GymFit Tracker System

Database Management System Mini Project

Group Members:

- Sujal Sokande - 1272250913
- Kavya Rane - 1262241582
- Megha Mahesh - 1262241643
- Raya Gangopadhyay - 1262241686

Project Title: GymFit Tracker - Gym Membership and Workout Management System

Technologies Used: MySQL, Python Flask, HTML5, CSS3, JavaScript, OpenAI API

Submission Date: November 1, 2025

Abstract

The GymFit Tracker System is a comprehensive database-driven application designed to manage gym memberships, trainer schedules, workout sessions, and member progress tracking in fitness centers. Built using MySQL as the backend database and a modern web stack (HTML, CSS, JavaScript, Python Flask) for the frontend, the system ensures efficient data management through normalized schemas (3NF), role-based access controls, and advanced features including AI-powered recommendations, automated notifications, and real-time progress tracking. The implementation demonstrates advanced DBMS concepts including ER modeling, normalization, DDL/DML/DCL operations, PL/SQL procedures and functions, triggers, and cursors. The system features three distinct user roles (Member, Trainer, Admin) with tailored dashboards, automated membership renewal checks, AI chatbot assistance for Gold members, and comprehensive analytics with visualization. This implementation successfully addresses inefficiencies in manual gym operations, promoting data integrity, scalability, and enhanced user experience for fitness facilities. (148 words)

Contents

List of Abbreviations

- **DBMS:** Database Management System
- **ER:** Entity-Relationship
- **DDL:** Data Definition Language
- **DML:** Data Manipulation Language
- **DCL:** Data Control Language
- **3NF:** Third Normal Form
- **PK:** Primary Key
- **FK:** Foreign Key
- **PL/SQL:** Procedural Language/SQL
- **UNF:** Unnormalized Form
- **1NF:** First Normal Form
- **2NF:** Second Normal Form
- **API:** Application Programming Interface
- **AI:** Artificial Intelligence
- **REST:** Representational State Transfer
- **CRUD:** Create, Read, Update, Delete

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Modern fitness centers face significant challenges in managing their operations efficiently. Traditional manual processes for handling memberships, scheduling training sessions, tracking member workouts, and monitoring health progress are prone to errors, data inconsistencies, and inefficient resource allocation. These challenges lead to poor member experience, difficulty in tracking fitness goals, and administrative overhead.

The GymFit Tracker System addresses these issues by providing a comprehensive, database-driven solution that automates and streamlines gym operations. By leveraging relational database management principles, the system ensures data integrity, reduces redundancy, and provides secure, role-based access to information. The integration of advanced features such as AI-powered recommendations and automated notifications enhances member engagement and operational effectiveness.

1.2 Objectives

The primary objectives of this project are:

- Design and implement a fully normalized database schema (3NF) with 9 interconnected tables representing gym entities
- Demonstrate comprehensive DDL operations including table creation with advanced constraints, foreign key relationships, and cascading actions
- Implement complex DML operations for data manipulation with transaction management and data validation
- Apply sophisticated DCL for multi-tier user privilege management with role-based access control
- Develop PL/SQL stored procedures, functions, triggers, and cursors for automated business logic
- Create a full-stack web application with role-specific dashboards and real-time features
- Integrate AI capabilities for personalized fitness recommendations and chatbot assistance
- Implement automated notification systems for membership renewals and session reminders
- Provide comprehensive progress tracking with analytical visualizations using Chart.js
- Ensure system scalability, performance optimization, and data security

2 Problem Definition

The GymFit Tracker System addresses the following key challenges in gym management:

- **Membership Management:** Tracking diverse membership types, durations, pricing, and renewal dates across multiple gym locations

- **User Management:** Handling three distinct user roles (Members, Trainers, Admins) with different access privileges and functionalities
- **Session Scheduling:** Managing training sessions with capacity constraints, trainer assignments, and member bookings
- **Workout Tracking:** Recording detailed workout logs including exercises, duration, calories burned, and progress notes
- **Health Monitoring:** Tracking member health metrics such as weight, height, sleep hours, water intake, and daily steps
- **Progress Analytics:** Providing meaningful insights through data visualization and trend analysis
- **Automated Notifications:** Sending timely alerts for membership renewals, session reminders, and progress milestones
- **AI Integration:** Delivering personalized workout recommendations based on member data and fitness patterns
- **Data Integrity:** Maintaining referential integrity through foreign key constraints and cascading operations
- **Concurrent Access:** Handling multiple simultaneous users with transaction management

3 Tools and Technologies Used

3.1 Backend Technologies

- **Database:** MySQL 8.0 - Relational database management system with advanced features
- **Application Server:** Python Flask - Lightweight WSGI web application framework
- **AI Integration:** OpenAI GPT-4o-mini API - For intelligent chatbot and recommendations
- **Security:** werkzeug.security - Password hashing and authentication
- **Database Connector:** mysql-connector-python - MySQL database adapter

3.2 Frontend Technologies

- **Markup:** HTML5 - Semantic structure and modern features
- **Styling:** CSS3 - Advanced styling with glassmorphism effects
- **Scripting:** JavaScript (ES6+) - Interactive functionality and AJAX requests
- **Visualization:** Chart.js - Data visualization for progress tracking
- **Icons:** Unicode Emojis - Visual indicators and icons

3.3 Development Tools

- **IDE:** Visual Studio Code - Primary development environment
- **Database Management:** phpMyAdmin (via XAMPP) - Database administration
- **Version Control:** Git - Source code management
- **Diagram Design:** draw.io - ER diagram and system architecture
- **API Testing:** Postman - REST API endpoint testing
- **Browser DevTools:** Chrome/Firefox - Frontend debugging

3.4 Architecture

- **Pattern:** Three-tier architecture (Presentation, Application, Data)
- **API:** RESTful architecture with JSON data exchange
- **Session Management:** Flask sessions with secure cookies
- **State Management:** sessionStorage for client-side persistence

4 Database Design (ER Diagram)

4.1 Entity Identification

The system comprises nine primary entities:

- **Gym:** Physical gym locations with capacity information
- **MembershipType:** Different membership plans with pricing and duration
- **Member:** Individual gym members with profile and membership details
- **Trainer:** Fitness trainers with specializations
- **Admin:** System administrators with full access
- **Session:** Training sessions conducted by trainers
- **WorkoutLog:** Individual workout records and session bookings
- **HealthMetrics:** Daily health and activity tracking data
- **Notifications:** Automated alerts and reminders for members

4.2 Relationship Analysis

Table 1: Relationship Matrix (Table 1)

