

Fitness Club Database Report

Student ID: 20067368

Name: Chinni Raj Paul Juthiga

A. Data Generation Method

The Python Faker library was used to produce data for the Fitness Club database, ensuring originality and relevance to the fitness industry. This strategy enabled me to create a realistic yet fictitious dataset that accurately reflects the operations and activities characteristic of a fitness club. The dataset contains detailed information about members, trainers, and classes offered by the club.

Using the Faker package, I was able to generate a variety of realistic names, locations, membership tiers, and class details automatically. This strategy made it much easier to create a diversified and high-quality dataset while also ensuring that the assignment's requirements for nominal, ordinal, interval, and ratio data types were met. The randomized data for dates and times was carefully selected to match plausible events in a fitness club context, bringing an extra layer of credibility to the database.

```
import sqlite3
from faker import Faker
import random

# Initialize Faker
fake = Faker()

# Connect to SQLite database
conn = sqlite3.connect('FitnessClubDB1.sqlite')
cursor = conn.cursor()

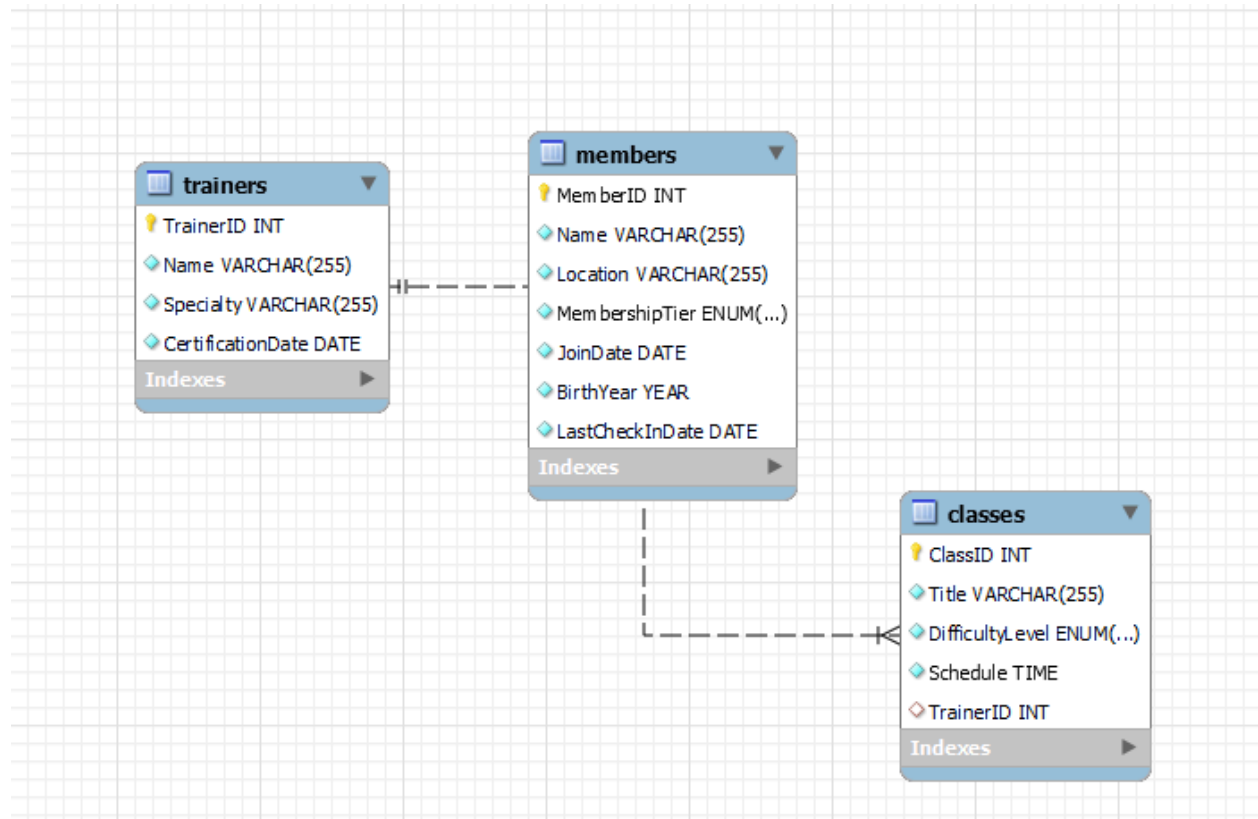
# Drop the existing Members table if it exists and recreate it
cursor.execute('DROP TABLE IF EXISTS Members')

cursor.execute('''
CREATE TABLE IF NOT EXISTS Members (
    MemberID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Location TEXT NOT NULL UNIQUE,
    MembershipTier TEXT NOT NULL CHECK(MembershipTier IN ('Basic', 'Premium', 'VIP')),
    JoinDate DATE NOT NULL,
    BirthYear INTEGER NOT NULL,
    LastCheckInDate DATE NOT NULL
)
''')

# Generate and insert 1000 rows of fake data
membership_tiers = ['Basic', 'Premium', 'VIP']

for _ in range(1000):
    try:
        cursor.execute('''
INSERT INTO Members (Name, Location, MembershipTier, JoinDate, BirthYear, LastCheckInDate)
VALUES (?, ?, ?, ?, ?, ?)
''', (
        fake.name(),
        fake.unique.city(),
```

