COMP 3005:

Seydi Cheikh Wade

101323727

Project Report:

Health and Fitness Club Management System

Group Size (Solo)

Table of contents

1. System Scope and Functional Requirements:

1.1. System Overview:

1 The Health and Fitness Club Management System is a centralized, database-driven application that supports the day-to-day operations of a modern fitness center. The club offers both **individual personal training sessions** and **group fitness classes**, and the system is responsible for coordinating schedules, enforcing room capacities, and managing trainer assignments in a consistent way.

2 The application is built on top of a **relational database (PostgreSQL)**, which stores all persistent data related to members, trainers, rooms, equipment, sessions, and other operational records. The design emphasizes **normalization, data integrity, and enforcement of business rules** such as preventing double-booking of rooms or trainers and retaining a historical record of health metrics and other operational events..

3 The system distinguishes three main user roles with different access privileges:

- **Members**, who manage personal information, fitness goals, health metrics, and bookings.
- **Trainers**, who manage their availability and review relevant member data to prepare for sessions.
- **Administrative staff**, who coordinate room allocation and equipment maintenance, and (in the full specification) oversee class schedules and billing activities.

4 This solo implementation focuses on a **core subset of these capabilities** that still reflects a realistic club workflow and provides enough complexity to demonstrate solid database design and ER modeling.

1.2. Implemented Functions by Role (This Project):

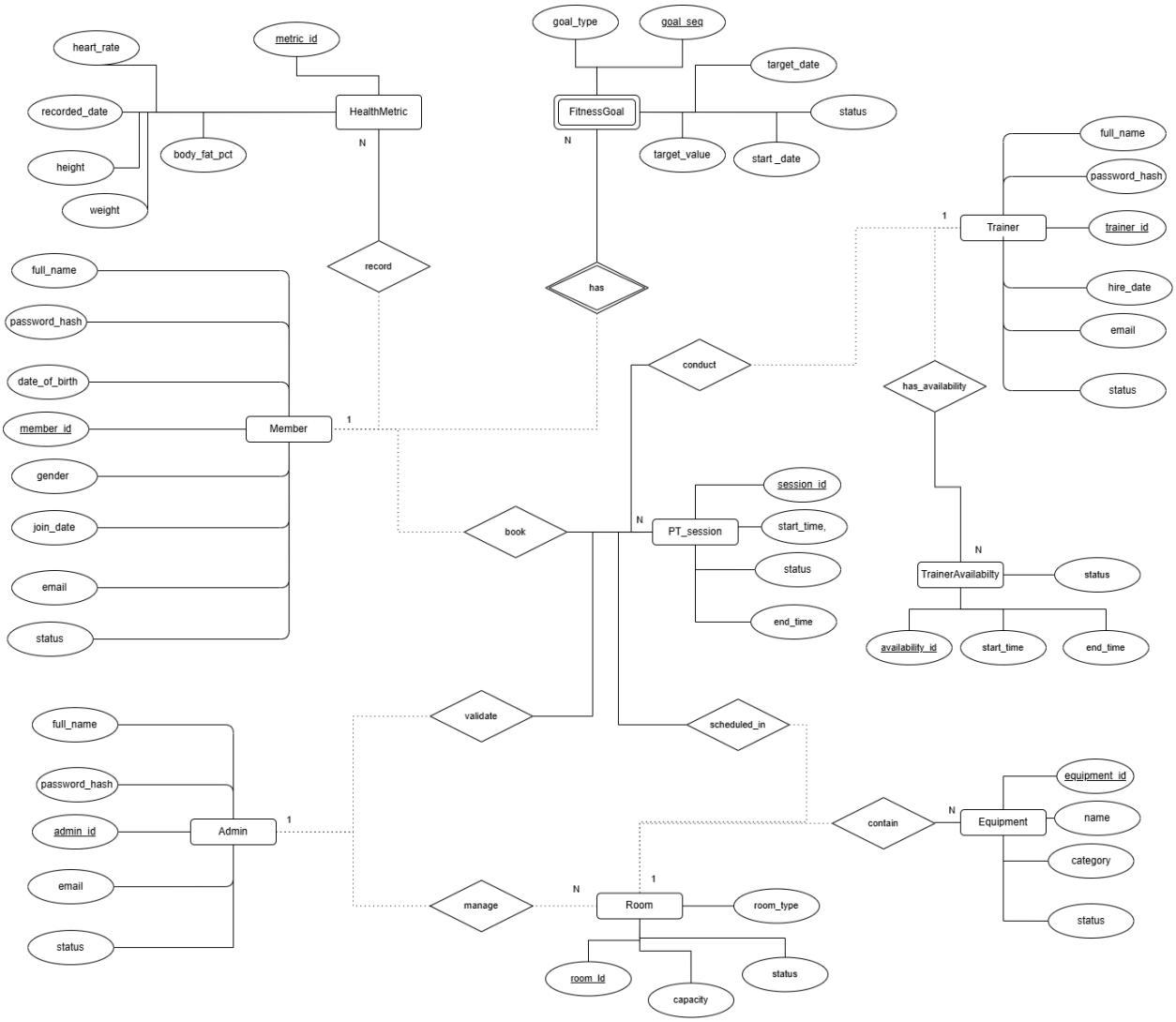
Table 1 summarizes the concrete operations that will be implemented in the application and, therefore, must be supported by the ER model and relational schema.

