Project Report

Name: Yun Hye Nam

Student ID: 101211656

1. Conceptual Design

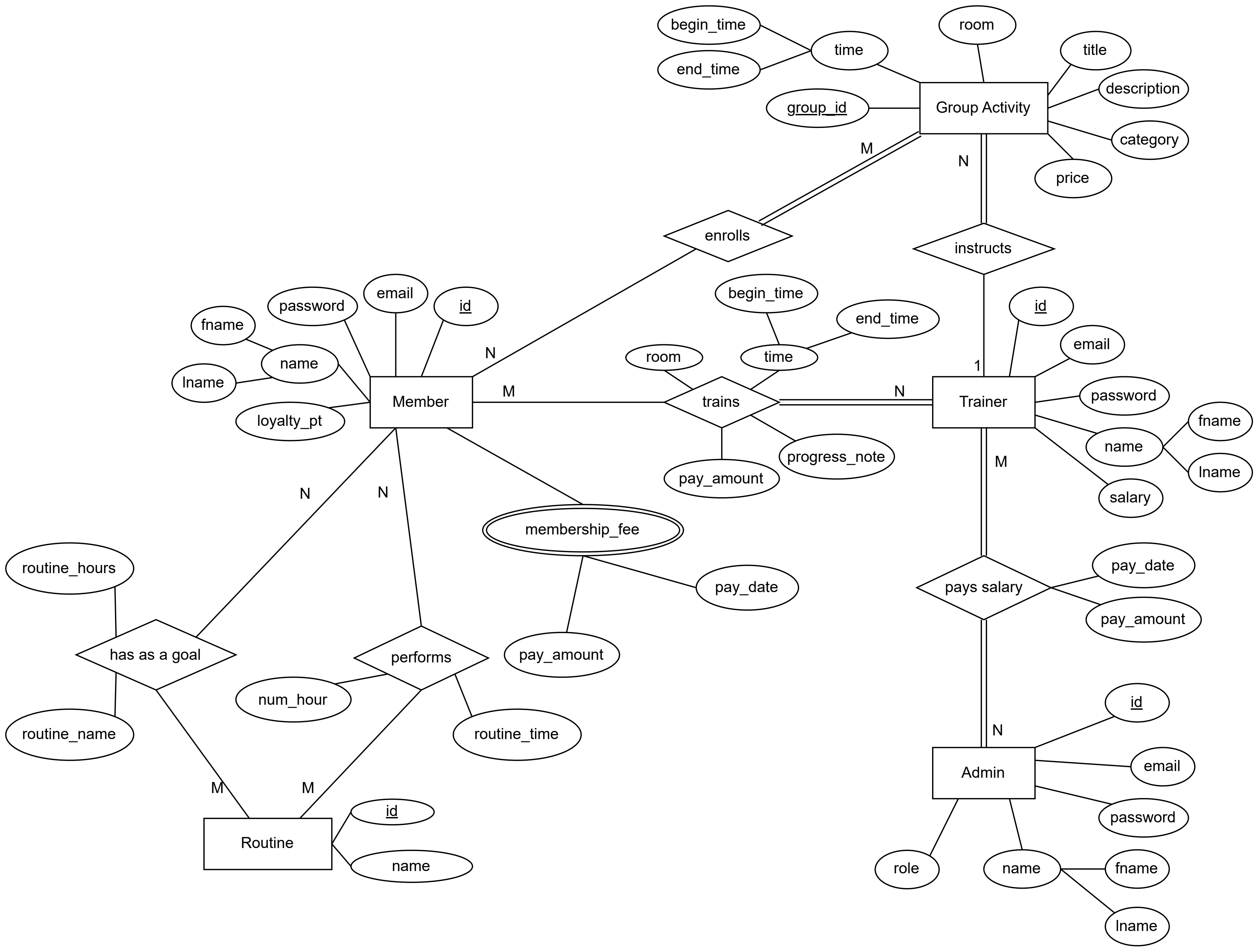
[The Entity Relationship Diagram is on the following page.]