B. Database Schema



In this database, I created three primary tables: Members, Trainers, and Classes. Each of these tables has a specific purpose within the database, containing information related to the operation and management of a fitness club. The Members table contains information on the club's members, the Trainers table has information about the fitness trainers, and the Classes table covers the various fitness classes available, including when they take place and who leads them.

To meet the requirements, the database has 1000 rows and 7 columns across these tables, with a diverse range of data types represented to ensure a comprehensive data model:

1. **Nominal Data:** This type is exemplified in the Name columns within both the Members and Trainers tables, as well as the Location column in the Members table and the Specialty column in the Trainers table. Nominal data is used for categorization or labeling without implying any numerical order.
2. **Ordinal Data:** The MembershipTier column in the Members table and the DifficultyLevel column in the Classes table represent ordinal data. These columns classify data into categories that do have an inherent order (e.g., Basic, Premium, VIP for membership tiers and Beginner, Intermediate, Advanced for class difficulty levels).
3. **Interval Data:** Dates such as JoinDate, CertificationDate, and LastCheckInDate across the Members and Trainers tables serve as interval data. This type of data is measured along a scale, where both the order and the exact differences between the values are meaningful.

4. **Ratio Data:** The MemberID and TrainerID fields act as ratio data since they not only order and measure the difference between values but also have a true zero point, indicating the absence of a member or trainer. This allows for operations such as multiplication and division to be meaningful.

C. Justification for Separate Tables and Ethical Discussion

I normalized the database by creating distinct tables for members, trainers, and classes, decreasing redundancy and improving data integrity. This deliberate split allows for more effective data handling and scalability. Trainer information, for example, is kept once in the Trainers table and then referenced in the Classes table, preventing data duplication.

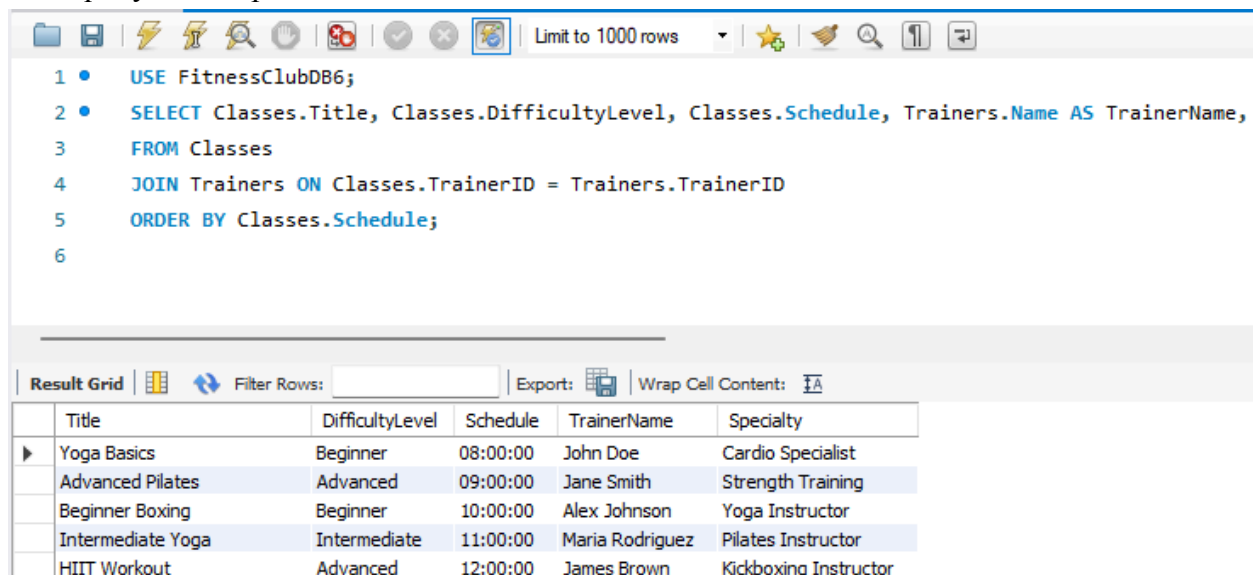
Ethically, this database design takes privacy and data protection seriously. Rather than retaining critical personal information like phone numbers, I opted to provide less sensitive data like addresses. This decision demonstrates my dedication to minimize potential harm or intrusions of privacy while allowing for important data analysis and operational functionality.

