



CMT 302:DATABASE SYSTEMS

FINAL PROJECT REPORT

GYM MEMBERSHIP MANAGEMENT

By GROUP 15

- 1. JESSE CHOMBA -1049057**
- 2. LEWIS SIMATWA -1049429**
- 3. SAM MUCHINA - 1049427**
- 4. OWEN WANJAMA - 1049430**
- 5. EMMANUEL NGETI - 1049396**

Submitted on 21st November 2024

1. Introduction

1.1.Overview

The Gym Membership Management Database is intended to streamline the operations of fitness centers(Gyms) by providing a centralized database for managing memberships and members, trainers, and sessions. The system enables efficient tracking of member details, subscription types, trainer specializations, session schedules as well as payments. The system simplifies membership management, enhances client-trainer interactions, and provides valuable insights through data queries. This ensures an organized and professional experience for both gym administrators and members.

1.2.Rationale

Modern gyms face numerous challenges in managing growing memberships, coordinating session schedules, and maintaining member and trainer records. A manual approach, mostly involving large volumes of books,binders or gigantic excel spreadsheets to these processes is prone to errors, inefficiencies, and data inconsistencies, which can negatively impact customer satisfaction and operational efficiency leading to an overall poor performance for fitness centers.

The Gym Membership Management Database addresses these challenges by automating critical processes such as membership and member tracking, session scheduling, and trainer assignments. It provides a reliable platform to handle memberships, enabling gyms to focus more on delivering quality services rather than administrative tasks. Furthermore, the system improves data accuracy, ensures compliance with operational policies, and supports informed decision-making through data analysis

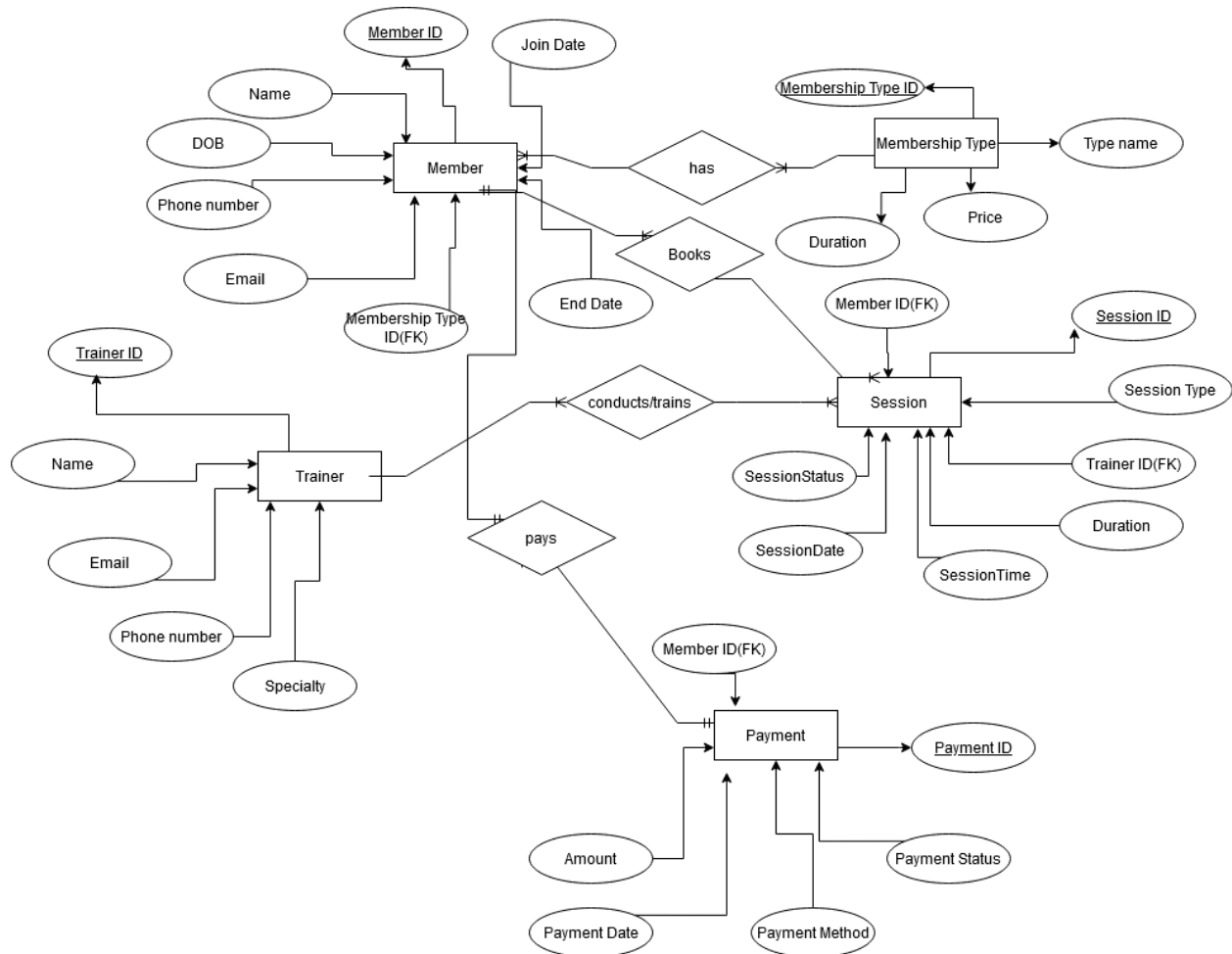
1.3.Objectives

The primary objectives of the Gym Membership Management System are:

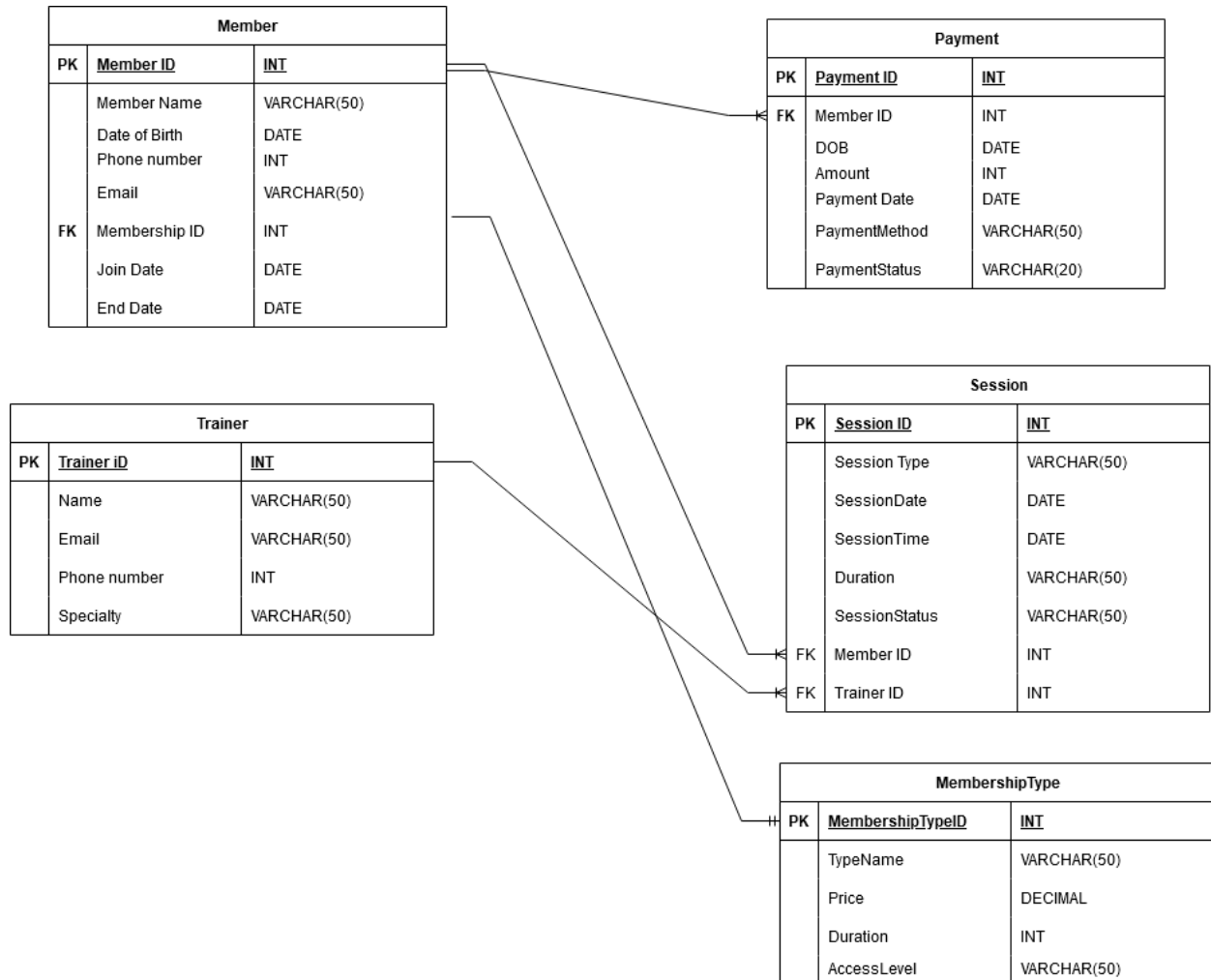
- I. To centralize and automate the management of gym memberships, including member registration, subscription tracking, and renewal processes.
- II. To facilitate the efficient scheduling and monitoring of sessions, including personal training, group fitness, and specialized classes.
- III. To maintain a comprehensive database of trainers, including their specializations, schedules, and contact information.
- IV. To enhance operational efficiency by minimizing manual data entry and reducing errors.
- V. To generate insights through reporting and analytics for improved decision-making.
- VI. To improve customer satisfaction by offering seamless membership management and transparent session tracking.

2. System Design

2.1. ER diagrams



2.2. Table Structures/ Schema



3. Implementation

3.1. CRUD Operations

3.1.1. Gym Trainer CRUD operations

I. Create Operations

- Add a Training Session

Trainers can create new sessions in the system, specifying details such as the session type, date, time, duration, and the member assigned eg:

```
INSERT INTO Session (MemberID, SessionType, SessionDate, SessionTime, Duration, SessionStatus, TrainerID)
```

```
VALUES (6, 'Yoga', CURDATE(), CURTIME(), 60, 'Confirmed', 7);
```

II. Read operations

Trainers can retrieve and view data relevant to their role. Examples include:

- View Assigned Sessions

Trainers can view their upcoming or completed sessions eg:

```
SELECT * FROM Session
WHERE TrainerID = ? AND SessionStatus = 'Scheduled';
```

- View Member Details

Trainers can access information about members attending their sessions eg:

```
SELECT Member.Name, Member.PhoneNumber, Member.Email
FROM Member INNER JOIN Session ON Member.MemberID =
Session.MemberID
WHERE Session.TrainerID = ?;
```

- Review Session History

Trainers can check their session history for performance tracking or evaluation eg to see the sessions the competed and the members that attended(a full session history):

```
SELECT * FROM Session WHERE TrainerID = ?;
```

III. Update Operations

Trainers working in tandem with gym administrators can modify existing information in the system for example:

- Update Session Details

Trainers can reschedule or adjust session details, such as changing the time or marking the session as "Completed."

```
UPDATE Session
SET SessionTime = '16:00:00', SessionStatus = 'Completed'
WHERE SessionID = ?;
```

- Update Trainer Information

Trainers can update their expertise or contact information if changes occur.

```
UPDATE Trainer
SET Specialty = 'Strength Training and Cardio', PhoneNumber = '0712345678'
WHERE TrainerID = ?;
```

IV. *Delete Operations*

- Delete a Session

Trainers can cancel and remove a session if it is no longer needed.(The session Id in this case will be input from the user)

```
DELETE FROM Session WHERE SessionID = ?;
```

3.1.2 Gym Member CRUD operation

I. *Create Operations*

- Schedule a New Session

The member can book a session by inserting a new record into the Session table with details about the session, member, and Session date.

```
INSERT INTO Session (MemberID, SessionType, SessionDate, SessionTime,
Duration, SessionStatus, TrainerID)
VALUES (?, 'Yoga', '2023-12-15', '09:00:00', 60, 'Confirmed', ?);
```

II. *Read Operations*

- View Upcoming Sessions

The member can check their future scheduled sessions, including session details and trainer names, using a SELECT query filtered by upcoming dates.

```
SELECT
    s.SessionID,
    s.SessionType,
    s.SessionDate,
    s.SessionTime,
    s.Duration,
    s.SessionStatus,
    t.Name AS TrainerName,
    t.Specialty AS TrainerSpecialty
FROM
    Session s
```

```

JOIN
    Trainer t ON s.TrainerID = t.TrainerID
JOIN
    Member m ON s.MemberID = m.MemberID
WHERE
    m.MemberID = ?
    AND s.SessionDate >= CURDATE()
    AND s.SessionStatus = 'Scheduled'
ORDER BY
    s.SessionDate ASC, s.SessionTime ASC;

```

- View Session History

The member can review past sessions, including session types, dates, times, and trainers, using a SELECT query to retrieve their booking and session details.

```

SELECT s.SessionID, s.SessionType, s.SessionDate, s.SessionTime,
s.Duration,
    t.Name AS TrainerName
FROM Session s
JOIN Trainer t ON s.TrainerID = t.TrainerID
WHERE s.MemberID = ? AND s.SessionDate < CURDATE();

```

- View Membership Details

The member can check their membership information, such as type, start and end dates, and current status, using a SELECT query.

```

SELECT m.MemberID, m.Name, mt.TypeName AS MembershipType,
mt.Price,
    m.JoinDate, m.EndDate
FROM Member m
JOIN MembershipType mt ON m.MembershipTypeID =
mt.MembershipTypeID
WHERE m.MemberID = ?;

