COMP 3005 - Fall 2025- Project: Health and Fitness Club Management System

Instructor: Abdelghny Orogat

Due: Dec 1, 2025 (11:59 PM) [**No extension**]

1 Problem Statement

The objective of this project is to design and implement a functional **Health and Fitness Club Management System** that operates as a centralized, database-driven platform for managing the daily activities and operations of a modern fitness center. The center offers both **individual personal training sessions** and **group fitness classes**, and the system must be capable of supporting both modalities with appropriate scheduling, capacity management, and trainer coordination. The application will support three distinct categories of users—**members**, **trainers**, and **administrative staff**—each with specialized access privileges and clearly defined functional responsibilities. It will rely on a **relational database (PostgreSQL)** for persistent storage and retrieval of data, with a focus on ensuring consistency, normalization, and data integrity across all functional modules.

From the **member's perspective**, the system should offer a complete experience for managing personal fitness data and interactions with the club's services. A member should be able to register by providing personal information such as name, date of birth, gender, and contact details, and later manage or update these details as needed. Members will have the ability to establish and track personalized fitness goals—such as achieving a target body weight or reducing body fat percentage—and store health metrics like height, weight, heart rate, and other measurable indicators of physical performance. These metrics should not overwrite previous entries but instead be recorded historically so that trends and progress can be analyzed over time. The member interface should also include a personalized dashboard that summarizes the most recent health metrics, progress toward fitness goals, participation in group classes, and upcoming personal training sessions. In addition, members should be able to schedule, reschedule, or cancel **personal training sessions** with certified trainers, provided that the trainer is available during the requested time slot, and they should be able to register for **group fitness classes**, subject to class capacity and schedule constraints. The system must ensure logical enforcement of business rules such as preventing overlapping bookings, ensuring that room capacities are not exceeded, and verifying trainer availability before confirming any reservation.

From the **trainer's perspective**, the system must provide features that allow them to define and manage their work schedules efficiently. Trainers should be able to specify their availability periods—either as recurring weekly slots or as individual time intervals—and update these schedules as needed. The system must prevent overlapping or inconsistent time slots for the same trainer, ensuring data accuracy in availability tracking. Trainers will also need to view their upcoming sessions and class assignments, allowing them to prepare for each session in advance. Additionally, trainers must have controlled access to **member profiles** to review relevant information such as health metrics, progress toward goals, and recent class attendance. This access enables trainers to tailor workout plans and monitor client improvement. However, trainers are not permitted to modify member data or access information unrelated to their assigned clients, preserving data privacy and role boundaries within the system.

From the **administrative staff's perspective**, the system should serve as an operational backbone that supports facility management and service coordination. Administrators will have the capability to manage **room bookings** to ensure that physical spaces—such as studios or training rooms—are properly allocated for classes and personal sessions, while avoiding conflicts in scheduling. They should also be able to oversee **equipment management**, including tracking the operational status of machines, logging maintenance issues, assigning repair tasks, and updating maintenance records once issues are resolved. Furthermore, administrative users are responsible for defining and maintaining the **class schedule**, which includes creating new classes, assigning trainers, setting class capacities, and updating or canceling sessions when necessary. The administrative component should also handle **billing and payment processes**, where the system can generate bills for various services such as membership subscriptions, personal training sessions, or class enrollments. Payments are to be simulated rather than processed through real financial gateways; however, the system should maintain a consistent record of invoices, amounts due, payment methods, and payment status, allowing for complete transaction tracking and financial reporting.

The overall system must maintain clear separation between these user roles, ensuring that each type of user only interacts with data and operations relevant to their responsibilities. In essence, this Health and Fitness Club Management System is expected to emulate a real-world operational workflow, offering a structured, reliable, and role-specific environment that demonstrates strong database design, proper SQL implementation, and logical interaction across all system components.

2 Functional Requirements

You are required to implement a **minimum number of distinct operations** based on your group size, divided across the three user roles: **members**, **trainers**, and **administrative staff**. Each operation must be backed by SQL logic and demonstrated independently. You are **not required to follow a specific sequence**, but each function must work in isolation and reflect real-world constraints (e.g., capacity limits, availability).

2.1 Member Functions

Members interact with the system to manage personal data, monitor progress, and schedule services. At least **4** operations must be implemented.

- **User Registration:** Create a new member with unique email and basic profile info.
- **Profile Management:** Update personal details, fitness goals (e.g., weight target), and input new health metrics (e.g., weight, heart rate).
- **Health History:** Log multiple metric entries; do not overwrite. Must support time-stamped entries.
- **Dashboard:** Show latest health stats, active goals, past class count, upcoming sessions.
- **PT Session Scheduling:** Book or reschedule training with a trainer, validating availability and room conflicts.
- **Group Class Registration:** Register for scheduled classes if capacity permits.