Assumptions:

1. Each trainer needs to train at least one member.
2. A member does not need a trainer but can have more than one trainer for different fitness routines.
3. There are multiple administrative staff that manage trainer payroll.
4. A member can choose to enroll in more than one group activity.
5. Group activities will have at least one participating member. If no one registers, administrative staff will cancel the group activity.
6. Each group activity is instructed by one trainer.
7. Members can have multiple fitness goals, but goals are unique for each routine.

Explanations

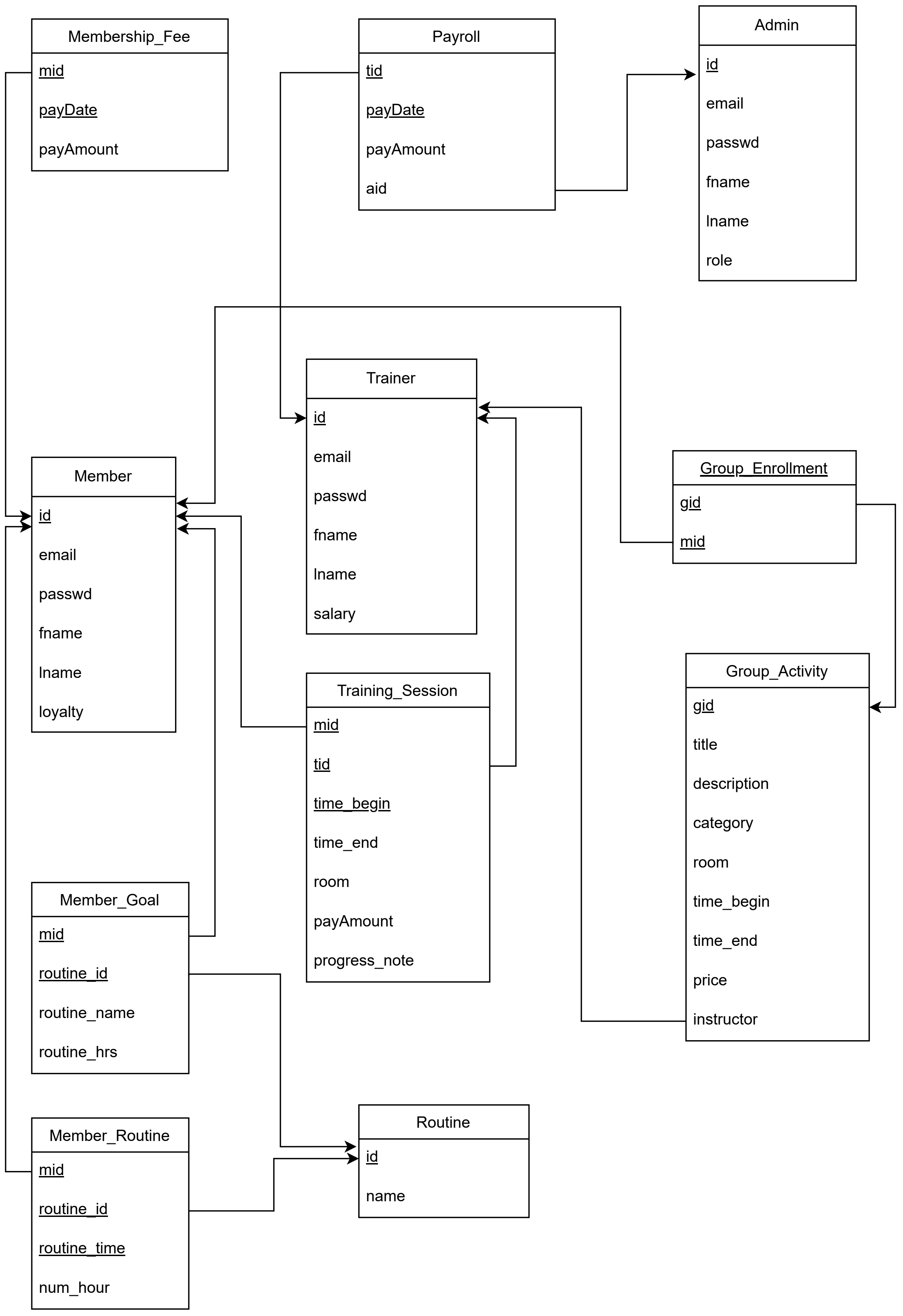
1. Members track their exercise routine by recording the fitness routine and the date of exercise.
2. Member's fitness goals measure the hours of a particular routine that the member aspires to perform.
3. Group Activity represents "group fitness classes, workshops, and other events" and the categories that classify the activity is recorded
4. Administrative staff manage the revenues coming from membership fees, training session registration payments, and group activity registration payments.
5. Administrative staff manage the payroll for trainers.


