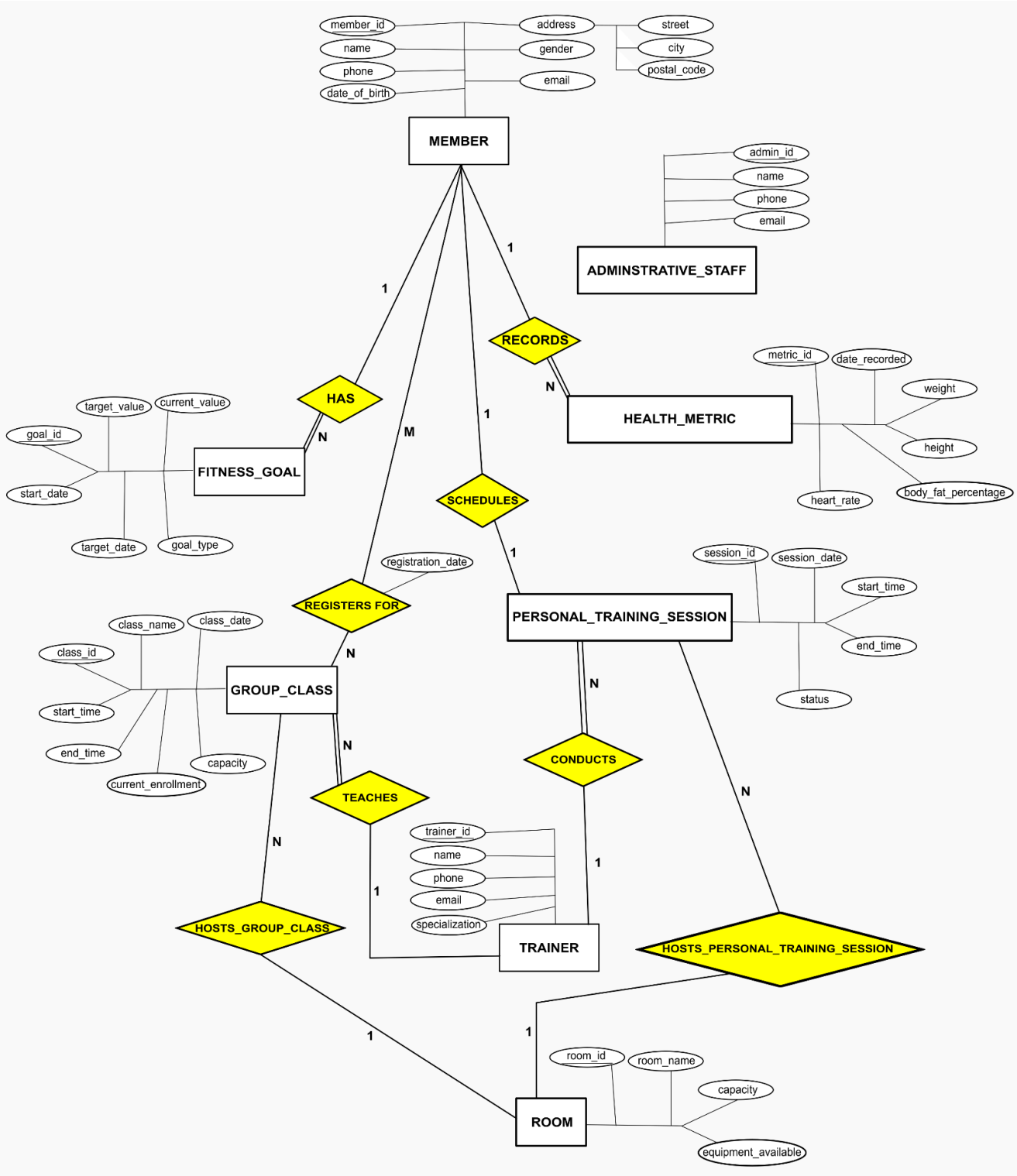
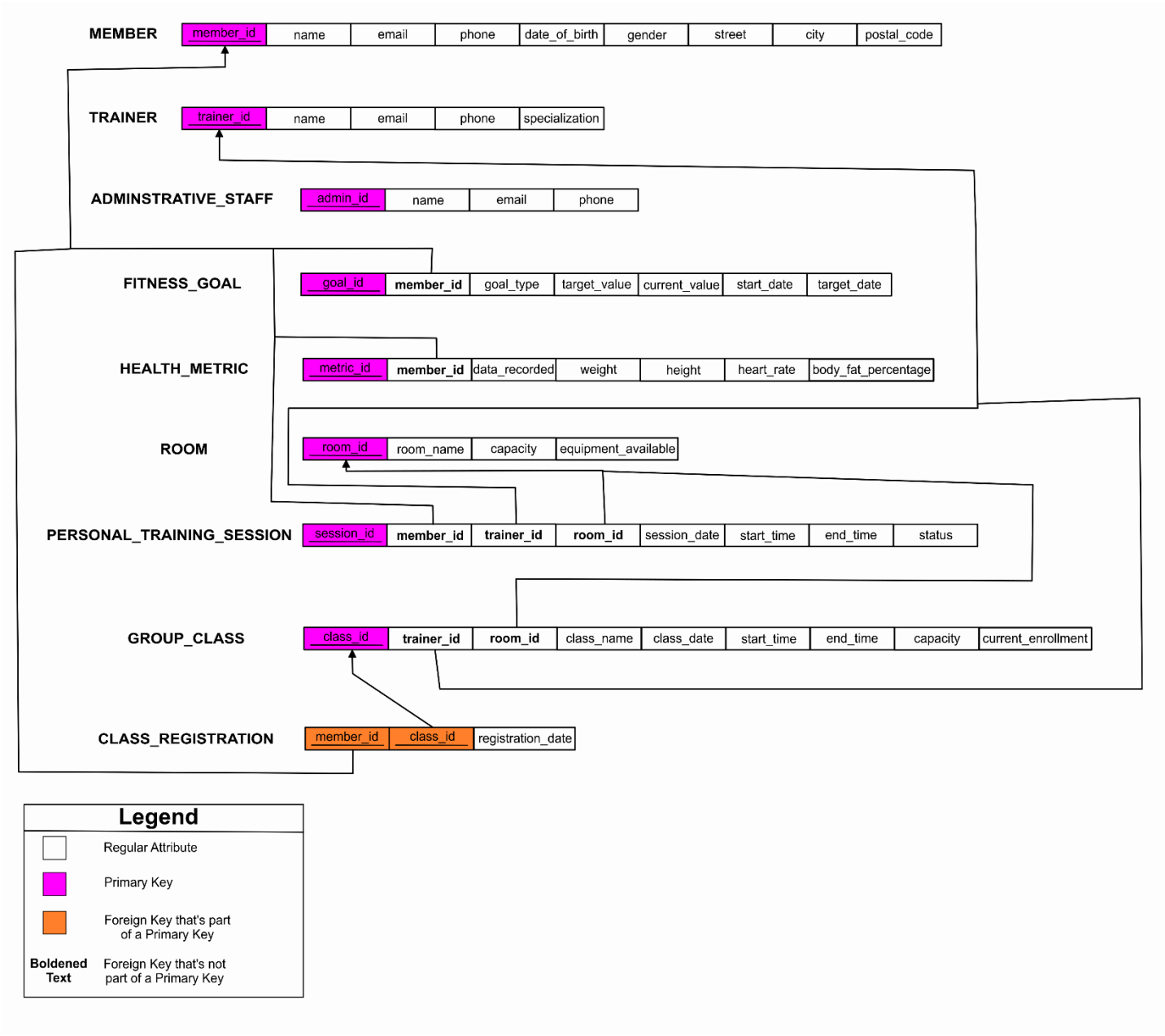
Health and Fitness Club Management System (Final Project for COMP3005A)