Table 1: Implemented Functional Requirements

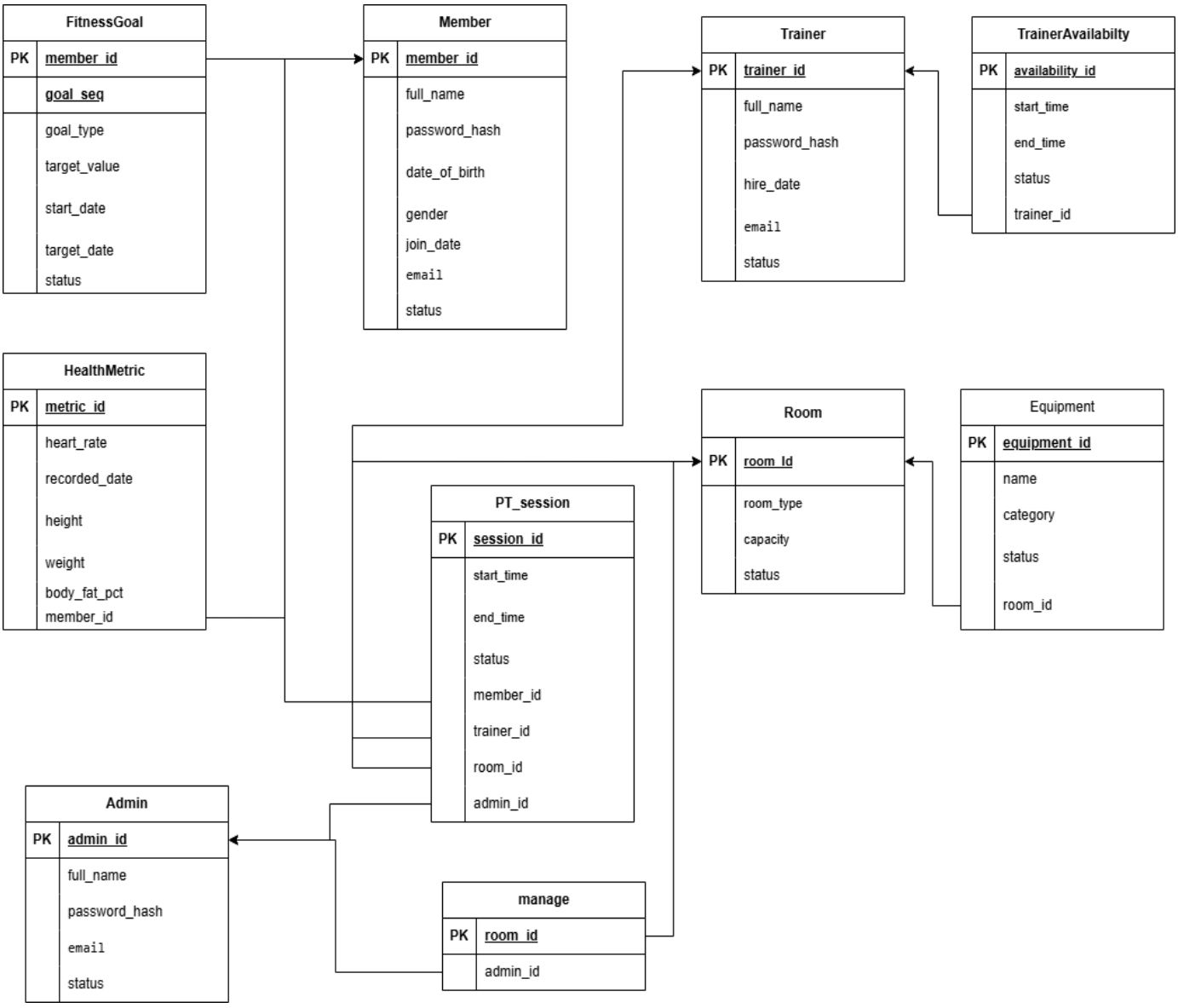
Role	Requirement ID	Operation name	Description
Member	M1	User Registration	Create a new member with unique email and basic profile info.
Member	M2	Profile Management	Update personal details, fitness goals (e.g., weight target), and input new health metrics (e.g., weight, heart rate).
Member	M3	Health History	Log multiple metric entries; do not overwrite. Must support time-stamped entries.
Member	M4	PT Session Scheduling	Book or reschedule training with a trainer, validating availability and room conflicts.
Trainer	T1	Set Availability	Define time windows when available for sessions or classes. Prevent overlap.
Trainer	T2	Schedule View	See assigned PT sessions and classes.
Admin	A1	Room Booking Management	Assign rooms for sessions or classes. Prevent double-booking
Admin	A2	Equipment Maintenance	Log issues, track repair status, associate with room/equipment

These eight operations form the implemented subset for this solo project; other features from the full problem statement (e.g., detailed dashboards, full group class lifecycle, and billing) are considered out of scope.