Relationship	Cardinality	Type	Description
Gym offers MembershipType	1:M	One-to-Many	Each gym offers multiple plans
MembershipType subscribed by Member	1:M	One-to-Many	Each plan has many members
Gym hosts Member	1:M	One-to-Many	Each gym has many members
Gym employs Trainer	1:M	One-to-Many	Each gym employs multiple trainers
Trainer conducts Session	1:M	One-to-Many	Each trainer conducts many sessions
Member attends Session	M:M	Many-to-Many	Via WorkoutLog (booking)
Member has WorkoutLog	1:M	One-to-Many	Each member has multiple logs
Session associated with WorkoutLog	1:M	One-to-Many	Optional session reference
Member tracks HealthMetrics	1:M	One-to-Many	Daily metrics per member
Member receives Notifications	1:M	One-to-Many	Multiple notifications per member

4.3 Primary Key Definitions

- **Gym:** Gym_ID (INT, AUTO_INCREMENT)
- **MembershipType:** Type_ID (INT, AUTO_INCREMENT)
- **Member:** M_ID (INT, AUTO_INCREMENT)
- **Trainer:** T_ID (INT, AUTO_INCREMENT)
- **Admin:** A_ID (INT, AUTO_INCREMENT)
- **Session:** S_ID (INT, AUTO_INCREMENT)
- **WorkoutLog:** L_ID (INT, AUTO_INCREMENT)
- **HealthMetrics:** Metric_ID (INT, AUTO_INCREMENT)
- **Notifications:** Notif_ID (INT, AUTO_INCREMENT)

4.4 ER Diagram

The complete ER diagram illustrates all entities, their attributes, relationships, and cardinalities. Key features include:

- All entities with primary keys (underlined)
- Foreign key relationships with referential integrity
- Crow's foot notation for cardinality representation
- Proper attribute placement showing atomic values
- Association between Member and Session via WorkoutLog

Note: The ER diagram image should be placed here as Figure 1

5 Normalization Details

5.1 Normalization Process Overview

All tables in the GymFit Tracker System have been normalized to Third Normal Form (3NF) to eliminate redundancy, prevent update anomalies, and ensure data integrity.

5.2 Gym Table

Schema: Gym(Gym_ID, Location, Capacity)

Normalization Steps:

- **UNF to 1NF:** Removed any repeating groups; ensured atomic values. Each gym location is stored as a separate row with unique Gym_ID.
- **1NF to 2NF:** With single attribute primary key (Gym_ID), all non-key attributes (Location, Capacity) are fully functionally dependent on the primary key. No partial dependencies exist.
- **2NF to 3NF:** Verified no transitive dependencies. Capacity does not depend on Location; both depend directly on Gym_ID.
- **Result:** 3NF achieved

5.3 MembershipType Table

Schema: MembershipType(Type_ID, Name, Duration, Price, Gym_ID)

Normalization Steps:

- **UNF to 1NF:** Separated membership types into individual rows with atomic attributes.
- **1NF to 2NF:** All attributes (Name, Duration, Price, Gym_ID) are fully dependent on Type_ID. No partial dependencies.
- **2NF to 3NF:** Price depends directly on Type_ID (the specific membership plan), not transitively through Gym_ID.
- **Result:** 3NF achieved

5.4 Member Table

Schema: Member(M_ID, Name, Email, Password, Age, JoinDate, Phone, MembershipType_ID, Gym_ID, IsActive, MembershipEndDate)

Normalization Steps:

- **UNF to 1NF:** Ensured each member has a single row with atomic values. Removed multi-valued attributes like multiple phone numbers.
- **1NF to 2NF:** All attributes depend on M_ID. No partial dependencies exist with the single-column primary key.
- **2NF to 3NF:** Verified no transitive dependencies. Age does not depend on JoinDate; IsActive does not depend on MembershipEndDate. All attributes depend directly on M_ID.
- **Result:** 3NF achieved

5.5 Trainer Table

Schema: Trainer(T_ID, Name, Email, Password, Specialization, Gym_ID)

Normalization Steps:

- **UNF to 1NF:** Atomic values for each attribute; no repeating groups.
- **1NF to 2NF:** Full functional dependency on T_ID.
- **2NF to 3NF:** Specialization depends on T_ID, not transitively through Gym_ID.
- **Result:** 3NF achieved

5.6 Admin Table

Schema: Admin(A_ID, Name, Email, Password)

Normalization Steps:

- **UNF to 1NF:** Simple table with atomic attributes.
- **1NF to 2NF:** All attributes fully dependent on A_ID.
- **2NF to 3NF:** No transitive dependencies exist.
- **Result:** 3NF achieved

5.7 Session Table

Schema: Session(S_ID, Details, SessionDate, SessionTime, Duration, T_ID, MaxParticipants, Status)

Normalization Steps:

- **UNF to 1NF:** Each session stored separately with atomic attributes.
- **1NF to 2NF:** All attributes depend fully on S_ID.
- **2NF to 3NF:** Duration and MaxParticipants depend on S_ID, not transitively through T_ID.
- **Result:** 3NF achieved

5.8 WorkoutLog Table

Schema: WorkoutLog(L_ID, M_ID, S_ID, Exercise, Date, Duration, CaloriesBurnt, Distance, Progress)

Normalization Steps:

- **UNF to 1NF:** Each workout log is a separate row with atomic values.
- **1NF to 2NF:** With single-column PK (L_ID), all attributes are fully dependent.
- **2NF to 3NF:** Progress notes depend on L_ID directly, not transitively through Date or M_ID.
- **Result:** 3NF achieved

5.9 HealthMetrics Table

Schema: HealthMetrics(Metric_ID, M_ID, Date, Weight, Height, SleepHours, WaterLiters, Steps)

Normalization Steps:

- **UNF to 1NF:** Daily metrics stored as separate rows.
- **1NF to 2NF:** All metrics depend on Metric_ID (unique entry).
- **2NF to 3NF:** Weight does not depend on Date or other metrics; all depend directly on Metric_ID.
- **Result:** 3NF achieved

5.10 Notifications Table

Schema: Notifications(Notif_ID, M_ID, Message, Type, IsRead, CreatedAt)

Normalization Steps:

- **UNF to 1NF:** Each notification as a separate record.
- **1NF to 2NF:** All attributes depend on Notif_ID.
- **2NF to 3NF:** No transitive dependencies; Type and IsRead depend directly on Notif_ID.
- **Result:** 3NF achieved

5.11 Functional Dependencies Summary

Table 2: Key Functional Dependencies (Table 2)

Table	Functional Dependency
Gym	Gym_ID → Location, Capacity
MembershipType	Type_ID → Name, Duration, Price, Gym_ID
Member	M_ID → Name, Email, Age, JoinDate, Phone, MembershipType_ID, Gym_ID, IsActive
Trainer	T_ID → Name, Email, Specialization, Gym_ID
Session	S_ID → Details, SessionDate, SessionTime, Duration, T_ID, MaxParticipants
WorkoutLog	L_ID → M_ID, S_ID, Exercise, Date, Duration, CaloriesBurnt, Distance
HealthMetrics	Metric_ID → M_ID, Date, Weight, Height, SleepHours, WaterLiters, Steps
Notifications	Notif_ID → M_ID, Message, Type, IsRead, CreatedAt

