Section 1: Project Direction Overview

I would like to build a social fitness tracking app which allows users to enter their health details and also connect with friends to motivate one another to reach their fitness goals. Users will have the ability to track their daily workouts and calorie intake. Friends can also send each other rewards like gift cards or gym sessions as a way to celebrate reaching workout goals. The app should be able to track different kind of workouts such as running, walking, cycling, swimming, yoga, weight training, and dance. The app should also be able to integrate with smart watches and health applications on mobile devices. App integrations will be a premium feature and will allow the app to collect more user data to get a better idea about their health. I also want to be able to connect the app to the user's primary physician so that the doctor's get a progress report for patients receiving critical care.

Additionally for some newer features that make my app more user-focused and unique from the existing fitness apps, I would like to add Smart Hydration alerting based on user activity levels, BMI and local weather conditions. So on hotter days, users will be reminded to drink water in more frequent intervals. It would also be powerful to be able to use user data, especially from those who have been diagnosed with some health conditions like cholesterol or diabetes or stroke, to be able to see their health trends and current statistics. Using machine learning we could be able to warn users when the machine learning models detect that users are at high risk of developing certain illnesses. Doctors could cross check the warnings and reports generated from the app to see if users are actually at risk and prescribe proper treatment or exercise routines accordingly. So the app would need to collect health metrics like systolic blood pressure, diastolic blood pressure, resting heart rate, cholesterol (Total, LDL, HDL), triglycerides. Also lifestyle metrics like hours of sleep, daily step count, smoking, alcohol consumption, BMI would be possible indicators of risks for conditions like cholesterol, diabetes or stroke through Machine Learning integration.
Additionally for female users, I want to add a menstrual cycle tracker which adds the ability to suggest workouts according to the user's menstrual phase. So for each of the 4 menstrual phases, namely Menstrual, Follicular, Ovulation and Luteal phases, we can suggest the best workouts to women. During the menstrual phase when estrogen and progesterone are low, women feel more tired and have more cramps, so only gentle workouts like yoga or stretching can be suggested. Whereas during Ovulation when women have the most energy, we can suggest weightlifting and high intensity workouts. To know a woman's menstrual cycle, we would need to keep track of metrics like Cycle start date, cycle length, period duration. Additionally, we can record the user's inputted fatigue level to give more user-specific recommendations.

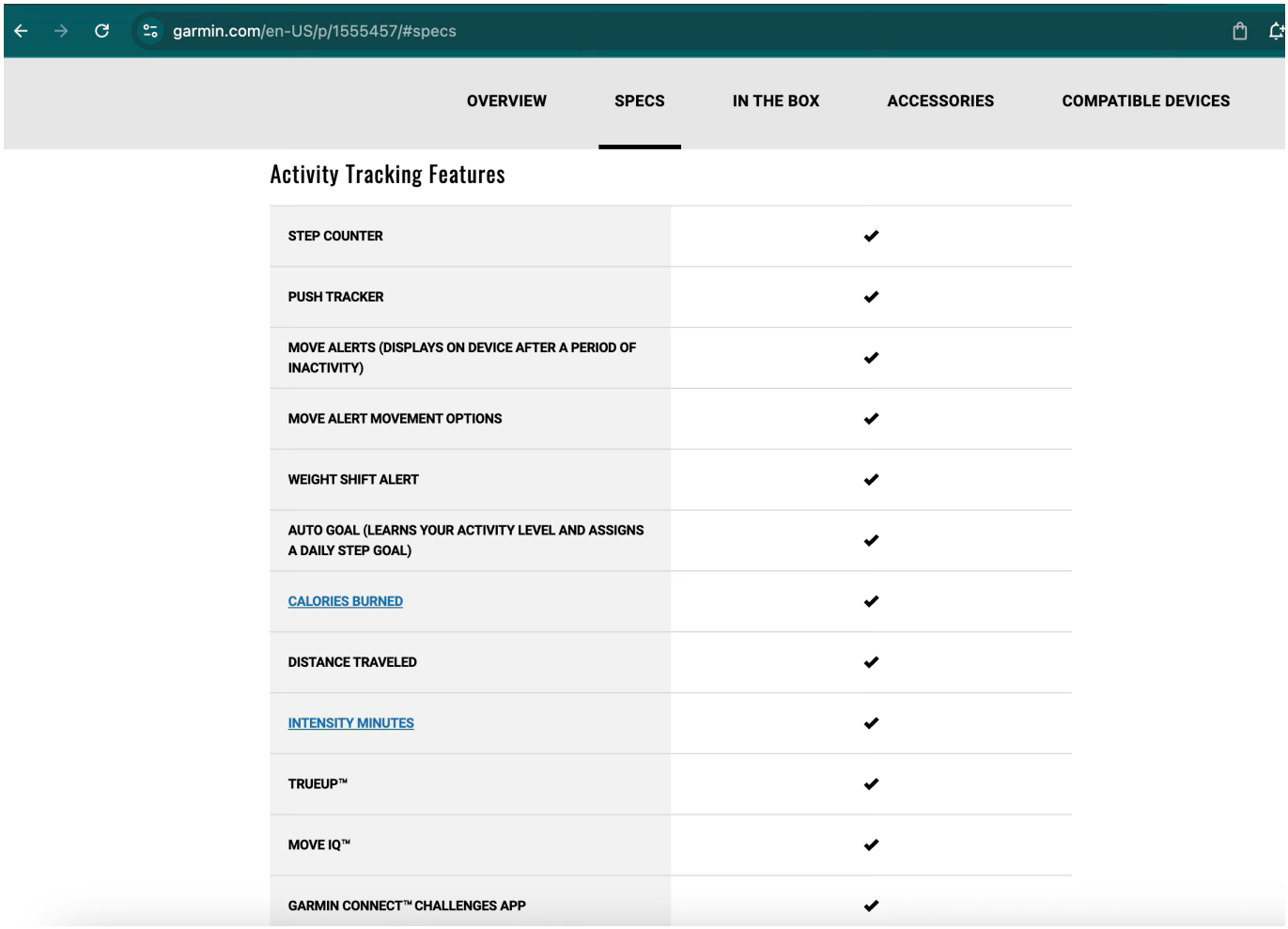
An example of someone using the app on a normal day for a patient like my mother who has diabetes would include recording her calorie intake for all meals including breakfast, lunch and dinner. It would also be important to note the timestamps of when she has her meals. I would also like to keep track of what medications she is consuming and marking the times of consumption too. She enjoys her morning walks and goes to the vegetable market on her stroll as well. The app should extract her step count on her walking workout in the morning. The amount of calories burned will also be extracted, along with the total miles she walked and the duration of the workout. Her heart rate would also be good to monitor and for patients who have glucose monitors, we can extract their glucose readings onto the app as well. Incase heart rate, glucose are at alarming levels, we can immediately notify her doctor by calling

the hospital as well. After seeing my mom walk for an hour as shown on mom's app post, my aunt would like to reward my mom by sending her a Starbucks gift card because she has been working so hard to reach 10k steps everyday. At the end of the month, when mom goes for her annual checkup, the doctor gets a full report of her yearly fitness habits from the app.

Through this project, I will be focusing on the database setup required to host the application. I will focus on the entities like users, workouts, meals, friends, exercise types, healthcare providers, billing information and rewards. Detailed information about each of the entities will be important to create an impactful health tracker. I will keep modifying the project as necessary in each iteration. I believe this will help people become more inspired to focus on their health since friends and doctors will be aware of a user's activities. The automated alerting system can also save patients in the case of emergencies.

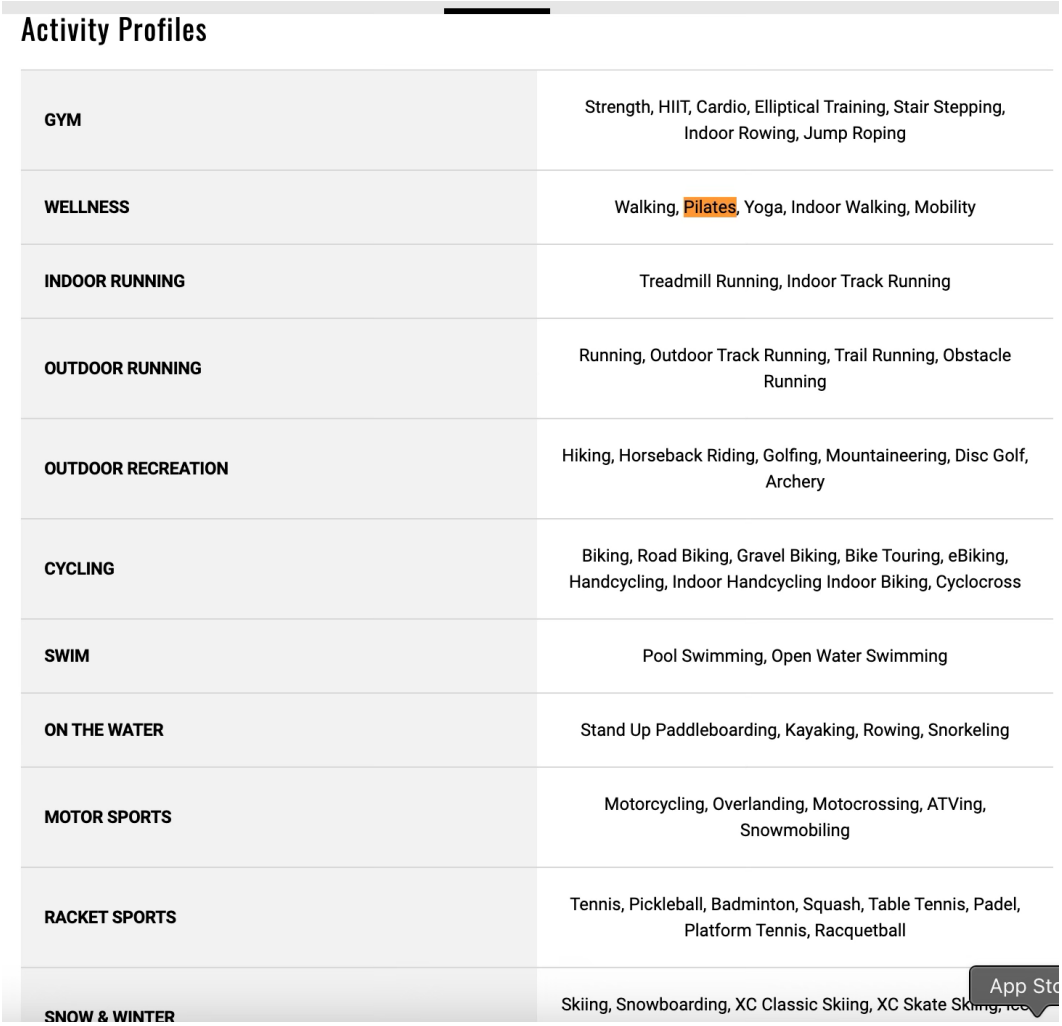
Section 2: Artifacts and Analysis

Replace this with your analysis of at least three artifacts. Redact, if necessary, information that cannot be shared.



Activity Tracking Features

STEP COUNTER	✔
PUSH TRACKER	✔
MOVE ALERTS (DISPLAYS ON DEVICE AFTER A PERIOD OF INACTIVITY)	✔
MOVE ALERT MOVEMENT OPTIONS	✔
WEIGHT SHIFT ALERT	✔
AUTO GOAL (LEARNS YOUR ACTIVITY LEVEL AND ASSIGNS A DAILY STEP GOAL)	✔
CALORIES BURNED	✔
DISTANCE TRAVELED	✔
INTENSITY MINUTES	✔
TRUEUP™	✔
MOVE IQ™	✔
GARMIN CONNECT™ CHALLENGES APP	✔

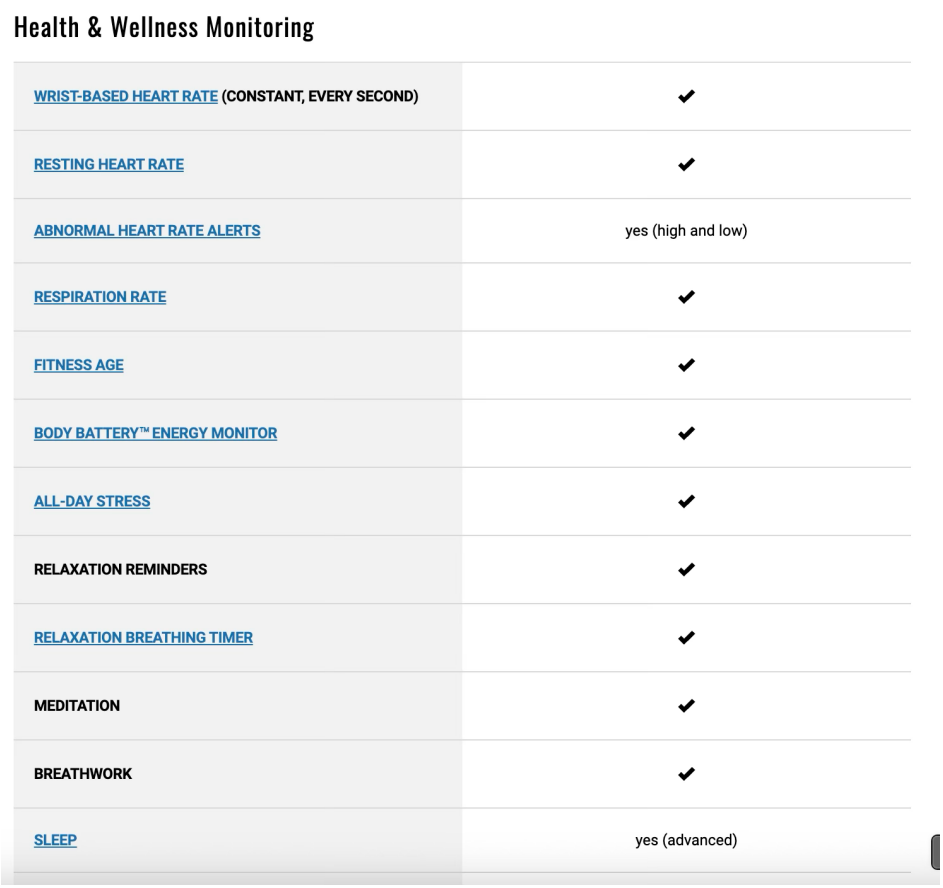


Activity Profiles

GYM	Strength, HIIT, Cardio, Elliptical Training, Stair Stepping, Indoor Rowing, Jump Roping
WELLNESS	Walking, Pilates, Yoga, Indoor Walking, Mobility
INDOOR RUNNING	Treadmill Running, Indoor Track Running
OUTDOOR RUNNING	Running, Outdoor Track Running, Trail Running, Obstacle Running
OUTDOOR RECREATION	Hiking, Horseback Riding, Golfing, Mountaineering, Disc Golf, Archery
CYCLING	Biking, Road Biking, Gravel Biking, Bike Touring, eBiking, Handcycling, Indoor Handcycling Indoor Biking, Cyclocross
SWIM	Pool Swimming, Open Water Swimming
ON THE WATER	Stand Up Paddleboarding, Kayaking, Rowing, Snorkeling
MOTOR SPORTS	Motorcycling, Overlanding, Motocrossing, ATVing, Snowmobiling
RACKET SPORTS	Tennis, Pickleball, Badminton, Squash, Table Tennis, Padel, Platform Tennis, Racquetball
SNOW & WINTER	Skiing, Snowboarding, XC Classic Skiing, XC Skate Ski

https://www.garmin.com/en-US/p/1555457/#specs

These are images of features of the Garmin smartwatch which will be considered as a data source that will be integrated into the application.



Health & Wellness Monitoring

WRIST-BASED HEART RATE (CONSTANT, EVERY SECOND)	✔
RESTING HEART RATE	✔
ABNORMAL HEART RATE ALERTS	yes (high and low)
RESPIRATION RATE	✔
FITNESS AGE	✔
BODY BATTERY™ ENERGY MONITOR	✔
ALL-DAY STRESS	✔
RELAXATION REMINDERS	✔
RELAXATION BREATHING TIMER	✔
MEDITATION	✔
BREATHWORK	✔
SLEEP	yes (advanced)

Entities:
User - We will need general information like user id, password, email for each registered user on the app. Along with this, general details like age, height, weight, name, location, gender are required for smart recommendations and summaries.

Workout - The smartwatch records each workout the user does so we need to care about workout duration, calories burned, type of workout.

WorkoutType - This table would hold additional information about each type of workout

DailyHealthMetric- Vital metrics like resting heart rate, total steps, hours of sleep will be recorded daily for each user.

MenstrualCycleMetrics- Cycle start date, cycle length, period duration, fatigue level are necessary for tracking the female user's menstrual phase to make appropriate recommendations.

HealthRiskMetrics- Information about the user's habits such as smoking frequency, alcohol consumption and medical information on systolic blood pressure, diastolic blood pressure, cholesterol (Total, LDL, HDL), triglycerides are all required to feed machine learning models to create warnings for risk of diseases like cholesterol and stroke. Existing medical conditions can also be included in this table itself.

