Textual Explanation Of the Relationships in ER diagram:

- Relationship between **Member entity and Personal Training Session** entity called “scheduling”. This encompasses scheduling, rescheduling and cancelling sessions. This is because rescheduling and cancelling are the same but instead of creating an entry we just modify it or remove it. Since this is a SQL query it doesnt need to be shown in the ER. This is 1 to N because a member can schedule multiple sessions, and each session has 1 member. Total participation for session and partial for member.
- Relationship between **Member entity and Event** called “Schedule” is for scheduling events. A member can schedule multiple events, and an event is scheduled by multiple members, so it's N to M. Total participation for event because every event has members, but partial participation for Member because not every member schedules an event
- Relationship between **Personal Training session and Trainer** “Teach” is 1 to N because a trainer can teach multiple sessions, and each session is taught by 1 trainer. Total participation for Personal Training Session because every session must have a trainer, but partial for Trainer because not every trainer has to teach a session
- Relationship between **Event and Trainer** “teach” is 1 to N, because a trainer can teach many events, and each event is taught by 1 trainer. Total participation for Event, partial for Trainer.
- Relationship between **Trainer and Member** “Views” is N to M because a trainer can view multiple member profiles, and a Member can be viewed by multiple trainers. Partial on both sides because not every Members profile has to be viewed, and not every trainer has to view profiles.
- Relationship between **Event and Room** called “In” is 1 to N because an event can be in 1 room, and a room can have multiple events. Total participation for Event, partial for Room.
- Relationship between **Admin Staff and Events** “Manage Schedule” is for managing event schedules as required in the requirements. Its 1 to N because a staff member can manage Many Events, and each event is managed by 1 staff member. Partial participation for Admin Staff, total for Events
- Relationship between **Admin Staff and Events** “Manage Rooms” is for managing rooms. The reason I chose this approach is because an event takes place in a room, so its easier to associate rooms with events. The other approach would be to make a relationship between Admin Staff and Room which means that rooms can be used for other things, and harder to associate rooms with specific events. Its 1 to N because an admin staff can manage multiple rooms, each room managed by 1 staff. Partial participation for Admin Staff, total for Events

- Relationship between **Admin Staff and Fitness Equipment maintenance** “Manage”. 1 to N because a maintenance can be managed by 1 staff member, and a staff member can manage multiple fitness maintenance. Total participation for Equipment Maintenance, Partial for Admin Staff.
- Relationship between **Admin Staff and Payment** “Oversee” is 1 to N because an Admin staff can oversee multiple payments, and each payment can be overseen by 1 admin staff.

- Relationship between **Admin Staff and Quality Assurance** “Monitor” is 1 to N because each staff member can monitor multiple QA issues, and each QA instance is monitored by 1 staff member. Admin staff is partial because not every member has to monitor QA, but each QA is monitored by a staff.
- Relationship between **Admin Staff and Billing** “Manage” is 1 to N because each Bill is managed by 1 Admin Staff and each staff can manage multiple billings. Partial for Staff because not every staff has to manage billings. Billings is Total because every billing has to be managed by a member. When the project specs say manage, I interpret that as creating the bills because a bill has to be created by staff.
- Relationship between **Billing and Personal training Session** “BilledFor” is to represent that a bill can be for a Personal training session. Its 1 to 1 because a bill can be for 1 session, and each session is billed for once. *Since we have the option to add nulls or make a new table, for simplicity we’ll add nulls, so foreign key that references SessionID*

- Relationship between **Billing and Events** “billedFor” is to represent that a bill can be for events. Its 1 to 1 because a bill can be for 1 event, and each event is billed for only once. *Since we have the option to either add nulls or make new table, for simplicity we’ll add nulls, so add a foreign key to Billing that references EventID*

- Relationship between **Billing and Member** “Receives” to show that a member receives a bill. 1 to N because Member can receive many bills, and bill is sent to only 1 member. This is represented as creating a Bill for a specific service in the Billing table. Total for both because every Bill is sent to Member, and every member receives a bill.

- Relationship between **Billing and Payments** “madeFor” to show that a payment is made FOR a billing. Its a 1 to 1 relationship because a payment is made for 1 bill, and a bill can be paid for ONCE. Total participation on both sides because every bill is paid for, and every payment is FOR a bill

- Relationship between **Payments and Loyalty Points** “EarnAndRedeem” this is one relationship that encompasses both earning and redeeming points. The reason its combined is because earning and redeeming is just adding or subtracting a value, its essentially the same process but different operator (+ vs -). 1 to 1 because a payment modifies a single loyalty points entry (incrementing or decrementing numPoints), and each loyalty points entry can be changed by multiple payments

- Relationship between **Payments and Member** “Makes” for a member making a payment. 1 to N, because a Member makes multiple payments, and each payment is made by 1 member.
- Relationship between **Member and Loyalty points** “has” for the fact that a member has loyalty points. In tables, this will be represented as a table that contains the memberID and the LoyaltyID.