6 Database Schema

Table 3: Complete Database Schema (Table 3)

Table	Attributes	Primary Key	Foreign Keys
Gym	Gym_ID, Location, Capacity	Gym_ID	None
MembershipType	Type_ID, Name, Duration, Price, Gym_ID	Type_ID	Gym_ID → Gym
Member	M_ID, Name, Email, Password, Age, JoinDate, Phone, MembershipType_ID, Gym_ID, IsActive, MembershipEndDate	M_ID	MembershipType_ID → MembershipType, Gym_ID → Gym
Trainer	T_ID, Name, Email, Password, Specialization, Gym_ID	T_ID	Gym_ID → Gym
Admin	A_ID, Name, Email, Password	A_ID	None
Session	S_ID, Details, SessionDate, SessionTime, Duration, T_ID, MaxParticipants, Status	S_ID	T_ID → Trainer
WorkoutLog	L_ID, M_ID, S_ID, Exercise, Date, Duration, CaloriesBurnt, Distance, Progress	L_ID	M_ID → Member, S_ID → Session
HealthMetrics	Metric_ID, M_ID, Date, Weight, Height, SleepHours, WaterLiters, Steps	Metric_ID	M_ID → Member
Notifications	Notif_ID, M_ID, Message, Type, IsRead, CreatedAt	Notif_ID	M_ID → Member

7 DDL (Data Definition Language)

7.1 Database Creation

```
-- Drop existing database and create fresh
DROP DATABASE IF EXISTS GymFitDB;
CREATE DATABASE GymFitDB CHARACTER SET utf8mb4 COLLATE
    utf8mb4_unicode_ci;
USE GymFitDB;
```

7.2 Table Creation with Constraints

```sql
-- Gym Table
CREATE TABLE Gym (
    Gym_ID INT PRIMARY KEY AUTO_INCREMENT,
    Location VARCHAR(50) NOT NULL,
    Capacity INT CHECK (Capacity > 0)
);

-- MembershipType Table
CREATE TABLE MembershipType (
    Type_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(30) NOT NULL,
    Duration INT NOT NULL COMMENT 'Duration in months',
    Price DECIMAL(8, 2) NOT NULL CHECK (Price >= 0),
    Gym_ID INT NOT NULL,
    FOREIGN KEY (Gym_ID) REFERENCES Gym(Gym_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Member Table with Enhanced Constraints
CREATE TABLE Member (
    M_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,
    Age INT CHECK (Age BETWEEN 10 AND 100),
    JoinDate DATE NOT NULL,
    Phone VARCHAR(15),
    MembershipEndDate DATE,
    IsActive BOOLEAN DEFAULT TRUE,
    MembershipType_ID INT NOT NULL,
    Gym_ID INT NOT NULL,
    FOREIGN KEY (MembershipType_ID)
        REFERENCES MembershipType(Type_ID)
        ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (Gym_ID) REFERENCES Gym(Gym_ID)
        ON DELETE SET NULL ON UPDATE CASCADE
);
```

```sql
-- Trainer Table
CREATE TABLE Trainer (
    T_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,
    Specialization VARCHAR(50),
    Gym_ID INT,
    FOREIGN KEY (Gym_ID) REFERENCES Gym(Gym_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Admin Table
CREATE TABLE Admin (
```

```
    A_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL
);

-- Session Table with Status Enum
CREATE TABLE Session (
    S_ID INT PRIMARY KEY AUTO_INCREMENT,
    Details VARCHAR(100),
    SessionDate DATE,
    SessionTime TIME,
    Duration INT COMMENT 'Duration in minutes',
    T_ID INT,
    MaxParticipants INT DEFAULT 10,
    Status ENUM('scheduled', 'completed', 'cancelled')
        DEFAULT 'scheduled',
    FOREIGN KEY (T_ID) REFERENCES Trainer(T_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
-- WorkoutLog Table
CREATE TABLE WorkoutLog (
    L_ID INT PRIMARY KEY AUTO_INCREMENT,
    M_ID INT NOT NULL,
    S_ID INT,
    Exercise VARCHAR(50),
    Date DATE,
    Duration INT COMMENT 'Duration in minutes',
    CaloriesBurnt DECIMAL(6,2),
    Distance DECIMAL(6,2) COMMENT 'Distance in km',
    Progress VARCHAR(255),
    FOREIGN KEY (M_ID) REFERENCES Member(M_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (S_ID) REFERENCES Session(S_ID)
        ON DELETE SET NULL ON UPDATE CASCADE
);

-- HealthMetrics Table
CREATE TABLE HealthMetrics (
    Metric_ID INT PRIMARY KEY AUTO_INCREMENT,
    M_ID INT NOT NULL,
    Date DATE NOT NULL,
    Weight DECIMAL(5,2),
    Height DECIMAL(5,2),
    SleepHours INT,
    WaterLiters DECIMAL(4,2),
    Steps INT,
    FOREIGN KEY (M_ID) REFERENCES Member(M_ID)
        ON DELETE CASCADE
);

-- Notifications Table
CREATE TABLE Notifications (
    Notif_ID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    M_ID INT,
    Message TEXT,
    Type ENUM('renewal', 'session_reminder',
              'progress', 'system'),
    IsRead BOOLEAN DEFAULT FALSE,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (M_ID) REFERENCES Member(M_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Key DDL Features:

- **Primary Keys:** AUTO_INCREMENT for automatic ID generation
- **Foreign Keys:** Proper referential integrity with CASCADE options
- **Constraints:** CHECK constraints for data validation (Age, Capacity, Price)
- **UNIQUE:** Email uniqueness across Member, Trainer, and Admin tables
- **ENUM:** Type-safe status values for notifications and sessions
- **DEFAULT:** Default values for IsActive, MaxParticipants, CreatedAt
- **NOT NULL:** Mandatory fields enforced at database level

8 DML (Data Manipulation Language)

8.1 Data Insertion

```
-- Insert Gym Data
INSERT INTO Gym (Gym_ID, Location, Capacity) VALUES
(1, 'Pune Downtown', 150),
(2, 'Mumbai Seaside', 200);

-- Insert Membership Types
INSERT INTO MembershipType (Type_ID, Name, Duration,
    Price, Gym_ID) VALUES
(1, 'Gold', 12, 12000.00, 1),
(2, 'Silver', 6, 7000.00, 1),
(3, 'Platinum', 12, 15000.00, 2);

-- Insert Admin User (password: admin123 - hashed)
INSERT INTO Admin (A_ID, Name, Email, Password) VALUES
(1, 'System Admin', 'admin@gymfit.in',
    'scrypt:32768:8:1$SKPoYjk1R9m14a6N$e89b5cacb766...');
```

```
-- Insert Members (passwords are hashed)
INSERT INTO Member (M_ID, Name, Email, Password, Age,
    JoinDate, MembershipType_ID, Gym_ID) VALUES
(1, 'Sujal Sokande', 'sujal.sokande@gmail.com',
    'scrypt:32768:8:1$yLd02krVD66uLAwc$b3a849f244d6...',
```