2.2 Trainer Functions

Trainers manage their working hours and access limited member data. At least **2** operations must be implemented.

- **Set Availability:** Define time windows when available for sessions or classes. Prevent overlap.
- **Schedule View:** See assigned PT sessions and classes.
- **Member Lookup:** Search by name (case-insensitive) and view current goal and last metric. No editing rights.

2.3 Administrative Staff Functions

Admins coordinate resources and simulate payments. At least **2** operations must be implemented.

- **Room Booking:** Assign rooms for sessions or classes. Prevent double-booking.
- **Equipment Maintenance:** Log issues, track repair status, associate with room/equipment.
- **Class Management:** Define new classes, assign trainers/rooms/time, update schedules.
- **Billing & Payment:** Generate bills, add line items, record payments. Simulate status updates.

The requirement lacks all the necessary details for building your system. Therefore, you must incorporate these details yourself as you follow the guidelines outlined in the requirements.

---

3 Group-Based Project Requirements

To ensure fairness and appropriate workload, the project deliverables scale with group size. All teams, regardless of size, must design a complete relational database schema, implement a working application, and submit a report and demo video. However, the **number of entities, relationships, and implemented functions** varies by team size.

Minimum Requirements by Group Size

Item	**Solo (1 student)**	**Team of 2**	**Team of 3**
Entities in ER Model	6	8	10
Relationships in ER Model	5	8	10
Application Operations	8 total	10 total	12 total
Mandatory Roles Covered	All 3 roles	All 3 roles	All 3 roles
View + Trigger + Index	1 of each	1 of each	1 of each

---

4 ER Model Requirements

Your ER model must be complete and fully aligned with the **functional requirements** your team plans to implement. For every function you include in your application—such as user registration, class scheduling, or billing—the ER model must contain all the necessary **entities, attributes, and relationships** to support those operations through proper data modeling.

For example, if your system includes **User Registration** and **Profile Management**, your ER model must define a *Member* entity with all relevant attributes (e.g., name, email, contact info), and relationships with other entities like *FitnessGoal* or *HealthMetric* if those are part of the profile. You **cannot hard-code any data** used by these functions; everything must be dynamically retrieved from or stored in your relational database using proper design.

In short, your ER diagram is not just documentation—it defines the **data foundation** of your system. Any function you implement in the application must have full support from the ER model and the relational schema derived from it.

5 ORM Integration Bonus (Optional - 10%)

To encourage deeper engagement with modern software development practices, a **bonus of up to 10%** of the total project mark is available for teams that successfully implement their project using an **Object-Relational Mapping (ORM) library**. This applies to technologies such as **Hibernate (Java)**, **SQLAlchemy (Python)**, **Entity Framework (C#)**, or any equivalent ORM framework appropriate to the programming language you are using.

ORM frameworks allow you to map database tables to application-level classes, enabling database operations to be performed through object manipulation instead of writing raw SQL. Successfully using an ORM demonstrates your ability to design a data-driven application using professional development tools that are widely adopted in industry.

To qualify for the bonus, your submission must meet the following criteria:

- The application must **use the ORM for the majority of database interactions**, including insertions, updates, queries, and deletions. Direct SQL queries may be used in limited cases where ORM support is impractical, but raw SQL should not be the primary method.
- Your codebase must include **clearly defined entity classes** that map to your relational schema (e.g., annotated Java classes if using Hibernate).
- The ORM must correctly handle **relationships between entities** (e.g., one-to-many, many-to-many), **constraints**, and **lazy/eager loading** where applicable.
- Your report should include a **brief section** (1–2 paragraphs) explaining how the ORM was used, along with **code samples** and a list of the major entities that were mapped.
- In your demo video, you must explicitly **show and explain how the ORM interacts with your database**, ideally through a walkthrough of entity definitions and how they are used in the code to perform operations like registration, scheduling, or billing.

The **full 10% bonus** will be awarded only if ORM usage is:

- **Complete** (used throughout the application),
- **Correct** (no major misuse or bypassing of the ORM layer), and
- **Clearly explained** in both the report and the demo video.

Partial bonus (e.g., 6–7%) may be awarded if ORM usage is limited, inconsistent, or not sufficiently demonstrated.

6. Instructions for Submission

All groups must submit a complete project that includes the implementation, report, SQL files (if applicable), and a demonstration video. Your grade will primarily be based on the quality, correctness, and clarity of the **video demonstration**, so make sure it reflects all aspects of your project.

