Name :- Tiasa Malitya

Student ID :- 24005709

Course :- M.Sc. Data Science

Module :- **7PAM2005-0105-2024 - Data Mining and Discovery**

Title :- **SQL Assignment**

Date :- 14/03/2025

Making of HSV Gym Database

A Brief Report on HSV Gym Database

Introduction

The report is based on the creation of a realistic yet well-structured relational gym database which has been synthetically generated using python. The database is effective for data management to track the overall gym operations like memberships, payments, class registrations etc which enables insightful analysis, such as identifying peak joining periods, preferred payment methods, and revenue trends helping in strategic decision-making.

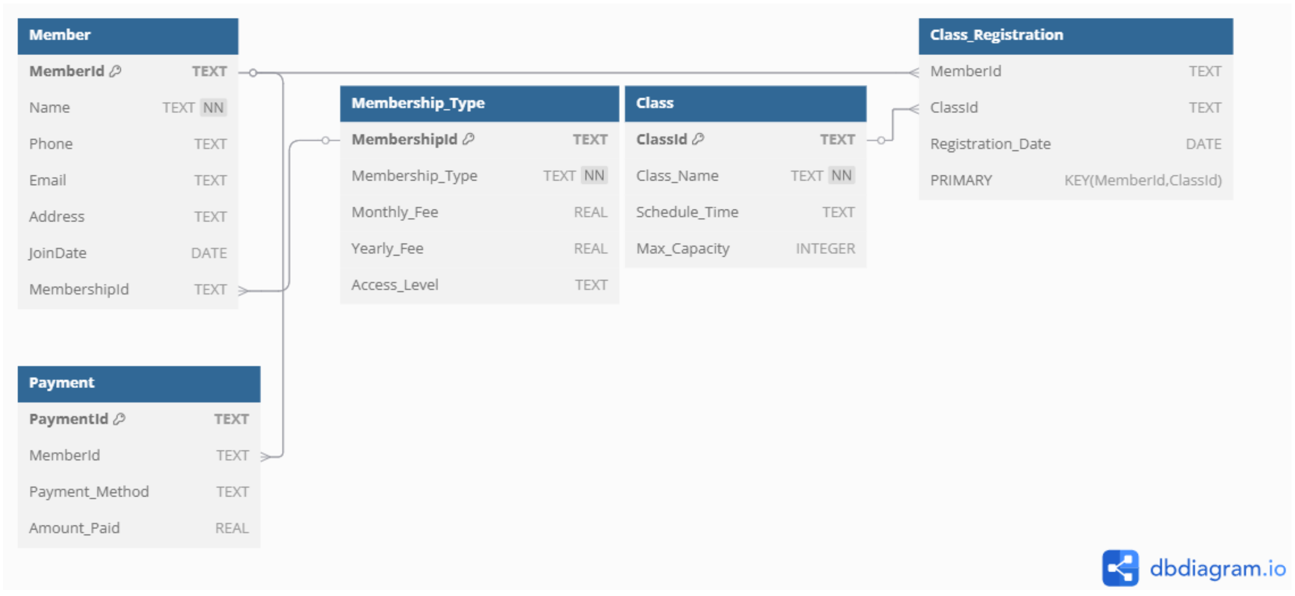
Process of Data Generation

To ensure the realism and randomness of the database different libraries have been imported. The **Faker** library was utilized to create realistic names, phone numbers, email ids, and addresses while **NumPy** was used for handling the distribution of numerical attributes and to introduce missing value, duplicate value simulating real-world human errors. After creating the data of all the columns, **Pandas** library helped to structure each table with their corresponding columns by formatting Data Frames. These Data Frames were then converted to **CSV files**, which were later imported into **SQLite** using **DB Browser** where the schema, constraints, and relationships were defined to maintain data consistency.

```
import numpy as np
import pandas as pd
from faker import Faker
fake = Faker()                    #initializing
```

