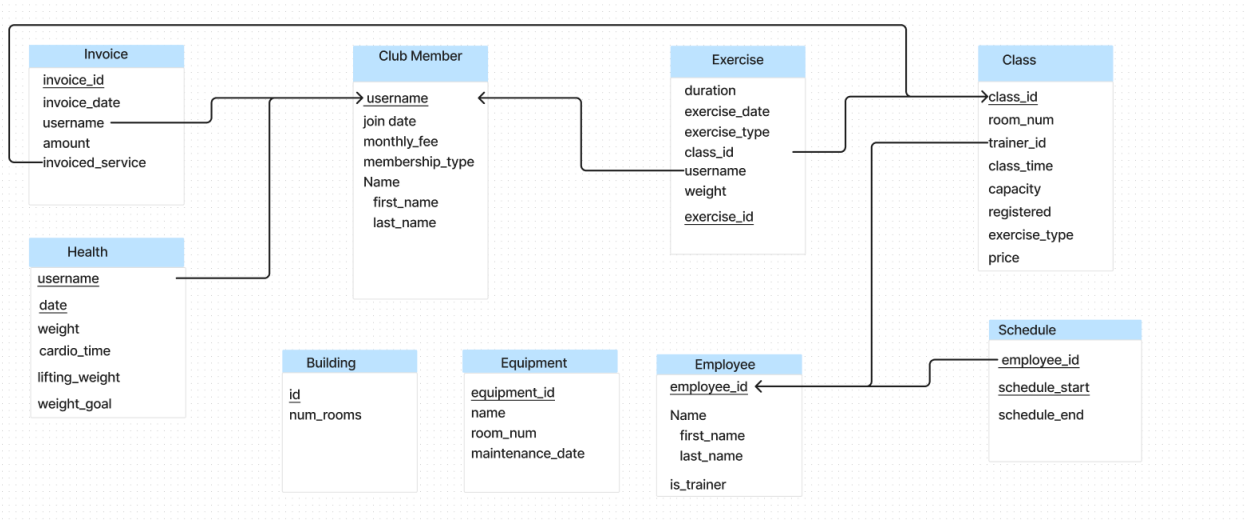
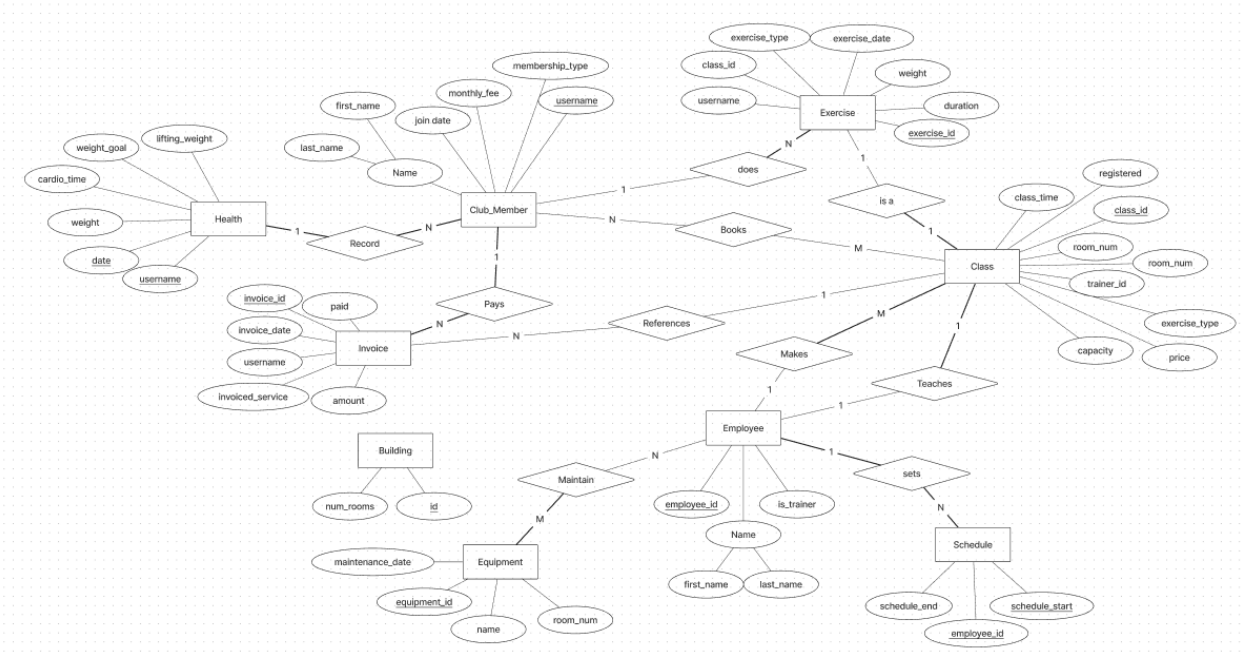
Project Report
By: Allie Stefanakis, Liam Stickle