Video Demonstration (Max: 15 minutes)

Your video must clearly and efficiently walk the viewer through the major components of your project. It should be well-paced, clearly narrated, and easy to follow. Each group member must participate and explain part of what they implemented.

The structure below must be followed:

Required Content for All Projects

1. **ER Model**:
   Briefly show your ER diagram and explain the key entities and relationships. Mention any assumptions made during design.

2. **ER to Relational Mapping**:
   Explain how your ER model was translated into relational tables. If using ORM, this means showing how entities and relationships are mapped using classes, annotations, or decorators.

3. **Database Definition**:

   - **If NOT using ORM**:
     Briefly show your `DDL.sql` **and** `DML.sql` files. You do **not** need to go line-by-line—just point out where they are located in the submission folder and show key tables and sample data to demonstrate completeness.
   - **If using ORM**:
     You are **not required to submit DDL or DML files**. Instead, you must:
     - Show the entity class definitions (e.g., `Member.java`, `Session.cs, Class.py`)
     - Explain how these classes are used to create and populate tables
     - Demonstrate how table creation, insertion, and queries are done via the ORM (not via raw SQL)

4. **Functionality Demonstration**:
   Walk through all the operations you have implemented (based on your group size). For each:
   - State the user role (Member, Trainer, Admin)
   - Show how the feature works in the application
   - Include both a **success case** and at least one **failure or edge case**
   - Show the corresponding backend logic:
     - If using SQL directly: show the SQL statement being executed
     - If using ORM: show the Python/Java/C#/etc. code that performs the database operation

5. **Code Structure**:
   Give a quick overview of how your code is organized. Highlight key files (e.g., database connection).

6. **Interface & User Flow**:
   Demonstrate how different users interact with the system (e.g., login flow, navigation across member/admin views), even if simple or CLI-based.

Video Upload Instructions

- Upload your video to a platform such as **YouTube** (unlisted) or **Vimeo**
- Include the link in your project report and **README**
- Ensure that the video is **accessible without sign-in** (test it from a different browser)
- The TAs will **grade your project based solely on the video**, so make sure all major features and interactions are visible and clear

---

Code Submission Requirements

Organize your submission as follows:

If Using SQL + Manual Schema

```
/project-root
  /sql
    DDL.sql        # CREATE TABLE statements with constraints
    DML.sql        # Sample data (at least 3–5 records per table)
  /app             # Application source code
  /docs
    ERD.pdf        # ER diagram + Mapping + Normalization (required)
  README.md        # How to run the project, video link
```

If Using ORM (e.g., Hibernate, SQLAlchemy, EF Core)

```
/project-root
  /models          # ORM entity classes (e.g., Member.java, Class.cs)
  /app             # Application logic and controllers
  /docs
    ERD.pdf        # ER diagram + Mapping + Normalization (required)
  README.md        # How to run the project, video link
```

Note: For ORM-based submissions, `DDL.sql` and `DML.sql` files are **not required**. Instead, the ORM code must clearly demonstrate how the database schema is defined and populated programmatically.

7 Final Project Grading Schema (Out of 100)

Category	Weight	Details
1. Conceptual Database Design (ER Model)	20 marks	Clear ER diagram with entities, attributes, and relationships; correctly reflects real-world use cases and functions; appropriate use of cardinality and participation constraints; avoids redundancy.
2. Mapping ER to Relational Schema	20 marks	Accurate and complete conversion of ER model to relational schemas; all keys and constraints properly defined; handles weak entities and relationships well.
3. Schema Quality and Normalization	10 marks	Tables are normalized (at least to 3NF); no unnecessary redundancy.
4. SQL Code (DDL, DML, Queries)	20 marks	High-quality `DDL.sql` and `DML.sql` (or equivalent ORM code); includes constraints, sample data, and test cases; use of advanced SQL features (e.g., views, triggers, joins).
5. Application Functionality	20 marks	Number and correctness of implemented functions based on group size; includes validation, constraints, and handles edge cases; proper user-role separation.
6. User Interface / CLI Flow	5 marks	Interface is functional, clear, and user-friendly; commands or menus are intuitive and role-specific; no major usability issues.
7. Demo Video Presentation	5 marks	Complete and well-paced walkthrough; demonstrates key components (ERD, mapping, code, functionalities); all members contribute; SQL or ORM code shown clearly.
Total	**100**	

Bonus Marks (Up to +10%)

Object-Relational Mapping (ORM)	+10%	ORM used throughout the project; entity classes fully replace DDL/DML; relationships mapped correctly; demo explains ORM and architecture.
