Database project

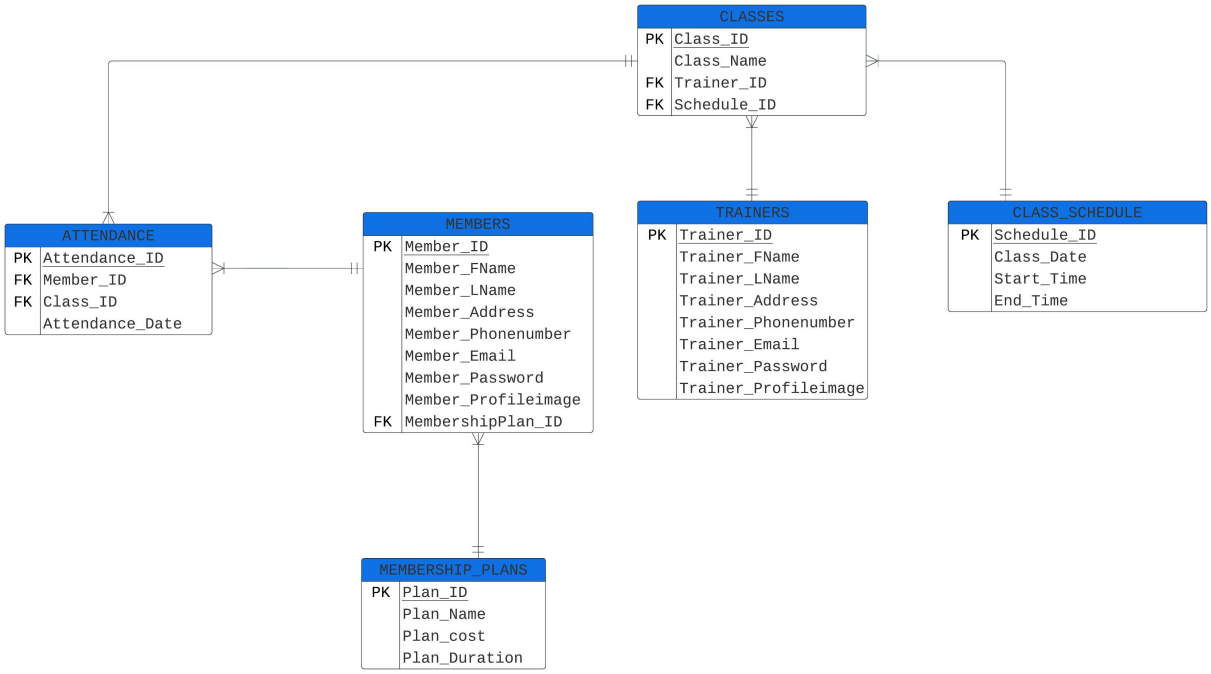
Akhigbe Paulinus

Application Description for Gym Membership Management System

The Gym Membership Management System is a comprehensive solution for efficiently managing gym operations, including memberships, trainers, classes, and attendance. Key features include:

- Membership Registration: Easily register new members with secure username/password login and profile picture uploads.
- Trainer Management: Assign trainers to members and classes, and maintain detailed trainer profiles.
- Class Scheduling: Schedule and manage various gym classes, ensuring optimal resource allocation and member satisfaction.
- Attendance Tracking: Monitor and record attendance for members and trainers, providing insights into participation and engagement.

This system ensures streamlined gym management with enhanced security and user-friendly functionalities.



Comment on Database Design

Considering the database design principles discussed during the lecture and in the text-book (Chapters 1-4):

- **Entity Integrity:**

Each table has a primary key (PK) that uniquely identifies each record. Primary keys are chosen appropriately to ensure entity integrity.

- **Referential Integrity:**

Foreign keys (FK) are used to link tables, ensuring that relationships between tables are maintained. Foreign keys are used correctly to represent the relationships between entities, such as between Members and MembershipPlans, Classes and Trainers, Classes and ClassSchedules, and Attendance and both Members and Classes.

- **Data Integrity:**

Data types are chosen appropriately for each attribute to ensure data integrity. Constraints such as NOT NULL and UNIQUE are applied where necessary to maintain data quality.

- **Normalization:**

The database design adheres to normalization principles to reduce redundancy and dependency. Tables are structured to ensure minimal redundancy and to support efficient querying.

Improvements

- **Indexing:**

Add indexes on frequently queried columns like Member_Email and Trainer_Email to enhance query performance.