begin_time
time
end_time
room
title
description
group_id
Group Activity
category
price
M
N
enrolls
instructs
begin_time
end_time
room
time
trains
progress_note
pay_amount
password
email
id
fname
name
lname
loyalty_pt
Member
N
M
N
1
Trainer
id
email
password
name
fname
lname
salary
M
N
N
membership_fee
pay_date
pay_amount
pays salary
pay_date
pay_amount
routine_hours
has as a goal
performs
num_hour
routine_time
routine_name
M
M
Routine
id
name
N
Admin
id
email
password
role
name
fname
lname

2. Reduction to Relation Schemas

[The schema is on the following page.]

Member (id, email, passwd, fname, lname, loyalty) [loyalty = loyalty points]
Trainer(id, email, passwd, fname, lname, salary)
Admin(id, email, passwd, fname, lname, role)
Routine(id, name)
Membership_Fee(mid, payDate, payAmount)
Member_Goal(mid, routine_id, routine_name, routine_hrs)
Member_Routine(mid, routine_id, routine_time, numHour)
Payroll(tid, payDate, payAmount, aid) [aid = admin id]
Training_Session(mid, tid, ts_room, time_begin, time_end, payAmount, progress)
[progress = progress notes]
Group_Activity(gid, title, description, category, price, instructor, time_begin, time_end, room)
Group_Enrollment(mid, gid)


Membership_Fee
mid
payDate
payAmount
Payroll
tid
payDate
payAmount
aid
Admin
id
email
passwd
fname
lname
role
Trainer
id
email
passwd
fname
lname
salary
Member
id
email
passwd
fname
lname
loyalty
Group_Enrollment
gid
mid
Training_Session
mid
tid
time_begin
time_end
room
payAmount
progress_note
Group_Activity
gid
title
description
category
room
time_begin
time_end
price
instructor
Member_Goal
mid
routine_id
routine_name
routine_hrs
Member_Routine
mid
routine_id
routine_time
num_hour
Routine
id
name

3. Normalization
1NF is achieved by decomposing multivalued attributes into another table and writing the composite attributes as separate attributes. Membership Fee is a multivalued attribute of Member, as members pay their fees periodically at different dates. So a new table “Membership Fee” was created to record the amount and date of payment. “Name” in Member, Trainer, Admin is a composite attribute, so the name is broken down as attributes “fname” and “lname”.

The functional dependencies for each table are
Member: id → email, passwd, fname, name, loyalty
Trainer: id → email, passwd, fname, name, salary
Admin: id → email, passwd, fname, name, role
Routine: id → name
Member_Goal:
    {mid, routine_id} → routine_hrs
    routine_id → routine_name
Membership_Fee: {mid, payDate} → payAmount
Payroll: {tid, payDate} → payAmount, aid
Training_Session: {mid, tid, time_begin} → time_end, ts_room, payAmount, progress
Group_Activity: gid → title, description, category, price, instructor, time_begin, time_end, room

All the dependencies except “routine_id → routine_name” are full dependencies, each depending on all the primary keys of the corresponding table.