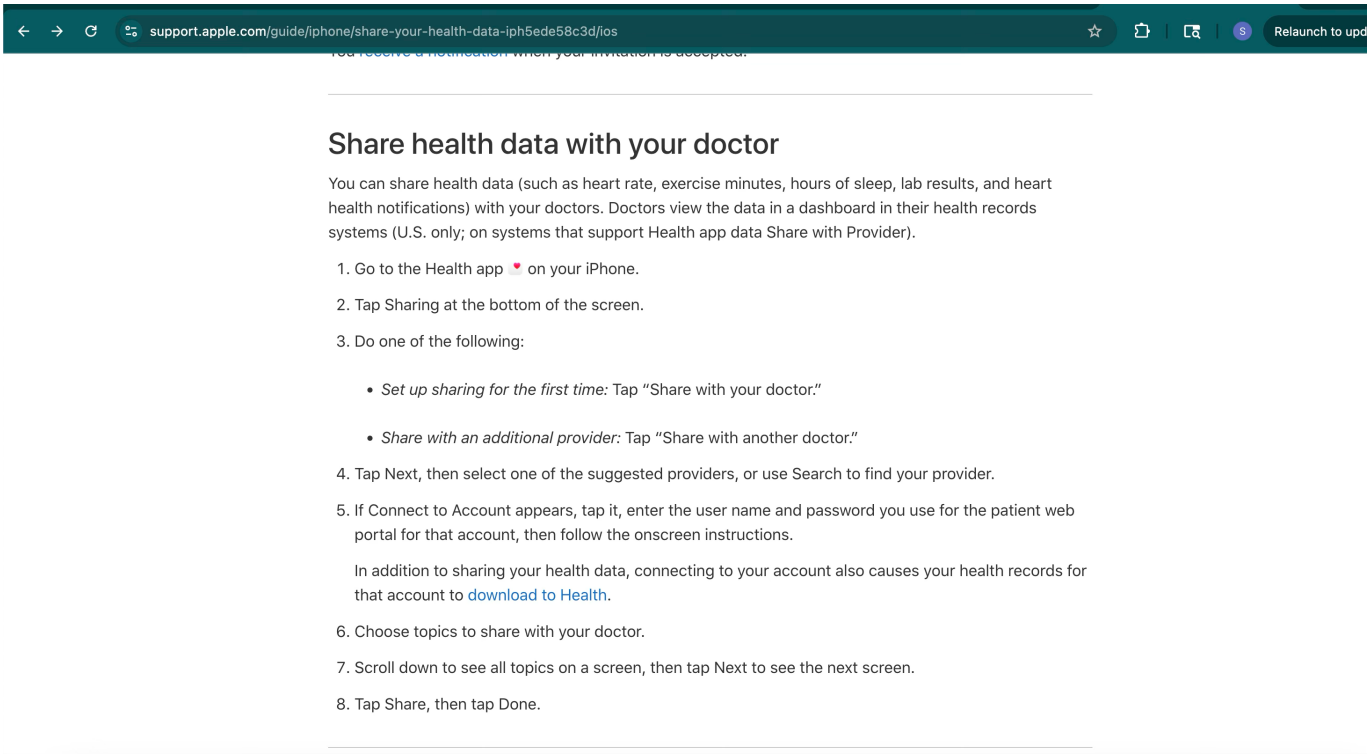
HydrationMetrics- Details like location, weather updates need to be recorded for environment and user activity-based hydration reminders.

Preconditions:
Capture health metrics daily- Collect and save daily metrics at the end of each day for the user
Capture workouts- Collect and save each workout done by the user

Actions:
User enters personal information- Details like age, height, weight, name, location, gender need to manually entered by the user for confirmation when setting up the app
User enters lifestyle preferences- Details like smoking frequency, alcohol consumption which are indicators of disease conditions required by the machine learning model
Doctor/user enters detailed medical information- systolic blood pressure, diastolic blood pressure, cholesterol (Total, LDL, HDL), triglycerides must be obtained from hospital records
User enters menstrual cycle information- Cycle start date, cycle length, period duration, fatigue level tell us more about the menstrual phase the user is in which will be used for workout recommendations
User chooses workout type on app- Select the appropriate workout type which will give the relevant summary.

Constraints:
1. The user must manually record the specific workout
2. The workout must be a valid workout type in our list
3. The user must be female to get menstrual cycle-based workout recommendations
4. The user needs to manually provide medical information and lifestyle preferences to get warnings about health risks
5. All the units of measurements for each manually entered metric should be appropriately matched by user
6. Hydration metrics like location of the user and activity level need to be kept up to date so they need to be refreshed every 30 minutes.

Helpfulness to my project: These existing smartwatch features show all the different kinds of metrics being recorded for users which would be beneficial to record. This section would only apply to smartwatch users, but people more conscious about monitoring their health should have such smartwatches. We will need a way to integrate and transfer all this information from the smartwatch to the fitness app. Since I currently use an Apple Watch myself, I know metrics like step count and workout metrics are already synced to the iPhone's Activity application. The app I am designing in this project would be similar to the Activity application with some additional features. A lot of the data/input needed to be saved in entity tables have been mentioned above.



https://support.apple.com/guide/iphone/share-your-health-data-iph5ede58c3d/ios
These are instructions to share health data with your doctor on an iPhone through the Health app.
Entities-
Doctor- If added by the user, the contact information for the doctor/hospital, doctor's name and specialty should be on file.
Insurance information- The patient's insurance provider, insurance ID should be on file.
Emergency contact- There should be the option to add an additional contact with their name and contact information.
Billing Information- In case of emergency, including a credit card and any other health savings account card information for immediate use would be helpful.
Calls- Each emergency call that is made should be recorded.
Messages- Record emails from app sent to doctor's office.

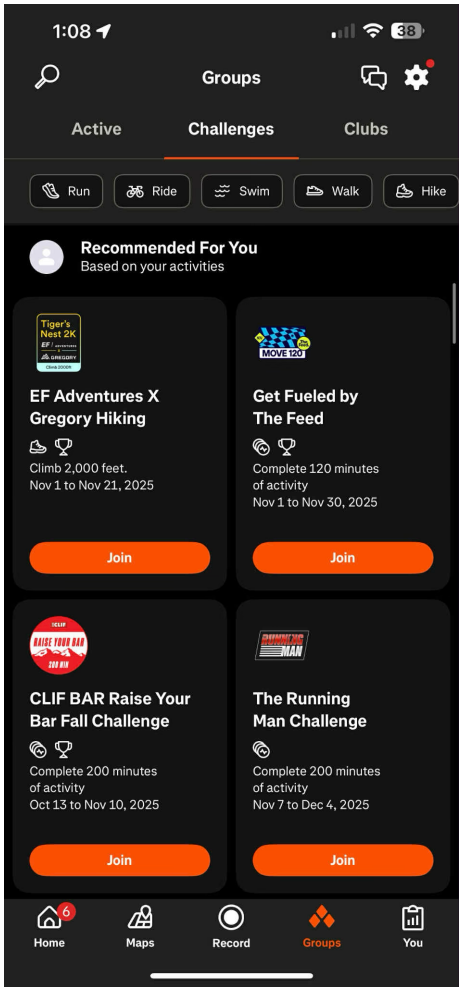
Actions-
User sends generated annual report for doctor- During the patient's annual physical, the app should be able to share the annual health report of the patient.
Respond to warnings from app- Based on the doctor's thresholds or predictions of disease risk from the machine learning models for certain critical patients, there should be a warning generated. User can respond to call the hospital for follow up or send a message requesting a visit through the app.
Call doctor or hospital- On the event of an emergency, based on any abnormal vital signs, the app should automatically call the 911, the hospital for the ambulance or user can set the app to only manually call the hospital.

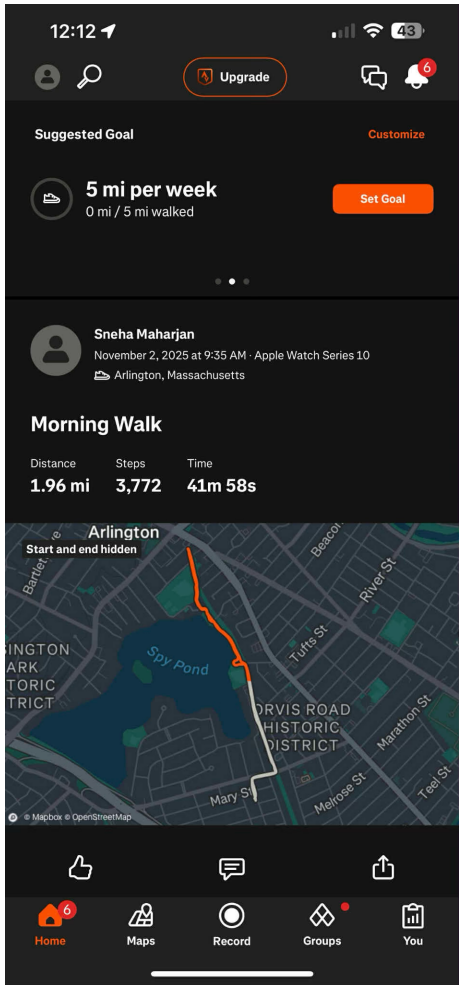
Constraints:
4. Hospitals who accept emergency calls on behalf of critical patients should only be called. If no hospital information is present, directly call 911.
5. Patient's medical conditions should be recorded when signing up by the doctor. This constraint won't apply for non-patients or people who don't have any health conditions.

6.Doctors should set the warning or emergency vital signs limits.
7.There should be at least 1 emergency contact.

Helpfulness to my project: There is also the ability to share health information to a personal contact on iPhones which seems like a great feature for the fitness app. In the same way, we should be able to select what information we want to reveal to friends on the app. Majority of this information will need to be stored securely and abide by HIPAA standards. My app should be considered a central platform to collect and analyze the user's health data and reach out to medical professionals directly. It helps to have 1 app that does it all rather than multiple scattered apps that provide selective functionalities only.

The images below are from my Strava app.





Entity - As mentioned above already, workout information will be recorded.
Workout - The smartwatch records each workout the user does so we need to worry about entering workout duration, calories burned, type of workout, distance covered, timestamp.
WorkoutType - This table would hold additional information about each type of workout
Rewards- The credits gifted to the user should be stored with information about the sender and the credit type and amount.
Friends- Store friendship relations of the users to select whom to share posts and credits with.
Badges- On each milestone, a badge is awarded to the user.
Posts- Store post related information when users share badges or workout information.

Challenges- Any challenges the user is participating in can optionally be recorded.

Actions:
Give permission to transfer workouts from smartwatch to app- On the app, users need to be prompted to transfer workout information from their smartwatches to the app after completing their workouts.
Post about workouts and badges to friends- We need to keep track of each milestone for users and provide badges when reaching those milestones.
Send rewards to friends- Friends on the fitness app need to be able to send gift cards to each other.

Constraints:
- Milestones/badges need to be reset every month.
6. Minimum workout time needs to be 1 minute for a workout to be considered valid.
7. Maximum of 5 workouts can be recorded for each user per day.
8.Only friends on the app are allowed to send and receive credit.
9.Only friends can view each other's posts.

Helpfulness to my project: Seeing a full-fledged app like Strava provides a lot of practical insight for my fitness app. Strava also allows users to network amongst each other and top athletes. There are also concepts like leadership boards, clubs and challenges which are great features for users to participate in for more health benefits and building community. I have introduced the unique concept of sending credits like fitness classes or giftcards in my app.

Section 3: Use Cases

Replace this with your use cases, updated, if necessary.

I would like to differentiate users based on how much of an active person they are so they can have less or more advanced features accordingly. I would like to provide 2 options for the type of user; they can be either beginner user or advanced user. Only the advanced users will have the option to join challenges.

1. User Information Entry when setting up the app Use Case
-The application will ask the user to create an account to use the app.
-The user application asks the user if they are a beginner or advanced user based on their workout frequency.
-The user enters their information, and a new account gets created in the database.
-All the user details like their general biological information, location zip code, lifestyle metrics, menstrual cycle metrics and all data needed for manual entry are stored.

2. Menstrual Cycle based workout recommendations use case
-The user of the app is female and would like to get recommendations catered to her menstrual phases.
-She will be asked to fill out data related to her cycle's start date, end date, duration.
-She will also be required to provide her fatigue level on each day of her menstrual phase to gauge her energy levels.
-She will then be recommended more intense or less intense workouts. For example, during her menstrual phase days 1 and 2 when estrogen and progesterone are low, she will be suggested to perform yoga for 30 minutes.

3. Workout Information Transfer Use Case
-The Fitness tracker app notices a workout once it has been completed and syncs to the smartwatch to extract the data.
-User gets request to transfer the workout information and must accept it on the app.
-User is also prompted to manually select the workout type for detailed summary based on the type.
-The new workout instance is recorded in the database.

I would like to distinguish the Credit/Rewards type. The app will allow users to choose from a gym class booking, monetary gift card or a new reward of their choice as options for the rewards they gift to friends. As the app expands, we could add other forms of rewards outside of gym classes and gift cards. To allow for more flexibility, I have set up the use case below accordingly-

4. Friends Credit Rewards Use Case
-A friend sends a request to her friend on the Fitness tracker app.
-Both friends can view each other's activities and want to send credit to each other after going on a group hike.
-The reward can be a gym class type or a gift card type or other new reward type, which must be selected by the user.
-The new credits being sent/received are recorded in the database. Both the sender and receiver are recorded.

5. Challenge Achieved Use Case
-A user signs up for a challenge and the challenge is recorded as in progress in the database.
-Each milestone achieved is recorded in the database.
-When the milestone is achieved, a badge is provided to the user and is also recorded in the database.

6. Medical Emergency Use Case
-When the patient's vitals are abnormal, the emergency contact person and hospital are notified.
-Both these calls are recorded in the database.

7. Hydration Reminder Use Case
-The user on sign up gets prompted to enter their location details
-The app notices user's activity levels, the weather in their zip code. In this case, the temperature is 90F on a hot summer day in Boston.
-The app sends hydration reminders every 2 hours to the user which is more frequent to avoid dehydration during summer.

Section 4: Structural Database Rules (revised)

Include your history table in the structural database rules.

Below is the structural database rules modified to include the new entities.

1. Each user has one or more health risk metric logs; each health risk metric log belongs to one user.
2. Each user may have many menstrual cycle metric logs; each menstrual cycle metric log belongs to one user.

Considering use cases 1 and 2, we think about what essential information is needed when a female user starts using the app to get customized workout recommendations. So, the main entities that are required to be setup here would be User, HealthRiskMetrics and MenstrualCycleMetrics to begin with. Each can have 1 to many HealthRiskMetrics records which contains information about their unique lifestyle related to their drinking and smoking habits and medical information related to their blood pressure and cholesterol. We must keep track of these details about the user over the entire year to create an annual summary, so if any of the details change we will need to insert a new record for the updated user information.
Only if the user is female, they will have data to enter related to their menstrual cycle, so each user may or may not have a single entry in MenstrualCycleMetrics. Also, for women, their menstrual cycle start and end times can vary each month, so those records may need to be updated monthly depending on each woman. For each update, we want to create a new metrics row with all the updated information to give recommendations based on up-to-date information.

3. Each user may have many workouts; each workout belongs to one user.
4. Each workout belongs to one workout type; each workout type may have many workouts.
5. Each user may have many daily health metric logs; each health metric log is associated with a single user.

Based on use case 3, we need to record each user's workout information. Additionally, each workout has to be related to a specific workout type. Every user can have a maximum of 5 workouts per day. If the user signs up for the app but doesn't workout or doesn't record their workouts properly through a smartwatch, they can have no workouts recorded too. Also, each user accumulates a daily health metric which keeps track of user's heart rate, step count, sleep hours every day that they have the app installed. So over the year, each user can have a year's worth of entries for such health metrics accumulated each day for their annual health record. Considering the user doesn't even keep the app till the end of the day for the metric log to be saved, the user may have no records stored too.

6. Each user may have many friendships with other users; each friendship relates exactly two distinct users.
7. Each user may have many credits/rewards from other users; each credit is associated with two distinct users.

For use case 4, we need to keep track of a user's friends, so we need an entity to maintain those friendships. A user can have any number of friends, some users may not even be very socially active on the app so they can have no friends too. Each user can possibly appear in many friendship records, and each friendship record provides information about the connection between 2 users. We also need to keep track of the credits or rewards that a user has been gifted by their friends. This same relationship holds for credits given by users. The credit is the gifting record between 2 users since there is a sender and receiver on each instance.

8. Each user has one hydration metric log; each hydration metric log belongs to a single user.

Hydration reminders, as mentioned in use case 7, are generated based on activity levels and the weather at their location. We don't need to keep track of all the hydration metrics, so we can just keep 1 updated row for every user that has the current location, weather at the location and activity levels. This should get updated every 30 minutes.

From the update to case 1, I derive a new structural database rule to support the change:

9. A user is a beginner user or an advanced user.

We only have 2 kinds of user options- beginner and advanced- and that is the complete list. The relationship is totally complete. The user must be either beginner or advanced, so the relationship is disjoint. Since there are no other allowed user types and the user must be one of the subtypes listed, the relationship is totally complete.

I derived a new structural database rule to support the update to use case 4-

10. A reward is a gym class, gift card, or none of these.

So, the reward is allowed to be either a gym class, a gift card or neither of the specified types since we can have a new option later as the app keeps expanding rewards. I made this relationship partially complete so that users have the flexibility to gift their friends in ways outside of the given list of rewards. The relationship is still disjoint since the reward must fall under a subtype listed or something entirely different.

I added rules 11-15 to reflect the new entities required after considering normalization.

11. Each gift card must be associated to one shop; each shop can be associated with many gift cards.
12. Each gym class is associated to a specific gym class type; each gym class type can be associated with many gym classes.
13. Each gym class type has one specific class type; each class type can be associated with many gym class types.
14. Each gym class type has one specific gym; each gym can be associated with many gym class types.
15. Each health risk metric log must have exactly one smoking frequency type and exactly one alcohol frequency type. Each Frequency Type value may be associated with many health risk metric logs through smoking frequency and alcohol frequency.

For the final project iteration, I am adding a new rule based on the history table setup explained below in Section 13.

