Entity Relation Diagram Description

Entities and Attributes:

Member:
Gym member who can register, manage profile, record health metrics, and join classes.
Primary key = member_id
Attributes: member_id, first_name, last_name, email (unique), phone, goal_description

Trainer:
Trainer who leads group classes, can view limited member info.
Primary key = trainer_id
Attributes: trainer_id, first_name, last_name, email (unique), phone, specialty

Room:
Physical room/studio where classes are held.
Primary key = room_id
Attributes: room_id, capacity

FitnessClass:
Scheduled group fitness class.
Primary key = class_id
Attributes: class_id, class_name, trainer_id, room_id, start_time, end_time, capacity

ClassRegisteration:
Registration of a member in specific class.
Primary key = (member_id, class_id)
Attributes: member_id, class_id, registered_at

HealthMetric:
Recorded health metric for a member at a certain time.
Primary key = metric_id
Attributes: metric_id, member_id, recorded_at, weight_kg, resting_heart_rate

Relationships:

Member-HealthMetric: One Member can have many HealthMetric records; each HealthMetric belongs to exactly one Member. Cardinality: Member (1) to HealthMetric (M).

Member-ClassRegisteration-FitnessClass: Members and FitnessClasses are in a many-to-many relationship, implemented via ClassRegistration. One Member can appear in many ClassRegistrations; one FitnessClass can appear in many ClassRegistrations.

Trainer-FitnessClass: One Trainer can lead many FitnessClasses; each FitnessClass is led by exactly one Trainer.Cardinality: Trainer (1) to FitnessClass (M).

Room-FitnessClass: One Room can host many FitnessClasses over time; each FitnessClass is assigned to exactly one Room. Cardinality: Room (1) to FitnessClass (M).

Room booking rule: Two FitnessClasses cannot use the same Room at overlapping times; this is enforced in the application/SQL checks when creating or updating classes.

Relational Schema (Mapping from ER Model)

Member table: Stores info of member
Member(member_id, first_name, last_name, email, phone, goal_description)

- Primary key = member_id
- Constraints: email is unique

Trainer table: Stores info of trainer
Trainer(trainer_id, first_name, last_name, email, phone, speciality)

- Primary key = trainer_id
- Constraints: email has to be unique to

Room table: Stores rooms where classes are held
Room(room_id, capacity)

- Primary key = room_id
- No constraints

FitnessClass Table: Stores scheduled fitness classes.
FitnessClass(class_id, class_name, trainer_id, room_id, start_time, end_time, capacity)

- Primary key = class_id
- Foreign keys = trainer_id → Trainer(trainer_id), room_id → Room(room_id)

ClassRegistration table : Stores registrations of members into classes.
ClassRegistration(member_id, class_id)

- Primary key = (class_id, member_id)
- Foreign keys = member_id → Member(member_id), class_id → FitnessClass(class_id)

HealthMetric Table: Stores records of health metrics for members.
HealthMetric(metric_id, member_id, recorded_at, weight_kg, resting_heart_rate)

- Primary key = metric_id
- Foreign keys = member_id → Member(member_id)

Normalization (2Nf and 3NF)

A)

Consider the relation:
Enrollment(member_id, class_id, member_name, email, phone, goal_description, class_name, trainer_id, trainer_name, room_id, room_capacity, start_time, end_time, registered_at) with the following functional dependencies:
member_id → member_name, email, phone, goal_description
trainer_id → trainer_name
room_id → room_capacity
class_id → class_name, trainer_id, room_id, start_time, end_time
(member_id, class_id) → registered_at
The primary key is (member_id, class_id), since that uniquely identifies each enrollment (one member in one class). There are partial dependencies because some attributes depend only on part of the composite key:
member_id → member_name, email, phone, goal_description
class_id → class_name, trainer_id, room_id, start_time, end_time
These violate 2NF, so we decompose Enrollment into:
Member(member_id, member_name, email, phone, goal_description)
FitnessClass(class_id, class_name, trainer_id, room_id, start_time, end_time)
ClassRegistration(member_id, class_id, registered_at)
Now in ClassRegistration, the non-key attribute registered_at depends on the whole key (member_id, class_id), and Member and FitnessClass have no partial dependencies on a composite key, so this decomposition is in 2NF.
B)
Next we check for transitive dependencies in the 2NF relations. In FitnessClass we have:
class_id → trainer_id, room_id
trainer_id → trainer_name
room_id → room_capacity
This shows trainer_name depending on class_id through trainer_id, and room_capacity depending on class_id through room_id. These are transitive dependencies and violate 3NF.
To remove them, we decompose further into:
Trainer(trainer_id, trainer_name)
Room(room_id, room_capacity)
FitnessClass(class_id, class_name, trainer_id, room_id, start_time, end_time)
Together with the other tables, the final schema is:

Member(member_id, member_name, email, phone, goal_description)
Trainer(trainer_id, trainer_name)
Room(room_id, room_capacity)
FitnessClass(class_id, class_name, trainer_id, room_id, start_time, end_time, capacity)
ClassRegistration(member_id, class_id, registered_at)ffoadjfoajd
HealthMetric(metric_id, member_id, recorded_at, weight_kg, resting_heart_rate)
In this final design, each non-key attribute in every table depends on the whole key, and not transitively on another non-key attribute, so the schema is in 3NF. So the schema stayed the same.