D. Example Queries

These queries demonstrate interactions with the database, showcasing the use of joins and selections across different data types:

Join Query to Find Classes with Trainers' Details:

Thee query and output are as follows:



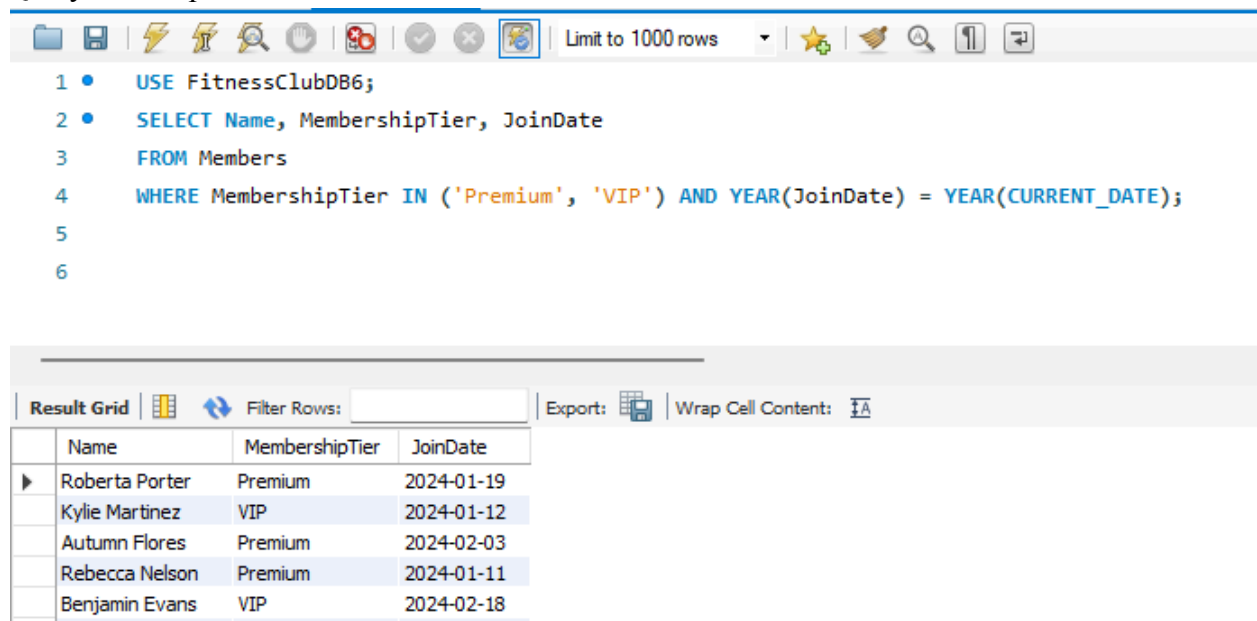
The screenshot shows a database query editor interface. At the top, there's a toolbar with various icons and a dropdown menu set to 'Limit to 1000 rows'. Below the toolbar, the query text is displayed in a monospaced font. The query is a JOIN query that selects class details and trainer information. Below the query, there's a 'Result Grid' section with a table of results. The table has six columns: Title, DifficultyLevel, Schedule, TrainerName, and Specialty. It contains five rows of data representing different classes and their respective trainers.

```
1 • USE FitnessClubDB6;  
2 • SELECT Classes.Title, Classes.DifficultyLevel, Classes.Schedule, Trainers.Name AS TrainerName,  
3 FROM Classes  
4 JOIN Trainers ON Classes.TrainerID = Trainers.TrainerID  
5 ORDER BY Classes.Schedule;  
6
```