By: Badr Ahmed (#101226464) & Faris Ahmed (#101142716)

ER Diagram:



Relational Schema:



Mapping Table:

Requirement	Assumption	Representation in ER Model	Representation in Relational Schema
Members need to register with personal information including name, contact details, and address.	Each member is uniquely identified by member_id	MEMBER Entity: member_id (Primary Key), name, email, phone, date_of_birth, gender, address (Composite Attribute: street, city, postal_code)	MEMBER Table: member_id (Primary Key), name, email (Unique), phone, date_of_birth, gender, street, city, postal_code (from composite address)
Trainers work at the club and have specializations.	Each trainer is uniquely identified by trainer_id	TRAINER Entity: trainer_id (Primary Key), name, email, phone, specialization	TRAINER Table: trainer_id (Primary Key), name, email (Unique), phone, specialization
Administrative staff manage operations.	Each admin is identified by admin_id	ADMINISTRATIVE_STAFF Entity: admin_id (Primary Key), name, email, phone	ADMINISTRATIVE_STAFF Table: admin_id (Primary Key), name, email (Unique), phone
Members can set multiple fitness goals.	One member can have many goals, each goal belongs to one member	FITNESS_GOAL Entity: goal_id (Primary Key), goal_type, target_value, current_value, start_date, target_date. HAS Relationship: MEMBER (1) to FITNESS_GOAL (N), total participation on FITNESS_GOAL side because every goal has to belong to someone.	FITNESS_GOAL Table: goal_id (Primary Key), member_id (Foreign Key references MEMBER), goal_type, target_value, current_value, start_date, target_date. The HAS relationship became member_id foreign key in this table. We put it here because it's the 'many' side of the 1:N relationship, which lets one member have multiple goals.