16. The new structural database rule would be- Each hydration metric log can have many hydration intake update logs, and each hydration intake update log can be associated to only 1 hydration metric.

Section 5: Conceptual ERD (revised)

Include your history table in your conceptual ERD.

Here are the specialization-generalization structural database rules I derived, listed again in this section-

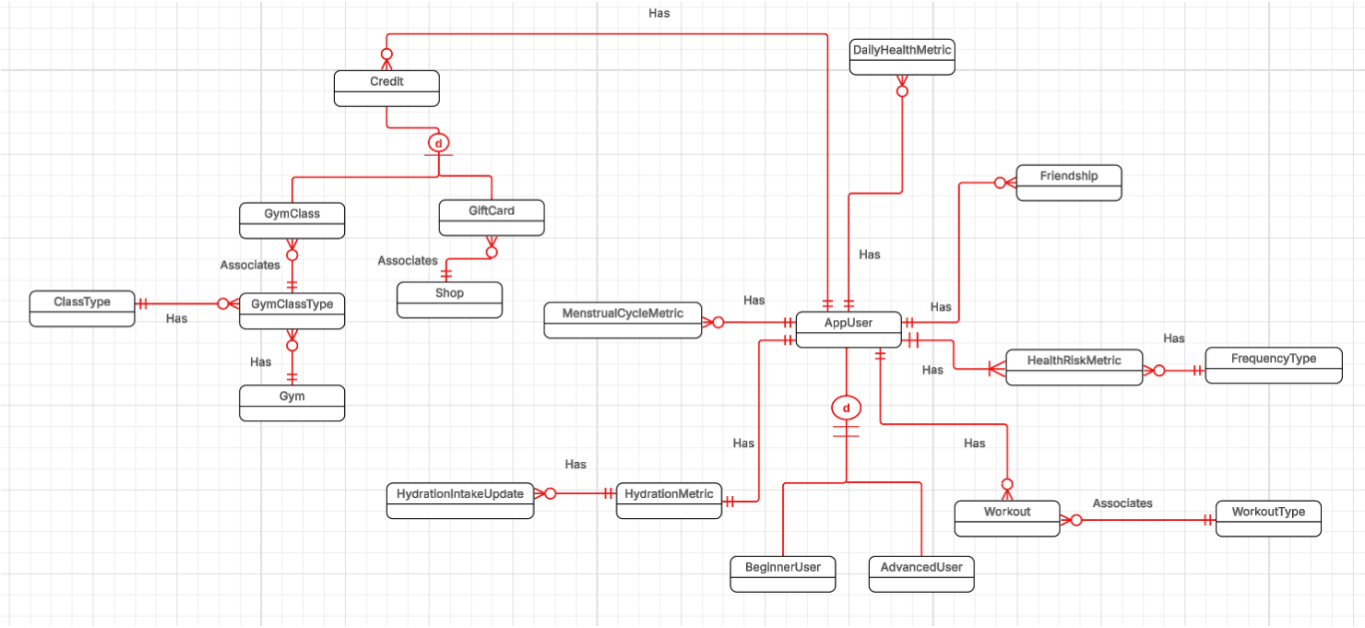
9. A user is a beginner user or an advanced user.
10. A reward is a gym class, gift card, or none of these.

I have added the 2 new entities under User- BeginnerUser and AdvancedUser – and the relationship is totally complete which is denoted by the double dashed lines. Also, the user relationship is disjoint as seen using the circular “d” symbol. I also have 2 additional new entities under Credit- GymClass and GiftCard- and the relationship is partially complete which is denoted by single dashed line, and it is disjoint denoted by the circular “d” symbol. I used crow’s feet notation for my ERD. The changes I mentioned above capture the 2 new structural database rules and use specialization-generalization.

I have also updated the User and Friendship relation based on the response from AI and the walkthrough guide. The Friendship defines the relationship between 2 users, so User is both entity 1 and 2 in this many to many relationship. Similarly, I updated the Credit entity relationship to User, since I need to store the sender and receiver of the credits on the app. Both the sender and receiver originate from the User entity, so Credit acts a bridge between the 2 users that depicts how the credit transfer happened.

I added the new entities necessary to meet Normalization needs. I added GymClassType and Gym for the GymClass. I also added Shop for GiftCard, and FrequencyType for HealthRiskMetric. The conceptual ERD is now in sync with the new database rules and the DBMS physical ERD.

Update to conceptual ERD for Iteration 6-
I have added the new HydrationIntakeUpdate entity to keep track of the changes in water intake throughout the day and compare daily water intake. I have also connected HydrationIntakeUpdate to HydrationMetric as a 1 to many relationship.
https://lucid.app/lucidchart/957d4674-0fd3-4443-8499-5378ffb069b8/edit?invitationId=inv_5c00b73e-6825-4a84-99ab-7eb4227b7666



Section 6: Physical ERD (revised)

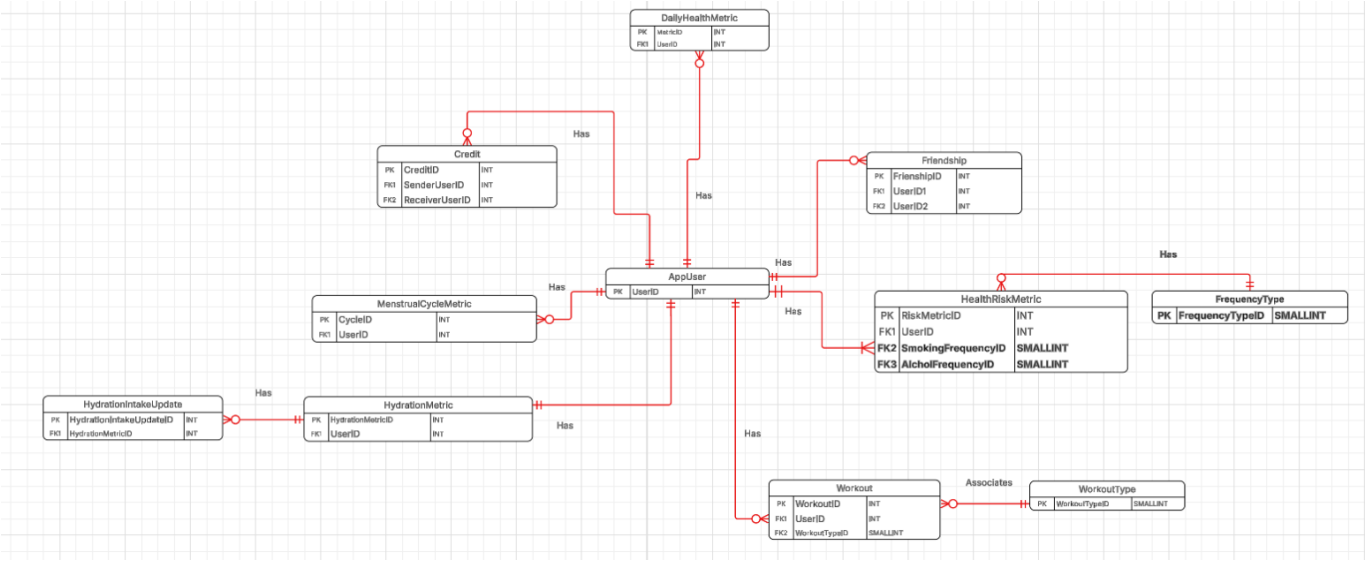
Include your history table in your physical ERD.

Relationship Classification and Associative Mapping

Based on the conceptual ERD, I have the following associative relationships:

- The User/DailyHealthMetric relationship is 1: M. Many daily health metrics log can be recorded for a user, but each daily health metric log can only belong to exactly 1 user.
- The User/User friends relationship is M: N. Each user can have many users as friends, and both entities are the user in this case, so the friends can have many users as their friends as well. It was necessary to create a bridging entity for this relationship. The entity was already created in the previous iteration and named as Friendship, which is the name of the relationship we consider in the real world. Both foreign keys in this bridging entity will be referencing the UserID from the User table, resulting in two 1:M relationships between Friendship and User.
- The User/HealthRiskMetric relationship is 1:M. Many health risk metrics logs can be recorded for a user, and each health risk metric log can only belong to 1 user.
- The User/Workout relationship is 1:M. Many workouts can be recorded for a user, and each workout can only belong to 1 user.
- The WorkoutType/Workout relationship is 1:M. Many workouts can belong to a workout type, and each workout can only fall under 1 workout type.
- The User/HydrationMetric relationship is 1:1. Each user can have 1 hydration metric log, and each hydration metric log can belong to only 1 user.
- The User/MenstrualCycleMetric relationship is 1:M. Each user can have many menstrual cycle metrics logs, and each menstrual cycle metric log can only belong to 1 user.
- The User/User credits relationship is M:N. Each user can have many credits from other users who are their friends. Both the sender user ID and the receiver user ID need to be recorded for each instance of a credit being given/received. Similar to the friendship relation, I used Credit itself as a bridging entity for this relationship. I have created 2 foreign keys, where one is for the sender and the other is for the receiver, and thus there are now two 1:M relationships between Credit and User entities.
- The GymClassType/GymClass relationship is 1:M. Each gym class type can be associated with many gym classes, and each gym class can belong to only 1 gym class type.
- The ClassType/GymClassType relationship is 1:M. Each class type can be associated with many gym class types, and each gym class type can only belong to only 1 class type.
- The Gym/GymClassType relationship is 1:M. Each gym can be present in many gym class types, and each gym class type can only be associated with 1 gym.
- The Shop/GiftCard relationship is 1:M. Each shop can be related to many gift cards, and each giftcard can belong to only 1 shop.
- The FrequencyType/HealthRiskMetric relationship is 1:M. Each frequency type can be associated with many health risk metric logs (with values across smoking frequency and alcohol frequency), and each health risk metric log can have only a single frequency type (for each smoking frequency and alcohol frequency).
- ***The HydrationMetric to HydrationIntakeUpdate relationship is 1:M. Each hydration metric log can have many hydration intake update logs, and each hydration intake update log can be associated to only 1 hydration metric.***

https://lucid.app/lucidchart/dbc802bd-4493-43a8-b779-0075cb86a817/edit?invitationId=inv_c8e48b3f-207a-4b8e-9aaa-3aa5a895a0a2



I followed the guideline for creating the best possible synthetic keys for all the tables. I also made the primary keys of datatype INT to hold many records. I added foreign keys to most of the entities on the many sides since they would need the reference to the unique foreign keys to identify the records. I added UserID in most tables as the foreign key since we will need to identify which user the record is for. Additionally, I created two foreign keys to capture each user in the two separate 1:M relationships on the bridging entities (Credit, Friendship).

I added FrequencyType as a new entity to avoid redundancy in the frequency values for smoking and alcohol frequencies. Extracting FrequencyType into its own table removes redundancy and satisfies the 3NF rule that every non-key attribute must depend directly on the primary key and nothing else. The frequencies had their own meaning that were not directly dependent on the PK, so creating a separate entity that stores the new meaning for frequencies is more preferrable.

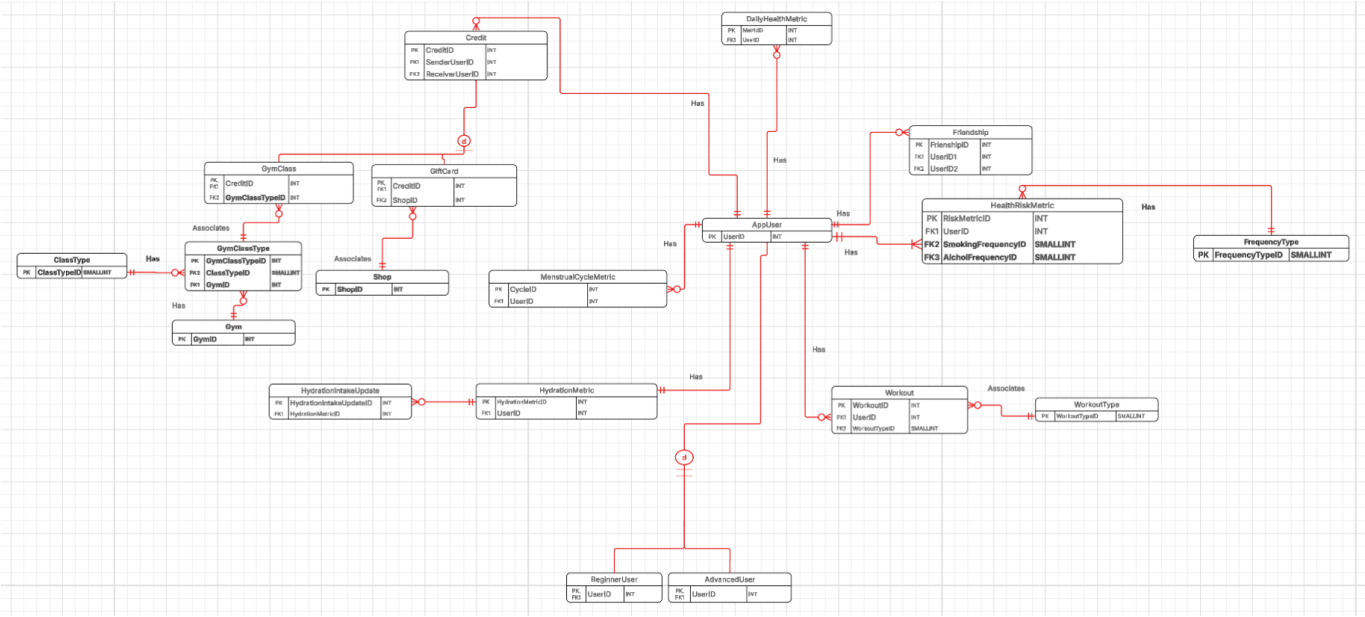
<u>Specialization-Generalization Mapping(updated)</u>

So, I have two specialization-generalization relationships in my conceptual ERD, one for the Credit entity and one for the User entity. Here is my DBMS physical ERD with these relationships mapped on it.
The additional entities under Credit are GymClass and GiftCard, each of which have a primary and foreign key of CreditID that reference the primary key of Credit.
The additional entities under User are BeginnerUser and AdvancedUser, which have primary and foreign keys of UserID that reference the primary key of User.
With these additional mappings, this DBMS physical now has all the relationships mentioned in the conceptual ERD.

I updated the PK and FK names for GymClassType, ClassType and Gymlass based on Iteration 4 feedback for clarity.

Update to the generalized ERD for Iteration 6-
I'm adding the new primary and foreign key setup for the new HydrationIntakeUpdate entity. I am making HydrationIntakeUpdateID which is a unique ID for the table as the primary key and I will use the HydrationMetricID referencing the HydrationMetric table's matching field as the foreign key.

https://lucid.app/lucidchart/c0f46ae1-5981-4b44-86de-1bbe871f4532/edit?view_items=Hd~76m2MZ8pS&page=0_0&invitationId=inv_6a361392-96d7-4860-b293-6f283f54a779



Final Physical ERD (updated)

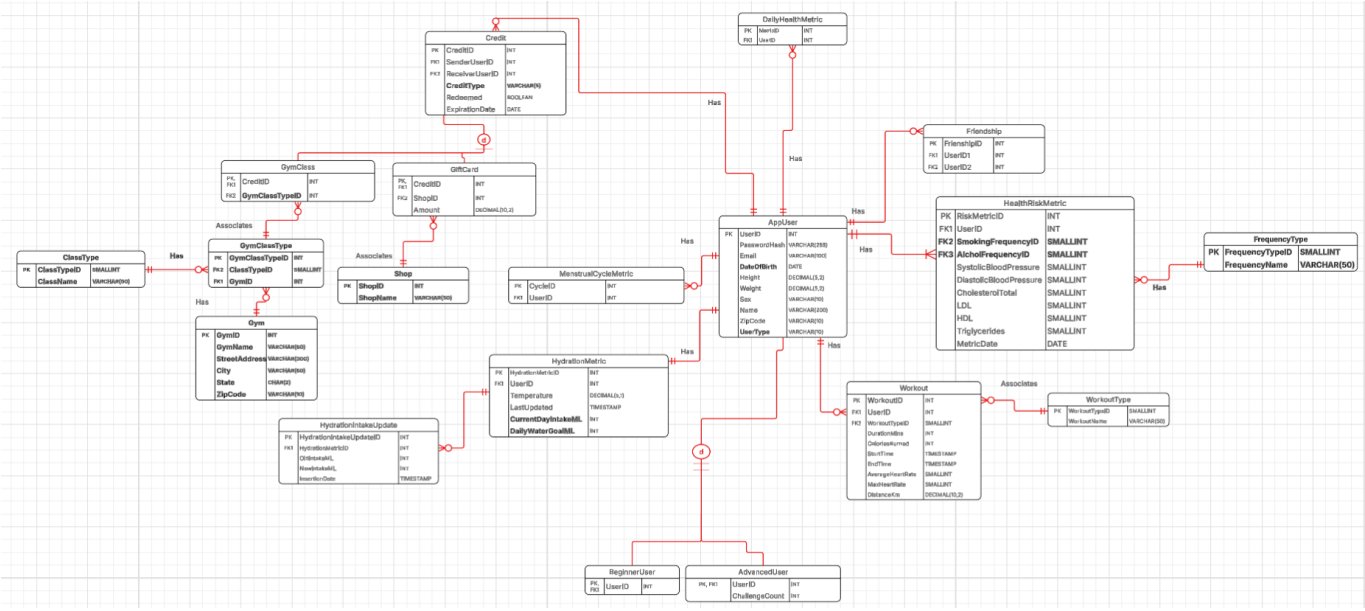
Iteration 4-
For the final ERD I chose to not expand the MenstrualCycleMetric, Friendship, DailyHealthMetric, BeginnerUser and AdvancedUser entities. I added attributes to all the other entities as explained in the attribute reasoning Section 7. The reasons for introducing the new entities GymClassType, ClassType, Gym, Shop, FrequencyType are described in the normalization reasoning Section 8. The conceptual ERD and final Physical ERD for the DBMS should now be in sync with the new structural database rules. I added new rules 11-15 in Section 4 above to reflect the new entities required after considering normalization.