```

III. Update Operations

- Renew or Upgrade Membership

The member can renew or upgrade their membership type, duration, and access level by modifying their record in the Member table with an UPDATE query.

```
UPDATE Member
SET MembershipTypeID = ?, JoinDate = CURDATE(),
    EndDate = DATE_ADD(CURDATE(), INTERVAL (
    SELECT Duration FROM MembershipType WHERE
    MembershipTypeID = ?
    ) DAY)
WHERE MemberID = ?;
```

- Update Personal Information

The member can modify their personal details, such as phone number or email, by updating their record in the Member table using an UPDATE query.

```
UPDATE Member
SET PhoneNumber = ?, Email = ?
WHERE MemberID = ?;
```

IV. Delete Operations

- Delete their own account

A member can request to remove their account from the system. This involves deleting their data from the Member table while also ensuring related records (e.g., bookings, payments) are handled properly to maintain data integrity.

```
DELETE FROM Payments WHERE MemberID = ?;
DELETE FROM Session WHERE MemberID = ?;
DELETE FROM Member WHERE MemberID = ?;
```


3.1.3 Gym Admin CRUD operation

I. Create a New Membership

The gym admin can add a new membership plan to the system e.g. The admin can enter details for a new membership e.g. Couples which gives a discount to members with spouses

This could be done using an SQL command like:

```
INSERT INTO MembershipType (TypeName, Duration, Price) VALUES ('Couples', 365, 36000);
```

II. Read Membership Information

The gym admin can view details of existing membership plans to assess their usage and popularity. E.g. The admin retrieves information about all available membership types.

This could be achieved using an SQL command like:

```
SELECT * FROM MembershipType;
```

III. Update Trainer Information

The admin can update the details of a trainer, such as their specialization. E.g. If a trainer completes a new certification, the admin updates this information in the system.

This could be done with an SQL command like:

```
UPDATE Trainer SET Specialty = 'Yoga' WHERE TrainerId = 1;
```

IV. Delete a Session Requirement

The admin can remove a session from the schedule if it is canceled or no longer offered. E.g. The admin deletes a specific training session that has been canceled due to low enrollment.

This could be executed using an SQL command like:

```
DELETE FROM Session WHERE SessionID = 1;
```

3.2. Advanced SQL queries

3.2.1 Stored Procedures:

I. Procedure to add new member

```
DELIMITER //
```

```
CREATE PROCEDURE `AddNewMember`(  
    IN p_Name VARCHAR(100),  
    IN p_DOB DATE,  
    IN p_PhoneNumber VARCHAR(20),  
    IN p_Email VARCHAR(100),  
    IN p_MembershipTypeID INT,  
    IN p_JoinDate DATE,  
    IN p_EndDate DATE  
  
)  
  
BEGIN  
  
    INSERT INTO Member (Name, DOB, PhoneNumber, Email, MembershipTypeID,  
JoinDate, EndDate)
```

```
VALUES (p_Name, p_DOB, p_PhoneNumber, p_Email, p_MembershipTypeID,  
p_JoinDate, p_EndDate);
```

```
END //
```

```
DELIMITER ;
```

ii.Procedure to add new Membership type

```
DELIMITER //
```

```
CREATE PROCEDURE `AddMembershipType`(  
    IN p_TypeName VARCHAR(50),  
    IN p_Price DECIMAL(10, 2),  
    IN p_Duration INT  
)  
BEGIN  
    INSERT INTO MembershipType (TypeName, Price, Duration)  
VALUES (p_TypeName, p_Price, p_Duration);  
END //
```

```
DELIMITER ;
```

iii.Procedure to Schedule New Session

```
DELIMITER //
```

```
CREATE PROCEDURE `ScheduleSession`(  
    IN p_SessionType VARCHAR(50),  
    IN p_SessionDate DATE,  
    IN p_SessionTime TIME,  
    IN p_Duration INT,  
    IN p_SessionStatus VARCHAR(50),
```

```

        IN p_TrainerID INT,

        IN p_MemberID INT

    )

BEGIN

    INSERT INTO Session (SessionType, SessionDate, SessionTime, Duration,
SessionStatus, TrainerID, MemberID)

        VALUES (p_SessionType, p_SessionDate, p_SessionTime, p_Duration, p_SessionStatus,
p_TrainerID, p_MemberID);

END //

DELIMITER ;

DELIMITER //

```

iv.Procedure to handle new payments

```

CREATE PROCEDURE AddPayment(

    IN p_member_id INT,

    IN p_amount DECIMAL(10, 2),

    IN p_payment_date DATE,

    IN p_payment_method VARCHAR(50)

)

BEGIN

    -- Insert a new payment record into the payments table

    INSERT INTO Payments (member_id, amount, PaymentDate, PaymentMethod)

        VALUES (p_member_id, p_amount, p_payment_date, p_payment_method);

END //

```

DELIMITER ;

3.2.2 Triggers

I. Automatically Update Membership End Date on Membership Renewal

When a member renews or upgrades their membership, the EndDate should be recalculated based on the new membership type.

DELIMITER \$\$

```
CREATE TRIGGER UpdateMembershipEndDate
AFTER UPDATE ON Member
FOR EACH ROW
BEGIN
    IF OLD.MembershipTypeID != NEW.MembershipTypeID THEN
        SET @new_end_date = DATE_ADD(NEW.JoinDate, INTERVAL (
            SELECT Duration FROM MembershipType
            WHERE MembershipTypeID = NEW.MembershipTypeID
        ) DAY);

        UPDATE Member
        SET EndDate = @new_end_date
        WHERE MemberID = NEW.MemberID;
    END IF;
END$$
```

DELIMITER ;

II. Update Booking Status if a Session Date is in the Past

If a session's date has passed, automatically set the Status of the associated bookings to "Completed."

DELIMITER \$\$

```
CREATE TRIGGER UpdateSessionStatus
AFTER UPDATE ON Session
FOR EACH ROW
```

```

BEGIN
  IF NEW.SessionDate < CURDATE() THEN
    UPDATE Session
    SET SessionStatus = 'Completed'
    WHERE SessionID = NEW.SessionID AND SessionStatus != 'Canceled';
  END IF;
END$$

DELIMITER ;

```

III. Automatically Insert a Payment Record After Booking a Paid Session

If a booking is made, create an entry in the Payments table for the session's fee (assuming you have fees tied to sessions or memberships).

```

DELIMITER $$

CREATE TRIGGER InsertPaymentOnSession
AFTER INSERT ON Session
FOR EACH ROW
BEGIN
  DECLARE session_fee DECIMAL(10,2);
  SET session_fee = (SELECT Price FROM MembershipType
    WHERE MembershipTypeID =
      (SELECT MembershipTypeID FROM Member
        WHERE MemberID = NEW.MemberID));

  INSERT INTO Payments (MemberID, DOB, Amount, PaymentDate,
    PaymentMethod, PaymentStatus)
  VALUES (NEW.MemberID,
    (SELECT DOB FROM Member WHERE MemberID = NEW.MemberID),
    session_fee,
    NOW(),
    'Pending',
    'Unpaid');
END$$

DELIMITER ;

```

IV. Prevent Overbooking a Session

Ensure a session cannot have more bookings than a defined limit (assuming session capacity is predefined in the Session table).