Members record health metrics over time for progress tracking.	One member can record many metrics, each metric belongs to one member	HEALTH_METRIC Entity: metric_id (Primary Key), date_recorded, weight, height, heart_rate, body_fat_percentage. RECORDS Relationship: MEMBER (1) to HEALTH_METRIC (N), total participation on HEALTH_METRIC side so every metric entry belongs to a member.	HEALTH_METRIC Table: metric_id (Primary Key), member_id (Foreign Key references MEMBER), date_recorded, weight, height, heart_rate, body_fat_percentage. Same idea as FITNESS_GOAL, the RECORDS relationship became a foreign key in the HEALTH_METRIC table. This way we can store multiple metric entries for each member over time.
Rooms are used for sessions and classes.	Each room is identifie d by room_id	ROOM Entity: room_id (Primary Key), room_name, capacity, equipment_available	ROOM Table: room_id (Primary Key), room_name, capacity, equipment_available
Members schedule personal training sessions with trainers in specific rooms.	One member schedule s one session at a time, one trainer can conduct many sessions , one room can host many sessions	PERSONAL_TRAINING_SESSI ON Entity: session_id (Primary Key), session_date, start_time, end_time, status. SCHEDULES: MEMBER (1) to SESSION (1). CONDUCTS: TRAINER (1) to SESSION (N), total participation on SESSION side. HOSTS_PERSONAL_TRAINING _SESSION: ROOM (1) to SESSION (N)	PERSONAL_TRAINING_SESSI ON Table: session_id (Primary Key), member_id (Foreign Key references MEMBER), trainer_id (Foreign Key references TRAINER), room_id (Foreign Key references ROOM), session_date, start_time, end_time, status. This table has three foreign keys because three different relationships from the ER diagram all point to it. The member_id comes from SCHEDULES, trainer_id comes from CONDUCTS, and room_id comes from HOSTS_PERSONAL_TRAINING _SESSION.
Members can register for multiple	Many member s can register	REGISTERS_FOR Relationship: M:N between MEMBER and GROUP_CLASS. Relationship attribute: registration_date	CLASS_REGISTRATION Table: member_id (Foreign Key references MEMBER, part of composite Primary Key),

group classes, and classes can have multiple members. We need to track when each member registered.	for many classes, many classes can have many members	stores when the member signed up	class_id (Foreign Key references GROUP_CLASS, part of composite Primary Key), registration_date. Since REGISTERS_FOR was a many-to-many relationship, we had to create a whole new junction table for it. The primary key is made from both member_id and class_id together, which makes sure a member can't register twice for the same class. The registration_date attribute from the relationship diamond ended up as a column in this table.

Normalization:

We made sure our database is normalized to Third Normal Form (3NF) like we learned in the course.

Here's how we checked each level: First Normal Form (1NF): The main thing for 1NF is that every column has to contain only one value (no lists or multiple items in a single cell). When we were designing our tables, we noticed that the 'address' attribute from our ER diagram was actually made up of multiple parts (street, city, postal code). So we split it into three separate columns to make sure each one only stores one piece of data. We also double-checked that none of our tables have any repeating columns.

Second Normal Form (2NF): 2NF is about getting rid of partial dependencies. Basically, if a table has a composite primary key (made from multiple columns), then all the other columns need to depend on the whole key, not just part of it.

Most of our tables use single-column primary keys (like member_id), so they're automatically good for 2NF. The one table we had to check more carefully was CLASS_REGISTRATION because it uses both member_id and class_id together as the primary key. We looked at the registration_date column and confirmed it really does need both IDs - you need to know which member and which class to figure out when that registration happened. So it passes 2NF.

Third Normal Form (3NF): 3NF means making sure that regular columns only depend on the primary key, and not on each other. We went through our tables one by one to check this. For example, in the MEMBER table, things like name and email only depend on the member_id - they don't depend on each other. Same thing for all our other tables. We didn't find any situations where one regular column determines another regular column, so we're good for 3NF.

Avoiding Redundancy: One of the big reasons for normalization is to avoid storing the same data multiple times. We made sure to do this by using foreign keys instead of copying information. For instance, when a member schedules a personal training session, we don't

copy their name and email into the session record - we just store their member_id and link back to the MEMBER table. That way if they update their email, it updates everywhere automatically.

We also decided not to store any calculated fields. Like, we could have put a "number of classes attended" column in the MEMBER table, but that number can just be counted from the CLASS_REGISTRATION table whenever we need it, so there's no point storing it.