Once I came up with the physical ERD, I asked AI to verify if all the entities were normalized according to BCNF and it explained how some entities only reached 3NF form but avoided unnecessary complexity while others do satisfy BCNF normalization. I mentioned the normalization reasoning in the section 8 below.

Iteration 5-
I updated the PK and FK names for GymClassType, ClassType and Gymlass based on Iteration 4 feedback for clarity.
I also added a new differentiating attribute to the AdvancedUser subtype based on (although I am not going to expand on the AppUser subtypes for the remainder of the project considering I expanded over 8 other entities as requested).

Iteration 6-
I'm adding the remaining attributes for the new HydrationIntakeUpdate entity. Now, we have fields for the old and new water intake amounts in ML, along with the record insertion datetime.

https://lucid.app/lucidchart/91bbc2b3-f4ee-447a-8466-cb19b1dba75e/edit?viewport_loc=-3566%2C-1316%2C4248%2C2141%2C0_0&invitationId=inv_c48501ff-431c-4782-be19-17676bcf4b2b



Section 7: Attribute Reasoning(revised)

Include your attribute reasoning as a reference (also update it if necessary).

I have created a table showing my attributes and their datatypes. I have also added the reasoning and example data in the table below. I chose to create attributes for these specific entities – User, HealthRiskMetric, FrequencyType, Workout, WorkoutType, HydrationMetric, Credit, GymClass, GiftCard, GymClassType, Shop, Gym, ClassType.
With the help of AI, I confirmed some of these datatypes, especially the use of SMALLINT was suggested by AI for fields that don't need huge amount of number ranges. I also asked for info on the real-world ranges of health metrics like cholesterol levels, heart rate and blood pressure. The attributes and datatypes in black were recommended by AI.

Update on Iteration 5: I updated the PK and FK names for GymClassType, ClassType and Gymlass based on Iteration 4 feedback for clarity.

Update on Iteration 6: I added the attributes and reasonings for all the new attributes of the HydrationIntakeUpdate entity.

Table	Attribute	Datatype	Reasoning	Example Data
AppUser	UserID	INT	Every new user will get a unique ID, so I want to allow many users to join with this datatype(INT) that can support millions of users upto 2,147,483,647. INT is also used because we need whole numbers for IDs, and it supports auto-incrementing primary keys.	12009

AppUser	PasswordHash	VARCHAR(255)	We need to store the user's password hash for authentication when they login. This version of the password will be encoded through a hashing algorithm so they can produce really long hashes. Thus, using 255 characters allows the required flexibility.	$2CSb$12$KIXJ3efQVn3W2C2jbn9kMx
AppUser	Email	VARCHAR(100)	With the need to form unique email addresses when signing up for popular email accounts like under gmail, some users could have really long email addresses. We will need to user's record email address for login authentication. To accomodate for longer email addresses, I am extending the character length.	snehamaharjan909501@gmail.com
AppUser	DateOfBirth	DATE	We need user's DOB to keep track of user's age. Age is generated from DOB. We can use age for generating better workout recommendations and to train machine learning models to create warnings for risk of diseases like cholesterol and stroke. DATE is the standard SQL datatype for storing dates such as birthdates.	2024-10-05
AppUser	Height	DECIMAL(5,2)	The maximum height this would allow is 999.99 cm which is reasonable since the maximum human height would be around 300 cm. We will use the user's height to determine BMI and any health risks.	190.01
AppUser	Weight	DECIMAL(5,2)	The maximum weight this would allow is 999.99 kg which can accommodate all realistic weights for humans. We will use the user's weight to determine BMI and any health risks.	170.58
AppUser	Sex	VARCHAR(10)	For the biological sex assigned at birth, we can have Male, Female, Intersex as the respective values. Thus, 10 characters is enough to store the sex. We maily need sex for differentiating if the user will require menstrual cycle based workout recommendations and for additional machine learning data to create warnings for health risks based on sex.	Female
AppUser	Name	VARCHAR(200)	Name is for the user's actual name which will be displayed in their profile for their friends to reference and will be needed for billing information and shipping information too. Based on people's family names, marital status or personal preferences, users can have short to very long names. 200 characters should give enough space for log user names.	Bridget Smith
AppUser	Zipcode	VARCHAR(10)	We need the zip code to know the user's current general location for hydration reminders and this can also be used for billing purposes later on. With 10 characters, we can accommodate the 5-digit base zip code, hyphen, 4-digit extension format which helps mailing services to do their jobs faster. VARCHAR also retains the leading 0 in the zip code.	02115-6701
AppUser	UserType	VARCHAR(10)	I am adding this new field as a subtype discriminator so we know what type of user	Beginner

			the record is for. The options can be either Beginner or Advanced. We need to know what the sort of user each person is to determine what app features we give them access to. Only advanced users can join challenges. 10 characters is enough for the options we have for this field.	
Workout	WorkoutID	INT	We need to have a unique identifier for each new workout. I want to support large number of workouts for our users with this datatype(INT) that can support millions of workouts up to 2,147,483,647. INT is also used because we need whole numbers for IDs and it supports auto-incrementing primary keys.	1029314
Workout	UserID	INT	This field is the foreign key, referencing the User table's primary key. Each workout recorded needs to be associated to a specific user so we need to store User ID which is originally an INT.	12009
Workout	WorkoutTypeID	INT	This field refers to the ID for the specific type of workout the user performed. It is a foreign key pointing to the Workout Type table's primary key. All the unique workouts will be assigned a type ID through this reference. INT is the best datatype to be used as an identifier to support large amount of unique records that are whole numbers.	10
Workout	DurationMins	INT	We need to record the duration of the workout to calculate the calories burned and time spent working out by the user for other metrics/calculations. We store this value in terms of minutes. If users pause and continue a long workout lasting multiple days the we need to use INT to handle larger range of values.	50
Workout	CaloriesBurned	INT	We need to store the calories burned per workout since this is the main motive to do a workout. Users will want to know how effective of a workout they did through this metric. INT allows us to store large values that are whole numbers if users take pauses and create a single long workout throughout multiple days.	500
Workout	StartTime	TIMESTAMP	The time the workout started is important to be noted to get the total workout duration. TIMESTAMP is used for saving time instances.	2025-04-01 12:30:00
Workout	EndTime	TIMESTAMP	The time the workout ended is important to be noted to get the total workout duration in respect to the start time. TIMESTAMP is used for saving time instances.	2025-04-01 14:30:00
Workout	**AverageHeartRate**	**SMALLINT**	The average heartrate of the user during the specific workout is an important health metric. It can show abnormalities in heart health if present so we record this metric in each workout. Usually during moderate exercises, the average heart rate lies between 120-160 bpm.	140

			SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	
Workout	**MaxHeartRate**	**SMALLINT**	The maximum heartrate of the user during the specific workout is another important health metric. With age, the maximum heart rate during intense workouts drops. For healthy young adults, the normal maximum heart rate is around 180-200 bpm. Abnormally high heart rates during workouts can indicate arrhythmias, underlying heart disease and other health conditions. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	199
Workout	DistanceKm	DECIMAL(10,2)	The distance travelled during the workout is important for the user to know how much they walked, and it is used to estimate the calories burned during the workout. We want to be able to store large distance values and have decimal precision so I am allowing 10 digits in total with 2 decimal places.	5.17
WorkoutType	**WorkoutTypeID**	**SMALLINT**	This identifier represents the type of workout performed such as swimming, running, jogging, Pilates and so on. Each workout type gets their own unique ID. SMALLINT is used for the datatype since the value is going to be a whole number, and it handles values up to 32,767 which is enough for our use case.	11
WorkoutType	WorkoutName	VARCHAR(50)	This field stores the actual name of the workout, so it needs to be broad enough to accommodate all the different workout names. 50 characters should be enough to store all the unique workout names	Swimming
HealthRiskMetric	RiskMetricID	INT	Each user will have multiple health risk metric logs recorded after each health checkup. The unique identifier for each health risk metric log should be able to expand for millions of records that are whole numbers so INT is a good datatype.	105
HealthRiskMetric	UserID	INT	This field is the foreign key, referencing the User table's primary key. Each workout recorded needs to be associated to a specific user, so we need to store User ID which is originally an INT.	12009
HealthRiskMetric	SmokingFrequencyID	**SMALLINT**	This field is a foreign key reference to the FrequencyTypeID field in the FrequencyType table. We reference the ID of the frequency name that we want to specify for the user. The frequency names are of types- Never, Occasionally, Weekly, Daily, Frequently. This value will be helpful for machine learning risk analysis of patient's health. SMALLINT is used for the datatype since the value is going to be a whole number and it	1

			handles values up to 32,767 which is enough for our use case. T	
HealthRiskMetric	AlcoholFrequencyID	**SMALLINT**	This field is a foreign key reference to the FrequencyTypeID field in the FrequencyType table. We reference the ID of the frequency name that we want to specify for the user. The frequency names are of types- Never, Occasionally, Weekly, Daily, Frequently. This value will be helpful for machine learning risk analysis of patient's health. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	2
HealthRiskMetric	SystolicBloodPressure	**SMALLINT**	The normal human range for systolic BP is 40-300 mmHg as seen in clinical monitors. We want to record the bloodpressure for machine learning to see if there is any health risk. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	200
HealthRiskMetric	DiastolicBloodPressure	**SMALLINT**	The normal human range for diastolic BP is 30-200 mmHg.We want to record the bloodpressure for machine learning to see if there is any health risk. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	180
HealthRiskMetric	CholesterolTotal	**SMALLINT**	The maximum cholesterol levels for unhealthy patients can go upto 500-600 mg/dL. We want to record this value from every checkup to use in machine learning risk analysis of patient's health. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	400
HealthRiskMetric	LDL	**SMALLINT**	The LDL/bad cholesterol can go up to 300-400 mg/dL. This value will be helpful for machine learning risk analysis of patient's health. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	200
HealthRiskMetric	HDL	**SMALLINT**	The high of HDL/good cholesterol is around 90-100 mg/dL. This value will be helpful for machine learning risk analysis of patient's health. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	90
HealthRiskMetric	Triglycerides	**SMALLINT**	The rare extreme range for triglycerides is around 2000 mg/dL. This value will be helpful for machine learning risk analysis of patient's health. SMALLINT is used for the datatype since the value is going to be a whole number and it handles values up to 32,767 which is enough for our use case.	1800

HealthRiskMetric	MetricDate	DATE	The date of the annual or regular checkup when these metrics were measured is recorded for knowing how relevant the record is. DATE datatype is the best way to save dates.	2024-10-05
FrequencyType	FrequencyTypeID	SMALLINT	This field will be the unique identifier for the frequency types. There is only a select list of options for frequency- Never, Occasionally, Weekly, Daily, Frequently. Thus, SMALLINT's range is enough for this use case.	1
FrequencyType	FrequencyName	VARCHAR(50)	This field holds the actual values of frequencies, namely Never, Occasionally, Weekly, Daily, Frequently, so 50 characters is more than enough to store our options.	Never
HydrationMetric	HydrationMetricID	INT	There will be 1 current hydration metric log for each user so we need to accomodate large number of values for this ID. INT matches the datatype used for UserID so it should suffice the maximum log count for us.	20012
HydrationMetric	UserID	INT	This field is the foreign key, referencing the User table's primary key. Each hydration log needs to be associated to a specific user, so we need to store User ID which is originally an INT.	12009
HydrationMetric	Temperature	DECIMAL(5,1)	Based on US temperatures, we want to store 5 total digits, allowing 1 decimal place for the temperature in Fahrenheit. The temperature outdoors is used to determine the frequency of the hydration logs.	102.9
HydrationMetric	LastUpdated	TIMESTAMP	We want to keep track of when the temperature was last updated to stay up to date with the outdoor temperature. We also want to keep the current day's intake amount up to date to make sure the user is keeping up with hydration reminders. The TIMESTAMP is the standard datatype to store date and time records together.	2025-04-01 4:30:00
HydrationMetric	**CurrentDayIntakeML**	SMALLINT	We want to keep track of the user's current daily water intake in milliliters, which can be stored in whole numbers based on how drinking containers are designed, so SMALLINT is the appropriate field to use here which stores 32,767 ml maximum whichs is more than enough for human limits. The user needs to mark the reminder as DONE on the app to update this value throughout the day	500
HydrationMetric	**DailyWaterGoalML**	SMALLINT	We want to store the user's desired daily water goal in millileters to keep sending hydration reminders until the goal is met. SMALLINT is big enough since it stores 32,767 ml maximum which is more than enough for human limits. It is recommended to drink around 3000 ml of water per day, so SMALLINT will suffice.	2500
Credit	CreditID	INT	This identifier represents each unique credit transaction between 2 users who are friends. INT is able to store millions of records so it should be big enough for our case.	5001

Credit	SenderUserID	INT	This is the 1st foreign key reference to the user ID who sent the credit or reward. We have to match the user ID's original datatype which is INT.	12007
Credit	ReceiverUserID	INT	This is the 2nd foreign key reference to the user ID who received the credit or reward. We again have to use INT which is the original datatype in the User table.	11390
Credit	**CreditType**	**VARCHAR(5)**	**We need to keep track of what type of credit was involved in each transaction to store proper additional details. We need to store whether the type is 'Class' or 'Card' which can be a maximum of 5 characters, so VARCHAR is used in this case.**	**Class**
Credit	Redeemed	BOOLEAN	We want to keep track of whether the credit has already been used by receiver or not, so we can use values TRUE or FALSE to mark the redemption status.	TRUE
Credit	ExpirationDate	DATE	We need to keep track of when the credit will expire so we are storing the exact date with the DATE datatype.	2024-10-25
GymClass	CreditID	INT	This field is the primary key for the GymClass subtype and is also the foreign key reference to the Credit supertype. It has to be the datatype of the original PK which was set as INT to handle large number of credits.	1234
GymClass	**GymClassTypeID**	INT	This field is a reference to the GymClassType that stores the information about the class and the gym the class is associated to. We need to be able to accommodate a large number of combinations of gyms with all of the classes they offer, so we use INT for this field.	9001
GymClassType	**GymClassTypeID**	INT	This primary key identifies each unique combination of a class offered at a specific gym. We need to be able to accommodate a large number of combinations of gyms with all of the classes they offer, so we use INT for this field.	5002
GymClassType	**ClassTypeID**	SMALLINT	This foreign key relates to the **ClassTypeID** primary key of ClassType that stores the unique mapping of each different type of class possible. There are only a select list of classes like Yoga, Pilates, Cardio, Boxing , etc that can be offered in gyms so SMALLINT can handle the list count.	102
GymClassType	GymID	INT	This field is a foreign key referring to the gym ID from the Gym entity. We need to keep track of what gym the class is associated to. INT is the original datatype of the primary key to accommodate the growing number of gyms.	100
ClassType	**ClassTypeID**	SMALLINT	This field is the primary key for the entity that identifies the unique class that a gym can offer. There are only a select list of classes like Yoga, Pilates, Cardio, Boxing , etc that can be offered in gyms so SMALLINT can handle the list count.	102

ClassType	ClassName	VARCHAR(50)	This field maps to the main names of the gym classes that can be offered by gyms like Yoga, Pilates, etc. 50 characters should be enough to store all the gym name options.	Pilates
Gym	GymID	INT	This field stores the gym ID for the gym that the class is hosted at. Since we want to accommodate all old and new gyms that are built in the future, we will need to store records for a large number of gyms. INT can hold millions of records so we use this datatype.	100
Gym	GymName	VARCHAR(50)	This field is used to store the gym's name which is important to know where the credit can be used. 50 characters should be enough to store the gym names.	Livfit
Gym	StreetAddress	VARCHAR(300)	We need to store the street address of the gym to give user's proper navigation to reach the gym. 300 characters should be enough to store long addresses. We also need a datatype that stores letters, numbers, spaces, hyphens, slashes and other symbols in addresses so VARCHAR works out great here.	500 Main St. Unit 5
Gym	City	VARCHAR(50)	The city name for the gym's address is stored here. City names are not usually too long so 50 characters should be long enough.	Arlington
Gym	State	CHAR(2)	The state for the gym's address is stored here. We are storing the US state abbreviations so 2 characters should handle all the state combinations.	MA
Gym	Zipcode	VARCHAR(10)	We need the zip code to know the gym's exact location. With 10 characters, we can accommodate the 5-digit base zip code, hyphen, 4-digit extension format. VARCHAR also retains the leading 0 in the zipcode.	02115-6701
GiftCard	CreditID	INT	This field is the primary key for the GiftCard subtype and is also the foreign key reference to the Credit supertype. It has to be the datatype of the original PK which was set as INT to handle large number of credits.	78800
GiftCard	ShopID	INT	This is the primary key of the Shop entity so this is a reference to that field. It identifies the shop where the giftcard can be used. We want to accommodate all the online and physical shops in America so INT is large enough for this use case since it supports millions of records.	12
GiftCard	Amount	DECIMAL(10,2)	We want to store the amount of credit on the giftcard. I am capping the maximum amount to be 10 digits long including 2 decimal places since people normally gift small cash quantities as giftcards. $99,999,999.99 is the maximum amount allowed which is more than enough for this real world scenario.	50.98
Shop	ShopID	INT	This field is the primary key of the Shop entity. It identifies the shop where the giftcard can be used. We need to support all the millions of online	12

			and physical stores, so INT will be large enough to handle this use case.	
Shop	ShopName	VARCHAR(50)	This field stores the shop's actual name that creates this Shop entity. 50 characters is usually long enough for any shop's name and to accommodate special characters we use VARCHAR.	Bryan's Candy Shop
HydrationIntakeUpdate	***HydrationIntakeUpdateID***	***INT***	***This field is the primary key of the HydrationIntakeUpdate entity. We expect to have multiple hydration update records for each user in a day, so we need to be able to support a lot of records which INT can handle.***	***200***
HydrationIntakeUpdate	***HydrationMetricID***	***INT***	***This field references the original HydrationMetricID from the HydrationMetric entity.*** ***There will be 1 current hydration metric log for each user so we needed to accommodate large number of values for this ID.***	***105***
HydrationIntakeUpdate	***OldIntakeML***	***INT***	***This field stores the old value of the user's current water intake amount in ML before the HydrationMetric record gets updated. We are using INT which matches the datatype of the same field (CurrentDayIntakeML) from the HydrationMetric table.***	***75***
HydrationIntakeUpdate	***NewIntakeML***	***INT***	***This field stores the new value of the user's current water intake amount in ML before the HydrationMetric record gets updated. We are using INT which matches the datatype of the same field (CurrentDayIntakeML) from the HydrationMetric table.***	***100***
HydrationIntakeUpdate	***InsertionDate***	***TIMESTAMP***	***This field stores the date and timestamp of when the new row in the HydrationIntakeUpdate history table gets inserted after the hydration metric record in the HydrationMetric entity gets updated and.***	***2025-12-05 12:30:00***

Section 8: Normalization Reasoning

Normalize your entities in your DBMS physical ERD and explain what you chose to normalize and why, and which entities did not need normalization. Update your structural database rules, conceptual ERD, and physical ERD to match.