DELIMITER \$\$

```
CREATE TRIGGER PreventOverbooking
BEFORE INSERT ON Session
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM Session WHERE SessionID = NEW.SessionID) >=
        (SELECT Capacity FROM Session WHERE SessionID = NEW.SessionID) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Session is fully booked.';
    END IF;
END$$
```

DELIMITER ;

V. Update Member Status When Membership Expires

Automatically deactivate a member when their EndDate passes.

DELIMITER \$\$

```
CREATE TRIGGER DeactivateMemberOnExpiry
AFTER UPDATE ON Member
FOR EACH ROW
BEGIN
    IF NEW.EndDate < CURDATE() THEN
        UPDATE Member
        SET MembershipTypeID = NULL
        WHERE MemberID = NEW.MemberID;
    END IF;
END$$
```

DELIMITER ;

VI. Track Trainer's Session Count

Keep a running count of how many sessions a trainer has conducted.

DELIMITER \$\$

```
CREATE TRIGGER UpdateTrainerSessionCount
AFTER INSERT ON Session
FOR EACH ROW
BEGIN
    UPDATE Trainer
    SET SessionCount = COALESCE(SessionCount, 0) + 1
    WHERE TrainerID = NEW.TrainerID;
END$$
```

DELIMITER ;

3.2.3 Views

I. Membership Summary

This view provides an overview of all membership types, the number of members in each type, and their total revenue contribution.

```
CREATE VIEW MembershipSummary AS
SELECT
    mt.MembershipTypeID,
    mt.TypeName AS MembershipType,
    mt.Price AS MembershipFee,
    COUNT(m.MemberID) AS TotalMembers,
    (COUNT(m.MemberID) * mt.Price) AS TotalRevenue
FROM
    MembershipType mt
LEFT JOIN
    Member m ON mt.MembershipTypeID = m.MembershipTypeID
GROUP BY
    mt.MembershipTypeID, mt.TypeName, mt.Price;
```


II. Active Sessions

This view lists all active sessions, including details about trainers and participating members.

```
CREATE VIEW ActiveSessions AS
SELECT
    s.SessionID,
    s.SessionType,
    s.SessionDate,
    s.SessionTime,
    s.Duration,
    s.SessionStatus,
    t.Name AS TrainerName,
    m.Name AS MemberName
FROM
    Session s
LEFT JOIN
    Trainer t ON s.TrainerID = t.TrainerID
LEFT JOIN
    Member m ON s.MemberID = m.MemberID
WHERE
    s.SessionStatus = 'Active';
```

III. Payments Summary

This view summarizes payments, showing total payments received, pending payments, and the number of transactions.

```
CREATE VIEW PaymentsSummary AS
SELECT
    PaymentStatus,
    COUNT(PaymentID) AS TotalTransactions,
    SUM(Amount) AS TotalAmount
FROM
    Payments
GROUP BY
    PaymentStatus;
```

IV. Trainer Performance

This view provides an overview of trainer activity, including the number of sessions conducted and total members trained.

```
CREATE VIEW TrainerPerformance AS
SELECT
    t.TrainerID,
    t.Name AS TrainerName,
    t.Specialty,
    COUNT(s.SessionID) AS TotalSessionsConducted,
    COUNT(DISTINCT s.MemberID) AS TotalMembersTrained
FROM
    Trainer t
LEFT JOIN
    Session s ON t.TrainerID = s.TrainerID
GROUP BY
    t.TrainerID, t.Name, t.Specialty;
```

V. Upcoming Sessions

This view displays all sessions scheduled for future dates.

```
CREATE VIEW UpcomingSessions AS
SELECT
    s.SessionID,
    s.SessionType,
    s.SessionDate,
    s.SessionTime,
    s.Duration,
    t.Name AS TrainerName
FROM
    Session s
LEFT JOIN
    Trainer t ON s.TrainerID = t.TrainerID
WHERE
    s.SessionDate > CURDATE()
ORDER BY
    s.SessionDate, s.SessionTime;
```

VI. Expired Memberships

This view lists members whose memberships have expired.

```
CREATE VIEW ExpiredMemberships AS
```

```
SELECT
```

```
    m.MemberID,
```

```
    m.Name AS MemberName,
```

```
    mt.TypeName AS MembershipType,
```

```
    m.EndDate
```

```
FROM
```

```
    Member m
```

```
JOIN
```

```
    MembershipType mt ON m.MembershipTypeID = mt.MembershipTypeID
```

```
WHERE
```

```
    m.EndDate < CURDATE();
```

VII. Session Attendance

This view tracks members and trainers participating in sessions for attendance tracking.

```
CREATE VIEW SessionAttendance AS
```

```
SELECT s.SessionID, s.SessionType, s.SessionDate, m.Name AS MemberName, t.Name AS  
TrainerName
```

```
FROM
```

```
    Session s
```

```
LEFT JOIN
```