- **Security:**

Ensure that passwords are stored using secure hashing techniques (bcrypt is used in the Python script).

Normalization

Current Normal Form:

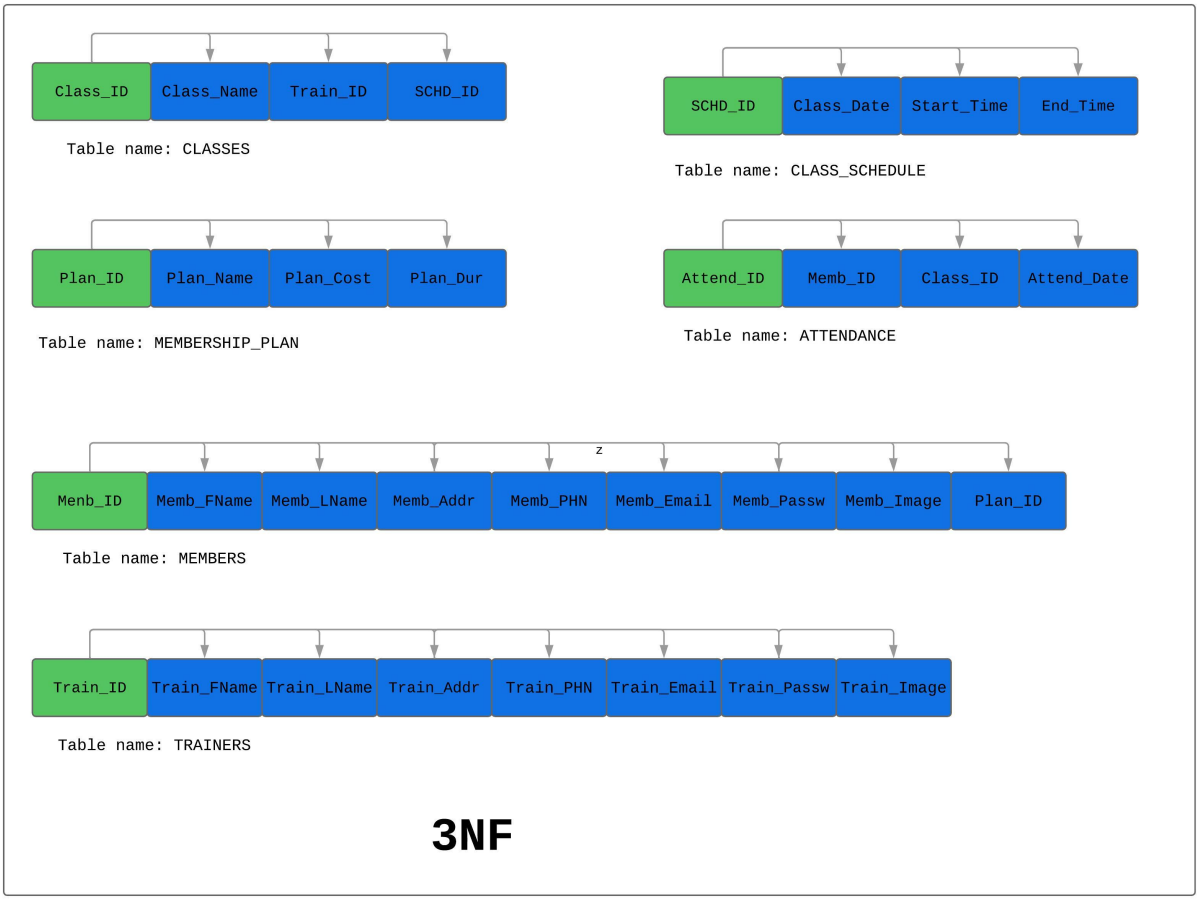
The database was designed to be in Third Normal Form (3NF):

1NF: All tables have atomic columns, and each column contains unique values.

2NF: All non-key attributes are fully functionally dependent on the primary key. There are no partial dependencies.

3NF: There are no transitive dependencies. Non-key attributes are not dependent on other non-key attributes.

Verification of 3NF:



Higher Normal Forms BCNF (Boyce-Codd Normal Form):

The database already conforms to BCNF because every determinant is a candidate key.

4NF (Fourth Normal Form):

The database does not have multi-valued dependencies, so 4NF is satisfied.

5NF (Fifth Normal Form):

The design ensures that all join dependencies are implied by candidate keys, so 5NF is satisfied.