I saw that the Gym Class Name and Gym Name could be repeated in instances of each gym class credit. Thus, I added 2 additional entities for the gym class setup which are the GymClassType and Gym. GymClassType now holds the combination of the class type ID and the gym ID. To support this combination, I created a new table that maps the gym ID to all the additional information about the gym. I also created another table that maps the class type ID to the actual class name. By having this setup, we don't repeat the gym name and class name, or the combination of the specific gym class for each credit.

I also saw that the ShopName related to the gift card can be repeated, so I created a separate Shop entity for the mapping of the shop name so that only the ShopID can be referenced and nothing else gets repeated in the GiftCard table.

I also introduced a Frequency Type table to hold the mapping of unique frequencies to make sure the frequency values weren't directly repeated in the HealthRiskMetric table. Both SmokingFrequencyID and AlcholFrequencyID are foreign keys referencing their own FrequencyTypeIDs to match the frequency at which users drink alcohol and smoke.

The database schema is fully normalized to Third Normal Form (3NF). All tables contain atomic attributes, and non-key fields depend solely on their respective primary keys, eliminating both partial and transitive dependencies. Repeating categories such as class types, workout types, gift card types, and healthrisk frequencies were extracted into separate lookup entities, which further reduces redundancy and maintains consistency across the schema. These lookup entities, along with associative tables such as GymClassType, also satisfy BCNF, since their determinants are always single attribute primary keys.

There was an intentional exception to full BCNF is the **Address** attributes within the User table. Because ZipCode functionally determines City and State, achieving BCNF would require decomposing these into a separate ZipCode reference table. In BCNF, every determinant must be a candidate key so the zipcode violates the rule. This level of decomposition was avoided for practicality and ease of use, while still maintaining 3NF. We are also not storing additional pieces of the address in the User table because that information is not relevant for the entities we have currently set up. The full user residential and billing address would need to be saved in a separate Address or Billing table later for the full-fledged implementation of the project.

Lastly, even for the Address fields in the Gym table, the same logic for zipcode mentioned above applies, so to make it BCNF I would need to create a separate zipcode table that maps cities and states with the zipcodes as primary keys. Additionally, in general, there is redundancy in cities, states in the main table and we would have to create further City and State entities holding city IDs and state IDs with mappings to the actual city names and state names. I could have created city and state and zipcode entities to avoid redundancy, but I chose not to normalize the Address fields in the Gym entity to full BCNF because it is not necessary for my database and just adds extra complexity.

Even though ZipCode functionally determines City and State, City and State do not depend on GymID through ZipCode. They depend directly on GymID. The relationship ZipCode → City, State is external real-world knowledge, not part of how the table is structured, so the Gym entity still meets 3NF.

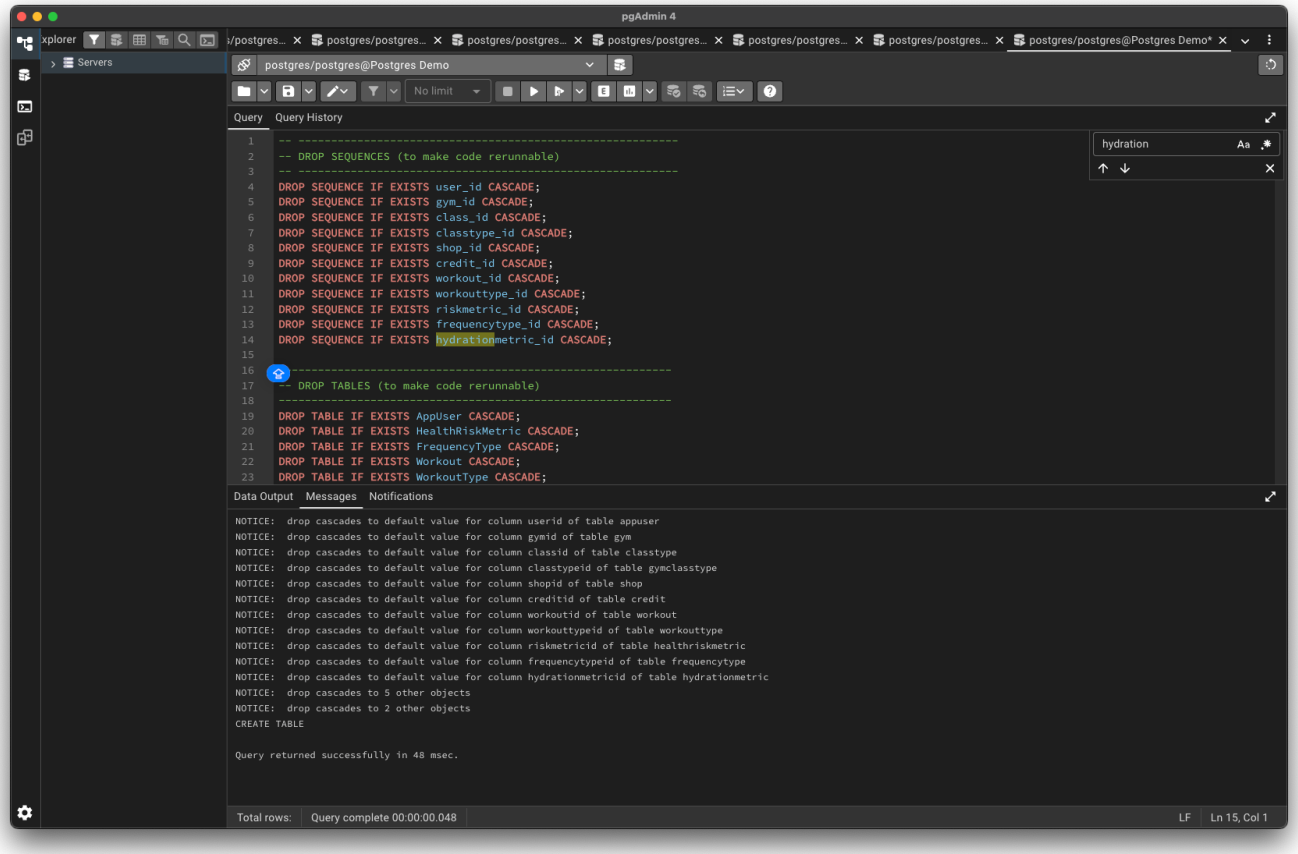
The rest of the entities that have attributes written out should be meeting the BCNF standards.

Section 9: SQL Script (revised)

Add your history table, trigger, and data queries to your SQL script.

I have attached the updated SQL script to the project submission zip folder. I also had to make some corrections to the PK and FK of the GymClassType and ClassType tables based on the feedback from Iteration4. I have added the new sections in the SQL script and you can see screenshots of each section below as well.

I have added the new history table that records changes to the HydrationMetric entries which is controlled by a trigger explained more in Section 13. I have also included the new data visualization queries explained more in Section 14.



Section 10: Transactions

Select two of your use cases that involve adding data to the database, create parameterized stored procedures that implement the transactions steps, and execute the stored procedures in the context of a transaction. All inserts should use sequences throughout for the primary and foreign key values. No values should be hardcoded. Provide screenshots of their creation and execution and update your SQL script to contain them.

I will be using this use case-

3. Workout Information Transfer Use Case

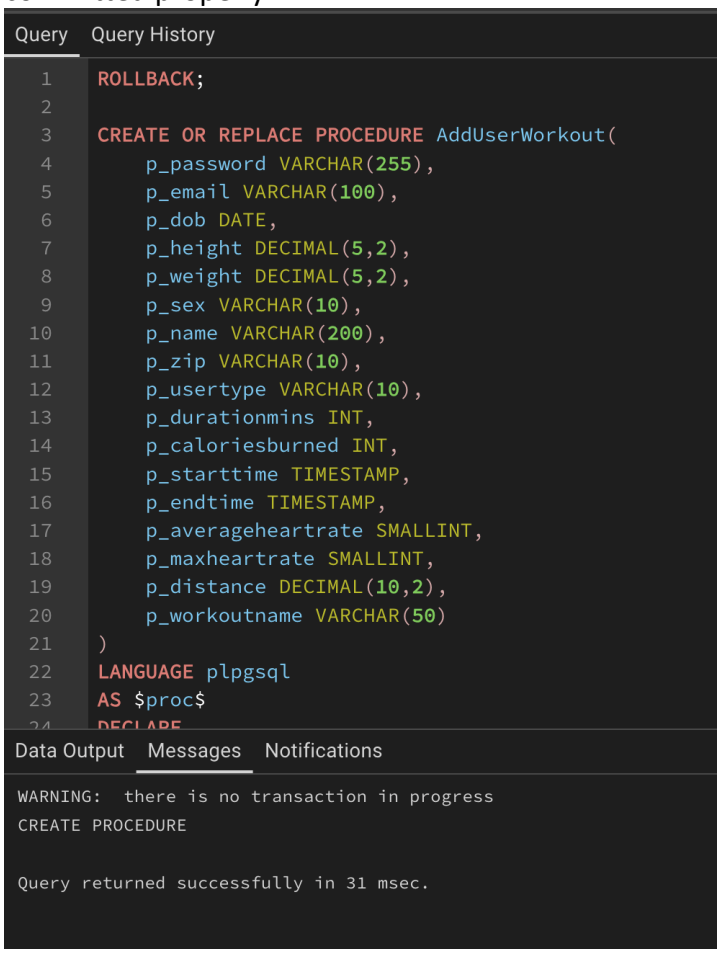
-The Fitness tracker app notices a workout once it has been completed and syncs to the smartwatch to extract the data.
-User gets request to transfer the workout information and must accept it on the app.
-User is also prompted to manually select the workout type for detailed summary based on the type.
-The new workout instance is recorded in the database.

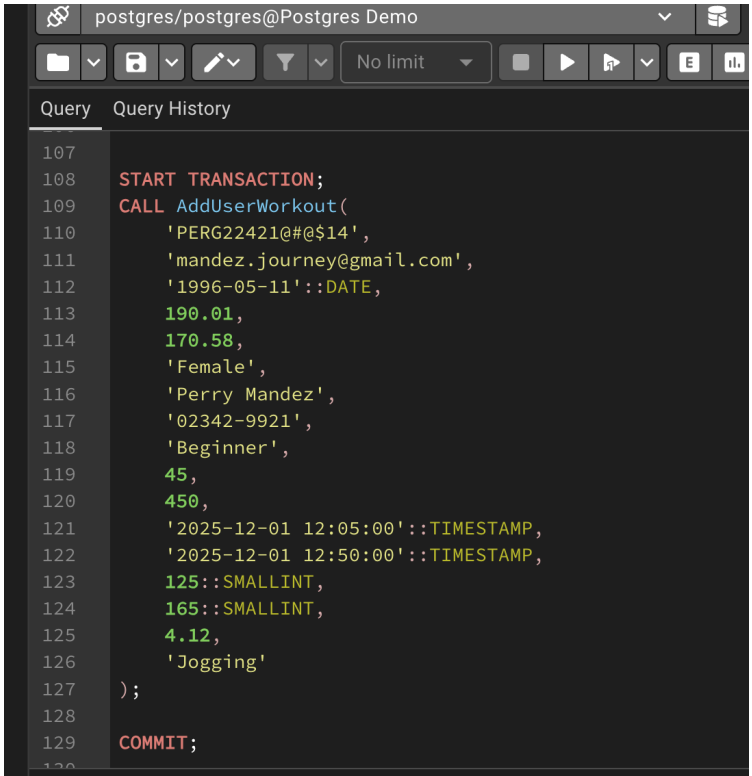
Considering the above use case, I implemented the scenario where a user completes a workout on their smartwatch and the fitness app transfers those workout details onto the app. During this process-

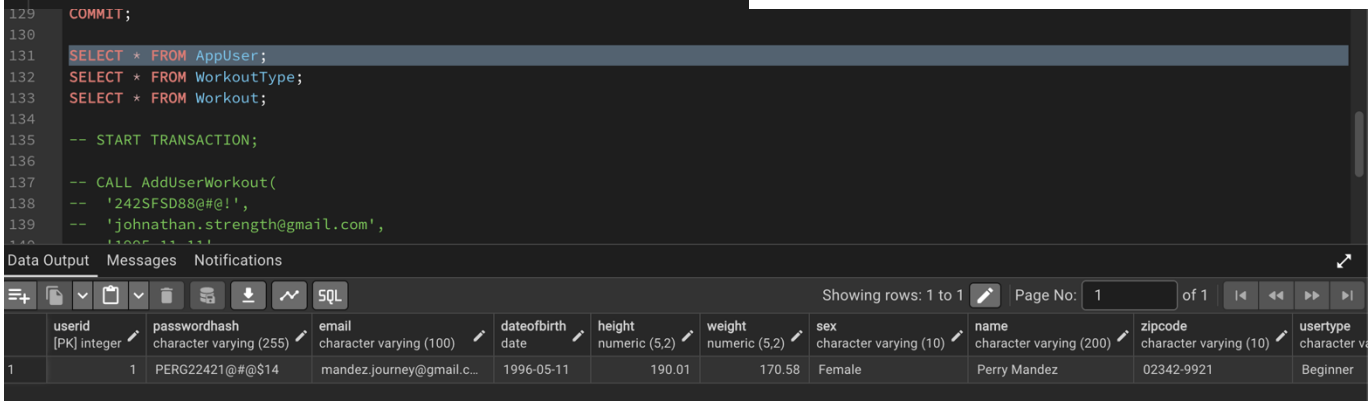
1. A new user gets created as part of the onboarding process.

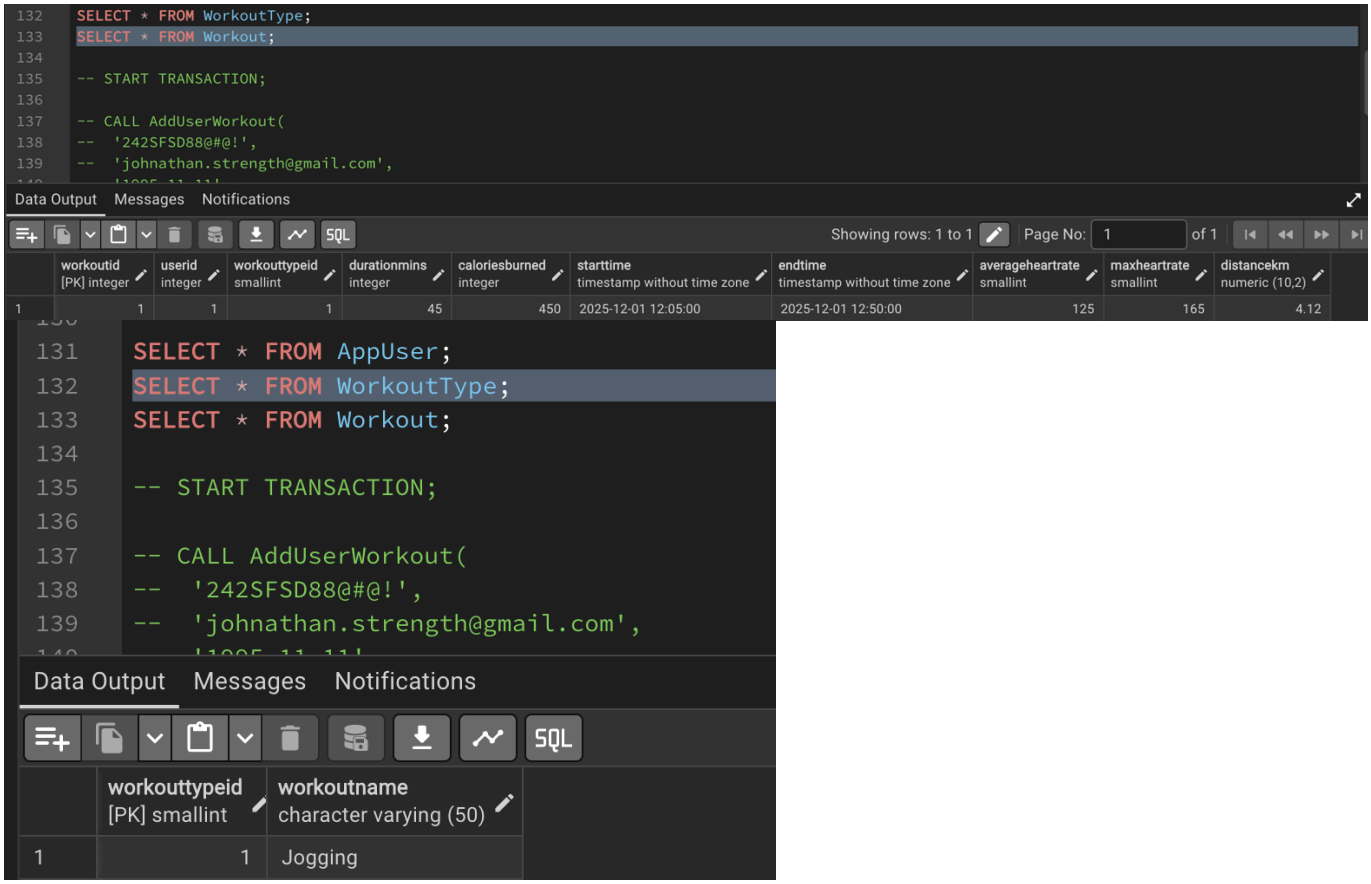
2. A new workout type is created if it did not exist before.
3. A new workout entry is added for the user's workout instance.