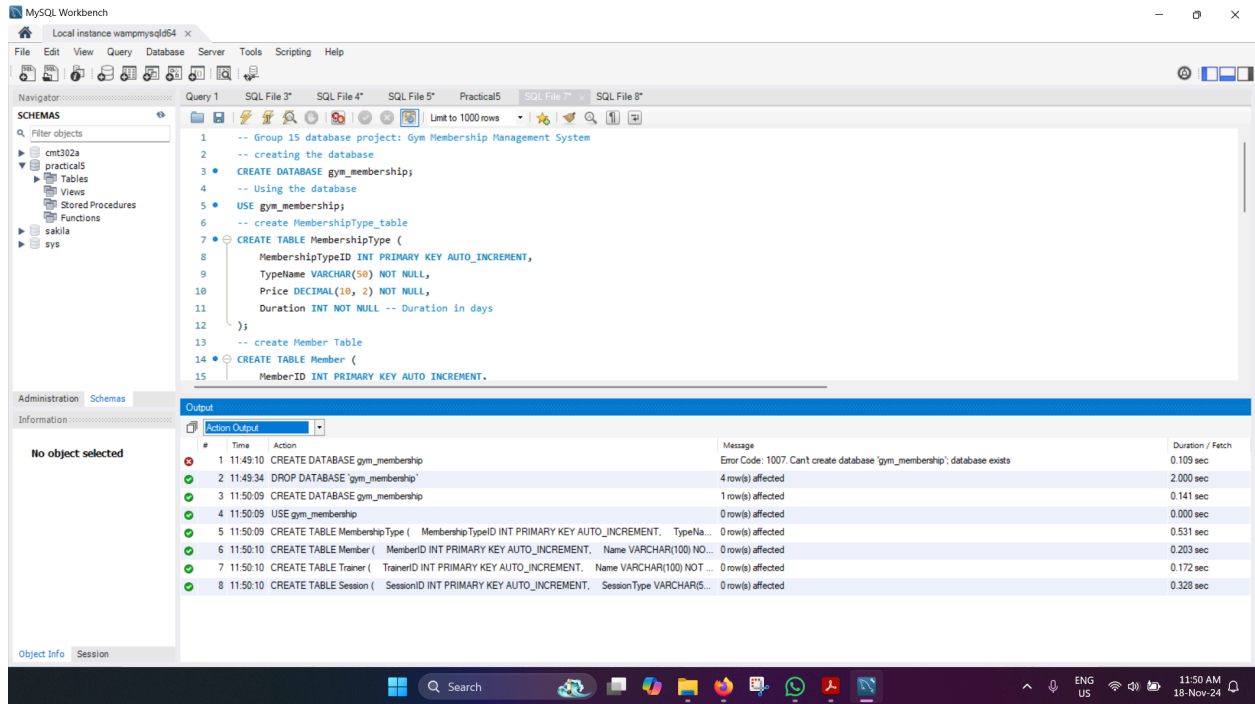
```
    Member m ON s.MemberID = m.MemberID
```

```
LEFT JOIN
```

```
    Trainer t ON s.TrainerID = t.TrainerID;
```

4. Testing And Validation

4.1 Table creation scripts

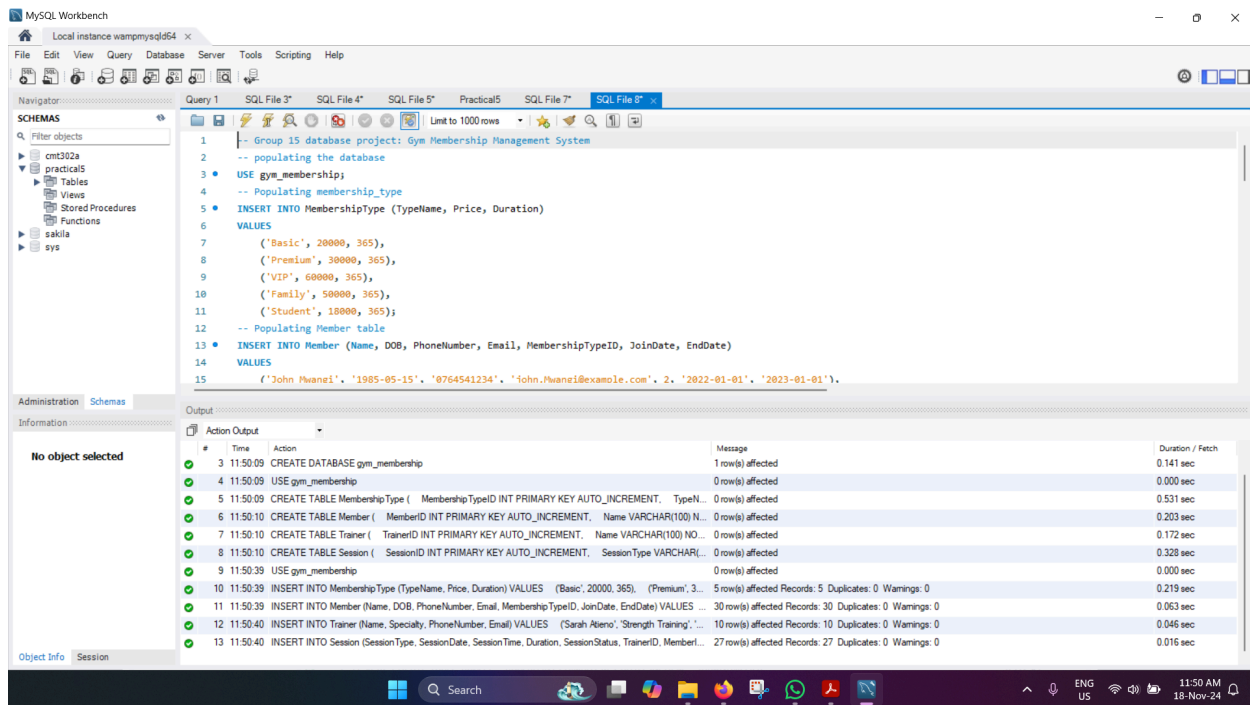


The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view containing 'cmt302a', 'practical5', 'Tables', 'Views', 'Stored Procedures', 'Functions', 'sakila', and 'sys'. The main editor window shows a SQL script for creating a database and tables. The script includes comments and SQL commands for creating the 'gym_membership' database, using it, and creating three tables: 'MembershipType', 'Member', and 'Trainer'. The 'Output' panel at the bottom shows the execution results of these commands.

```
1 -- Group 15 database project: Gym Membership Management System
2 -- creating the database
3 CREATE DATABASE gym_membership;
4 -- Using the database
5 USE gym_membership;
6 -- create MembershipType_table
7 CREATE TABLE MembershipType (
8     MembershipTypeID INT PRIMARY KEY AUTO_INCREMENT,
9     TypeName VARCHAR(50) NOT NULL,
10    Price DECIMAL(10, 2) NOT NULL,
11    Duration INT NOT NULL -- Duration in days
12 );
13 -- create Member Table
14 CREATE TABLE Member (
15     MemberID INT PRIMARY KEY AUTO_INCREMENT,
```

#	Time	Action	Message	Duration / Fetch
1	11:49:10	CREATE DATABASE gym_membership	Error Code: 1007. Can't create database 'gym_membership'; database exists	0.109 sec
2	11:49:34	DROP DATABASE 'gym_membership'	4 row(s) affected	2.000 sec
3	11:50:09	CREATE DATABASE gym_membership	1 row(s) affected	0.141 sec
4	11:50:09	USE gym_membership	0 row(s) affected	0.000 sec
5	11:50:09	CREATE TABLE MembershipType (MembershipTypeID INT PRIMARY KEY AUTO_INCREMENT, TypeName VARCHAR(50) NOT NULL, Price DECIMAL(10, 2) NOT NULL, Duration INT NOT NULL -- Duration in days)	0 row(s) affected	0.531 sec
6	11:50:10	CREATE TABLE Member (MemberID INT PRIMARY KEY AUTO_INCREMENT, Name VARCHAR(100) NOT NULL)	0 row(s) affected	0.203 sec
7	11:50:10	CREATE TABLE Trainer (TrainerID INT PRIMARY KEY AUTO_INCREMENT, Name VARCHAR(100) NOT NULL)	0 row(s) affected	0.172 sec
8	11:50:10	CREATE TABLE Session (SessionID INT PRIMARY KEY AUTO_INCREMENT, SessionType VARCHAR(50) NOT NULL)	0 row(s) affected	0.328 sec

4.2 Database population scripts



5. Conclusion and Recommendations

5.1 Conclusion:

The Gym Membership Management System has successfully been developed as a comprehensive system to streamline the operations of a gym or fitness center. By utilizing a relational database approach, the system effectively manages the key aspects of gym management, including member information, trainer records, class schedules, and payment(subscription) tracking

The implementation of the system provides several benefits; It eliminates the errors and inefficiencies associated with manual record-keeping, allowing the gym management to maintain accurate and up-to-date information. The user-friendly interface and minimal training requirements ensure a smooth user experience for both staff and members.

