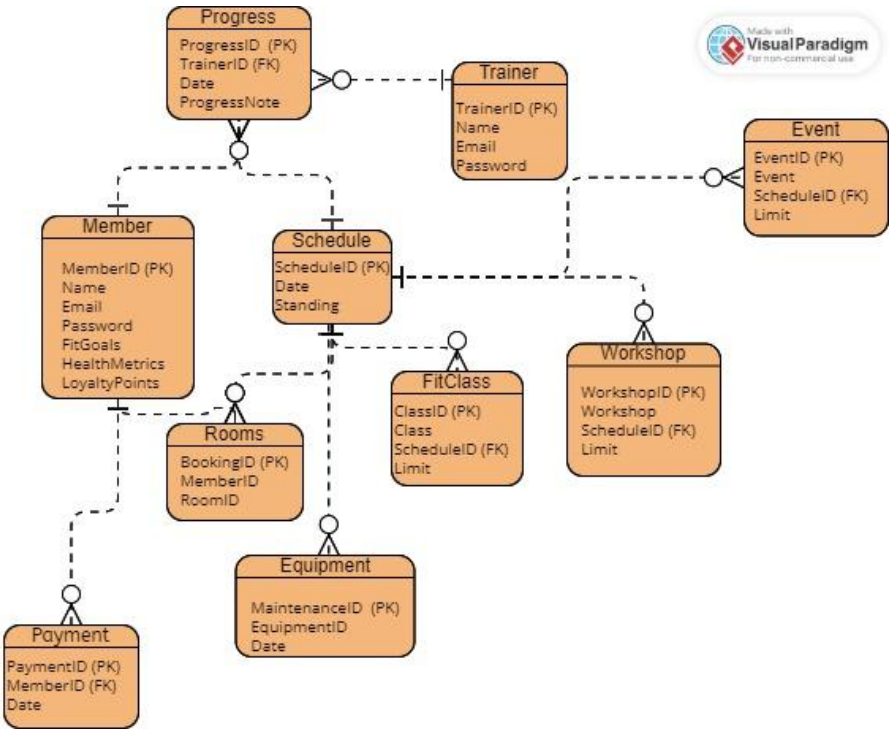
2.4



2.5

My applications architecture is designed to be very user friendly. With that in mind I kept the design fairly simple meaning that every

is easily accessible, and not questionable. I made it so that a ten year old would be able to follow along. This is important because without a great application, everything falls apart. With that being said, most of the entities are one to many with few many to many connections. This allows for more flow, direct control, and easier usage. All in all, the would boost sales, and encourage users to interact more within the actual application.



2.6:

N/A - Not enough time unfortunately.

Github Link:

https://github.com/MAHCORP/3005.git