Thus, we need to insert into 3 tables, namely AppUser, WorkoutType and Workout for this use case. I did not choose to fully expand and create the User's subtypes with all the attributes considering I already expanded 8 other entities as required in the previous sections. I created a user's workout scenario with realistic information below. I have added screenshots of the sample transaction that has the nested stored procedure call for AddUserWorkout within it to make sure the transaction is committed properly.




postgres/postgres@Postgres Demo
No limit
Query
Query History
START TRANSACTION;
CALL AddUserWorkout(
'PERG22421@#@$14',
'mandez.journey@gmail.com',
'1996-05-11'::DATE,
190.01,
170.58,
'Female',
'Perry Mandez',
'02342-9921',
'Beginner',
45,
450,
'2025-12-01 12:05:00'::TIMESTAMP,
'2025-12-01 12:50:00'::TIMESTAMP,
125::SMALLINT,
165::SMALLINT,
4.12,
'Jogging'
);
COMMIT;
Data Output
Messages
Notifications
COMMIT
Query returned successfully in 29 msec.


COMMIT;
SELECT * FROM AppUser;
SELECT * FROM WorkoutType;
SELECT * FROM Workout;
-- START TRANSACTION;
-- CALL AddUserWorkout(
-- '242SFSD88@#@!',
-- 'johnathan.strength@gmail.com',
Data Output
Messages
Notifications
SQL
Showing rows: 1 to 1
Page No: 1
of 1
userid [PK] integer
passwordhash character varying (255)
email character varying (100)
dateofbirth date
height numeric (5,2)
weight numeric (5,2)
sex character varying (10)
name character varying (200)
zipcode character varying (10)
usertype character v
1 1 PERG22421@#@$14 mandez.journey@gmail.c... 1996-05-11 190.01 170.58 Female Perry Mandez 02342-9921 Beginner



For the second transaction, I am going to use the Friends Credit Rewards use case.

4. Friends Credit Rewards Use Case (New)

-A friend sends a request to her friend on the Fitness tracker app.
-Both friends can view each other's activities and want to send credit to each other after going on a group hike.
-The reward can be a gym class type or a gift card type or other new reward type, which must be selected by the user.
-The new credits being sent/received are recorded in the database. Both the sender and receiver are recorded.

With the current setup of entities and attributes, we will need to perform the following steps when friends send each other credits as rewards-

1. Check if sender and receiver users exist yet, and if they don't, create records for both users.
2. Create a Credit record for the transaction between sender and receiver.
3. Based on whether the credit is a gym class or a gift card reward, a row must also be inserted into the GymClass or GiftCard tables.
4. A GymClassType entry, along with the matching ClassType and Gym should be set up for the Credit if it is a GymClass credit.
5. A Shop entry should be set up for the Credit if it is a GiftCard credit.

I created a credit transaction scenario between two users with realistic information below. I have added screenshots of the sample transaction that has the nested stored procedure call within it to make sure the transaction is committed properly. On running the stored procedure AddCreditReward, I created this credit record along with the related gym class records. I created the gym, class, then the record between the gym and class for gymclasstype. I then created the record for the specific credit based on it being a gymclass subtype.

```
-- Executing the sproc
START TRANSACTION;

CALL AddCreditReward(
    108,
    104,
    'Class',
    FALSE,
    '2025-12-01'::DATE,
    'Yoga',
    'Bars and Bells',
    '239 Brenton St',
    'Franconia',
    'NH',
    '03213-5352',
    NULL,
    NULL,
    'REGE231@#@$14',
    'harrington.smiles@gmail.com',
    '1991-05-11'::DATE,
    190.01,
    170.58,
    'Female',
    'Emilia Harrington',
    '02342-9921',
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 39 msec.
```

```
SELECT * FROM Credit;
SELECT * FROM GymClass;
SELECT * FROM GymClassType;
SELECT * FROM ClassType;
SELECT * FROM Gym;
SELECT * FROM GiftCard;
SELECT * FROM Shop;
```

Data Output | Messages | Notifications

	creditid [PK] integer	senderuserid integer	receiveruserid integer	credittype character varying (5)	redeemed boolean	expirationdate date
1	1	2	3	Class	false	2025-12-01

```
233
234  SELECT * FROM Credit;
235  SELECT * FROM GymClass;
236  SELECT * FROM GymClassType;
237  SELECT * FROM ClassType;
238  SELECT * FROM Gym;
239  SELECT * FROM GiftCard;
240  SELECT * FROM Shop;
241
242
```

Data Output | Messages | Notifications

	classtypeid [PK] smallint	classname character varying (50)
1	1	Yoga

```
238  SELECT * FROM Gym;
239  SELECT * FROM GiftCard;
240  SELECT * FROM Shop;
241
242
```

Data Output | Messages | Notifications

Showing rows: 1

	gymid [PK] integer	gymname character varying (50)	streetaddress character varying (300)	city character varying (50)	state character (2)	zipcode character varying (10)
1	1	Bars and Bells	239 Brenton St	Franconia	NH	03213-5352

```
234  SELECT * FROM Credit;
235  SELECT * FROM GymClass;
236  SELECT * FROM GymClassType;
237  SELECT * FROM ClassType;
238  SELECT * FROM Gym;
239  SELECT * FROM GiftCard;
240  SELECT * FROM Shop;
241
242
```

Data Output | Messages | Notifications

	gymclasstypeid [PK] integer	classtypeid smallint	gymid integer
1	1	1	1

```
235  SELECT * FROM GymClass;
236  SELECT * FROM GymClassType;
237  SELECT * FROM ClassType;
238  SELECT * FROM Gym;
239  SELECT * FROM GiftCard;
240  SELECT * FROM Shop;
241
242
```

Data Output | Messages | Notifications

	creditid [PK] integer	gymclasstypeid integer
1	1	1

Section 11: Questions and Queries

Create three questions useful to your organization or application, then write queries to address the questions. Make sure to follow the specific requirements given in the walkthrough. Capture screenshots. Make sure to update the SQL script to contain these.

First Query

This query should retrieve information from at least four tables joined by associative relationships. Ensure that the question driving this query is applicable and useful for the intended use of the database.

Question 1: What kind of users have higher levels of Systolic Blood Pressure and Diastolic Blood Pressure?

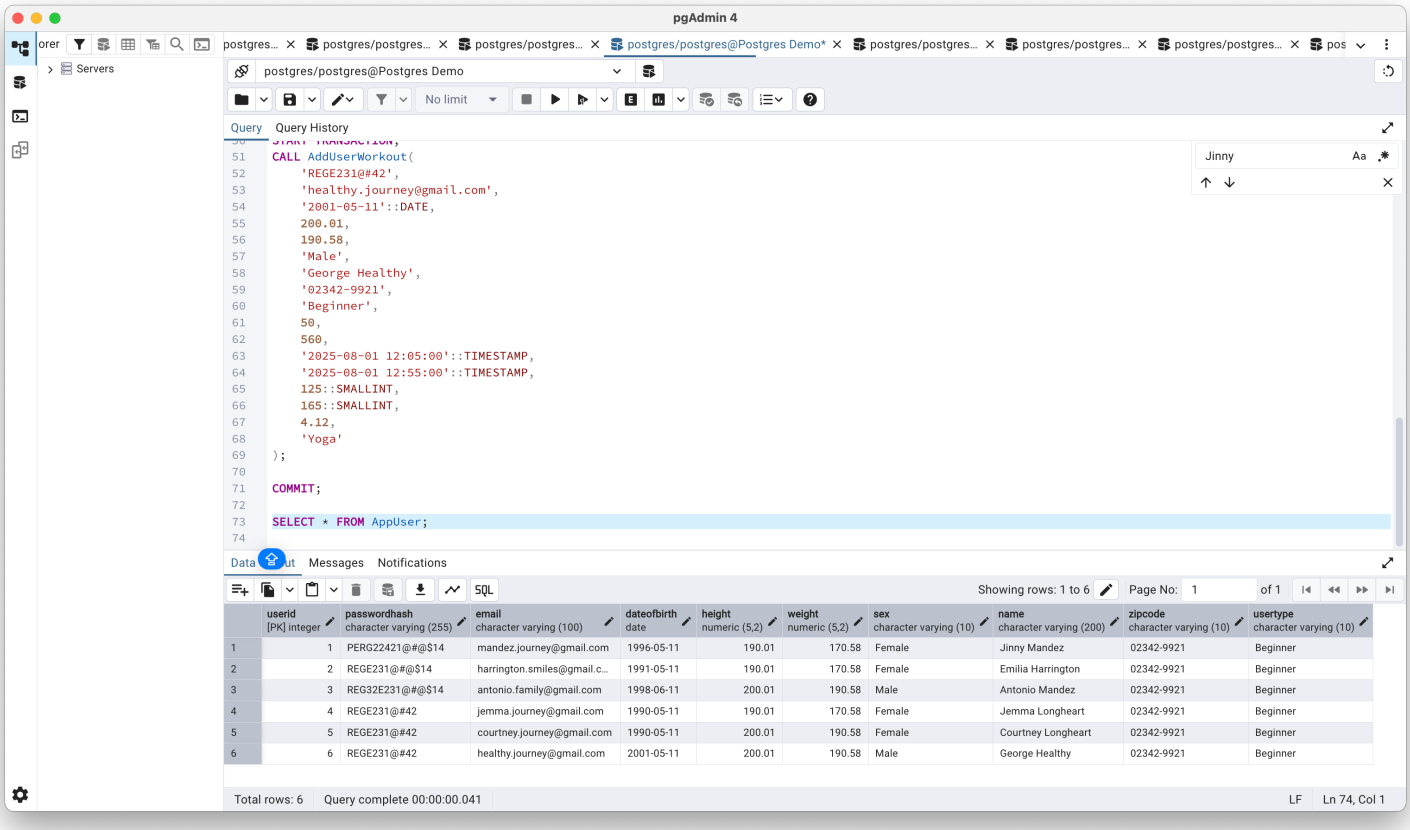
We want to know which users are at risk of Stage 1 Hypertension. When Systolic BP is between the range 130-139 mmHg and Diastolic BP is between the range 80-89 mmHg, it indicates that user has Stage 1 Hypertension. It increases the risk of having heart disease, stroke and kidney stress. The body is working harder to push blood through vessels. This condition can be related to stress, poor sleep, medications, genetics or other health issues, so we want to warn users that they need to take care of their lifestyle choices. We can send additional health reminders to sleep on time and do more meditation accordingly. We also want to see if people who do more cardio workouts have better blood pressure meaning better health.

I needed to create some new users and their workouts for this scenario using the AddUserWorkout stored procedure created above. I created multiple users that performed different workouts for more variety. Then I created the frequency type options for the healthriskmetric entry. I created data for users that have both healthy and unhealthy ranges of blood pressures. My query shows users that have higher blood pressure levels, and it shows what workouts lead to such health situations. Trying to match the real world, where users who perform more cardio workouts have better blood pressure levels, in my data results there are 2 users who perform jogging and yoga that have moderate to high levels of blood pressure signifying the risk of having Stage 1 Hypertension. The user who performed running workouts had lower blood pressure, so this shows the need to do more cardio for better overall health.

```
51
52  START TRANSACTION;
53  CALL AddUserWorkout(
54      'REGE231@#42',
55      'healthy.journey@gmail.com',
56      '2001-05-11'::DATE,
57      200.01,
58      190.58,
59      'Male',
60      'George Healthy',
61      '02342-9921',
62      'Beginner',
63      50,
64      560,
65      '2025-08-01 12:05:00'::TIMESTAMP,
66      '2025-08-01 12:55:00'::TIMESTAMP,
67      125::SMALLINT,
68      165::SMALLINT,
69      4.12,
70      'Yoga'
71  );
72
73  COMMIT;
74
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 31 msec.
```



	userid [PK] integer	passwordhash character varying (255)	email character varying (100)	dateofbirth date	height numeric (5,2)	weight numeric (5,2)	sex character varying (10)	name character varying (200)	zipcode character varying (10)	usertype character varying (10)
1	1	PERG22421@#@$14	mandez.journey@gmail.com	1996-05-11	190.01	170.58	Female	Jinny Mandez	02342-9921	Beginner
2	2	REGE231@#@$14	harrington.smiles@gmail.c...	1991-05-11	190.01	170.58	Female	Emilia Harrington	02342-9921	Beginner
3	3	REG32E231@#@$14	antonio.family@gmail.com	1998-06-11	200.01	190.58	Male	Antonio Mandez	02342-9921	Beginner
4	4	REGE231@#42	jemma.journey@gmail.com	1990-05-11	190.01	170.58	Female	Jemma Longheart	02342-9921	Beginner
5	5	REGE231@#42	courtney.journey@gmail.com	1990-05-11	200.01	190.58	Female	Courtney Longheart	02342-9921	Beginner
6	6	REGE231@#42	healthy.journey@gmail.com	2001-05-11	200.01	190.58	Male	George Healthy	02342-9921	Beginner

```sql
INSERT INTO FrequencyType (FrequencyName)
VALUES
    ('Never'),
    ('Occasionally'),
    ('Weekly'),
    ('Daily'),
    ('Frequently');


INSERT INTO HealthRiskMetric (
    UserID,
    SmokingFrequencyID,
    AlcoholFrequencyID,
    SystolicBloodPressure,
    DiastolicBloodPressure,
    CholesterolTotal,
    LDL,
    HDL,
    Triglycerides,
    MetricDate
)
VALUES
    -- healthy reading
    (2, 1, 2, 118, 76, 165, 90, 55, 120, NOW()),

    -- mildly elevated BP
    (5, 2, 3, 132, 84, 190, 120, 45, 160, NOW() - INTERVAL '30 days'),

    -- higher risk reading
    (6, 3, 4, 148, 92, 220, 150, 40, 200,NOW() - INTERVAL '60 days');
```

Data Output | Messages | Notifications

```
INSERT 0 3

Query returned successfully in 31 msec.
```

```sql
SELECT
    u.UserID,
    u.Name AS UserName,
    wt.WorkoutName,
    hm.SystolicBloodPressure,
    hm.DiastolicBloodPressure
FROM AppUser u
JOIN Workout w
    ON w.UserID = u.UserID
JOIN WorkoutType wt
    ON wt.WorkoutTypeID = w.WorkoutTypeID
JOIN HealthRiskMetric hm
    ON hm.UserID = u.UserID
WHERE hm.SystolicBloodPressure>=130
  AND hm.DiastolicBloodPressure>=80;
```

Data Output | Messages | Notifications

	userid integer	username character varying (200)	workoutname character varying (50)	systolicbloodpressure smallint	diastolicbloodpressure smallint
1	5	Courtney Longheart	Jogging	132	84
2	6	George Healthy	Yoga	148	92

Second Query

This query should retrieve information from the subtypes and supertype you have in your project, the entities that participate in the specialization-generalization relationship. The query should require joins to one or more of the subtypes, as well as the supertype, to fully address the question.
Ensure that the question driving this query is applicable and useful for the intended use of the database.

Question 2 - Which gym classes do each user have available through the credits gifted by their friends?

This question is very relevant to the fitness app since we allow users to gift credits in the form of subtypes gift cards or gym classes. The app should be able to display the gift cards that a user has available so they can use them as desired. For this use case, I am going to create some credit transactions among some users that we are considering as friends. I am creating a mix of class and card type credits to create some variety in the database. I am going to be using the AddCreditReward stored procedure to do the necessary insertions for credit setup.