```
    22, '2024-01-10', 1, 1),
(2, 'Priya Deshmukh', 'priya.d@email.com',
    'scrypt:32768:8:1$RVb8qiW26oramF5y$ace264c4d5aa...',
    25, '2024-03-15', 2, 1),
(3, 'Kavya Rane', 'kavya.rane@gmail.com',
    'scrypt:32768:8:1$RVb8qiW26oramF5y$ace264c4d5aa...',
    19, '2025-01-01', 2, 1);

-- Insert Trainers
INSERT INTO Trainer (T_ID, Name, Email, Password,
    Specialization, Gym_ID) VALUES
(1, 'Joshua Thompson', 'joshua.t@gymfit.in',
    'scrypt:32768:8:1$RVb8qiW26oramF5y$ace264c4d5aa...',
    'Strength Training', 1),
(2, 'Anjali Mehta', 'anjali.m@gymfit.in',
    'scrypt:32768:8:1$RVb8qiW26oramF5y$ace264c4d5aa...',
    'Yoga', 2);
```

```
-- Insert Sessions
INSERT INTO Session (S_ID, Details, SessionDate,
    SessionTime, Duration, T_ID, MaxParticipants) VALUES
(1, 'Morning Cardio', CURDATE(), '07:00:00', 60, 1, 15),
(2, 'Evening Yoga', CURDATE() + INTERVAL 1 DAY,
    '18:30:00', 45, 2, 10),
(3, 'Advanced Weightlifting', CURDATE() + INTERVAL 2 DAY,
    '17:00:00', 75, 1, 5);

-- Insert Workout Logs
INSERT INTO WorkoutLog (L_ID, M_ID, S_ID, Exercise, Date,
    Duration, CaloriesBurnt, Distance, Progress) VALUES
(1, 1, 1, 'Treadmill', CURDATE(), 30, 300, 5,
    'Good pace'),
(2, 2, NULL, 'Freestyle Weights',
    CURDATE() - INTERVAL 1 DAY, 45, 250, NULL,
    'Felt strong'),
(3, 1, NULL, 'Cycling', CURDATE() - INTERVAL 2 DAY,
    60, 500, 20, 'New personal best');

-- Insert Health Metrics
INSERT INTO HealthMetrics (M_ID, Date, Weight, Height,
    SleepHours, WaterLiters, Steps) VALUES
(1, CURDATE(), 70.5, 175, 7, 2.5, 8000),
(2, CURDATE(), 60.0, 165, 8, 3.0, 10000);
```

8.2 Data Retrieval (SELECT Queries)

```
-- Query 1: Member Dashboard Data
SELECT m.M_ID, m.Name, m.Email, m.Age, m.JoinDate,
       mt.Name AS MembershipType,
       g.Location AS GymLocation
FROM Member m
LEFT JOIN MembershipType mt
```

```
    ON m.MembershipType_ID = mt.Type_ID
LEFT JOIN Gym g ON m.Gym_ID = g.Gym_ID
WHERE m.M_ID = 1;

-- Query 2: Recent Workouts with Session Details
SELECT wl.L_ID, wl.Exercise, wl.Date, wl.Duration,
       wl.CaloriesBurnt, s.Details AS SessionDetails
FROM WorkoutLog wl
LEFT JOIN Session s ON wl.S_ID = s.S_ID
WHERE wl.M_ID = 1
ORDER BY wl.Date DESC, wl.L_ID DESC
LIMIT 5;
```

```
-- Query 3: Trainer Performance Analytics
SELECT t.Name, t.Specialization,
       COUNT(s.S_ID) as TotalSessions,
       AVG(s.Duration) as AvgSessionDuration,
       COUNT(DISTINCT wl.M_ID) as UniqueClients
FROM Trainer t
LEFT JOIN Session s ON t.T_ID = s.T_ID
LEFT JOIN WorkoutLog wl ON s.S_ID = wl.S_ID
GROUP BY t.T_ID, t.Name, t.Specialization;

-- Query 4: Member Progress Summary
SELECT m.Name, m.M_ID,
       COUNT(wl.L_ID) as TotalWorkouts,
       AVG(wl.Duration) as AvgDuration,
       SUM(wl.CaloriesBurnt) as TotalCalories,
       MAX(hm.Weight) as CurrentWeight,
       AVG(hm.SleepHours) as AvgSleep
FROM Member m
LEFT JOIN WorkoutLog wl ON m.M_ID = wl.M_ID
LEFT JOIN HealthMetrics hm ON m.M_ID = hm.M_ID
WHERE m.M_ID = 1
GROUP BY m.M_ID, m.Name;
```

8.3 Data Update Operations

```
-- Update 1: Bulk Membership Renewal
UPDATE Member
SET MembershipEndDate = DATE_ADD(JoinDate,
    INTERVAL 12 MONTH)
WHERE MembershipType_ID = 1
    AND MembershipEndDate IS NULL;

-- Update 2: Session Status Management
UPDATE Session s
SET Status = 'Completed'
WHERE s.SessionDate < CURDATE()
    AND s.Status = 'scheduled'
    AND EXISTS (SELECT 1 FROM WorkoutLog wl
                WHERE wl.S_ID = s.S_ID);
```

```sql

-- Update 3: Mark Notifications as Read
UPDATE Notifications
SET IsRead = TRUE
WHERE M_ID = 1 AND IsRead = FALSE;
```

8.4 Data Deletion Operations

```sql
-- Delete 1: Remove Old Notifications (90+ days)
DELETE FROM Notifications
WHERE CreatedAt < DATE_SUB(CURDATE(), INTERVAL 90 DAY)
    AND IsRead = TRUE;

-- Delete 2: Remove Inactive Members (Cascades)
DELETE FROM Member
WHERE IsActive = FALSE
    AND DATEDIFF(CURDATE(), JoinDate) > 365;
```

Table 4: Sample Query Results (Table 5)

Member Name	Total Workouts	Total Calories
Sujal Sokande	15	4500
Priya Deshmukh	12	3600
Kavya Rane	8	2400

9 DCL (Data Control Language)

9.1 User Creation and Privilege Management

```sql
-- Create Admin User with Full Privileges
CREATE USER 'gymfit_admin'@'localhost'
    IDENTIFIED BY 'secure_admin_123';
GRANT ALL PRIVILEGES ON GymFitDB.*
    TO 'gymfit_admin'@'localhost';
GRANT CREATE USER, RELOAD, PROCESS ON *.*
    TO 'gymfit_admin'@'localhost';

-- Create Trainer User with Limited Access
CREATE USER 'gymfit_trainer'@'localhost'
    IDENTIFIED BY 'trainer_secure_456';
GRANT SELECT, INSERT, UPDATE ON GymFitDB.Session
    TO 'gymfit_trainer'@'localhost';
GRANT SELECT ON GymFitDB.Member
    TO 'gymfit_trainer'@'localhost';
GRANT SELECT ON GymFitDB.WorkoutLog
    TO 'gymfit_trainer'@'localhost';
```

