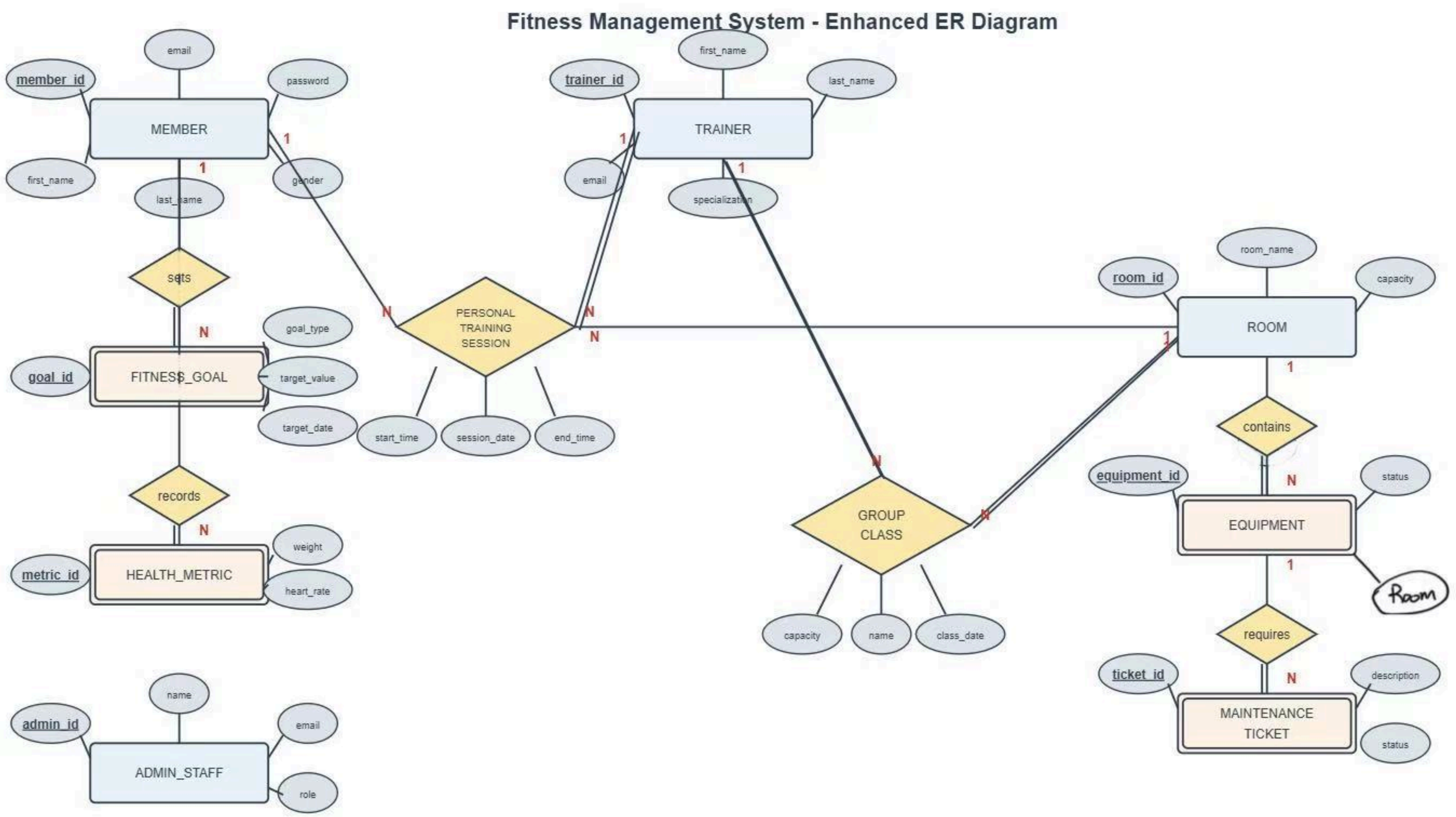