Given the current requirements, further normalization beyond 3NF (such as BCNF, 4NF, or 5NF) may not provide significant benefits and might complicate the design unnecessarily. The database is well-structured and normalized to 3NF, ensuring data integrity and efficient querying.

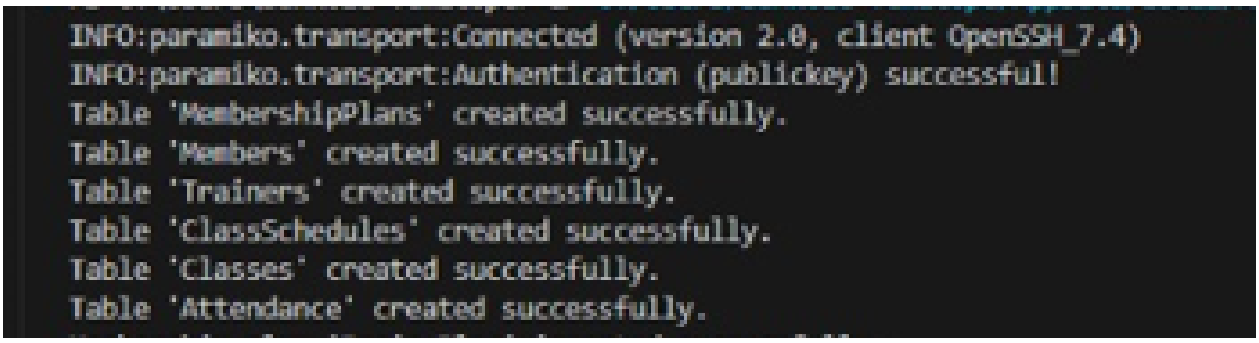
Test Table Creation

Description: Verify that the tables are created correctly.

Input: Code for table creation.

Expected Output: Tables created successfully without errors.

Actual Output:



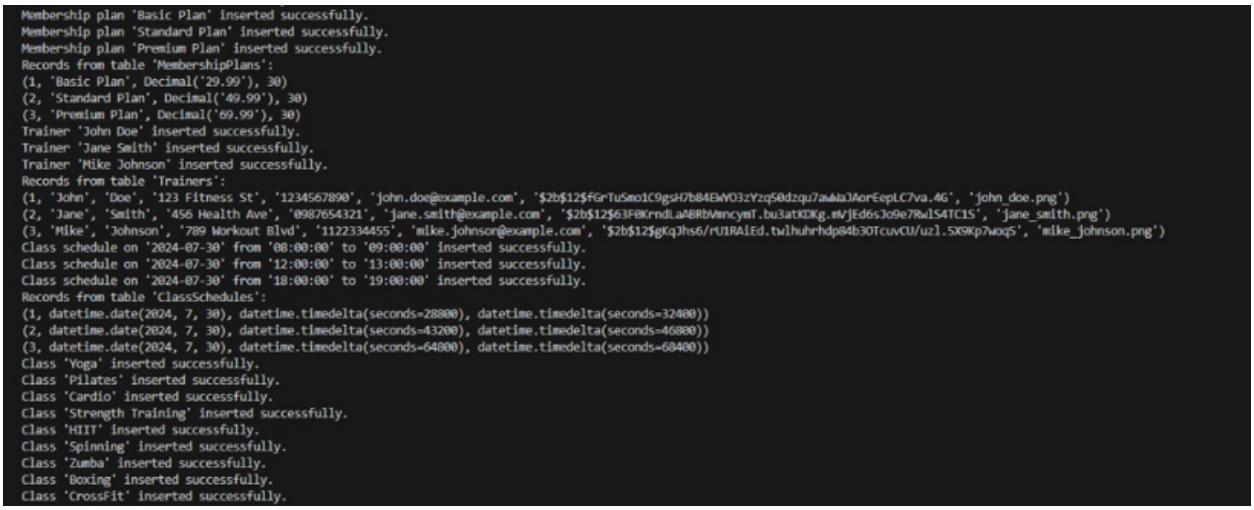
Test Insert Methods

Description: Verify that data is inserted into each table correctly.

Input: Values to be inserted.

Expected Output: Data inserted successfully without errors.

Actual Output:



Test Update Membership Cost: Verify that the cost of a membership plan can be updated.