The robust database design, with its well-defined entities and relationships, ensures the system's scalability and flexibility to handle the growing needs of the gym. The incorporation of advanced database features, such as views, triggers, and stored procedures, enhances the system's functionality and data integrity.

5.2 Recommendations:

To further enhance the Gym Membership Management system, the following recommendations are suggested:

1. Mobile Application Development:

- Develop a mobile application that allows members to access their account information, schedule classes, and receive notifications on their smartphones. This would improve the overall user experience and convenience for gym members.

2. Integrated Billing and Payment System:

- Integrate a secure and user-friendly payment gateway to handle membership fees, class payments, and other financial transactions. This would streamline the billing process and provide a seamless experience for members.

3. Fitness Tracking and Analytics:

- Implement features that allow members to track their fitness progress, such as workout history, progress charts and personalized recommendations. Providing the gym management with comprehensive analytics to identify usage patterns, popular classes, and member engagement levels.

4. Automated Membership Renewal and Notifications:

- Implement automated processes to handle membership renewals and send timely reminders to members about upcoming expirations. This would improve customer retention and reduce the administrative burden on the gym staff.

5. Integration with Wearable Devices and Fitness Trackers:

- Explore the possibility of integrating the system with popular wearable devices and fitness trackers to seamlessly sync member data and provide a more holistic fitness experience.

6. Enhance System Security:

Implement robust security measures, such as:

- Role-based access controls to restrict user access based on their responsibilities.

- Secure data storage and encryption techniques to protect sensitive member and employee information.
- Regular security audits and updates to stay ahead of evolving cyber threats.
- Comprehensive logging and monitoring mechanisms to detect and respond to security incidents promptly.

Implementation of these security measures ensures compliance with relevant data privacy regulations and industry best practices to build trust and maintain the integrity of the system.

By implementing these recommendations, the Gym Membership Management system can further enhance its capabilities, improve member satisfaction, and provide valuable insights to the gym management, while also prioritizing the security and protection of the system and its data. These enhancements will contribute to the growth and success of the fitness center, ensuring a safe and trusted environment for both members and staff.

6. References

- *W3Schools.com*. (n.d.). <https://www.w3schools.com/sql/>
- *Mastering Gym Membership Database Management: A Comprehensive guide for gym owners - Felvedere*. (n.d.).
<https://www.joinfitnessflow.com/blog/mastering-gym-membership-database-management-a-comprehensive-guide-for-gym-owners>

7. Appendices

7.1. Code Snippets

7.1.1. Table Creation scripts

-- Group 15 database project: Gym Membership Management System

```

-- creating the database
CREATE DATABASE gym_membership;
-- Using the database
USE gym_membership;
-- create MembershipType_table
CREATE TABLE MembershipType (
    MembershipTypeID INT PRIMARY KEY AUTO_INCREMENT,
    TypeName VARCHAR(50) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
    Duration INT NOT NULL -- Duration in days
);
-- create Member Table
CREATE TABLE Member (
    MemberID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    DOB DATE NOT NULL,
    PhoneNumber VARCHAR(20),
    Email VARCHAR(100) UNIQUE,
    MembershipTypeID INT,
    JoinDate DATE NOT NULL,
    EndDate DATE,
    FOREIGN KEY (MembershipTypeID) REFERENCES
MembershipType(MembershipTypeID)
);
-- create Trainer Table
CREATE TABLE Trainer (
    TrainerID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Specialty VARCHAR(100),
    PhoneNumber VARCHAR(20),
    Email VARCHAR(100) UNIQUE
);
-- create Session table
CREATE TABLE Session (
    SessionID INT PRIMARY KEY AUTO_INCREMENT,
    SessionType VARCHAR(50) NOT NULL,
    SessionDate DATE NOT NULL,
    SessionTime TIME NOT NULL,
    Duration INT NOT NULL, -- Duration in minutes
    SessionStatus VARCHAR(50) NOT NULL,

```



```

    TrainerID INT,
    MemberID INT,
    FOREIGN KEY (TrainerID) REFERENCES Trainer(TrainerID),
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
);
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY AUTO_INCREMENT, -- Primary Key
    MemberID INT, -- Foreign Key referencing Member table
    DOB DATE NOT NULL, -- Date of Birth of the member
    Amount INT NOT NULL, -- Payment amount
    PaymentDate DATE NOT NULL, -- Date of the payment
    PaymentMethod VARCHAR(50) NOT NULL, -- Payment method (e.g., Cash, Card)
    PaymentStatus VARCHAR(20) NOT NULL, -- Status of payment (e.g.,
Completed, Pending)

    -- Foreign Key Constraint
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
);

```

7.1.2 Populating The database

```

-- Group 15 database project: Gym Membership Management System
-- populating the database
USE gym_membership;
-- Populating membership_type
INSERT INTO MembershipType (TypeName, Price, Duration)
VALUES
    ('Basic', 20000, 365),
    ('Premium', 30000, 365),
    ('VIP', 60000, 365),
    ('Family', 50000, 365),
    ('Student', 18000, 365);
-- Populating Member table
INSERT INTO Member (Name, DOB, PhoneNumber, Email, MembershipTypeID, JoinDate,
EndDate)
VALUES
    ('John Mwangi', '1985-05-15', '0764541234', 'john.Mwangi@example.com', 2,
'2022-01-01', '2023-01-01'),
    ('Jane Smith', '1990-09-23', '0725815678', 'jane.smith@example.com', 1, '2023-03-15',
'2024-03-15'),