Title	DifficultyLevel	Schedule	TrainerName	Specialty
Yoga Basics	Beginner	08:00:00	John Doe	Cardio Specialist
Advanced Pilates	Advanced	09:00:00	Jane Smith	Strength Training
Beginner Boxing	Beginner	10:00:00	Alex Johnson	Yoga Instructor
Intermediate Yoga	Intermediate	11:00:00	Maria Rodriguez	Pilates Instructor
HIIT Workout	Advanced	12:00:00	James Brown	Kickboxing Instructor

Selection Query to Find Premium or VIP Members Who Joined This Year:

Query and output are as shown below:



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

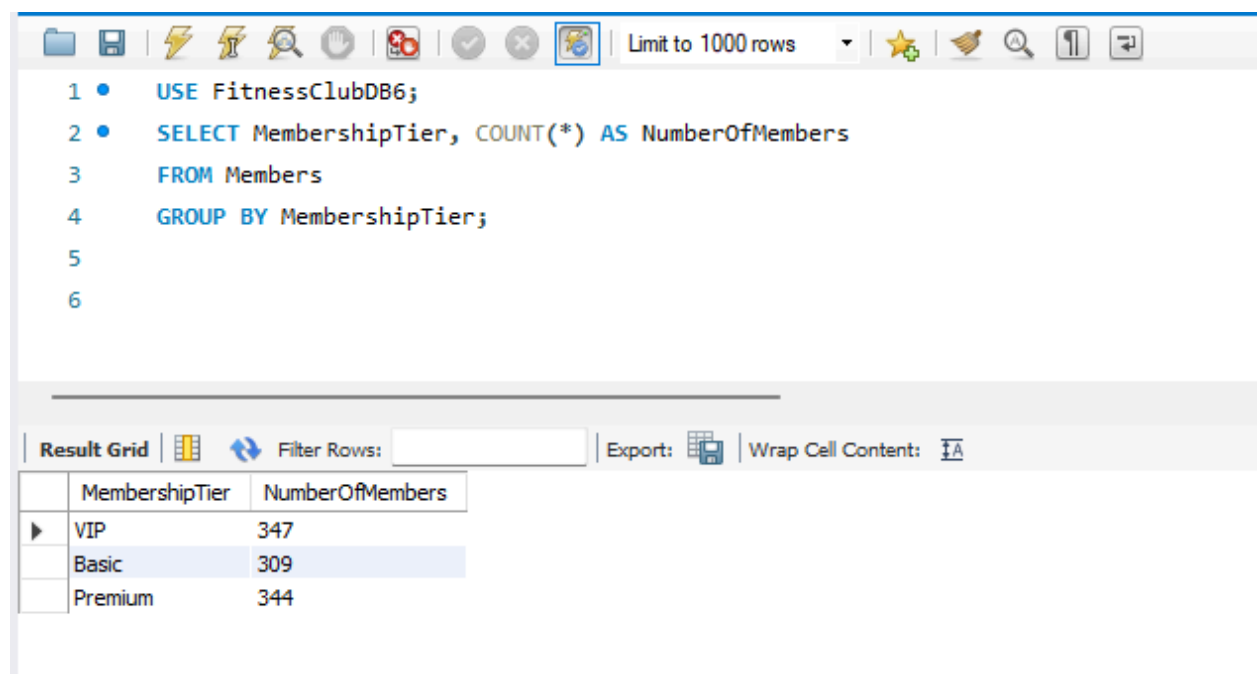
```
1 • USE FitnessClubDB6;  
2 • SELECT Name, MembershipTier, JoinDate  
3 FROM Members  
4 WHERE MembershipTier IN ('Premium', 'VIP') AND YEAR(JoinDate) = YEAR(CURRENT_DATE);  
5  
6
```

Below the query editor is the result grid. It has a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The grid displays the following data:

	Name	MembershipTier	JoinDate
▶	Roberta Porter	Premium	2024-01-19
	Kylie Martinez	VIP	2024-01-12
	Autumn Flores	Premium	2024-02-03
	Rebecca Nelson	Premium	2024-01-11
	Benjamin Evans	VIP	2024-02-18

Aggregate Query to Count Members by MembershipTier:

Query and output are as shown below:



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
1 • USE FitnessClubDB6;  
2 • SELECT MembershipTier, COUNT(*) AS NumberOfMembers  
3 FROM Members  
4 GROUP BY MembershipTier;  
5  
6
```

Below the query editor is the result grid. It has a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The grid displays the following data:

	MembershipTier	NumberOfMembers
▶	VIP	347
	Basic	309
	Premium	344