```sql
-- Create Member User with Restricted Access
CREATE USER 'gymfit_member'@'localhost'
    IDENTIFIED BY 'member_secure_789';
GRANT SELECT ON GymFitDB.Member
    TO 'gymfit_member'@'localhost';
GRANT SELECT, INSERT, UPDATE ON GymFitDB.WorkoutLog
    TO 'gymfit_member'@'localhost';
GRANT SELECT, INSERT ON GymFitDB.HealthMetrics
    TO 'gymfit_member'@'localhost';
GRANT SELECT ON GymFitDB.Session
    TO 'gymfit_member'@'localhost';

-- Revoke Dangerous Privileges
REVOKE DELETE, DROP, CREATE ON GymFitDB.*
    FROM 'gymfit_member'@'localhost';
REVOKE DELETE ON GymFitDB.*
    FROM 'gymfit_trainer'@'localhost';

-- Flush Privileges
FLUSH PRIVILEGES;
```

9.2 View Privileges

```sql
-- View Granted Privileges
SHOW GRANTS FOR 'gymfit_admin'@'localhost';
SHOW GRANTS FOR 'gymfit_trainer'@'localhost';
SHOW GRANTS FOR 'gymfit_member'@'localhost';
```

Table 5: User Role Permissions (Table 4)

Role	SELECT	INSERT	UPDATE	DELETE
Admin	All Tables	All Tables	All Tables	All Tables
Trainer	Limited	Session Only	Session Only	None
Member	Personal Data	Personal Data	Personal Data	None

10 Triggers

10.1 Trigger 1: Session Capacity Check

```sql
DELIMITER //
CREATE TRIGGER CheckSessionCapacity
BEFORE INSERT ON WorkoutLog
FOR EACH ROW
BEGIN
    DECLARE current_participants INT;
    DECLARE max_participants INT;

    IF NEW.S_ID IS NOT NULL
        AND NEW.Exercise = 'Session Booking' THEN
```

```
11
12          SELECT COUNT(*), s.MaxParticipants
13          INTO current_participants, max_participants
14          FROM WorkoutLog wl
15          JOIN Session s ON wl.S_ID = s.S_ID
16          WHERE wl.S_ID = NEW.S_ID
17              AND wl.Exercise = 'Session Booking';
18
19          IF current_participants >= max_participants THEN
20              SIGNAL SQLSTATE '45000'
21              SET MESSAGE_TEXT =
22                  'Session is full. Cannot book.';
23          END IF;
24      END IF;
25  END //
26  DELIMITER ;
```

Purpose: Prevents session overbooking by checking current participant count against maximum capacity before allowing new session bookings.

10.2 Trigger 2: Membership Renewal Notification

```
1  DELIMITER //
2  CREATE TRIGGER CheckMembershipRenewal
3  BEFORE UPDATE ON Member
4  FOR EACH ROW
5  BEGIN
6      DECLARE membership_duration INT;
7      DECLARE renewal_date DATE;
8
9      SELECT Duration INTO membership_duration
10     FROM MembershipType
11     WHERE Type_ID = NEW.MembershipType_ID;
12
13     SET renewal_date = DATE_ADD(NEW.JoinDate,
14         INTERVAL membership_duration MONTH);
15
16     IF renewal_date BETWEEN CURDATE()
17         AND DATE_ADD(CURDATE(), INTERVAL 7 DAY) THEN
18
19         INSERT INTO Notifications (M_ID, Message, Type)
20         VALUES (NEW.M_ID,
21             CONCAT('Your membership expires on ',
22                    renewal_date,
23                    '. Please renew to continue.'),
24             'renewal');
25     END IF;
26
27     IF renewal_date < CURDATE() THEN
28         SET NEW.IsActive = FALSE;
29     END IF;
30 END //
31 DELIMITER ;
```

Purpose: Automatically creates renewal notifications for memberships expiring within 7 days and deactivates expired memberships.

10.3 Trigger 3: Workout Progress Tracking

```
DELIMITER //
CREATE TRIGGER TrackWorkoutProgress
AFTER INSERT ON WorkoutLog
FOR EACH ROW
BEGIN
    DECLARE total_workouts INT;
    DECLARE progress_message VARCHAR(255);

    SELECT COUNT(*) INTO total_workouts
    FROM WorkoutLog
    WHERE M_ID = NEW.M_ID;

    IF total_workouts = 10 THEN
        SET progress_message =
            'Congratulations! 10 workouts completed!';
    ELSEIF total_workouts = 25 THEN
        SET progress_message =
            'Amazing! 25 workouts completed!';
    ELSEIF total_workouts = 50 THEN
        SET progress_message =
            'Incredible! 50 workouts - You are a champion!';
    END IF;

    IF progress_message IS NOT NULL THEN
        INSERT INTO Notifications (M_ID, Message, Type)
        VALUES (NEW.M_ID, progress_message, 'progress');
    END IF;
END //
DELIMITER ;
```

Purpose: Creates motivational progress notifications when members reach workout milestones (10, 25, 50 workouts).

11 PL/SQL Procedures and Functions

11.1 Function 1: Check Membership Expiry

```
DELIMITER //
CREATE FUNCTION CheckMembershipExpiry(member_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE days_left INT;
    DECLARE join_date DATE;
    DECLARE duration_months INT;
```

```
10
11      SELECT m.JoinDate, mt.Duration
12      INTO join_date, duration_months
13      FROM Member m
14      JOIN MembershipType mt
15          ON m.MembershipType_ID = mt.Type_ID
16      WHERE m.M_ID = member_id;
17
18      SET days_left = DATEDIFF(
19          DATE_ADD(join_date, INTERVAL duration_months MONTH),
20          CURDATE()
21      );
22
23      RETURN days_left;
24  END //
25  DELIMITER ;
26
27  -- Usage Example
28  SELECT M_ID, Name, CheckMembershipExpiry(M_ID)
29      AS DaysUntilExpiry
30  FROM Member WHERE M_ID = 1;
```

Purpose: Calculates and returns the number of days remaining until a member's membership expires.

11.2 Procedure 1: Get Member Progress Summary

```
1  DELIMITER //
2  CREATE PROCEDURE GetMemberProgressSummary(
3      IN member_id INT
4  )
5  BEGIN
6      SELECT
7          m.Name,
8          m.M_ID,
9          (SELECT COUNT(*) FROM WorkoutLog
10          WHERE M_ID = member_id
11          AND Date >= DATE_SUB(CURDATE(), INTERVAL 7 DAY)
12         ) AS WeeklyWorkouts,
13         (SELECT AVG(CaloriesBurnt) FROM WorkoutLog
14          WHERE M_ID = member_id
15          AND Date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
16         ) AS AvgMonthlyCalories,
17         (SELECT Weight FROM HealthMetrics
18          WHERE M_ID = member_id
19          ORDER BY Date DESC LIMIT 1
20         ) AS CurrentWeight,
21         (SELECT Weight FROM HealthMetrics
22          WHERE M_ID = member_id
23          ORDER BY Date ASC LIMIT 1
24         ) AS StartingWeight,
25         CheckMembershipExpiry(member_id)
26             AS DaysUntilExpiry
27     FROM Member m
```