```

('Michael Omondi', '1978-12-07', '0715159012', 'michael.Omondi@example.com', 3, '2021-06-01', '2022-06-01'),
('Emily Williams', '1992-02-18', '0715423456', 'emily.williams@example.com', 2, '2022-09-01', '2023-09-01'),
('David Brown', '1987-06-30', '0751857890', 'david.brown@example.com', 1, '2023-02-01', '2024-02-01'),
('Sarah Davis', '1995-11-04', '0758542468', 'sarah.davis@example.com', 2, '2022-04-15', '2023-04-15'),
('Alex Kiprotich', '1993-08-11', '0746523690', 'alex.Kiprotich@example.com', 1, '2023-01-01', '2024-01-01'),
('Emily Taylor', '1991-03-22', '0715657531', 'emily.taylor@example.com', 3, '2021-12-01', '2022-12-01'),
('Tom Anderson', '1988-07-14', '0715488642', 'tom.anderson@example.com', 2, '2022-06-01', '2023-06-01'),
('Olivia Thompson', '1989-10-28', '0764829753', 'olivia.thompson@example.com', 1, '2023-03-15', '2024-03-15'),
('Jessica Lee', '1996-04-05', '0765514567', 'jessica.lee@example.com', 1, '2023-11-01', '2024-11-01'),
('Robert Kyalo', '1984-12-19', '0721548901', 'robert.Kyalo@example.com', 2, '2022-08-01', '2023-08-01'),
('Sophia Hernandez', '1991-05-27', '0714846543', 'sophia.hernandez@example.com', 3, '2021-09-01', '2022-09-01'),
('Christopher Gonzalez', '1988-02-14', '0755542109', 'christopher.gonzalez@example.com', 1, '2023-05-01', '2024-05-01'),
('Megan Rodriguez', '1997-08-09', '0721657654', 'megan.rodriguez@example.com', 2, '2022-12-01', '2023-12-01'),
('Emma Davis', '1993-06-20', '0714651357', 'emma.davis@example.com', 2, '2022-07-01', '2023-07-01'),
('Jacob Gonzalez', '1985-11-12', '0764582468', 'jacob.gonzalez@example.com', 3, '2021-02-01', '2022-02-01'),
('Sophia Herrera', '1988-03-28', '0761823579', 'sophia.herrera@example.com', 1, '2023-06-15', '2024-06-15'),
('Ethan Ramirez', '1991-09-05', '0764984680', 'ethan.Ramirez@example.com', 2, '2022-11-01', '2023-11-01'),
('Isabella Njeri', '1994-04-18', '0725455791', 'isabella.Njeri@example.com', 1, '2023-08-01', '2024-08-01'),
('Lucas Mueni', '1986-12-24', '0796456802', 'lucas.Mueni@example.com', 3, '2021-05-01', '2022-05-01'),
('Mia Khalifa', '1997-02-14', '0769697913', 'mia.Khalifa@example.com', 2, '2022-10-01', '2023-10-01'),

```

('Sebastian Castillo', '1990-07-31', '0736568024', 'sebastian.castillo@example.com', 1,
'2023-09-15', '2024-09-15'),
('Olivia Diaz', '1992-05-09', '0754949135', 'olivia.diaz@example.com', 2, '2022-12-01',
'2023-12-01'),
('Daniel Flores', '1988-11-23', '0764490246', 'daniel.flores@example.com', 3,
'2021-08-01', '2022-08-01'),
('Camila Mwangi', '1995-03-07', '0758461357', 'camila.Mwangi@example.com', 1,
'2023-04-01', '2024-04-01'),
('Mark Kimingi', '1987-08-19', '0795842468', 'Mark.Kimingi@example.com', 2,
'2022-09-01', '2023-09-01'),
('Valeria Nana', '1993-01-02', '0764643579', 'valeria.Nana@example.com', 1,
'2023-07-01', '2024-07-01'),
('Gabriel Ramos', '1989-06-14', '0788414680', 'gabriel.ramos@example.com', 2,
'2022-03-01', '2023-03-01'),
('Emilia Mwangi', '1996-10-27', '0715485791', 'emilia.Mwangi@example.com', 1,
'2023-12-01', '2024-12-01');

```

-- Populating Trainers table

```

INSERT INTO Trainer (Name, Specialty, PhoneNumber, Email)

```

VALUES

```

('Sarah Atieno', 'Strength Training', '0765452468', 'sarah.Atieno@example.com'),
('Muli Wilson', 'Cardio', '0764523690', 'Muli.wilson@example.com'),
('Emily Taylor', 'Yoga', '0761557531', 'emily.taylor@example.com'),
('Tom Kipruto', 'Strength Training', '0765628642', 'tom.Kipruto@example.com'),
('Olivia Wairimu', 'Cardio', '0766499753', 'olivia.Wairimu@example.com'),
('Julia Muiruri', 'Pilates', '0764540987', 'julia.Muiruri@example.com'),
('Michael kamau', 'Martial Arts', '0785346543', 'michael.Kamau@example.com'),
('Isabella Sang', 'Nutrition', '0784852109', 'isabella.Sang@example.com'),
('Daniel Kamau', 'Weight Loss', '0768927654', 'daniel.Kamau@example.com'),
('Camila Mohamed', 'Flexibility', '0725650321', 'camila.Mohamed@example.com');

```

-- Populating Session table

```

INSERT INTO Session (SessionType, SessionDate, SessionTime, Duration,
SessionStatus, TrainerID, MemberID)

```

```

VALUES ('Personal Training', '2024-12-01', '10:00:00', 60, 'Scheduled', 1, 6),
('Group Fitness', '2024-12-02', '14:00:00', 45, 'Scheduled', 2, 7),
('Yoga', '2024-12-03', '18:30:00', 90, 'Scheduled', 3, 8),
('Pilates', '2024-12-04', '09:30:00', 75, 'Scheduled', 6, 9),
('Martial Arts', '2024-12-05', '19:00:00', 90, 'Scheduled', 7, 10),
('Nutrition Consultation', '2024-12-06', '14:00:00', 45, 'Scheduled', 8, 11),
('Weight Loss Program', '2024-12-07', '11:30:00', 120, 'Scheduled', 9, 12),

```

('Flexibility Training', '2024-12-08', '17:00:00', 60, 'Scheduled', 10, 13),
('Personal Training', '2024-12-09', '08:00:00', 60, 'Scheduled', 1, 14),
('Group Fitness', '2024-12-10', '16:30:00', 45, 'Scheduled', 2, 15),
('Yoga', '2024-12-11', '07:00:00', 90, 'Scheduled', 3, 16),

('Personal Training', '2024-12-12', '12:00:00', 60, 'Scheduled', 1, 17),
('Group Fitness', '2024-12-13', '15:30:00', 45, 'Scheduled', 2, 18),
('Pilates', '2024-12-14', '10:00:00', 75, 'Scheduled', 6, 19),
('Martial Arts', '2024-12-15', '18:30:00', 90, 'Scheduled', 7, 20);