Test Get Member by Email: Verify that a member's information can be retrieved using their email.

```
#pip install pymysql
import pymysql
#pip install passlib and pip install argon2_cffi
from passlib.hash import argon2
#you don't have to run a pip insall csv its a default package.
import bcrypt
#pip install sshtunnel
from sshtunnel import SSHTunnelForwarder

# Step 1: Establishing a connection
ssh_host = '192.197.151.116'
ssh_port = 22
ssh_username = 'paulinusakhigbe'
ssh_key_path = 'paulinusakhigbe.private'
ssh_password = 'Ilovefood@123'
mysql_host = '127.0.0.1'
mysql_port = 3306
mysql_user = 'paulinusakhigbe'
mysql_password = 'Iphone15@pro'
mysql_db = 'paulinusakhigbe'

# Create an SSH tunnel
with SSHTunnelForwarder(
    (ssh_host, ssh_port),
    ssh_username=ssh_username,
    ssh_password=ssh_password,
    ssh_pkey=ssh_key_path,
    remote_bind_address=(mysql_host, mysql_port)
) as tunnel:
    connection = pymysql.connect(
        host='127.0.0.1',
        user=mysql_user,
        password=mysql_password,
        database=mysql_db,
        port=tunnel.local_bind_port,
    )

    cursor = connection.cursor()

    # Step 2: Creating tables and relationships
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS MembershipPlans (
        Plan_ID INT PRIMARY KEY AUTO_INCREMENT,
        Plan_Name VARCHAR(100) NOT NULL,
        Plan_Cost DECIMAL(10, 2) NOT NULL,
```

```
    Plan_Duration INT NOT NULL
)
''')
print("Table 'MembershipPlans' created successfully.")

cursor.execute('''
CREATE TABLE IF NOT EXISTS Members (
    Member_ID INT PRIMARY KEY AUTO_INCREMENT,
    Member_FName VARCHAR(100) NOT NULL,
    Member_LName VARCHAR(100) NOT NULL,
    Member_Address VARCHAR(255),
    Member_Phonenumber VARCHAR(20),
    Member_Email VARCHAR(100) NOT NULL UNIQUE,
    Member_Password VARCHAR(255) NOT NULL,
    Member_ProfileImage VARCHAR(255),
    MembershipPlan_ID INT,
    FOREIGN KEY (MembershipPlan_ID) REFERENCES MembershipPlans(Plan_ID)
)
''')
print("Table 'Members' created successfully.")

cursor.execute('''
CREATE TABLE IF NOT EXISTS Trainers (
    Trainer_ID INT PRIMARY KEY AUTO_INCREMENT,
    Trainer_FName VARCHAR(100) NOT NULL,
    Trainer_LName VARCHAR(100) NOT NULL,
    Trainer_Address VARCHAR(255),
    Trainer_Phonenumber VARCHAR(20),
    Trainer_Email VARCHAR(100) NOT NULL UNIQUE,
    Trainer_Password VARCHAR(255) NOT NULL,
    Trainer_ProfileImage VARCHAR(255)
)
''')
print("Table 'Trainers' created successfully.")

cursor.execute('''
CREATE TABLE IF NOT EXISTS ClassSchedules (
    Schedule_ID INT PRIMARY KEY AUTO_INCREMENT,
    Class_Date DATE NOT NULL,
    Start_Time TIME NOT NULL,
    End_Time TIME NOT NULL
)
''')
print("Table 'ClassSchedules' created successfully.")
```

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Classes (
    Class_ID INT PRIMARY KEY AUTO_INCREMENT,
    Class_Name VARCHAR(100) NOT NULL,
    Trainer_ID INT,
    Schedule_ID INT,
    FOREIGN KEY (Trainer_ID) REFERENCES Trainers(Trainer_ID),
    FOREIGN KEY (Schedule_ID) REFERENCES ClassSchedules(Schedule_ID)
)
''')
print("Table 'Classes' created successfully.")

cursor.execute('''
CREATE TABLE IF NOT EXISTS Attendance (
    Attendance_ID INT PRIMARY KEY AUTO_INCREMENT,
    Member_ID INT,
    Class_ID INT,
    Attendance_Date DATE NOT NULL,
    FOREIGN KEY (Member_ID) REFERENCES Members(Member_ID),
    FOREIGN KEY (Class_ID) REFERENCES Classes(Class_ID)
)
''')
print("Table 'Attendance' created successfully.")

connection.commit()

