COMP 3005 Final Project Report

Health and Fitness Club Management System

Solo Project Submission

Fall 2025

Contents

1 Overview

This report documents the architectural design and implementation of the Health and Fitness Club Management System. The application serves as a centralized platform for managing the core operational activities of a fitness center, including member management, class scheduling, and personal training coordination. The system is built using relational database backend (PostgreSQL) interacted with via Python and **SQLAlchemy ORM**.

2 Conceptual Database Design (ER Model)

2.1 Entity Definition and Rationale

The conceptual model identifies 9 core entities necessary to support my system's functional scope.

- **User Entities (Member, Trainer, Admin):** Instead of a single generic user table, three distinct entities were defined. This decision enforces strict role separation at the schema level, allowing each role to maintain distinct attributes (e.g., only Members have health metrics; only Trainers have availability schedules).
- **Operational Entities (Room, Group Class, PTSession):** These entities represent the physical and temporal resources of the club. `Room` is modeled as a distinct entity to enforce capacity constraints and prevent double-booking across different activity types (Classes vs. Sessions).
- **Dependent Entities (HealthMetric, FitnessGoal, Availability):** These are modeled as **Weak Entities** because they dependent on their respective users.
  - *Rationale:* A health metric record (example: "Weight: 150lbs") has no semantic meaning without an associated Member. Modeling these as weak entities ensures that if a user is removed, their associated personal data is automatically cascaded and cleaned up, thsi will help me to maintaining referential integrity for the system.

2.2 Relationships and Cardinality Analysis

Below are the relationships I defined for the interaction logic between system components:

- **Member registers for Group Class (M:N):** Modeled as a Many-to-Many relationship because a single class serves multiple members, and a member can attend multiple classes.
- **Trainer leads Activity (1:N):** Both `Group Class` and `PTSession` have a 1:N relationship with `Trainer`.
- *Rationale:* A trainer can lead many sessions, but a specific session instance is led by exactly one trainer. by applying this constraint, it can help me to prevent scheduling conflicts where a session might be left without an instructor.

- **Room hosts Activity (1:N):** Similar to trainers, a room accommodates many events over time, but a specific event occurs in exactly one room. This is critical for the conflict-checking logic implemented in the application layer.
- **Participation Constraints:**
  - **Total Participation:** Entities like `PTSession` and `GroupClass` have total participation with Trainers and Rooms (a class cannot exist without an instructor or a location).
  - **Partial Participation:** Users (Members/Trainers) have partial participation in activities, allowing the system to register users who have not yet booked a session.