```
    WHERE m.M_ID = member_id;
END //
DELIMITER ;

-- Usage Example
CALL GetMemberProgressSummary(1);
```

Purpose: Retrieves comprehensive progress metrics for a member including workout frequency, calories, weight changes, and membership status.

11.3 Procedure 2: Generate Workout Recommendations

```
DELIMITER //
CREATE PROCEDURE GenerateWorkoutRecommendations(
    IN member_id INT
)
BEGIN
    DECLARE recent_workout_count INT;
    DECLARE avg_calories_burned DECIMAL(6,2);
    DECLARE current_sleep_hours DECIMAL(4,2);
    DECLARE current_steps INT;
    DECLARE workout_variety INT;

    SELECT COUNT(*), AVG(CaloriesBurnt)
    INTO recent_workout_count, avg_calories_burned
    FROM WorkoutLog
    WHERE M_ID = member_id
        AND Date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);

    SELECT SleepHours, Steps
    INTO current_sleep_hours, current_steps
    FROM HealthMetrics
    WHERE M_ID = member_id
    ORDER BY Date DESC LIMIT 1;

    SELECT COUNT(DISTINCT Exercise)
    INTO workout_variety
    FROM WorkoutLog
    WHERE M_ID = member_id
        AND Date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);

    IF recent_workout_count < 3 THEN
        SELECT 'frequency' as Type,
               'Increase workout frequency' as Message,
               'Mixed Cardio & Strength' as Exercise,
               45 as Duration;
    ELSEIF current_steps < 5000 THEN
        SELECT 'cardio' as Type,
               'Boost daily step count' as Message,
               'Brisk Walking' as Exercise,
               30 as Duration;
    ELSEIF current_sleep_hours < 6 THEN
        SELECT 'recovery' as Type,
               'Improve sleep quality' as Message,
```

```
                'Evening Yoga' as Exercise,
                20 as Duration;
    ELSEIF workout_variety < 2 THEN
        SELECT 'variety' as Type,
                'Add exercise variety' as Message,
                'HIIT Training' as Exercise,
                30 as Duration;
    ELSE
        SELECT 'maintenance' as Type,
                'Maintain current routine' as Message,
                'Current Program' as Exercise,
                45 as Duration;
    END IF;
END //
DELIMITER ;

-- Usage Example
CALL GenerateWorkoutRecommendations(1);
```

Purpose: Analyzes member data and generates personalized workout recommendations based on activity patterns, health metrics, and exercise variety.

11.4 Function 2: Calculate Member Engagement Score

```
DELIMITER //
CREATE FUNCTION CalculateMemberEngagement(
    member_id INT
)
RETURNS DECIMAL(5,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE engagement_score DECIMAL(5,2);
    DECLARE workout_frequency INT;
    DECLARE session_attendance INT;
    DECLARE metric_consistency INT;

    SELECT COUNT(*) INTO workout_frequency
    FROM WorkoutLog
    WHERE M_ID = member_id
        AND Date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);

    SELECT
        CASE
            WHEN COUNT(*) = 0 THEN 0
            ELSE (SUM(CASE WHEN wl.L_ID IS NOT NULL
                      THEN 1 ELSE 0 END) / COUNT(*)) * 100
        END
    INTO session_attendance
    FROM Session s
    LEFT JOIN WorkoutLog wl
        ON s.S_ID = wl.S_ID AND wl.M_ID = member_id
    WHERE s.SessionDate >=
        DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

```
31
32      SELECT COUNT(*) INTO metric_consistency
33      FROM HealthMetrics
34      WHERE M_ID = member_id
35          AND Date >= DATE_SUB(CURDATE(), INTERVAL 7 DAY);
36
37      SET engagement_score =
38          (workout_frequency * 0.4) +
39          (session_attendance * 0.4) +
40          (metric_consistency * 0.2);
41
42      RETURN engagement_score;
43  END //
44  DELIMITER ;
45
46  -- Usage Example
47  SELECT M_ID, Name,
48         CalculateMemberEngagement(M_ID) AS EngagementScore
49  FROM Member;
```

Purpose: Calculates a composite engagement score (0-100) based on workout frequency, session attendance, and health metric logging consistency.

12 Cursor Implementation

12.1 Cursor: Membership Renewal Check

```
1  DELIMITER //
2  CREATE PROCEDURE CheckAllMembershipRenewals()
3  BEGIN
4      DECLARE done INT DEFAULT FALSE;
5      DECLARE member_id INT;
6      DECLARE member_name VARCHAR(100);
7      DECLARE renewal_date DATE;
8      DECLARE days_until_renewal INT;
9
10     DECLARE member_cursor CURSOR FOR
11         SELECT m.M_ID, m.Name,
12                DATE_ADD(m.JoinDate,
13                    INTERVAL mt.Duration MONTH)
14                    AS RenewalDate
15         FROM Member m
16         JOIN MembershipType mt
17             ON m.MembershipType_ID = mt.Type_ID
18         WHERE m.IsActive = TRUE
19             AND DATE_ADD(m.JoinDate,
20                 INTERVAL mt.Duration MONTH)
21                 BETWEEN CURDATE()
22                 AND DATE_ADD(CURDATE(), INTERVAL 7 DAY);
23
24     DECLARE CONTINUE HANDLER FOR NOT FOUND
25         SET done = TRUE;
26
```

```
    OPEN member_cursor;

    renewal_loop: LOOP
        FETCH member_cursor INTO member_id, member_name,
            renewal_date;
        IF done THEN
            LEAVE renewal_loop;
        END IF;

        SET days_until_renewal =
            DATEDIFF(renewal_date, CURDATE());

        INSERT INTO Notifications (M_ID, Message, Type)
        VALUES (member_id,
                CONCAT('Hello ', member_name,
                       '! Your membership renews in ',
                       days_until_renewal, ' days on ',
                       renewal_date, '.'),
                'renewal');
    END LOOP;

    CLOSE member_cursor;

    SELECT CONCAT('Renewal check completed. ',
                  ROW_COUNT(),
                  ' notifications created.') AS Result;
END //
DELIMITER ;

-- Usage Example
CALL CheckAllMembershipRenewals();
```

Purpose: Uses a cursor to iterate through all members with upcoming renewals (within 7 days) and creates personalized renewal notification for each.