# Step 3: Insert methods with validation and error handling

def insert_membership_plan(plan_name, plan_cost, plan_duration):
    try:
        cursor.execute('''
        INSERT INTO MembershipPlans (Plan_Name, Plan_Cost, Plan_Duration)
        VALUES (%s, %s, %s)
        ''', (plan_name, plan_cost, plan_duration))
        connection.commit()
        print(f"Membership plan '{plan_name}' inserted successfully.")
    except Exception as e:
        print(f"Error inserting membership plan: {e}")

def insert_member(first_name, last_name, address, phone, email, password, profile_im
    try:
        hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
        cursor.execute('''
        INSERT INTO Members (Member_FName, Member_LName, Member_Address, Member_Phon
        Member_Email, Member_Password, Member_ProfileImage, MembershipPlan_ID)
```

```
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            ''', (first_name, last_name, address, phone, email, hashed_password, profile
            connection.commit()
            print(f"Member '{first_name} {last_name}' inserted successfully.")
        except Exception as e:
            print(f"Error inserting member: {e}")

    def insert_trainer(first_name, last_name, address, phone, email, password, profile_i
        try:
            hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
            cursor.execute('''
            INSERT INTO Trainers (Trainer_FName, Trainer_LName, Trainer_Address, Trainer
            Trainer_Email, Trainer_Password, Trainer_ProfileImage)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
            ''', (first_name, last_name, address, phone, email, hashed_password, profile
            connection.commit()
            print(f"Trainer '{first_name} {last_name}' inserted successfully.")
        except Exception as e:
            print(f"Error inserting trainer: {e}")

    def insert_class_schedule(class_date, start_time, end_time):
        try:
            cursor.execute('''
            INSERT INTO ClassSchedules (Class_Date, Start_Time, End_Time)
            VALUES (%s, %s, %s)
            ''', (class_date, start_time, end_time))
            connection.commit()
            print(f"Class schedule on '{class_date}' from '{start_time}' to '{end_time}'
        except Exception as e:
            print(f"Error inserting class schedule: {e}")

    def insert_class(class_name, trainer_id, schedule_id):
        try:
            cursor.execute('''
            INSERT INTO Classes (Class_Name, Trainer_ID, Schedule_ID)
            VALUES (%s, %s, %s)
            ''', (class_name, trainer_id, schedule_id))
            connection.commit()
            print(f"Class '{class_name}' inserted successfully.")
        except Exception as e:
            print(f"Error inserting class: {e}")

    def insert_attendance(member_id, class_id, attendance_date):
        try:
            cursor.execute('''
```

```
            INSERT INTO Attendance (Member_ID, Class_ID, Attendance_Date)
            VALUES (%s, %s, %s)
            ''', (member_id, class_id, attendance_date))
            connection.commit()
            print(f"Attendance for member ID '{member_id}' in class ID '{class_id}' on '
        except Exception as e:
            print(f"Error inserting attendance: {e}")

    # Step 4: Fetch all records from a table

    def fetch_all_records(table_name):
        try:
            cursor.execute(f'SELECT * FROM {table_name}')
            records = cursor.fetchall()
            print(f"Records from table '{table_name}':")
            for record in records:
                print(record)
        except Exception as e:
            print(f"Error fetching records from {table_name}: {e}")

    # Step 5: Remove methods

    def remove_membership_plan(plan_id):
        try:
            cursor.execute('DELETE FROM MembershipPlans WHERE Plan_ID = %s', (plan_id,))
            connection.commit()
            print(f"Membership plan ID '{plan_id}' deleted successfully.")
        except Exception as e:
            print(f"Error removing membership plan: {e}")

    def remove_member(member_id):
        try:
            cursor.execute('DELETE FROM Members WHERE Member_ID = %s', (member_id,))
            connection.commit()
            print(f"Member ID '{member_id}' deleted successfully.")
        except Exception as e:
            print(f"Error removing member: {e}")

    def remove_trainer(trainer_id):
        try:
            cursor.execute('DELETE FROM Trainers WHERE Trainer_ID = %s', (trainer_id,))
            connection.commit()
            print(f"Trainer ID '{trainer_id}' deleted successfully.")
        except Exception as e:
            print(f"Error removing trainer: {e}")
```