2. Conceptual Database Design (ER Model):



3. Mapping ER to Relational Schema:



4. Schema Quality and Normalization:

All relations in the schema were derived from the ER model using the standard ER-to-relational mapping rules. As a result, the tables are already in **3rd Normal Form (3NF)** and in most cases also satisfy **BCNF**.

- **1NF**: Every attribute stores atomic values only. Repeating information (fitness goals, health metrics, trainer availability slots, equipment items, PT sessions) is stored in separate tables (FITNESS_GOAL, HEALTH_METRIC, TRAINER_AVAILABILITY, EQUIPMENT, PT_SESSION), so there are no multivalued attributes.
- **2NF**: The only relation with a composite key is FITNESS_GOAL(member_id, goal_seq). All non-key attributes (goal_type, target_value, start_date, target_date, status) depend on the **whole** key (member_id, goal_seq), not on just one component. The other relations have single-attribute primary keys, so 2NF holds trivially.
- **3NF**: In every relation, non-key attributes depend **directly** on the key and not on other non-key attributes. For example, in PT_SESSION(session_id, member_id, trainer_id, room_id, admin_id, start_time, end_time, status) all attributes describe the session identified by session_id; there are no dependencies such as room_id → status or category → status in EQUIPMENT. Similarly, MEMBER, TRAINER, ADMIN, HEALTH_METRIC, TRAINER_AVAILABILITY, ROOM, and MANAGE have a single key and no non-key→non-key functional dependencies.

Because there are no partial or transitive dependencies in any table, **no further decomposition is required**. The schema therefore avoids unnecessary redundancy and helps prevent update, insertion, and deletion anomalies.

5. Use of ORM and Entity Mapping (Hibernate):

The implementation uses **Hibernate** as an Object–Relational Mapping (ORM) framework to bridge the gap between the Java domain model and the PostgreSQL database. Instead of issuing raw SQL for every operation, the application manipulates persistent objects such as `Member`, `PTSession`, and `TrainerAvailability`, while Hibernate translates these operations into the appropriate `INSERT`, `UPDATE`, `DELETE`, and `SELECT` statements. Each domain class is annotated with `@Entity` and `@Table`, and attributes are mapped to columns using `@Column`, `@Id`, and `@GeneratedValue`. Relationships are captured explicitly using `@ManyToOne`, `@OneToMany`, and composite identifiers, which keeps the codebase closely aligned with the ER model and reduces the risk of inconsistencies between the object model and the relational schema.

A representative example is the mapping of the **Member** and **PTSession** entities. A member can participate in many PT sessions, while each session is linked to exactly one member and one trainer. This is implemented by storing foreign keys (`member_id`, `trainer_id`, `room_id`, `admin_id`) and mapping them to object references using `@ManyToOne`:

```
@Entity
@Table(name = "member")
public class Member {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "member_id")
    private Long memberId;

    @Column(name = "full_name", nullable = false)
    private String fullName;

    @Column(name = "email", nullable = false, unique = true)
    private String email;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    // ... dateOfBirth, gender, joinDate, status, getters/setters
}
```

```
@Entity
@Table(name = "pt_session")
public class PTSession {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "session_id")
    private Long sessionId;

    @ManyToOne(optional = false)
    @JoinColumn(name = "member_id", nullable = false)
    private Member member;

    @ManyToOne(optional = false)
    @JoinColumn(name = "trainer_id", nullable = false)
    private Trainer trainer;

    @ManyToOne
    @JoinColumn(name = "room_id")
    private Room room;        // assigned by admin

    @ManyToOne
    @JoinColumn(name = "admin_id")
    private Admin admin;      // admin who validated the booking

    @Column(name = "start_time", nullable = false)
    private LocalDateTime startTime;
```

```
    @Column(name = "end_time", nullable = false)
    private LocalDateTime endTime;


    @Column(name = "status", nullable = false)
    private String status;    // PENDING, VALIDATED, RESCHEDULED, CANCELLED,  ...
}
```

The **major entities** mapped via Hibernate are:

- Member – club member profile and authentication data.
- Trainer – personal trainers and their employment status.
- Admin – administrative staff responsible for rooms and equipment.
- Room – physical training rooms and studios.
- Equipment – machines and devices located in rooms.
- PTSession – personal training sessions linking a member, trainer, room, and admin.
- TrainerAvailability – time slots when trainers are available for booking.
- HealthMetric – historical health and fitness measurements per member.
- FitnessGoal and FitnessGoalId – member goals, modeled as a weak entity with a composite key (member_id, goal_seq).
- Manage – association entity linking Admin to Room to capture which admin manages which room.

By using Hibernate to map these entities and their associations directly to the relational schema, the system can enforce business rules (such as foreign-key relationships and status transitions) at the object level while still benefiting from the robustness and integrity guarantees of the underlying PostgreSQL database.