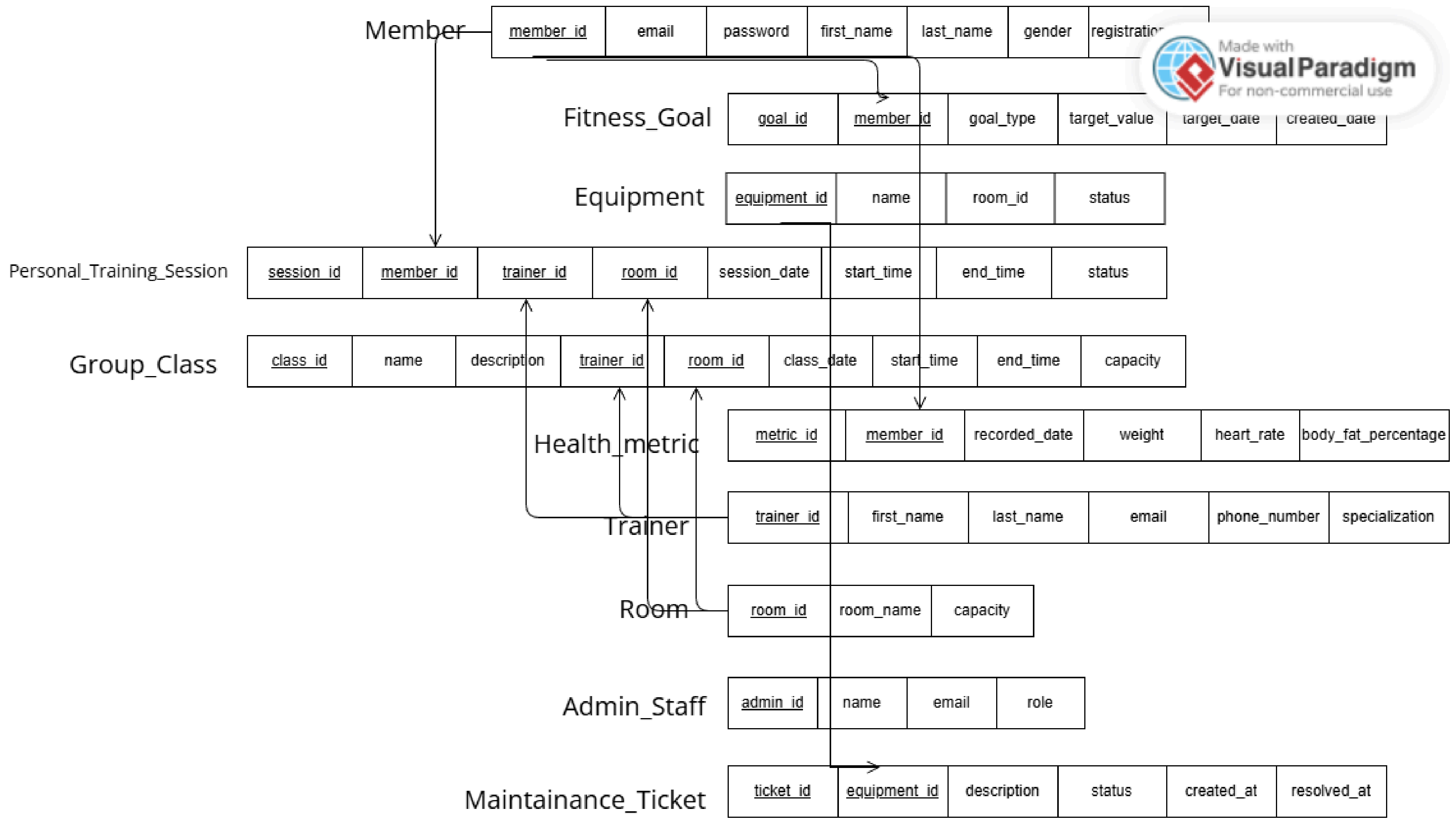
ER Diagram:


Fitness Management System - Enhanced ER Diagram
MEMBER
member_id
email
password
first_name
last_name
gender
sets
FITNESS_GOAL
goal_id
goal_type
target_value
target_date
records
HEALTH_METRIC
metric_id
weight
heart_rate
ADMIN_STAFF
admin_id
name
email
role
PERSONAL TRAINING SESSION
start_time
session_date
end_time
TRAINER
trainer_id
first_name
last_name
email
specialization
GROUP CLASS
capacity
name
class_date
ROOM
room_id
room_name
capacity
contains
EQUIPMENT
equipment_id
status
Room
requires
MAINTENANCE TICKET
ticket_id
description
status
1
N

Schema Diagram:



Mapping Table:

Requirement	Assumptions	Representation in ER Model
Members are uniquely identified and have basic contact information recorded. Each member has an age attribute.	Each member has exactly one member_id (unique identifier), email, password, first name, last name, and calculated/stored age.	**Member Entity** — Attributes: member_id (PK), email, password, first_name, last_name, age.
Members can set multiple fitness goals to track their fitness journey. Each goal has specific details.	A member can have zero or many fitness goals. Each goal belongs to exactly one member. Goals are tracked with target values, deadlines, and goal types.	**Weak Entity: Fitness_Goal** — Attributes: goal_id (Partial Key), goal_type, target_value, deadline, goal_date. **Relationship: sets** (1:N between Member and Fitness_Goal). Participation: partial on Member, total on Fitness_Goal.
Members' health metrics need to be tracked over time for progress monitoring.	A member can have multiple health metric records. Each record belongs to exactly one member and captures weight and heart rate at a specific time.	**Weak Entity: Health_Metric** — Attributes: metric_id (Partial Key), weight, heart_rate. **Relationship: records** (1:N between Member and Health_Metric). Participation: partial on Member, total on Health_Metric.
Trainers are uniquely identified and have contact details and specialization recorded.	Each trainer has exactly one trainer_id (unique identifier), one email, name, and specialization area (e.g., Yoga, Strength Training, Cardio).	**Trainer Entity** — Attributes: trainer_id (PK), email, name, specialization.
Members can have personalized training sessions with trainers. These sessions need to capture session details and scheduling.	A member can attend multiple personal training sessions. A trainer can conduct multiple personal training sessions. Sessions have specific dates, times, and durations.	**Relationship: Personal_Training_Session** (M:N between Member and Trainer). **Associative Entity** with attributes: session_date, session_time, session_type, min_price. Composite PK: (member_id, trainer_id, session_date, session_time).

Group fitness classes are offered with different types and capacities. Trainers lead these classes.	Each group class has a unique identifier, capacity limit, name, and class type. A trainer can lead multiple group classes, but each class is led by exactly one trainer.	**Group_Class Entity** — Attributes: class_id (implied PK), capacity, name, class_type. **Relationship: leads** (1:N between Trainer and Group_Class). Participation: partial on Trainer, total on Group_Class.
Rooms are identified and have capacity and name recorded. Rooms may contain equipment.	Each room has a unique room_id, room name, and capacity. A room can have zero or many pieces of equipment.	**Room Entity** — Attributes: room_id (PK), room_name, capacity. **Relationship: contains** (1:N between Room and Equipment). Participation: partial on both sides.
Equipment needs to be tracked for maintenance purposes. Each piece of equipment is located in a specific room.	Each equipment has a unique equipment_id, name, and status (e.g., available, maintenance, out of service). Equipment belongs to exactly one room.	**Equipment Entity** — Attributes: equipment_id (PK), name, status. **Relationship: contains** (1:N from Room). Participation: total on Equipment side (equipment must be in a room).
Equipment requires regular maintenance to ensure safety and functionality.	Each equipment can have multiple maintenance records. Each maintenance ticket tracks issues and service dates.	**Weak Entity: Maintenance_Ticket** — Attributes: ticket_id (Partial Key), description, status. **Relationship: requires** (1:N between Equipment and Maintenance_Ticket). Participation: partial on Equipment, total on Maintenance_Ticket.
Administrative staff manage the fitness center operations and need to be tracked.	Each admin staff has a unique admin_id, name, email, and role (e.g., Manager, Front Desk, Operations).	**Admin_Staff Entity** — Attributes: admin_id (PK), name, email, role.