```
def remove_class_schedule(schedule_id):
    try:
        cursor.execute('DELETE FROM ClassSchedules WHERE Schedule_ID = %s', (schedul
        connection.commit()
        print(f"Class schedule ID '{schedule_id}' deleted successfully.")
    except Exception as e:
        print(f"Error removing class schedule: {e}")

def remove_class(class_id):
    try:
        cursor.execute('DELETE FROM Classes WHERE Class_ID = %s', (class_id,))
        connection.commit()
        print(f"Class ID '{class_id}' deleted successfully.")
    except Exception as e:
        print(f"Error removing class: {e}")

def remove_attendance(attendance_id):
    try:
        cursor.execute('DELETE FROM Attendance WHERE Attendance_ID = %s', (attendanc
        connection.commit()
        print(f"Attendance ID '{attendance_id}' deleted successfully.")
    except Exception as e:
        print(f"Error removing attendance: {e}")

# Step 6: Additional useful operations

def get_member_by_email(email):
    try:
        cursor.execute('SELECT * FROM Members WHERE Member_Email = %s', (email,))
        return cursor.fetchone()
    except Exception as e:
        print(f"Error fetching member by email: {e}")

def update_membership_plan_cost(plan_id, new_cost):
    try:
        cursor.execute('UPDATE MembershipPlans SET Plan_Cost = %s WHERE Plan_ID = %s
        connection.commit()
        print(f"Membership plan ID '{plan_id}' cost updated to '{new_cost}' successf
    except Exception as e:
        print(f"Error updating membership plan cost: {e}")

# Insert Membership Plans
membership_plans = [
    ('Basic Plan', 29.99, 30),
```

```
    ('Standard Plan', 49.99, 30),
    ('Premium Plan', 69.99, 30)
]

for plan in membership_plans:
    insert_membership_plan(plan[0], plan[1], plan[2])

fetch_all_records('MembershipPlans')

# Insert Trainers
trainers = [
    ('John', 'Doe', '123 Fitness St', '1234567890', 'john.doe@example.com', 'passwor
    ('Jane', 'Smith', '456 Health Ave', '0987654321', 'jane.smith@example.com', 'pas
    ('Mike', 'Johnson', '789 Workout Blvd', '1122334455', 'mike.johnson@example.com'
]

for trainer in trainers:
    insert_trainer(trainer[0], trainer[1], trainer[2], trainer[3], trainer[4], train

fetch_all_records('Trainers')

# Insert Class Schedules
class_schedules = [
    ('2024-07-30', '08:00:00', '09:00:00'), # Morning
    ('2024-07-30', '12:00:00', '13:00:00'), # Afternoon
    ('2024-07-30', '18:00:00', '19:00:00')  # Evening
]

for schedule in class_schedules:
    insert_class_schedule(schedule[0], schedule[1], schedule[2])

fetch_all_records('ClassSchedules')

# Insert Classes
classes = [
    ('Yoga', 1, 1), # Morning class with John Doe
    ('Pilates', 2, 1), # Morning class with Jane Smith
    ('Cardio', 3, 1), # Morning class with Mike Johnson
    ('Strength Training', 1, 2), # Afternoon class with John Doe
    ('HIIT', 2, 2), # Afternoon class with Jane Smith
    ('Spinning', 3, 2), # Afternoon class with Mike Johnson
    ('Zumba', 1, 3), # Evening class with John Doe
    ('Boxing', 2, 3), # Evening class with Jane Smith
    ('CrossFit', 3, 3) # Evening class with Mike Johnson
]
```

```
for cls in classes:
    insert_class(cls[0], cls[1], cls[2])

fetch_all_records('Classes')

# Insert Members
members = [
    ('Alice', 'Wonderland', '101 Fantasy Road', '5551234567', 'alice@example.com', '
    ('Bob', 'Builder', '202 Construction Lane', '5557654321', 'bob@example.com', 'pa
    ('Charlie', 'Brown', '303 Comic Strip', '5559988776', 'charlie@example.com', 'pa
]

for member in members:
    insert_member(member[0], member[1], member[2], member[3], member[4], member[5],

fetch_all_records('Members')

# Insert Attendance Records
attendance_records = [
    (1, 1, '2024-07-30'),
    (2, 2, '2024-07-30'),
    (3, 3, '2024-07-30')
]

for record in attendance_records:
    insert_attendance(record[0], record[1], record[2])

fetch_all_records('Attendance')

# Update a membership plan cost
update_membership_plan_cost(1, 34.99)

# Fetch a member by email
member = get_member_by_email('alice@example.com')
print(f"Fetched member by email: {member}")

# Remove records (demonstrating deletion)
remove_attendance(1)
remove_class(1)
remove_member(1)
remove_membership_plan(1)

# Close the connection
cursor.close()
```

```
    connection.close()
```