Query Query History

```
80  CALL AddCreditReward(
81      12,
82      13,
83      'Class',
84      FALSE,
85      '2026-03-01'::DATE,
86      'Yoga',
87      'Minis Yoga Studio',
88      '21 Fullerton St',
89      'Beverly',
90      'MA',
91      '02313-5352',
92      NULL,
93      NULL,
94      'REGE111@#@$1314',
95      'handyman.smiles@gmail.com',
96      '2001-05-11'::DATE,
97      200.01,
98      180.58,
99      'Female',
100     'Bonita Frenza',
101     '02342-9921',
102     'Beginner',
103     'REG32E23111@#@$14',
104     'fitness.queen@gmail.com',
105     '1992-01-03'::DATE,
106     210.01,
107     180.58,
108     'Female',
109     'Samantha Medina',
110     '02342-9921',
111     'Beginner'
112 );
113
114 COMMIT;
115
116 SELECT * FROM Credit;
```

Data Output Messages Notifications

SQL

	creditid [PK] integer	senderuserid integer	receiveruserid integer	credittype character varying (5)	redeemed boolean	expirationdate date
1	1	2	3	Class	false	2025-12-01
2	2	7	8	Class	false	2025-12-01
3	3	9	10	Card	false	2025-12-12
4	4	11	12	Class	false	2026-03-01

Query | Query History

```sql
SELECT
    u.UserID AS ReceiverID,
    u.Name AS ReceiverName,
    s.Name AS SenderName,
    g.GymName,
    ct.ClassName,
    c.Redeemed,
    c.ExpirationDate

FROM Credit c --supertype

JOIN AppUser u
    ON u.UserID = c.ReceiverUserID

JOIN AppUser s
    ON s.UserID = c.SenderUserID

JOIN GymClass gc
    ON gc.CreditID = c.CreditID

JOIN GymClassType gct
    ON gct.GymClassTypeID = gc.GymClassTypeID

JOIN ClassType ct
    ON ct.ClassTypeID = gct.ClassTypeID

JOIN Gym g
    ON g.GymID = gct.GymID

WHERE c.Redeemed IS FALSE;
```

Data Output | Messages | Notifications

	receiverid integer	receivername character varying (200)	sendername character varying (200)	gymname character varying (50)	classname character varying (50)	redeemed boolean	expirationdate date
1	3	Antonio Mandez	Emilia Harrington	Bars and Bells	Yoga	false	2025-12-01
2	8	Marcus Medina	Julia Serrano	Bars and Bells	Pilates	false	2025-12-01
3	12	Samantha Medina	Bonita Frenza	Minis Yoga Studio	Yoga	false	2026-03-01

We can see that there are 3 users who have gymclass type credits gifted by their friends. Since the credits have not been redeemed yet, they are still available for the users and will show up in the app's credits section for them.

Third Query

Create a view that captures information that needs be accessed regularly based upon the use of your database. Utilize the view in the query. To ensure sufficient complexity, the query (or underlying view) should contain at least two of the constructs below. There should be at least one from each group.

Group 1 (choose one or more)

- **joins of at least two tables.**
- **one or more restrictions in the WHERE clause.**
- **an order by statement.**

Group 2 (choose one or more)

- **at least one aggregate function.**

- at least one subquery.
- a having clause.
- a left or right join.
- joins of four or more tables.
- a union of two queries

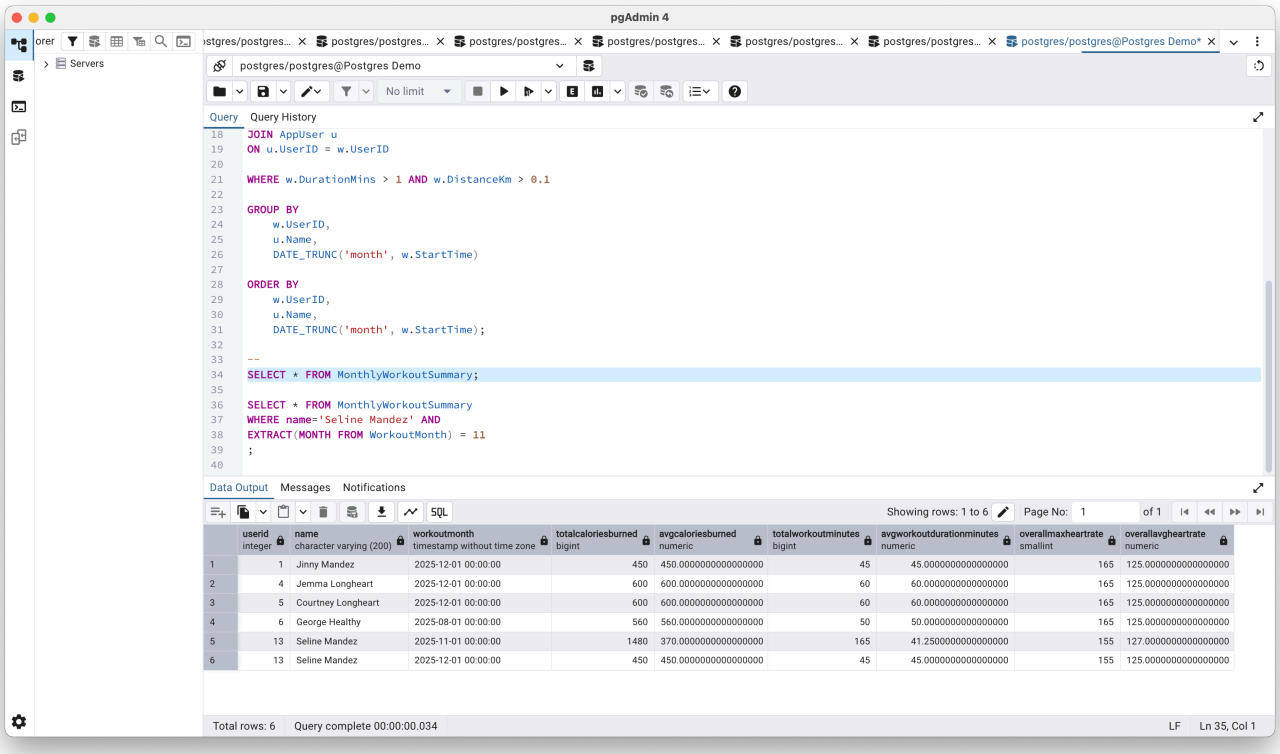
Question 3- "What is the monthly workout summary for each user including calories burned, heart health and workout performance?"

I will be creating a View for a user's monthly workout summary. For the app, we want to give the user a snapshot of their entire month's workout metrics. The metrics we see normally for such workout summaries in other fitness apps are average and total calories burned, average workout durations, total number of workouts performed, overall maximum heart rate and overall average heart rate.
I used the **AddUserWorkout** stored procedure to add new workouts for my test user.

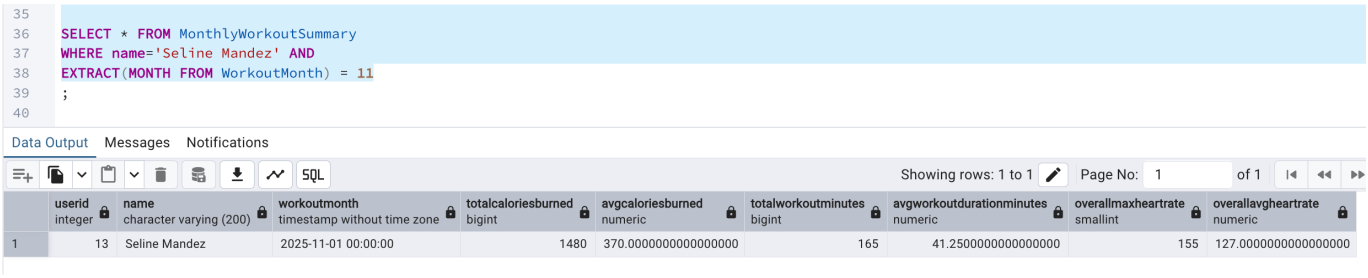
```
START TRANSACTION;
CALL AddUserWorkout(
    'PERG22421@#@$14',
    'mandez.journey@gmail.com',
    '1996-05-11'::DATE,
    190.01,
    170.58,
    'Female',
    'Seline Mandez',
    '02342-9921',
    'Beginner',
    45,
    450,
    '2025-12-02 12:05:00'::TIMESTAMP,
    '2025-12-02 12:50:00'::TIMESTAMP,
    125::SMALLINT,
    155::SMALLINT,
    6,
    'Jogging'
);

COMMIT;

```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 29 msec.
```



So we can see above the monthly workout summaries of each user grouped by each month (shown by month's starting date). For specific user Seline Mandez in the month of November, this was the summary extracted through the view.



Section 12: Indexes(revised)

Include your indexes as a reference (also update it if necessary).

The primary keys are as follows-
AppUser.UserID
BeginnerUser.UserID
AdvancedUser.UserID
HealthRiskMetric.RiskMetricID
FrequencyType. FrequencyTypeID
DailyHealthMetric. MetricID
Friendship. FrienshipID
WorkoutType. WorkoutTypeID
Workout. WorkoutID
MenstrualCycleMetric. CycleID

HydrationMetric. HydrationMetricID
Credit. CreditID
GymClass. CreditID
GiftCard. CreditID
GymClassType. GymClassTypeID
Shop. ShopID
Gym. GymID
ClassType. ClassTypeID
HydrationIntakeUpdate. HydrationIntakeUpdateID

The foreign keys being used are as follows-

Column	**Unique?**	**Description**
Workout.UserID	Non-unique	The foreign key in Workout referencing AppUser is not unique because there can be many workouts for a user.
Workout. WorkoutTypeID	Non-unique	The foreign key in Workout referencing WorkoutType is not unique because there can be many workouts that are of the same workout type.
HealthRiskMetric. UserID	Non-unique	The foreign key in HealthRiskMetric referencing AppUser is not unique because there can be health records for each user. Each annual checkup should create a new health metric log.
HealthRiskMetric. SmokingFrequencyID	Non-unique	The foreign key in HealthRiskMetric referencing FrequencyType is not unique because there can be many users whose health records have the same frequency of drinking alcohol or smoking.
HealthRiskMetric. AlcholFrequencyID	Non-unique	The foreign key in HealthRiskMetric referencing FrequencyType is not unique because there can be many users whose health records have the same frequency of drinking alcohol or smoking.
Friendship. UserID1	Non-unique	The foreign key in Friendship referencing AppUser is not

		unique because there can be many friendships that a user has.
Friendship. UserID2	Non-unique	The foreign key in Friendship referencing AppUser is not unique because there can be many friendships that a user has.
DailyHealthMetric.MetricID	Non-unique	The foreign key in DailyHealthMetric referencing AppUser is not unique since there can be many health metric logs for each user. There should be 1 log daily for each user.
Credit.SenderUserID	Non-unique	The foreign key in Credit referencing AppUser is not unique since the same user can give many credits to their friends
Credit.ReceiverUserID	Non-unique	The foreign key in Credit referencing AppUser is not unique since the same user can receive many credits from their friends
GymClass.CreditID	Unique	The foreign key in GymClass referencing Credit is unique since each credit should be defined by a subtype once and must specifically be of type gym class or gift card or outside of the given list of rewards (partially complete). This field is both a PK, FK in GymClass.
GymClass.GymClassTypeID	Non-unique	The foreign key in GymClass referencing GymClassType is non-unique since there can be many gym classes that can be of the same gym class type combination.
GiftCard.CreditID	Unique	The foreign key in GymClass referencing Credit is unique since each credit should be defined by a subtype once and must specifically be of type gym class or gift card or outside of the given list of rewards

		(partially complete). This field is both a PK, FK in GymClass.
GiftCard.ShopID	Non-unique	The foreign key in GiftCard referencing Shop is not unique since there can be many gift cards that can be used at the same store.
GymClassType.GymClassTypeID	Non-unique	The foreign key in GymClassType referencing ClassType is not unique since there can be many gym classes that are of the same class type.
GymClassType.GymID	Non-unique	The foreign key in GymClassType referencing Gym is not unique since there can be many gym classes that are hosted at the same gym.
MenstrualCycleMetric.UserID	Non-unique	The foreign key in MenstrualCycleMetric referencing AppUser is not unique since there can be many menstural cycle metric instances recorded for the same user.
HydrationMetric.UserID	Unique	The foreign key in HydrationMetric referencing AppUser is unique since there should only be one hydration metric log for each user that gets update frequently.
BeginnerUser.UserID	Unique	The foreign key in BeginnerUser referencing AppUser should be unique since the user has to be either in the Beginner or Advanced category and there should only be one occurrence of the user in the subtype. UserID is both the PK and FK in the subtype.
AdvancedUser.UserID	Unique	The foreign key in AdvancedUser referencing AppUser should be unique since the user has to be either in the Beginner or Advanced category and there should only be one occurrence of the user in the

		subtype. UserID is both the PK and FK in the subtype.
HydrationIntakeUpdate. HydrationMetricID	***Non-unique***	***The foreign key in HydrationIntakeUpdate referencing HydrationMetric is non-unique since there can be many hydration intake update logs for the same hydration metric ID which is assigned to a specific user.***

For query driven index 1, I am going to choose Workout.DurationMins since it is used in the WHERE clause of the MonthlyWorkoutSummary view to only obtain valid impactful workouts. It filters out very short and impossible workouts that the user might have accidentally recorded on their smartwatches that provide no real value. This indexing improves the performance of the filtering and avoids full table scans when generating monthly summaries. Many workouts can be of similar durations so this index would be non-unique.

For query driven index 2, I am going to use the Credit.ReceiverUserID since we have Query 2 above which checks what gym classes are available for each user. Thus, the receiver user ID is going to be a heavily used search column in such queries. This indexing would speed up all the credit-lookup queries for each receiving user. Multiple credits can be received by the same user, so the index would be non-unique.

For query driven index 3, I am going to use the Workout.StartTime since this is a main column used in the MonthlyWorkoutSummary view. It is used for grouping workouts by month and filtering workouts by date. This indexing approach improves the performance of DATE_TRUNC-based grouping and month-range filters. Multiple workouts can occur at the same start time so this index would be non-unique.

```
1  -- Indexes
2  CREATE INDEX Workout_DurationMins_Idx ON Workout(DurationMins);
3  CREATE INDEX Credit_ReceiverUserID_QIdx ON Credit(ReceiverUserID);
4  CREATE INDEX Workout_StartTime_Idx ON Workout(StartTime);
5
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 35 msec.
```

Section 13: Attribute History

Select a column or columns that would benefit from a record of historical values in your database and explain. Design a new history table by creating a new structural database rule and updating your ERDs. Create the history table and sequence in SQL, and define a trigger that maintains the history table. Provide proof that your trigger maintains the history correctly with screenshots.

I am going to select the CurrentDayIntakeML attribute from the HydrationMetric column. Each user will only have 1 hydration metric log which keeps getting updated throughout the day as they consume more water. By keeping track of the changes in total water intake throughout the day, we will have a hydration history for each user. We can use this data to observe and later visualize hydration trends. We can detect what days a user has insufficient intake. We can compare hydration trends over weeks, months or years. We also discover at what times of the day users drink more water. This data helps us give more personalized hydration recommendations to users.

I am adding screenshots of the execution of the create trigger function and create trigger->

```
-- Function linked to trigger setup

CREATE OR REPLACE FUNCTION HydrationIntakeUpdateFunction()
RETURNS TRIGGER LANGUAGE plpgsql
AS $trigfunc$
BEGIN
    INSERT INTO HydrationIntakeUpdate(
        HydrationIntakeUpdateID,
        OldIntakeML,
        NewIntakeML,
        HydrationMetricID,
        InsertionDate
    )
    VALUES(
        nextval('hydrationupdate_id'),
        OLD.CurrentDayIntakeML,
        NEW.CurrentDayIntakeML,
        NEW.HydrationMetricID,
        NOW()
    );

    RETURN NEW;
END;
$trigfunc$;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 26 msec.
```

```
-- Trigger setup
DROP TRIGGER IF EXISTS HydrationIntakeUpdateTrigger ON HydrationMetric;

CREATE TRIGGER HydrationIntakeUpdateTrigger
BEFORE UPDATE OF CurrentDayIntakeML ON HydrationMetric
FOR EACH ROW
EXECUTE PROCEDURE HydrationIntakeUpdateFunction();
```

Data Output | Messages | Notifications

```
NOTICE:  trigger "hydrationintakeupdatetrigger" for relation "hydrationmetric" does not exist, skipping
CREATE TRIGGER

Query returned successfully in 32 msec.
```

To test the trigger, we are setting up the HydrationMetric table with some initial rows for some users.

```
1163  -- Inserting the initial sample HydrationMetric rows
1164
1165  SELECT * FROM AppUser;
1166
1167  INSERT INTO HydrationMetric(
1168      UserID,
1169      Temperature,
1170      LastUpdated,
1171      CurrentDayIntakeML,
1172      DailyWaterGoalML)
1173  VALUES(1, 100.5, NOW(), 300, 2800),
1174  (2, 87, NOW(), 500, 2600);
1175
1176  SELECT * FROM HydrationMetric;
```

Data Output | Messages | Notifications

Showing rows: 1 to 2

	hydrationmetricid [PK] integer	userid integer	temperature numeric (5,1)	lastupdated timestamp without time zone	currentdayintakeml integer	dailywatergoalml integer
1	1	1	100.5	2025-12-07 22:52:01.387671	300	2800
2	2	2	87.0	2025-12-07 22:52:01.387671	500	2600

When the users drink water throughout the day, their daily intake field gets updated.

```
1179
1180  UPDATE HydrationMetric
1181  SET CurrentDayIntakeML = 450, LastUpdated = NOW()
1182  WHERE HydrationMetricID = 1;
1183
1184  UPDATE HydrationMetric
1185  SET CurrentDayIntakeML = 650, LastUpdated = NOW()
1186  WHERE HydrationMetricID = 2;
1187
1188  UPDATE HydrationMetric
1189  SET CurrentDayIntakeML = 600, LastUpdated = NOW()
1190  WHERE HydrationMetricID = 1;
1191
1192  UPDATE HydrationMetric
1193  SET CurrentDayIntakeML = 800, LastUpdated = NOW()
1194  WHERE HydrationMetricID = 2;
1195
1196  UPDATE HydrationMetric
1197  SET CurrentDayIntakeML = 700, LastUpdated = NOW()
1198  WHERE HydrationMetricID = 1;
1199
1200  UPDATE HydrationMetric
1201  SET CurrentDayIntakeML = 990, LastUpdated = NOW()
1202  WHERE HydrationMetricID = 2;
1203
```

Data Output | Messages | Notifications

Show

	hydrationintakeupdateid [PK] integer	hydrationmetricid integer	oldintakeml integer	newintakeml integer	insertiondate timestamp without time zone
1	1	1	300	450	2025-12-07 22:52:39.112834
2	2	2	500	650	2025-12-07 22:52:39.112834
3	3	1	450	600	2025-12-07 22:52:39.112834

We can then see that the history table has recorded each new and old intake quantity for the users along with their insertion timestamps.