Strong Entities

1. **Member** (member_id, email, password, first_name, last_name, age)
2. **Trainer** (trainer_id, email, name, specialization)
3. **Room** (room_id, room_name, capacity)
4. **Equipment** (equipment_id, name, status)
5. **Group_Class** (class_id, capacity, name, class_type)
6. **Admin_Staff** (admin_id, name, email, role)

Weak Entities

1. **Fitness_Goal** (goal_id, goal_type, target_value, deadline, goal_date) — depends on Member
2. **Health_Metric** (metric_id, weight, heart_rate) — depends on Member
3. **Maintenance_Ticket** (ticket_id, description, status) — depends on Equipment

Associative Entities

1. **Personal_Training_Session** (member_id, trainer_id, session_date, session_time, session_type, min_price) — M:N between Member and Trainer with attributes

Proof of Normalization:

1. **MEMBER**
   i. Relation Schema
      i. MEMBER(member_id, email, password, first_name, last_name, gender, registration_date)
   ii. Functional Dependencies
      i. FD1: member_id → {email, password, first_name, last_name, gender, registration_date}
      ii. FD2: email → {member_id, password, first_name, last_name, gender, registration_date}
2. **FITNESS_GOAL**
   i. Relation Schema
      i. FITNESS_GOAL(goal_id, member_id, goal_type, target_value, target_date, created_date)
   ii. Functional Dependencies
      i. FD1: goal_id → {member_id, goal_type, target_value, target_date, created_date}
3. **EQUIPMENT**
   i. Relation Schema
      i. EQUIPMENT(equipment_id, name, room_id, status)
   ii. Functional Dependencies
      i. FD1: equipment_id → {name, room_id, status}
4. **HEALTH_METRIC**
   i. Relation Schema
      i. HEALTH_METRIC(metric_id, member_id, recorded_date, weight, heart_rate, body_fat_percentage)
   ii. Functional Dependencies
      i. FD1: metric_id → {member_id, recorded_date, weight, heart_rate, body_fat_percentage}
5. **TRAINER**
   i. Relation Schema
      i. TRAINER(trainer_id, first_name, last_name, email, phone_number, specialization)
   ii. Functional Dependencies
      i. FD1: trainer_id → {first_name, last_name, email, phone_number, specialization}
      ii. FD2: email → {trainer_id, first_name, last_name, phone_number, specialization}
6. **ROOM**
   i. Relation Schema
      i. ROOM(room_id, room_name, capacity)
   ii. Functional Dependencies
      i. FD1: room_id → {room_name, capacity}
      ii. FD2: room_name → {room_id, capacity}

7. **PERSONAL_TRAINING_SESSION**
   i. Relation Schema
      i. PERSONAL_TRAINING_SESSION(session_id, member_id, trainer_id, room_id, session_date, start_time, end_time, status)
8. **ADMIN_STAFF**
   i. Relation Schema
      i. ADMIN_STAFF(admin_id, name, email, role)
   ii. Functional Dependencies
      i. FD1: admin_id → {name, email, role}
      ii. FD2: email → {admin_id, name, role}
9. **GROUP_CLASS**
   i. Relation Schema
      i. GROUP_CLASS(class_id, name, description, trainer_id, room_id, class_date, start_time, end_time, capacity)
   ii. Functional Dependencies
      i. FD1: class_id → {name, description, trainer_id, room_id, class_date, start_time, end_time, capacity}
10. **MAINTAINANCE_TICKET**
   i. Relation Schema
      i. MAINTENANCE_TICKET(ticket_id, equipment_id, description, status, created_at, resolved_at)
   ii. Functional Dependencies
      i. FD1: ticket_id → {equipment_id, description, status, created_at, resolved_at}