Schema of Gym database

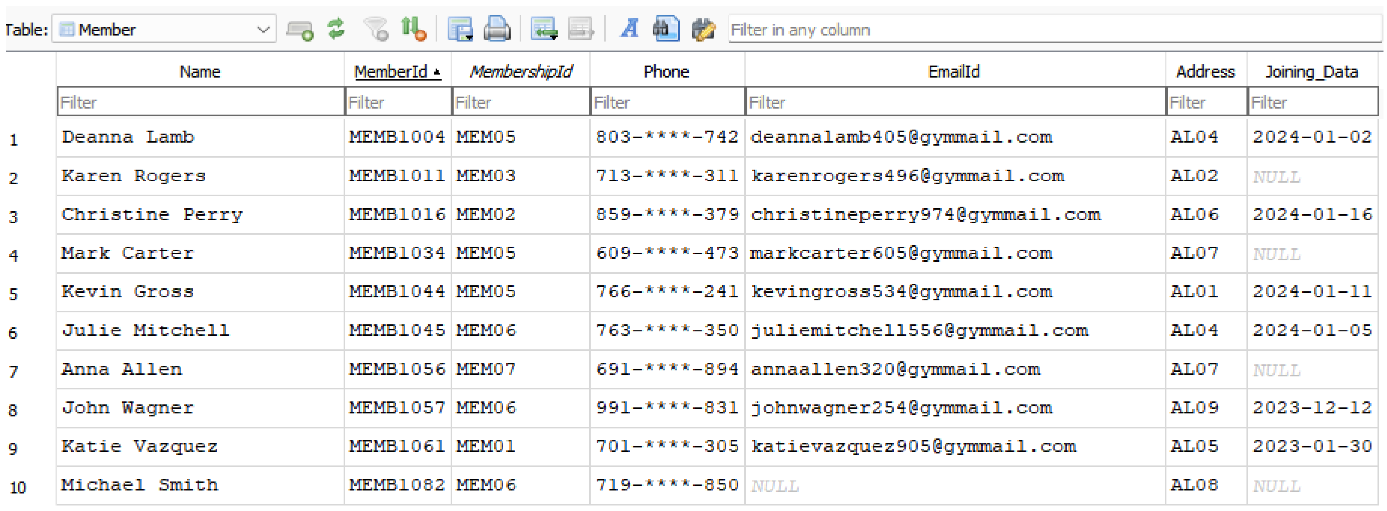
The database follows a relational schema with multiple tables connected via foreign keys. Following is the Graphical Representation of Gym database Schema.



Justification behind structure of Tables and constraint of attributes

➢ **Members Table [1000 rows]**

Designed to store essential details about the members of HSV gym.



	Name	MemberId	MembershipId	Phone	EmailId	Address	Joining_Data
1	Deanna Lamb	MEMB1004	MEM05	803-****-742	deannalamb405@gymmail.com	AL04	2024-01-02
2	Karen Rogers	MEMB1011	MEM03	713-****-311	karenrogers496@gymmail.com	AL02	NULL
3	Christine Perry	MEMB1016	MEM02	859-****-379	christineperry974@gymmail.com	AL06	2024-01-16
4	Mark Carter	MEMB1034	MEM05	609-****-473	markcarter605@gymmail.com	AL07	NULL
5	Kevin Gross	MEMB1044	MEM05	766-****-241	kevingross534@gymmail.com	AL01	2024-01-11
6	Julie Mitchell	MEMB1045	MEM06	763-****-350	juliemitchell556@gymmail.com	AL04	2024-01-05
7	Anna Allen	MEMB1056	MEM07	691-****-894	annaallen320@gymmail.com	AL07	NULL
8	John Wagner	MEMB1057	MEM06	991-****-831	johnwagner254@gymmail.com	AL09	2023-12-12
9	Katie Vazquez	MEMB1061	MEM01	701-****-305	katievazquez905@gymmail.com	AL05	2023-01-30
10	Michael Smith	MEMB1082	MEM06	719-****-850	NULL	AL08	NULL