The table Member_Goal(mid, routine_id, routine_name, routne_hrs) has a partial dependency “routine_id → routine_name”. Member_Goal has a composite primary key: “routine_id” and “mid”, but the attribute “routine_name” only depends on “routine_id”. Thus, we can decompose the Member_Goal table into two tables:

Member_Goal1(mid, routine_id, routne_hrs)

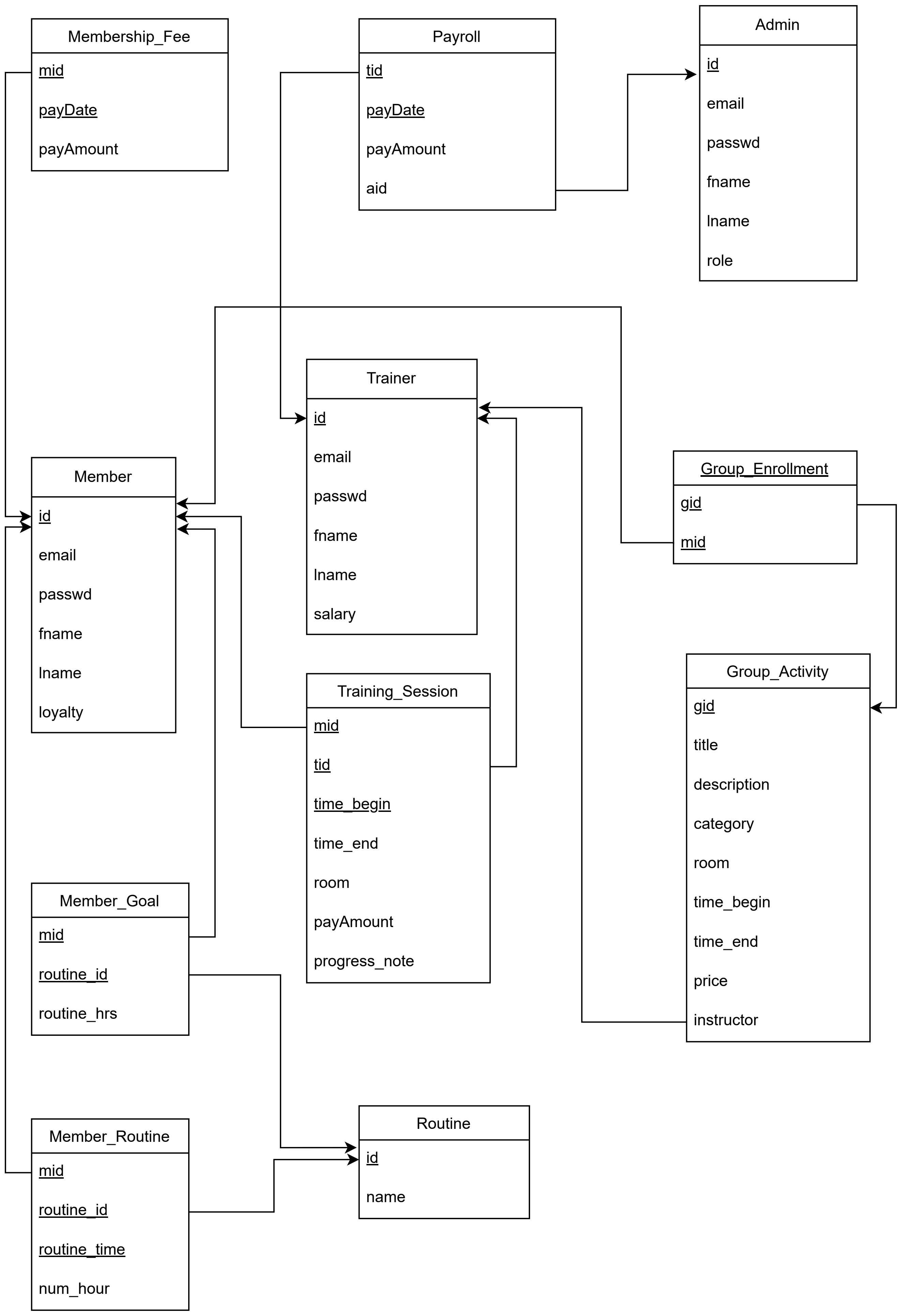
mid	routine_id	routine_hrs

Member_Goal2(routine_id, routine_name)