```
1204    -- Checking for history of changes for the 2 users in the new table
1205    SELECT * FROM HydrationIntakeUpdate;
1206
```

Data Output | Messages | Notifications

	hydrationintakeupdateid [PK] integer	hydrationmetricid integer	oldintakeml integer	newintakeml integer	insertiondate timestamp without time zone
1	1	1	300	450	2025-12-07 22:52:39.112834
2	2	2	500	650	2025-12-07 22:52:39.112834
3	3	1	450	600	2025-12-07 22:52:39.112834
4	4	2	650	800	2025-12-07 22:52:39.112834
5	5	1	600	700	2025-12-07 22:53:26.986608
6	6	2	800	990	2025-12-07 22:53:26.986608

To explain the code steps line by line->

Code	Description
CREATE OR REPLACE FUNCTION HydrationIntakeUpdateFunction()	Defining the trigger function that gets executed every time the trigger condition is satisfied
RETURNS TRIGGER LANGUAGE plpgsql	Specifies that the function is used within a trigger and written in PL/pgSQL procedural language.
AS $trigfunc$	Denotes where the function body starts.
BEGIN	Starts the executable portion of the trigger function.
INSERT INTO HydrationIntakeUpdate(HydrationIntakeUpdateID, OldIntakeML, NewIntakeML, HydrationMetricID, InsertionDate) VALUES(nextval('hydrationupdate_id'), OLD.CurrentDayIntakeML, NEW.CurrentDayIntakeML, NEW.HydrationMetricID, NOW());	Setup to insert a row into the history table which includes the old and new hydration values, the hydration metric ID matching the user and the insertion timestamp of the change.
RETURN NEW;	Required in BEFORE UPDATE triggers. Returns the modified NEW row so the actual update on HydrationMetric can continue.

END;	Closes the trigger function's executable block
$trigfunc$;	Tells PostgreSQL that the function definition is complete.
DROP TRIGGER IF EXISTS HydrationIntakeUpdateTrigger ON HydrationMetric;	Removes any trigger matching the given name on the given entity if it already exists.
CREATE TRIGGER HydrationIntakeUpdateTrigger	Creating a new trigger object with the given name.
BEFORE UPDATE OF CurrentDayIntakeML ON HydrationMetric	Specifies that the trigger will fire before an update occurs on the CurrentDayIntakeML field of the HydrationMetric entity. This guarantees that history row is written before the new hydration amount is saved.
FOR EACH ROW	This assures that the trigger fires once for every row that gets updated.
EXECUTE PROCEDURE HydrationIntakeUpdateFunction();	Implements the trigger function defined above to do the actual history row insertion.

Section 14: Data Visualizations

Create two visualizations of the data in your database. Clearly explain the data story conveyed by each visualization. Ensure that the visualizations and data stories are useful and appropriate given the intended use of your database. Add the SQL to your SQL script.

Visualization 1

The first visualization I would like to capture is the changes in a user's hydration intake quantity throughout the day. It would be insightful to see the user's hydration trend over time as well to understand how the user is taking care of their hydration needs. We will be able to see days when the user drinks less water or more water throughout the day. According to the results, we can give hydration reminders. For example, if we notice that a user drinks less water during the daytime we can send more frequent hydration reminders from the app during the afternoon. So the question that will be answered here is, ***how many hydration reminders do we need to send users based on their hydration trends throughout the day?***

So, to create more user hydration updates, I have inserted more HydrationMetric rows for some of the app users and updated the CurrentDayIntakeML to show how users drink more water throughout the day. We can see the entire history of 4 different users throughout a day below. For our visualization, let's look at the average change in water intake for each user by day.

```
1265  SELECT * FROM HydrationIntakeUpdate;
1266  SELECT * FROM HydrationMetric;
```

Data Output | Messages | Notifications

Showing rows: 1 to 15 | Page No: 1 of 1

	hydrationintakeupdateid [PK] integer	hydrationmetricid integer	oldintakeml integer	newintakeml integer	insertiondate timestamp without time zone
1	1	1	300	450	2025-12-08 13:40:33.298545
2	2	2	500	650	2025-12-08 13:40:33.298545
3	3	1	450	600	2025-12-08 13:40:33.298545
4	4	2	650	800	2025-12-08 13:40:33.298545
5	5	1	600	700	2025-12-08 13:40:33.298545
6	6	2	800	990	2025-12-08 13:40:33.298545
7	7	3	0	450	2025-12-08 13:41:25.662729
8	8	4	0	300	2025-12-08 13:41:25.662729
9	9	3	450	500	2025-12-08 13:52:56.63207
10	10	4	300	400	2025-12-08 13:52:56.63207
11	11	3	500	700	2025-12-08 14:02:30.258871
12	12	4	400	600	2025-12-08 14:02:30.258871
13	13	3	700	800	2025-12-08 14:40:37.578388
14	14	4	600	800	2025-12-08 14:40:37.578388
15	15	4	800	900	2025-12-08 14:45:45.185806

Using the SELECT query I am finding the average hydration intake changes for each user based on each day by subtracting the old intake amount from the new intake amount in each record. I am counting the total number of hydration update rows there are for each user which shows how many times they drank water throughout the day so far, considering that they log their hydration each time on the app. I am also joining with the HydrationMetric and AppUser tables to get the user's name through their HydrationMetricID for more practical data for visualization.

```
1268  -- Visualization Query 1
1269
1270  SELECT
1271      InsertionDate::date AS DateRecordeed,
1272      AVG(NewIntakeML - OldIntakeML) AS AvgIntakeIncrease,
1273      COUNT(*) AS totalDrinks,
1274      u.Name
1275  FROM HydrationIntakeUpdate i
1276  JOIN HydrationMetric h
1277  ON i.HydrationMetricID = h.HydrationMetricID
1278  JOIN AppUser u
1279  ON h.UserID = u.UserID
1280  GROUP BY InsertionDate::date, u.Name
1281  ORDER BY DateRecordeed, u.Name;
1282
1283
```

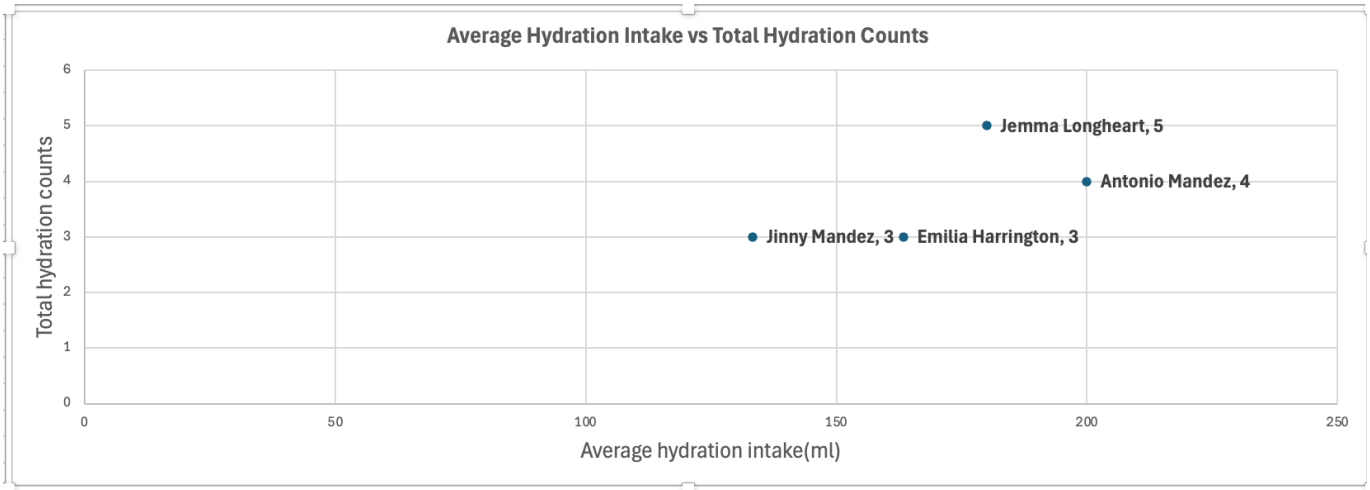
Data Output | Messages | Notifications

Showing rows: 1 to 4 | Pag

	daterecordeed date	avgintakeincrease numeric	totaldrinks bigint	name character varying (200)
1	2025-12-08	200.0000000000000000	4	Antonio Mandez
2	2025-12-08	163.3333333333333333	3	Emilia Harrington
3	2025-12-08	180.0000000000000000	5	Jemma Longheart
4	2025-12-08	133.3333333333333333	3	Jinny Mandez

Through the scatterplot we see each user's hydration patterns. For example, user Jinny Mandez seems to have the least amount of water intakes which was just 3 times a day and she has the least average intake which was 133 ml. Seeing that Jinny drank very little amount of water only few times a day, she

has a high chance of becoming dehydrated, so we need to send her more hydration reminders compared to our other users. On the other hand, Jemma seems to have 5 daily water intakes averaging 180 ml each time which is a better hydration pattern for that day. Since Jemma drinks a good amount of water more frequently, she should be better hydrated. We don't need to send Jemma that many hydration reminders as a result. Thus, through this visualization we know that combining average water intake and intake frequency helps us send more user-specific hydration reminders.



Visualization 2

It would be very insightful to know if users have reached their hydration goals for the day. We are allowing users to set their own daily hydration goals as they see fit. This information is very important to send them hydration reminders within the remaining hours of the day to reach their hydration goals. So the question we are going to ask for this case is- ***Have users reached their daily hydration goals?*** Based on the users that are already in the database so far, we can get the information on the current total water intake of a user throughout the day along with their hydration goals for the day from the HydrationMetric table. To get the usernames, we join to the AppUser table based on UserID. We get the hydration information of 4 users that show their current total intake along with their daily water goals.

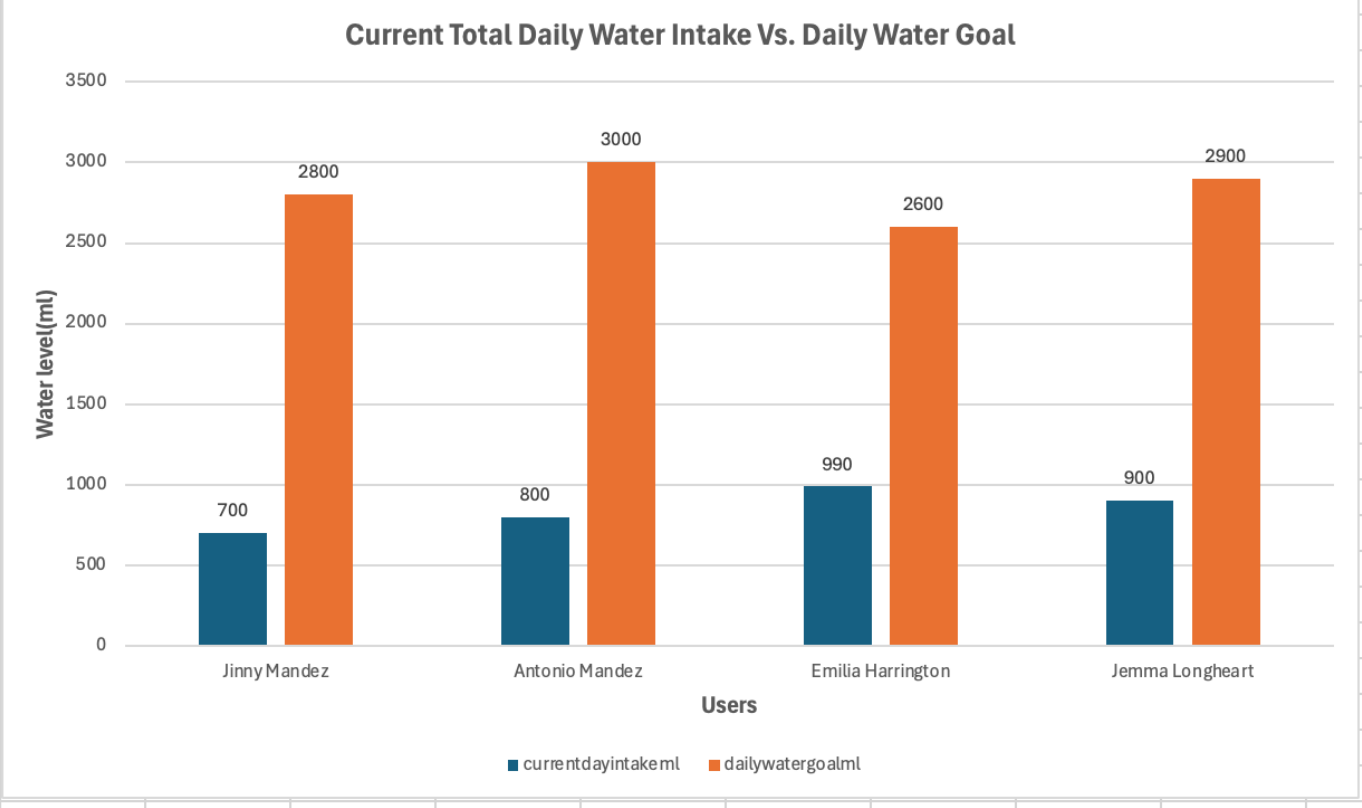
```
1283  -- Visualization Query 2
1284
1285  SELECT h.CurrentDayIntakeML, h.DailyWaterGoalML, u.Name
1286  FROM HydrationMetric h
1287  JOIN AppUser u
1288  ON h.UserID = u.UserID
1289  WHERE h.CurrentDayIntakeML < h.DailyWaterGoalML
1290
1291
```

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page

	currentdayintakeml integer	dailywatergoalml integer	name character varying (200)
1	700	2800	Jinny Mandez
2	990	2600	Emilia Harrington
3	800	3000	Antonio Mandez
4	900	2900	Jemma Longheart

With the help of this bar chart, we see how much of a gap each user must reach their daily water goals. The blue bars represent the current total daily intake for users, and the orange bars represent the daily water intake goals for users. Emilia is the closest to her daily hydration goal with a gap of 1610 ml left to drink throughout the day. Whereas Jinny and Antonio have a large gap of over 2000 ml to meet their daily hydration goals. We see that we need the app to send more hydration reminders to Jinny and Antonio since they are falling behind on their hydration goals. Thus, this hydration visualization helps us to keep track of each user's performance and needs throughout the day making it easier for us to know which users need how many hydration reminders.



Section 15: AI Collaboration Summary for Iteration 6

Summarize your generative AI collaboration for Iteration 6. Do not include summaries for your prior iterations again.

15.1 AI Collaboration Overview

I used AI to learn more about trigger function and actual trigger setup. AI also helped come up with meaningful business ideas using the hydration update history table. AI also gave me suggestions on what column to choose to make a history table that is most beneficial for app users. I also got help with debugging some code to unblock me.

15.2 Two Key Prompts

Prompt 1: "how to update this to not get timezone error- VALUES(
nextval('hydrationupdate_id'),
OLD.CurrentDayIntakeML,

NEW.CurrentDayIntakeML,
NEW.HydrationMetricID,
current_date
);"

I was not getting the proper DATETIME values with current_date, so checked with AI to debug the code to get the proper value to use to get the accurate insertion date and time recorded. It turned out there was no time component in current_date.
PostgreSQL does not allow inserting a DATE into a TIMESTAMP column,so it was triggering a casting or timezone error. I ended up using NOW() because it showed the timestamp I required for the app's use cases.

Prompt 2: "can I ask 'Show users daily hydration trend to determine how hydrated users are and how many hydration reminders we need to send?'"

I wanted to see what AI would suggest as a valid question to ask for the business queries meant for the first visualization. I wanted to see if it was a good practical question and get a more polished version from AI to include in the report. I ended up using the same idea for the query since AI also thought it was a good idea but worded the question abit differently based on what I thought sounded best.

15.3 Two Key Iterations

Replace this with descriptions of two key iterations with AI.

Iteration 1- Suggestions for Visualization 2
AI gave useful suggestions on what to use for my second visualization. It had suggested that I use the daily hydration goal to see **which users are meeting their water goals** and which ones fall short. It matched the business goals of the app since the app needs to send hydration reminders to users based on their hydration intake throughout the day. It said that since my Visualization 1 focuses on hydration behavior, it makes more sense that my Visualization 2 should focus on hydration outcomes. I ended up going with the recommended idea for my second query and visualization.

Iteration 2- Line-By-Line Trigger code explanation
I had understood the concept of triggers but was new to writing out a trigger and followed the walkthrough as well as AI suggestions on the setup of the trigger and explanation of the code line-by-line. AI explained all the parts of the trigger setup clearly which helped me write up my explanation table in Section 13. It was also interesting to know that the trigger can do the insertion into the history table before the original table that is being monitored gets updated.

15.4 AI Transcripts (Separate Documents or Links)

Attach your raw, unedited transcripts as separate PDFs or Word docs, or provide shared links.

https://chatgpt.com/s/t_69377dd6b7b48191b9448c3894b9551e
https://chatgpt.com/s/t_69377f327a6481919dbd7794df90d996
https://chatgpt.com/s/t_693781f986548191acca99141b303f23
https://chatgpt.com/s/t_693785ccc3d08191952032c24ed559a1