https://youtu.be/rjsQuPuiqIU
https://github.com/a-llie/COMP3005-Final

2.1
We made several assumptions that formed certain design decisions, including: :
Every group class is taught by one of the gym's trainers
Gym members are billed manually at the start of every month
Trainers can set their schedules through the menu options available to them
Admin can view trainer schedules, and create classes based on their available times
Users are only prompted to pay for classes after the class has already occurred

We started with the 3 main actor tables: member, trainer, and administrator. From there, we added the exercise table to fulfill the logging and to track progress for goals and metrics. We created a table which tracks schedule availability for trainers, which is used as the available times for admins to create classes. We added the building to store the room data for classes and equipment, which is expandable for multiple locations. The building entity was created with a singleton pattern in mind. Gym equipment is represented by a table that tracks its previous maintenance date, as well as a room number representing its location. The trainer and admin were merged into employees for simplicity, with a boolean that distinguishes their separate functionality. An invoice table was added to track balances owing and paid from classes and membership fees. All dates are stored as timestamps.




exercise_type
exercise_date
class_id
weight
username
Exercise
duration
exercise_id
membership_type
monthly_fee
first_name
join date
username
last_name
Name
does
N
1
weight_goal
lifting_weight
cardio_time
Health
weight
date
username
1
Record
N
Club_Member
1
N
Books
is a
registered
class_time
class_id
room_num
room_num
trainer_id
Class
M
1
invoice_id
paid
invoice_date
username
invoiced_service
amount
Invoice
N
Pays
N
References
1
M
Makes
1
Teaches
exercise_type
capacity
price
Building
num_rooms
id
N
Maintain
M
Employee
1
sets
N
employee_id
is_trainer
Name
first_name
last_name
Schedule
maintenance_date
Equipment
equipment_id
name
room_num
schedule_end
schedule_start
employee_id

2.2


Equipment
equipment_id
name
room_num
maintenance_date
Building
id
num_rooms
Schedule
employee_id
schedule_start
schedule_end
Employee
employee_id
first_name
last_name
is_trainer
Class
class_id
room_num
trainer_id
class_time
capacity
registered
exercise_type
price
Exercise
duration
exercise_date
exercise_type
class_id
username
weight
exercise_id
Club_Member
username
join date
monthly_fee
membership_type
first_name
last_name
Invoice
invoice_id
invoice_date
username
invoiced_service
paid
Health
username
date
weight
cardio_time
lifting_weight
weight_goal

2.3-2.4
See SQL directory

2.5
Our application uses PostgreSQL to manage the data associated with the gym. It is a command-line interface application which uses Python to facilitate the interactive component. With Python, we leveraged OOP design concepts to create entity classes for the actors, a class for the gym server system and then a class for some helper functions.

Case sensitive

2.6
We felt that it would be beneficial to have the ability to create a large amount of randomized data in order to properly test the application's functionality, and so we implemented several 'generators' in the .ddl file to populate several tables.

Along with that, we also implemented several triggers that keep the data of a certain format (for example, all classes start exactly on the hour), or that set other attributes based on certain variables (for example, the 'monthly_fee' variable on a club member is determined by the type of membership they choose).

Additionally, we used transactions to modify foreign key values, specifically the username can be changed by the user, requiring modifying table constraints.