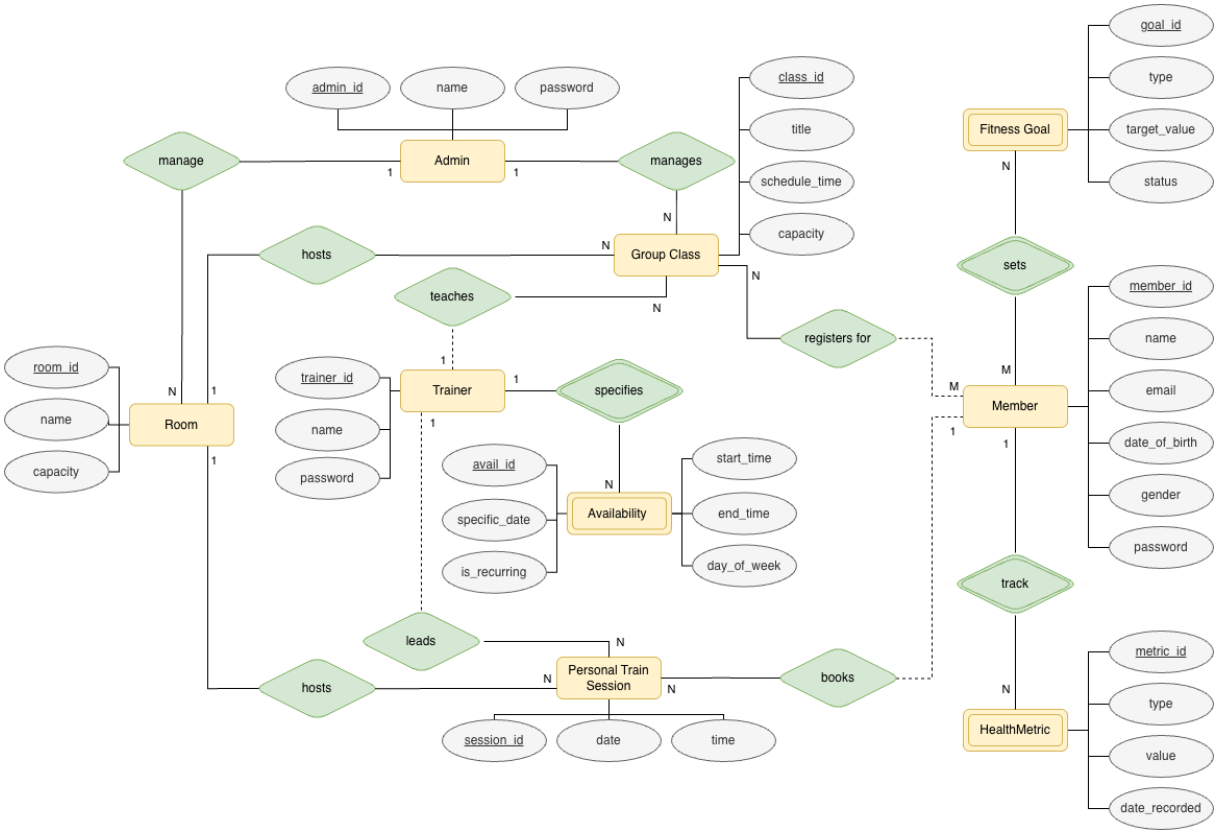

Figure 1: Entity-Relationship Diagram (Chen Notation)

3 Mapping ER to Relational Schema

3.1 ORM Mapping Strategy

The system utilizes an **Object-Relational Mapping (ORM)** strategy via SQLAlchemy (I chosed Python for the program language). This abstracts the raw SQL DDL commands into Python classes, ensuring that the application logic remains tightly coupled with the data definition.

3.2 Transformation Logic

1. **Entity to Table:** Each regular entity in the ER diagram was mapped to a class inheriting from `DeclarativeBase`. Attributes were converted to strongly-typed columns (e.g., `String`, `Integer`, `DateTime`).

2. **Handling Weak Entities:** Weak entities were implemented using a combination of a Foreign Key to the owner and a `relationship()` directive with `cascade="all, delete-orphan"`. This programmatically enforces the "existence dependency" defined in the ER model.

3. **Association Table for M:N:** The M:N relationship between `Member` and `GroupClass` was resolved by creating an association object, `ClassRegistration`. This intermediate table holds foreign keys to both parent tables, adhering to relational design principles.

4 Schema Quality and Normalization

4.1 Normalization Analysis

The schema was rigorously designed to satisfy **Third Normal Form (3NF)** to minimize redundancy and update anomalies.

- **First Normal Form (1NF):** All attributes contain atomic values. The composite `name` attribute from the conceptual design was decomposed into `first_name` and `last_name` columns.

- **Second Normal Form (2NF):** All tables utilize a single-column surrogate Primary Key (ex: `member_id`). This ensures that every non-key attribute is fully functionally dependent on the entire primary key, eliminating partial dependencies.

- **Third Normal Form (3NF):** The schema contains no transitive dependencies. Attributes depend solely on the primary key. For example, room capacity is stored only in the `Room` table and referenced by ID in the session tables, rather than being duplicated in every session record.

4.2 Strategic Design Decision: Vertical Metric Storage

A key design decision was the implementation of `HealthMetric` as a separate entity rather than columns in the `Member` table.

- **Issue:** A flat table design (columns for Weight, Height) would result in a sparse matrix with many NULL values if metrics are not recorded simultaneously. It also limits the system to a fixed set of metrics.
- **Solution:** The vertical design (rows for each measurement) allows for infinite extensibility (new metric types can be added without DDL changes) and accurate historical tracking (each measurement has its own timestamp).

5 View, Trigger, Index Database Features

To enhance performance and data integrity, the following database objects were implemented:

- **View (`v_member_dashboard_stats`):** A virtual table was created to pre-aggregate complex join data (member details + class attendance counts). This simplifies the query logic for the dashboard and improves read performance.
- **Trigger (`check_room_capacity`):** A database-level trigger was implemented to enforce the business rule that class registrations cannot exceed room capacity. This ensures data integrity at the database level, preventing race conditions that might occur in the application layer.
- **Index (`idx_room_date_time`):** A composite index was applied to the `PTSession` table on columns (`room_id`, `date`, `start_time`). This optimizes the performance of conflict-checking queries, which are executed frequently during the booking process, further more the booking process is a central core transaction of fitness system, therefore it is necessary to optimize.

6 Project Scope

These specific functional modules were excluded from my ER model and implementation scope :

- **Equipment Maintenance:** The tracking of machine status was omitted to focus on the core scheduling and user management workflows.
- **Billing System:** Invoice generation and payment processing were excluded to prioritize the complexity of the booking conflict logic.