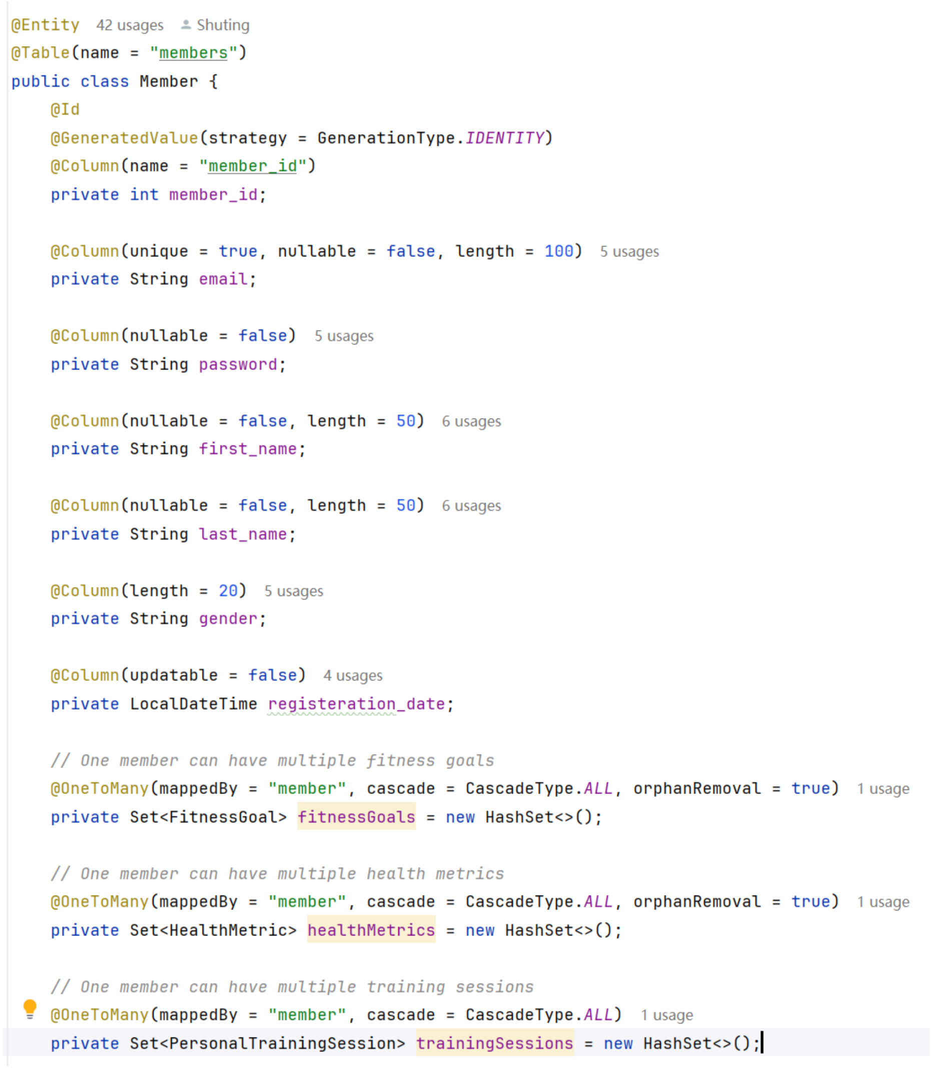
Explanation:

All of the above primary key is a single attribute, there are no subsets of the primary key hence there are no partial dependencies. All non-prime attributes are fully functionally dependent on the entire primary key in each class. Therefore that makes it 2NF.