Cursor Features Demonstrated:

- Declaration of cursor with complex JOIN query
- Variable declarations for fetched data
- CONTINUE HANDLER for loop termination
- OPEN, FETCH, and CLOSE cursor operations
- Loop control with LEAVE statement
- Dynamic message generation within loop

13 Frontend GUI

13.1 System Architecture

The GymFit Tracker implements a modern three-tier architecture:

- **Presentation Tier:** Responsive HTML5/CSS3/JavaScript interface with role-based views

- **Application Tier:** Python Flask RESTful API server with business logic
- **Data Tier:** MySQL database with stored procedures and triggers

13.2 Key Frontend Features

13.2.1 Role Selection Screen

- Modern glassmorphism design with gradient backgrounds
- Three role options: Member, Trainer, Admin
- Hover effects and smooth transitions
- Intuitive visual hierarchy with emojis

13.2.2 Login Interface

- Role-specific login with dynamic title updates
- Password hashing with werkzeug.security
- Demo credentials display for testing
- Form validation and error handling
- Session management with secure cookies

13.2.3 Member Dashboard

- **Stats Cards:** Sleep hours, water intake, daily steps
- **Profile Section:** Member details with age, height, weight
- **Recent Workouts:** Last 5 workouts with duration and calories
- **Upcoming Sessions:** Booked sessions with cancel functionality
- **AI Recommendations:** Personalized workout suggestions
- **Progress Charts:** Interactive Chart.js visualizations
- **AI Chatbot:** GPT-4 powered fitness assistant (Gold members)

13.2.4 Trainer Dashboard

- Total clients and today's sessions statistics
- Session management with participant tracking
- Client list with last workout information
- Add session functionality (planned feature)

13.2.5 Admin Dashboard

- System-wide statistics (members, trainers, sessions, revenue)
- Member management with add/delete operations
- Trainer management with CRUD operations
- Revenue analytics and reporting

13.3 Interactive Features

13.3.1 Add Workout Modal

- Exercise name input with validation
- Date picker (defaults to today)
- Duration, calories, distance tracking
- Progress notes textarea
- Real-time form validation

13.3.2 Session Booking System

- Available sessions display with trainer info
- Participant count and capacity tracking
- One-click booking functionality
- Session cancellation with confirmation

13.3.3 AI Chatbot Interface

- Modern chat UI with message bubbles
- Markdown support for formatted responses
- Typing indicators for better UX
- Context-aware responses using member data
- Personalized recommendations and insights

13.4 Progress Visualization

- **Workout Frequency Chart:** Bar chart showing daily workouts (30 days)
- **Weight Progress Chart:** Line chart tracking weight changes
- **Calorie Trend Chart:** Line chart showing daily calorie burn
- Interactive tooltips and legends
- Responsive design for mobile devices

13.5 Technology Stack

- **HTML5:** Semantic markup, data attributes
- **CSS3:** Flexbox/Grid layouts, animations, glassmorphism
- **JavaScript ES6+:** Async/await, arrow functions, modules
- **Chart.js:** Data visualization library
- **Fetch API:** RESTful API communication
- **Session Storage:** Client-side state persistence

13.6 Security Features

- Password hashing using scrypt algorithm
- Session-based authentication
- Role-based access control (RBAC)
- SQL injection prevention through parameterized queries
- XSS protection through input sanitization
- CSRF protection with SameSite cookies

13.7 Screenshots

Figure 4: Login Screen - Shows role selection and login interface with demo credentials
Figure 5: Member Dashboard - Displays stats cards, workout logs, sessions, and navigation
Figure 6: Add Workout Modal - Form for logging new workout with all fields
Figure 7: AI Chatbot Interface - Chat window with message history and markdown rendering
Figure 8: Progress Analytics - Three interactive charts showing workout frequency, weight, and calories

14 Advanced Features Implementation

14.1 AI Integration

14.1.1 Workout Recommendations Engine

The system implements a rule-based AI recommendation engine that analyzes member data:

- **Frequency Analysis:** Recommends increasing workout frequency if less than 3 workouts per week
- **Step Count Monitoring:** Suggests cardio exercises when daily steps fall below 5,000
- **Sleep Quality:** Recommends recovery exercises when sleep is below 6 hours
- **Exercise Variety:** Suggests new exercise types when variety is low
- **Calorie Optimization:** Recommends higher intensity workouts for better calorie burn

14.1.2 GPT-4 Powered Chatbot

Available exclusively for Gold members:

- Analyzes member's complete fitness history
- Provides personalized advice based on actual data
- Generates weekly workout plans
- Offers nutrition and recovery guidance
- Supports natural language conversations
- Markdown formatting for better readability

14.2 Automated Notification System

- **Membership Renewals:** Automatic notifications 7 days before expiry
- **Session Reminders:** Reminders sent day before booked sessions
- **Progress Milestones:** Notifications at 10, 25, 50 workout milestones
- **System Messages:** Admin broadcasts and important updates
- Real-time notification badge updates
- Read/unread status tracking

14.3 Analytics and Reporting

- Member engagement scoring algorithm
- Trainer performance metrics
- Revenue calculations and projections
- Workout pattern analysis
- Health metric trend visualization
- Export capabilities (planned feature)

15 Testing and Validation

15.1 Database Testing

- **Integrity Testing:** Verified foreign key constraints and cascading operations
- **Trigger Testing:** Validated session capacity checks and notification generation
- **Procedure Testing:** Tested all stored procedures with various input scenarios
- **Cursor Testing:** Verified renewal notification batch processing
- **Transaction Testing:** Ensured ACID properties under concurrent access

15.2 Application Testing

- **Authentication:** Tested login/logout for all user roles
- **Authorization:** Verified role-based access restrictions
- **CRUD Operations:** Tested create, read, update, delete for all entities
- **API Endpoints:** Validated all REST API endpoints
- **Session Management:** Tested session persistence and expiration
- **Error Handling:** Verified graceful error handling and user feedback

15.3 Frontend Testing

- **Responsive Design:** Tested on various screen sizes and devices
- **Browser Compatibility:** Verified on Chrome, Firefox, Safari, Edge
- **Form Validation:** Tested input validation and error messages
- **Chart Rendering:** Validated data visualization accuracy
- **Modal Functionality:** Tested all modal open/close operations
- **AI Chatbot:** Validated markdown rendering and response accuracy

15.4 Performance Testing

- Query optimization with EXPLAIN analysis
- Connection pooling for database efficiency
- Lazy loading for improved page load times
- Chart rendering optimization
- API response time measurement
- Concurrent user load testing

16 Challenges and Solutions

16.1 Technical Challenges

- **Challenge:** Managing complex many-to-many relationships between Members and Sessions
- **Solution:** Used WorkoutLog as an association table with additional attributes (Exercise type differentiates between workouts and bookings)
- **Challenge:** Preventing session overbooking with concurrent requests
- **Solution:** Implemented database trigger with capacity check before insertion

- **Challenge:** Maintaining session state across API calls
- **Solution:** Implemented Flask sessions with secure cookies and sessionStorage backup
- **Challenge:** Handling SQL code blocks in LaTeX breaking across pages
- **Solution:** Used breakable tcolorbox with optimized padding and margins