Entities in the ER:

- The **Member** entity has all the attributes and an attribute called joinDate to store the date the person joined so that every year at this date we can bill the Member for yearly membership. Primary key is MemberID.
- **Event** has all the attributes for itself and the attributes for all the child entities. Its primary is EventID. It has RoomID and TrainerID as foreign keys, RoomID referencing the RoomID attribute in Room entity, and TrainerID referencing TrainerID in Trainer entity for the Trainer that taught the specific Event.
- **Trainer** Entity has all the attributes listed in the specs (Fname, Lname, TrainerID, Phone number, email). Primary key is TrainerID and no Foreign keys
- **Personal Training session** entity has SessionID (Primary key), Date, Time, Duration, Progress Notes, MemberID (foreign key to Member Entity) for the member that registered, TrainerID (Foreign key to Trainer Entity) for the Trainer teaching this session
- **Room** Entity has R_ID and Available attributes, and StaffID . Available is a boolean set to False if booked, True otherwise. R_ID is the primary key
- **Admin Staff** Entity has StaffID (Primary key), Fname, Lname, Email, Phone number
- **Fitness Equipment Maintenance** has MaintenanceID (Primary key), EquipmentName, Date, Description, and StaffID (Foreign key to Admin Staff table for the staff member who made the maintenance record)
- **Quality Assurance** has QAID, Date, Feedback, Evaluation, and StaffID which is a foreign key to the Admin Staff table for the staff member who made the QA record
- **Billing** has amount (the amount of bill), BillingID, Date (date the bill was issued) description, and billingType. Has StaffID as foreign key to show who was overseeing the creation of the bill. Staff are the ones who create the bill. Also has foreign keys to SessionID, MemberID, and EventID. The reason I chose this method is that the specific types of bills (Training session, event, membership fees) dont have their own attributes that arent found in the Bills entity, so its easier to have foreign keys to all 3 entities.

- **Payment** has PaymentID, Date, Description and a foreign key to Bills because you make a payment FOR a bill, also foreign key to MemberID to show who made the payment, and StaffID fk to show who was overseeing that transaction, and loyaltyID reference to loyalty points table. pointsEarned, pointsRedeemed are for representing the number of points earned, and redeemed. These attributes are added to the numPoints attribute in loyaltyPoints, or subtracted when redeeming.
- **Loyalty points** has numPoints, loyaltyID (Primary key), and foreign key to MemberID

Assumptions:

- Booking a room means booking a room for an event, i.e., associating an event with a room at a specific time
- Not every member has to schedule personal training sessions or Events
- Not every personal trainer has to teach training sessions (they could be just hired so doesnt make sense to assign them to a training session)
- Not every room has an event going on

- Not every staff member has to manage a Billing, Payment, Room, Event, Fitness Maintenance or Quality Assurance

2.3 FUNCTIONAL DEPENDENCIES:

1. **The functional dependencies for the Member entity are:**
   MemberID → {Fname,Lname,Email,StreetPCode,HomeNum,Gender,DOB,Phone Number,joinDate,Username,Password}

2. **The functional dependencies for the Event entity are:**
   EventID → {EventDate, EventTime, EventDuration, StaffID, RoomNumber, TrainerID, EventType}

3. **The functional dependencies for the Trainer entity are:**
   TrainerID → {Fname, Lname, Phone Number, Email, Username, Password}

4. **The functional dependencies for the Personal Training session entity are:**
   SessionID → {SessionDate, SessionTime, Duration, ProgressNotes, TrainerID, MemberID}

5. **The functional dependencies for the Room entity are:**
   R_ID → Available

6. **The functional dependencies for the Admin Staff entity are:**
   StaffID → {Fname, Lname, PhoneNumber, Email, Username, Password}

7. **The functional dependencies for the Fitness Equipment Maintenance entity are:**
   MaintenanceID → {EquipmentName, Date, Description, StaffID}

8. **The functional dependencies for the Quality Assurance entity are:**
   QAID → {Evaluation, Feedback, Date, StaffID}

9. **The functional dependencies for the Billing entity are:**
   BillingID → {Amount, Date, Description, BillingType, Status, StaffID, MemberID}
   BillingID → {EventID, SessionID}
   EventID → StaffID
   SessionID → MemberID

10. **The functional dependencies for the Loyalty points entity are:**
    LoyaltyID -> {numPoints, MemberID}

11. **The functional dependencies for the Payment entity are:**
    {PaymentID, LoyaltyID} → {pointsEarned, pointsRedeemed}
    PaymentID → {Date, StaffID, MemberID, Description}
    PaymentID → BillingID
    BillingID → {StaffID, MemberID, Description}

Decomposing into 2NF:

Payment entity:

- We create a new separate entity, let's say Payment2 where PaymentID is a sole primary key and the attributes fully functionally dependent on it (Date, MemberID, StaffID, Description, BillingID).
- The original Payment entity would then just have the composite key {PaymentID, LoyaltyID} and the attributes that are dependent on the full composite key (pointsEarned, pointsRedeemed).
- I.e.
- FD1: {PaymentID, LoyaltyID} → {pointsEarned, pointsRedeemed}
- FD2: PaymentID → {Date, MemberID, StaffID, Description, BillingID}

Now they both apply full functional dependency

- Every other entity is already in 2NF

Decomposing into 3NF:

We can normalize Billing by decomposing it into the three 3NF relation schemas Billing1, Billing2, Billing3 where:

- Billing1 has a primary key BillingID and with non-transitively dependent attributes (Amount, Billing1Date, Billing1Description, BillingType, Status, EventID, SessionID).
- Billing2 has a primary key EventID and with non-transitively dependent attributes : (StaffID).
- Billing3 has a primary key SessionID and the attributes MemberID with non-transitively dependent attributes .

We can normalize Payment by decomposing it into the three 3NF relation schemas Payment1, Payment2, Payment3 where:

- Payment1 has a composite primary key {PaymentID, LoyaltyID} and with non-transitively dependent attributes (pointsEarned, pointsRedeemed).
- Payment2 has a primary key PaymentID with non-transitively dependent attributes : (Payment2Date, BillingID).
- Payment3 has a primary key BillingID and the attributes MemberID with non-transitively dependent attributes {StaffID, MemberID, PaymentDescription3}