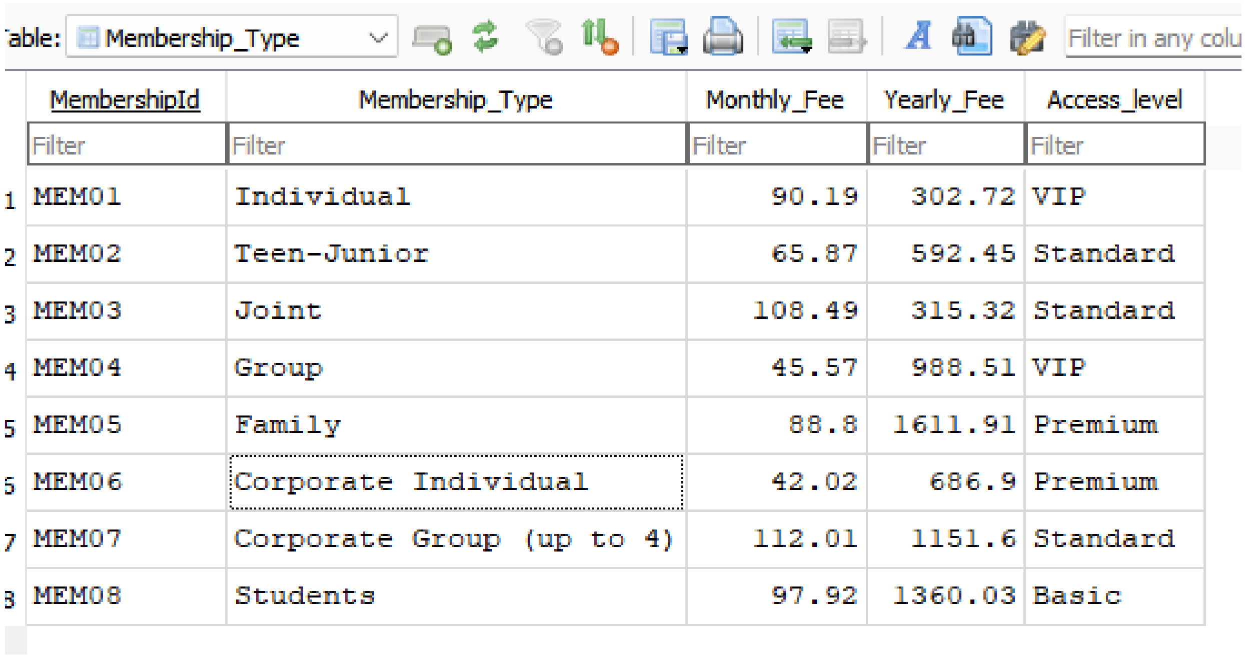
MemberId [Primary Key]: The primary identifier to uniquely track members.

```
#Creating Member ids
member_ids = [f"MEMB{num}" for num in np.random.choice(range(1000, 9999), size=n, replace=False)]
```

MembershipId [FK]: Linked to the Membership_Types table to categorize members based on their subscription.
Phone Number & Email: Set as unique for email where available to prevent duplicate entries. However, nullable to accommodate users who do not provide contact details.
Address: To maintain data privacy, only the first four characters of the postcode were stored in the address column instead of the full postal code. This ensures that location data remains general while still allowing for geographic analysis.
Join Date: A required field to track start dates of members which has been skewed toward January to reflect real-world seasonal trends.

- **Membership_Type Table [8 rows]**

Stored different membership plans and their pricing.



	MembershipId	Membership_Type	Monthly_Fee	Yearly_Fee	Access_level
1	MEM01	Individual	90.19	302.72	VIP
2	MEM02	Teen-Junior	65.87	592.45	Standard
3	MEM03	Joint	108.49	315.32	Standard
4	MEM04	Group	45.57	988.51	VIP
5	MEM05	Family	88.8	1611.91	Premium
5	MEM06	Corporate Individual	42.02	686.9	Premium
7	MEM07	Corporate Group (up to 4)	112.01	1151.6	Standard
3	MEM08	Students	97.92	1360.03	Basic

MembershipId [Primary Key]: Identifies each membership type uniquely.

```
#Creating membership ids
membership_id = ["MEM01", "MEM02", "MEM03", "MEM04", "MEM05", "MEM06", "MEM07", "MEM08"]
probabilities = [0.25, 0.10, 0.10, 0.10, 0.20, 0.10, 0.10, 0.05]
membership_ids= np.random.choice(membership_id, size=n, p=probabilities)
```

Monthly & Yearly Fees: Numerical values of membership's price to facilitate financial calculations which are constrained to positive values to prevent incorrect entries. By defining an appropriate minimum and maximum range, the uniform distribution ensures that prices remain within reasonable and expected limits.