16.2 Design Challenges

- **Challenge:** Creating intuitive UI for three different user roles
- **Solution:** Developed role-specific dashboards with tailored information architecture
- **Challenge:** Visualizing complex fitness data meaningfully
- **Solution:** Implemented multiple chart types with interactive features using Chart.js
- **Challenge:** Providing personalized recommendations at scale
- **Solution:** Hybrid approach using rule-based engine for real-time suggestions and GPT-4 for detailed analysis

17 Future Enhancements

17.1 Feature Additions

- **Mobile Application:** Native iOS and Android apps with offline capabilities
- **Wearable Integration:** Sync with Fitbit, Apple Watch, Garmin devices
- **Payment Gateway:** Online payment processing for membership renewals
- **Social Features:** Member challenges, leaderboards, social sharing
- **Advanced Analytics:** Machine learning for predictive analytics and churn prevention
- **Video Library:** Exercise demonstration videos with proper form instructions
- **Meal Planning:** Nutrition tracking and meal plan recommendations
- **Virtual Training:** Live video sessions with trainers

17.2 Technical Improvements

- Migration to cloud infrastructure (AWS/Azure)
- Implementation of caching layer (Redis)
- Database replication for high availability
- Microservices architecture for better scalability
- GraphQL API for flexible data querying

- Progressive Web App (PWA) capabilities
- Automated testing suite (unit, integration, E2E)
- CI/CD pipeline for automated deployments

18 Conclusion

The GymFit Tracker System successfully demonstrates the practical application of advanced database management concepts in solving real-world problems. The project achieves all specified objectives:

18.1 Technical Accomplishments

- **Database Design:** Implemented fully normalized database schema (3NF) with 9 interconnected tables, ensuring data integrity and eliminating redundancy
- **DDL Mastery:** Demonstrated comprehensive DDL operations including table creation with complex constraints, foreign key relationships, and cascading actions
- **DML Proficiency:** Implemented sophisticated data manipulation operations with transaction management, complex queries, and efficient data retrieval
- **DCL Implementation:** Applied role-based access control with granular privilege management for three distinct user types
- **PL/SQL Expertise:** Developed stored procedures, functions, triggers, and cursors for automated business logic and data processing
- **Full-Stack Development:** Created complete web application with modern frontend technologies and RESTful API architecture

18.2 Functional Achievements

- Successfully manages gym operations including memberships, trainers, sessions, and member progress
- Provides personalized AI-powered recommendations and chatbot assistance
- Automates critical workflows like membership renewals and notifications
- Delivers comprehensive analytics with visual representations
- Ensures data security through role-based access control and password hashing
- Handles concurrent users with proper transaction management

18.3 Learning Outcomes

This project provided hands-on experience with:

- Entity-Relationship modeling and normalization theory
- Complex SQL query optimization and indexing strategies
- Transaction management and concurrency control
- Stored procedures and trigger development
- Full-stack web development with database integration
- RESTful API design and implementation
- Security best practices for web applications
- AI integration for intelligent features

18.4 Impact and Applicability

The GymFit Tracker System demonstrates how database-driven applications can transform manual processes into efficient, automated systems. The principles and techniques used are applicable to various domains including:

- Healthcare management systems
- Educational institutions
- E-commerce platforms
- Hotel and restaurant management
- Employee management systems
- Inventory and supply chain management

The project successfully bridges theoretical DBMS concepts with practical implementation, creating a scalable and maintainable solution that addresses real-world business requirements.

19 References

1. C. J. Date, *An Introduction to Database Systems*, 8th ed. Boston: Addison-Wesley, 2003.
2. R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Boston: Pearson, 2015.
3. MySQL Documentation, "MySQL 8.0 Reference Manual," Oracle Corporation, 2024. [Online]. Available: https://dev.mysql.com/doc/
4. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, June 1970.

5. Flask Documentation, "Flask Web Development," Pallets Projects, 2024. [Online]. Available: https://flask.palletsprojects.com/

6. OpenAI, "GPT-4 Technical Report," OpenAI, 2024. [Online]. Available: https://platform.openai.com/docs/

7. Chart.js Documentation, "Chart.js - Open Source HTML5 Charts," 2024. [Online]. Available: https://www.chartjs.org/

8. M. Fowler, *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley Professional, 2002.

9. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

10. OWASP Foundation, "OWASP Top Ten Web Application Security Risks," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

Appendix A: Installation and Setup

Prerequisites

- Python 3.8 or higher
- MySQL 8.0 or higher
- Web browser (Chrome, Firefox, Safari, or Edge)
- OpenAI API key (for AI chatbot functionality)

Installation Steps

1. Database Setup:

```
- Install MySQL Server
- Import gymfit_database.sql using phpMyAdmin or command line
- Verify all tables are created successfully
```

2. Python Dependencies:

```
pip install flask flask-cors mysql-connector-python
pip install python-dotenv werkzeug openai
```

3. Environment Configuration:

```
Create .env file with:
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=your_password
DB_NAME=GymFitDB
SECRET_KEY=your_secret_key
OPENAI_API_KEY=your_openai_key
```

4. Run Application:

```
python app.py
Access at http://localhost:5000
```

Appendix B: API Endpoints

Authentication:

- POST /api/login - User authentication
- POST /api/logout - User logout

Member Endpoints:

- GET /api/dashboard/member/¡id¿ - Member dashboard data
- GET /api/member/¡id¿/progress - Progress analytics
- GET /api/member/¡id¿/recommendations - AI recommendations
- POST /api/member/¡id¿/chat - AI chatbot interaction
- POST /api/workouts - Add workout log

Trainer Endpoints:

- GET /api/dashboard/trainer/¡id¿ - Trainer dashboard

Admin Endpoints:

- GET /api/dashboard/admin/¡id¿ - Admin dashboard
- POST /api/admin/member - Add new member
- DELETE /api/admin/member/¡id¿ - Delete member
- POST /api/admin/trainer - Add new trainer
- DELETE /api/admin/trainer/¡id¿ - Delete trainer

Session Endpoints:

- GET /api/sessions/available - Get available sessions
- POST /api/sessions/book - Book a session
- POST /api/sessions/cancel - Cancel booking

Notification Endpoints:

- GET /api/notifications - Get user notifications
- POST /api/admin/check_renewals - Trigger renewal check

Appendix C: Demo Credentials

Members:

- Email: sujal.sokande@gmail.com — Password: theSujal866
- Email: priya.d@email.com — Password: password123
- Email: kavya.rane@gmail.com — Password: password123

Trainers:

- Email: joshua.t@gymfit.in — Password: password123
- Email: anjali.m@gymfit.in — Password: password123

Admin:

- Email: admin@gymfit.in — Password: admin123

End of Report

Submitted by:
Sujal Sokande, Kavya Rane, Megha Mahesh, Raya Gangopadhyay

Database Management System Mini Project
November 1, 2025