-- Populating Payments Table

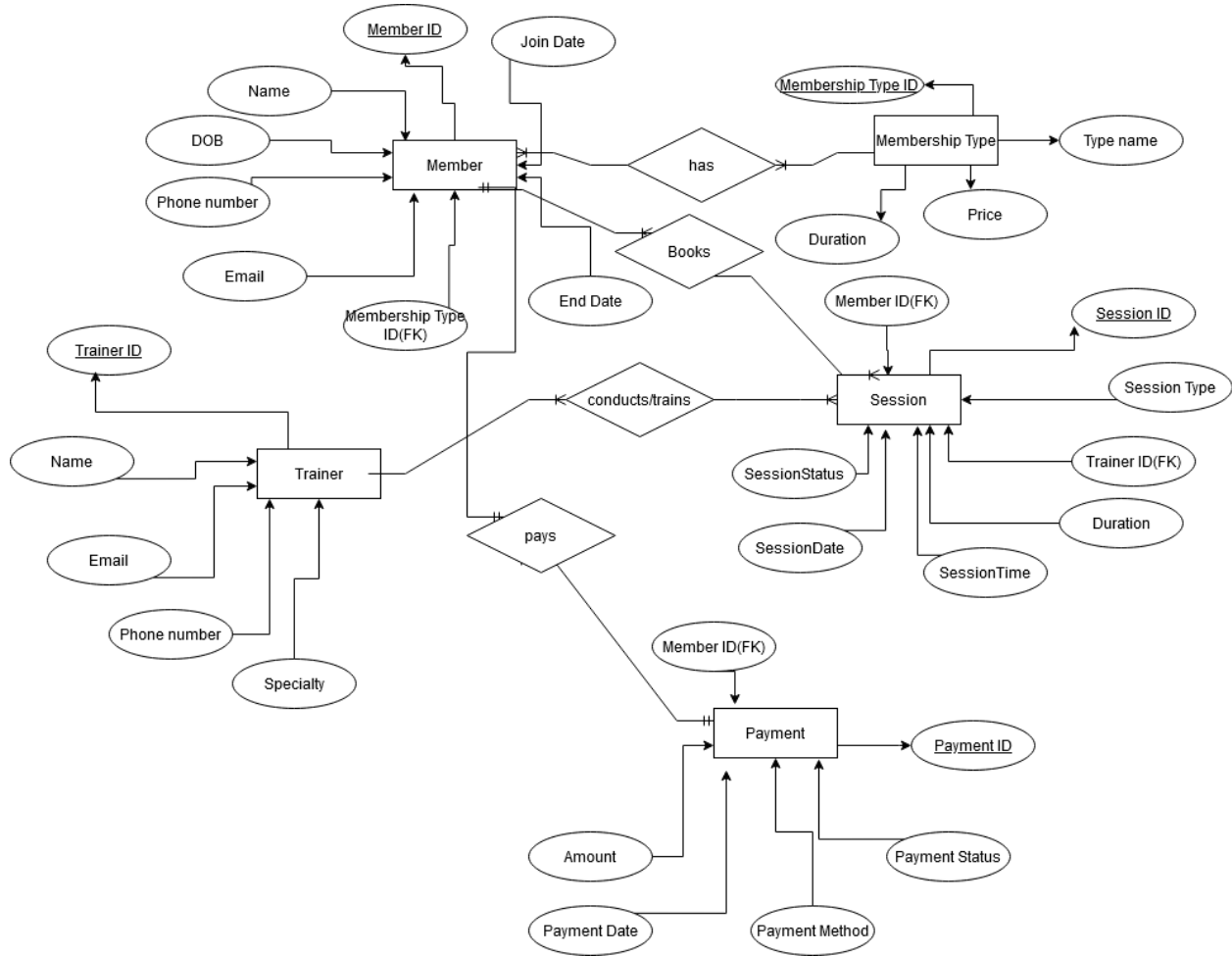
INSERT INTO Payments (MemberID, DOB, Amount, PaymentDate, PaymentMethod,
PaymentStatus)

VALUES

(1, '1985-05-15', 20000, '2023-01-15', 'Cash', 'Completed'),
(2, '1990-09-23', 18000, '2023-02-01', 'Card', 'Completed'),
(3, '1978-12-07', 30000, '2023-01-25', 'Mpesa', 'Completed'),
(4, '1992-02-18', 20000, '2023-03-10', 'Card', 'Pending'),
(5, '1987-06-30', 30000, '2023-03-20', 'Cash', 'Completed'),
(6, '1995-11-04', 20000, '2023-04-05', 'Mpesa', 'Completed'),
(7, '1993-08-11', 18000, '2023-04-15', 'Card', 'Completed'),
(8, '1991-03-22', 30000, '2023-05-10', 'Cash', 'Completed'),
(9, '1988-07-14', 20000, '2023-05-20', 'Mpesa', 'Completed'),
(10, '1989-10-28', 18000, '2023-06-01', 'Card', 'Pending'),
(11, '1996-04-05', 20000, '2023-06-15', 'Mpesa', 'Completed'),
(12, '1984-12-19', 30000, '2023-07-01', 'Card', 'Completed'),
(13, '1991-05-27', 20000, '2023-07-20', 'Cash', 'Completed'),
(14, '1988-02-14', 18000, '2023-08-05', 'Mpesa', 'Completed'),
(15, '1997-08-09', 20000, '2023-08-15', 'Cash', 'Completed'),
(16, '1993-06-20', 30000, '2023-09-01', 'Card', 'Completed'),
(17, '1985-11-12', 20000, '2023-09-10', 'Mpesa', 'Pending'),
(18, '1988-03-28', 18000, '2023-10-01', 'Card', 'Completed'),
(19, '1991-09-05', 30000, '2023-10-10', 'Cash', 'Completed'),
(20, '1994-04-18', 20000, '2023-10-20', 'Mpesa', 'Completed'),
(21, '1986-12-24', 18000, '2023-11-01', 'Card', 'Completed'),
(22, '1997-02-14', 20000, '2023-11-10', 'Mpesa', 'Completed'),
(23, '1990-07-31', 30000, '2023-11-15', 'Cash', 'Pending'),
(24, '1992-05-09', 20000, '2023-11-20', 'Card', 'Completed');

7.2. Diagrams

- ERD diagram:



- SQL Schema

Member		
PK	<u>Member ID</u>	INT
FK	Member Name	VARCHAR(50)
	Date of Birth	DATE
	Phone number	INT
	Email	VARCHAR(50)
	Membership ID	INT
	Join Date	DATE
	End Date	DATE

Payment		
PK	<u>Payment ID</u>	INT
FK	Member ID	INT
	DOB	DATE
	Amount	INT
	Payment Date	DATE
	PaymentMethod	VARCHAR(50)
	PaymentStatus	VARCHAR(20)

Trainer		
PK	<u>Trainer ID</u>	INT
	Name	VARCHAR(50)
	Email	VARCHAR(50)
	Phone number	INT
	Specialty	VARCHAR(50)

Session		
PK	<u>Session ID</u>	INT
	Session Type	VARCHAR(50)
	SessionDate	DATE
	SessionTime	DATE
	Duration	VARCHAR(50)
	SessionStatus	VARCHAR(50)
FK	Member ID	INT
FK	Trainer ID	INT

MembershipType		
PK	<u>MembershipTypeID</u>	INT
	TypeName	VARCHAR(50)
	Price	DECIMAL
	Duration	INT
	AccessLevel	VARCHAR(50)