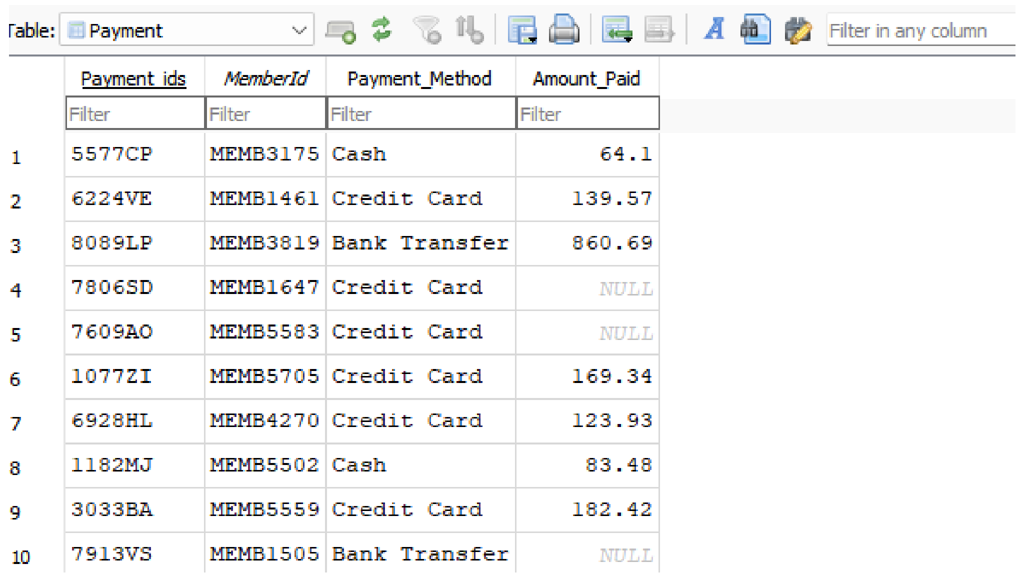
```
monthly_price = np.round(np.random.uniform(30, 150, 8), 2)
yearly_price = np.round(np.random.uniform(300, 1800 ,8),2)
```

Access Level: Indicates facilities available for each membership. By including this field, the gym can enforce policies—ensuring that members only use the facilities they are entitled to.

```
CREATE TABLE "Membership_Type" (
    "MembershipId"  TEXT NOT NULL UNIQUE,
    "Membership_Type"   TEXT NOT NULL,
    "Monthly_Fee"   REAL CHECK("Monthly_Fee" >= 0),
    "Yearly_Fee"    REAL CHECK("Yearly_Fee" >= 0),
    "Access_level"  TEXT CHECK("access_level" IN ('Basic', 'Standard', 'Premium', 'VIP')),
    PRIMARY KEY("MembershipId")
);
```

➢ **Payment Table [1000 rows]**

Tracks transactions of members.



	Payment_ids	MemberId	Payment_Method	Amount_Paid
	Filter	Filter	Filter	Filter
1	5577CP	MEMB3175	Cash	64.1
2	6224VE	MEMB1461	Credit Card	139.57
3	8089LP	MEMB3819	Bank Transfer	860.69
4	7806SD	MEMB1647	Credit Card	NULL
5	7609AO	MEMB5583	Credit Card	NULL
6	1077ZI	MEMB5705	Credit Card	169.34
7	6928HL	MEMB4270	Credit Card	123.93
8	1182MJ	MEMB5502	Cash	83.48
9	3033BA	MEMB5559	Credit Card	182.42
10	7913VS	MEMB1505	Bank Transfer	NULL

PaymentId [Primary Key]: To keep each transaction unique for accurate records.
MemberId [FK]: Links to the **Member** table to associate payments with individuals.
Payment_Method: How members complete their transactions, whether through Credit Card, Cash, or Bank Transfer which helps analyse payment trends, offering discounts for specific payment methods and to verifying transaction details.

```
CREATE TABLE "Payment" (
    "Payment_ids"   TEXT,
    "MemberId"  TEXT NOT NULL,
    "Payment_Method"    TEXT CHECK("Payment_Method" IN ('Credit Card', 'Cash', 'Bank Transfer')),
    "Amount_Paid"   REAL CHECK("amount_paid" > 0),
    PRIMARY KEY("Payment_ids"),
    FOREIGN KEY("MemberId") REFERENCES "Member"("MemberId")
);
```