Then we checked for transitive dependency, which we can see that any other non-prime attributes do not functionally determine any attribute. We can also separate some of the schema into a natural join of functional dependencies . Hence they all satisfy 3NF.

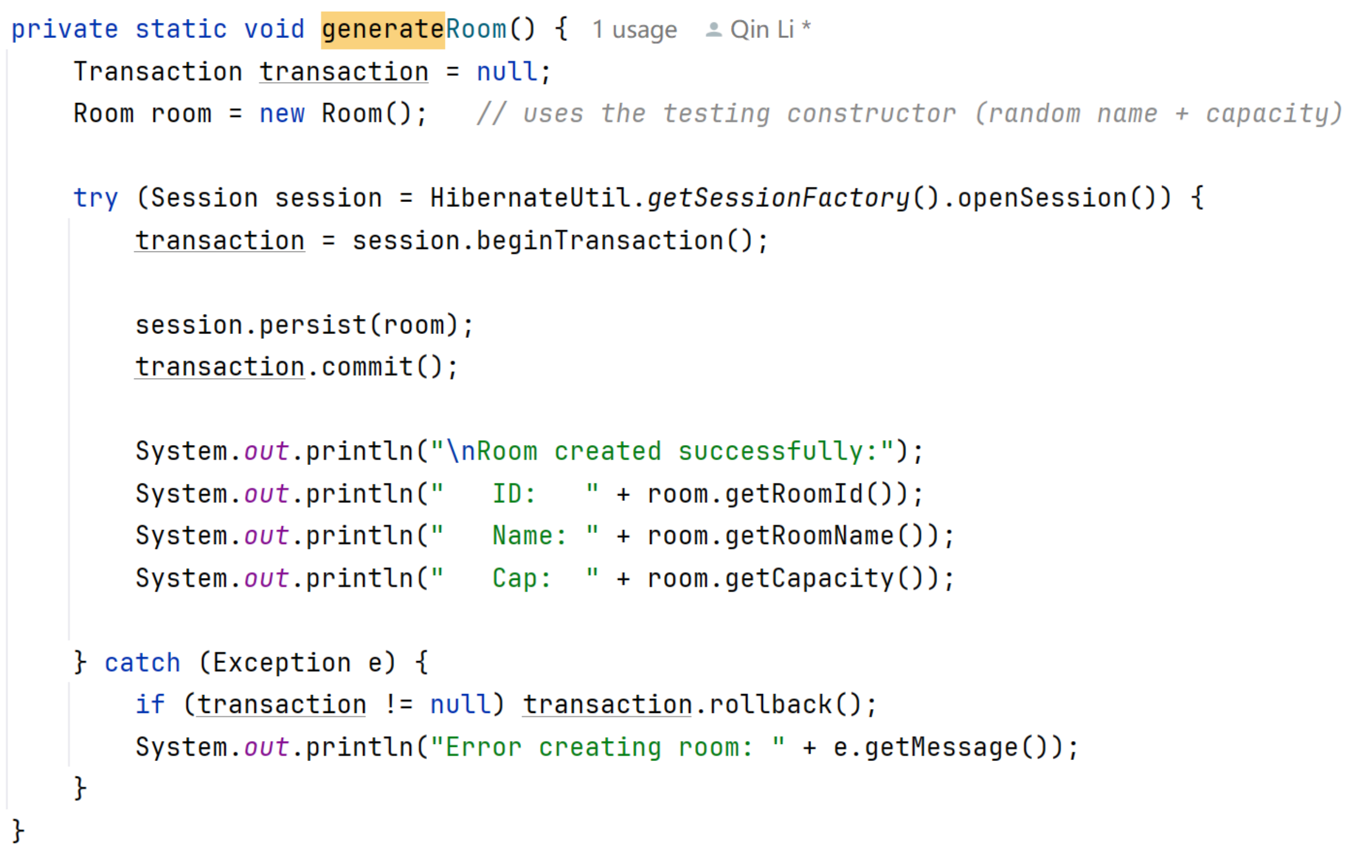
The use of ORM:

In our project, we started with an ER model that described the major objects in a fitness club, such as Members, Trainers, Rooms, Classes, and Equipment. Using Hibernate as our ORM, we translated every entity and relationship directly into annotated Java classes. For example, each table became a Java class annotated with `@Entity` and `@Table`, and all primary keys were defined using `@Id` and `@GeneratedValue`.



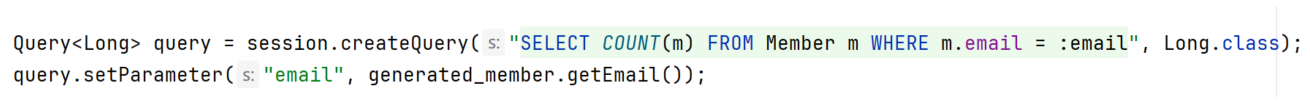
```java
@Entity  42 usages  Shuting
@Table(name = "members")
public class Member {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "member_id")
    private int member_id;

    @Column(unique = true, nullable = false, length = 100)  5 usages
    private String email;

    @Column(nullable = false)  5 usages
    private String password;

    @Column(nullable = false, length = 50)  6 usages
    private String first_name;

    @Column(nullable = false, length = 50)  6 usages
    private String last_name;

    @Column(length = 20)  5 usages
    private String gender;

    @Column(updatable = false)  4 usages
    private LocalDateTime registeration_date;

    // One member can have multiple fitness goals
    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL, orphanRemoval = true)  1 usage
    private Set<FitnessGoal> fitnessGoals = new HashSet<>();

    // One member can have multiple health metrics
    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL, orphanRemoval = true)  1 usage
    private Set<HealthMetric> healthMetrics = new HashSet<>();

    // One member can have multiple training sessions
    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL)  1 usage
    private Set<PersonalTrainingSession> trainingSessions = new HashSet<>();
```

Relationships from the ER model (like one Member having many HealthMetrics, or one Trainer having many TrainingSessions) were expressed using `@OneToMany`, `@ManyToOne`, and `@JoinColumn`. Because of these mappings, Hibernate was able to automatically generate and maintain our relational schema in PostgreSQL without us manually writing the table creation SQL.



```java
private static void generateRoom() {  1 usage  Qin Li *
    Transaction transaction = null;
    Room room = new Room();   // uses the testing constructor (random name + capacity)

    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        transaction = session.beginTransaction();

        session.persist(room);
        transaction.commit();

        System.out.println("\nRoom created successfully:");
        System.out.println("   ID:    " + room.getRoomId());
        System.out.println("   Name:  " + room.getRoomName());
        System.out.println("   Cap:   " + room.getCapacity());

    } catch (Exception e) {
        if (transaction != null) transaction.rollback();
        System.out.println("Error creating room: " + e.getMessage());
    }
}
```

Once the mappings were in place, we used these classes to insert, update, delete, and query data through Hibernate's Session API rather than raw SQL. For example, creating a new member simply required `session.persist(member)` and Hibernate handled the actual INSERT statement. Queries such as searching for members or listing scheduled classes were written using HQL (Hibernate Query Language), which Hibernate converted into efficient SQL behind the scenes. This allowed us to focus on the application logic while Hibernate managed the database interactions, relationships, and transactions. Overall, ORM let us build the system in an object-oriented way while still maintaining full relational database functionality.



```java
Query<Long> query = session.createQuery( s: "SELECT COUNT(m) FROM Member m WHERE m.email = :email", Long.class);
query.setParameter( s: "email", generated_member.getEmail());
```

Which shows all CRUD operations being done through Hibernate, not raw SQL as we wanted.