COMP 3005 Final Project Report

Trisha Toocaram, 101267603

Video link: Here

1. ER Model

The ER model for this project is included in **ERD.pdf**. It contains all the required entities for the three user roles defined in the project:

Key Entities

- **Member** – personal information, health metrics, fitness goals, class participation.
- **Trainer** – instructor information, availability, assigned sessions/classes.
- **AdminStaff** – manages rooms, equipment, and maintenance.
- **Room** – locations where sessions/classes take place.
- **FitnessGoal** – member's goal type, target, and timeline.
- **HealthMetric** – timestamped weight/HR/body fat entries.
- **PersonalTrainingSession** – links member, trainer, room, and time.
- **FitnessClass** + **ClassRegistration** – class offerings and member registrations.
- **TrainerAvailability** – recurring weekly time availability.
- **Equipment** + **EquipmentMaintenance** – equipment status and issue tracking.

Key Relationships

- Member → FitnessGoal, HealthMetric (1 to many)

- Member ↔ FitnessClass (many to many via ClassRegistration)
- Trainer → PersonalTrainingSession, FitnessClass (1 to many)
- Trainer → TrainerAvailability (1 to many)
- Room → FitnessClass, PersonalTrainingSession, Equipment (1 to many)
- Equipment → EquipmentMaintenance (1 to many)

Assumptions

- Member, Trainer, and AdminStaff are kept as separate entities for clarity.
- No derived attributes are stored; all metrics are logged historically.
- All scheduling conflicts (trainer availability, room booking) are enforced at the application level.

2. ER to Relational Mapping

Each entity becomes a relational table. Every relationship is implemented via foreign keys in the ORM models such as

- FitnessGoal(member_id → Member.member_id)
- HealthMetric(member_id → Member.member_id)
- PersonalTrainingSession(member_id, trainer_id, room_id)
- FitnessClass(trainer_id, room_id, created_by_admin_id)
- ClassRegistration(class_id, member_id) with UNIQUE(class_id, member_id)

Many-to-many relationships (Member–Class) are mapped using a separate join table (ClassRegistration).

All participation constraints from the ERD (e.g., goals must belong to a member) are enforced using nullable=False on foreign keys.

3. Schema Quality & Normalization

The system is fully normalized to 3NF:

- All attributes are atomic (no repeating groups).
- All non key attributes depend solely on the primary key of their table.
- No transitive dependencies.
- No redundant storage (eg. trainer name is not copied into sessions).
- Many to many relationships avoid redundancy using a join table.

4. Database Definition Using ORM

4.1 Entity Class Definitions

All tables are defined in the **models/** folder using SQLAlchemy **declarative_base()**. Below is an examples for **Member**:

```
class Member(Base):
  __tablename__ = "member"
  member_id = Column(Integer, primary_key=True)
  email = Column(String, unique=True, nullable=False)
  first_name = Column(String, nullable=False)
  last_name = Column(String, nullable=False)
```

Each model includes its foreign keys and relationship() definitions to reflect the ERD.

4.2 Table Creation (DDL via ORM)

In **create_db.py**:

```
Base.metadata.create_all(bind=engine)
```

This automatically generates all relational tables.

4.3 Insertions & Queries (DML via ORM)

Example **insert (seeding)**:

```
session.add(Member(first_name="Mia", email="mia@club.com", password_hash="pass"))
session.commit()
```

Example **query (dashboard)**:

```
latest = session.query(HealthMetric).filter_by(member_id=m_id).order_by(
   HealthMetric.recorded_at.desc()
).first()
```

All CRUD logic in the application uses ORM instead of raw SQL.

5. Functionality Demonstration Overview

The application implements the required number of 8 operations.

Member

1. Register a new member
2. Update profile
3. Add health metric
4. View dashboard

Trainer

5. Set weekly availability
6. View schedule of assigned sessions

Admin

7. Log equipment issue
8. Resolve equipment issue or view equipment by room

Each operation has:

- a success case (eg. metric added)
- an edge case (eg. duplicate email, trainer not found, resolving non-existing issue)

All operations are shown in the demo video through the terminal menu.

6. Code Structure

```
app/
  main.py             # main program + menus
  member_service.py     # Member operations
  trainer_service.py    # Trainer operations
  admin_service.py      # Admin operations
  db.py               # DB connection + session

models/
  base.py             # declarative_base()
  *.py                # all ORM entity classes (all included in README.md)

create_db.py          # create tables
seed.py           # sample data

docs/
3005projectreport.pdf   # this file
ERD.pdf               # data model
README.md             # run instructions + other details
```

The code is separated clearly into:

- **Model layer** (ORM definitions)
- **Service layer** (role-based features)
- **Database layer** (connection/session)

7. Interface & User Flow

The system uses a simple CLI driven interface.

Main menu:

1. Member
2. Trainer
3. Admin Staff
0. Exit

Each user selects their role and is taken to a role specific menu.