routine_id	routine_name

The second table is identical to the Routine(id, name) table. Thus, the Member_Goal1 can be used in lieu of the original table but Member_Goal2 should be absorbed into the Routine table.

Now, 2NF is achieved with all the dependencies being full dependencies.


Membership_Fee
mid
payDate
payAmount
Payroll
tid
payDate
payAmount
aid
Admin
id
email
passwd
fname
lname
role
Trainer
id
email
passwd
fname
lname
salary
Member
id
email
passwd
fname
lname
loyalty
Group_Enrollment
gid
mid
Training_Session
mid
tid
time_begin
time_end
room
payAmount
progress_note
Group_Activity
gid
title
description
category
room
time_begin
time_end
price
instructor
Member_Goal
mid
routine_id
routine_hrs
Member_Routine
mid
routine_id
routine_time
num_hour
Routine
id
name

3NF is also achieved, since there is no transitive dependency.

4. Database Schema Diagram

[The schema is on the following page.]

5. Implementation

I would use this to create a three-tier architecture. The presentation layer would be the front-end allowing clients to interact with the app. The logic layer would be the back-end dealing with the business logic. The business logic will interact with the database if the client's input fits the right condition.

In the project, there will be a separate frontend directory and backend directory, each having different port numbers. CORS will be configured to allow the different parts to communicate. The database will be set up on PostgreSQL server. The host, port number, username, password, and database name will be set up as environment variables for the backend app to communicate with the database.

The frontend would use React.js and HTML/CSS to design what the client sees, clicks, and writes input to. When a client loads a particular page, the frontend side will call the appropriate data from the backend server using GET request. When a client interacts with the view with a button click or writes a data input, this will trigger a POST, UPDATE, DELETE request to the backend.

The backend will be run on JavaScript, Node and Express. It will use 'node-postgres' modules to communicate with the PostgreSQL server and prevent SQL injection attacks with parameterized queries.
Using the principle of abstraction, the backend will be divided into different layers

1. Express routing will receive the HTTP requests and pass them to controller functions.
2. The controller will receive the request object, then send it to the corresponding middleware function. Upon receiving a new object from middleware, the controller will send it as a response object.
3. The middleware functions perform the business logic and extract values from or transform the request object so that they can be used by the model layer.
4. The model layer directly interacts with the database by performing SQL queries. It will return a new object to be used as a response object by the other layers

The frontend will have separate authentication pages for members, trainers, and administrative staff. When a user registers, their password will be hashed using the bcrypt algorithm with a salt to be stored securely in the database. When the user tries to authenticate, it will match the hashed input with the stored hashed password. If the user successfully authenticates, user id and their type (member, trainer or admin) will be stored in the session until the user logs out, to keep track of the user's info and login status.

If the user’s type is Member, they will be able to see their own profile but not other members’. The Trainers will be able to see the profiles of members they train. The administrative staff will be able to see every member and trainer’s profile.

Member profiles will show email, name, fitness goals and achievements. Trainer profiles will show their email, name, training members and schedule. Admin profile will see links to administrative functionalities - such as (re-)scheduling and canceling training sessions and group activities, salary payments to trainers, and membership fee management.

The sidebar will have links to the page that allow members to schedule or reschedule or cancel group activity registrations and training sessions. If there is a time and room conflict during scheduling, the model layer will return no data. The middleware will then create an error object, which will be sent by the router to the frontend. The frontend will show error messaging. If scheduling is successful, the response object will contain the inserted or updated data, which will be modified by the frontend to show the change to the client.