Amount Paid: Numeric field where amounts varied based on the **payment method**, ensuring realistic financial transactions with non-negative payment constraint.

```
amount_paid = np.where(
    payment_methods == 'Credit Card',
    np.round(np.random.uniform(100, 200, n), 2),  # Middle range for Credit Card
    np.where(
        payment_methods == 'Cash',
        np.round(np.random.uniform(30, 100, n), 2),  # Low range for Cash
        np.round(np.random.uniform(200, 1000, n), 2)  # High range for Bank Transfer
```

- **Class Table [20 rows]**

Maintain information about gym classes.

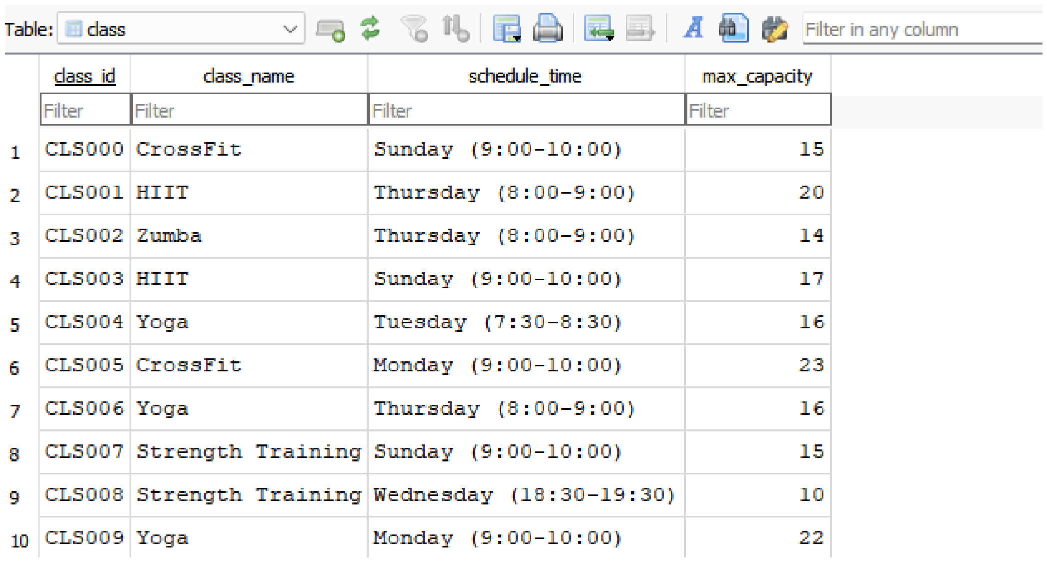


Table: class

	class_id	class_name	schedule_time	max_capacity
1	CLS000	CrossFit	Sunday (9:00-10:00)	15
2	CLS001	HIIT	Thursday (8:00-9:00)	20
3	CLS002	Zumba	Thursday (8:00-9:00)	14
4	CLS003	HIIT	Sunday (9:00-10:00)	17
5	CLS004	Yoga	Tuesday (7:30-8:30)	16
6	CLS005	CrossFit	Monday (9:00-10:00)	23
7	CLS006	Yoga	Thursday (8:00-9:00)	16
8	CLS007	Strength Training	Sunday (9:00-10:00)	15
9	CLS008	Strength Training	Wednesday (18:30-19:30)	10
10	CLS009	Yoga	Monday (9:00-10:00)	22

classId [Primary Key]: Unique identifier for each class.

class_name & schedule_time: Captures session details for members by making a reasonable scheduling.

Max Capacity: Constraint applied to maintain the data type of the number of participants per session as to prevent float numbers.

```
CREATE TABLE "class" (
    "class_id"  TEXT NOT NULL UNIQUE,
    "class_name"    TEXT NOT NULL,
    "schedule_time" TEXT,
    "max_capacity"  INTEGER CHECK("max_capacity" > 0),
    PRIMARY KEY("class_id")
);
```

- **Class_Registration Table [1000 rows]**

Manages class enrolments.

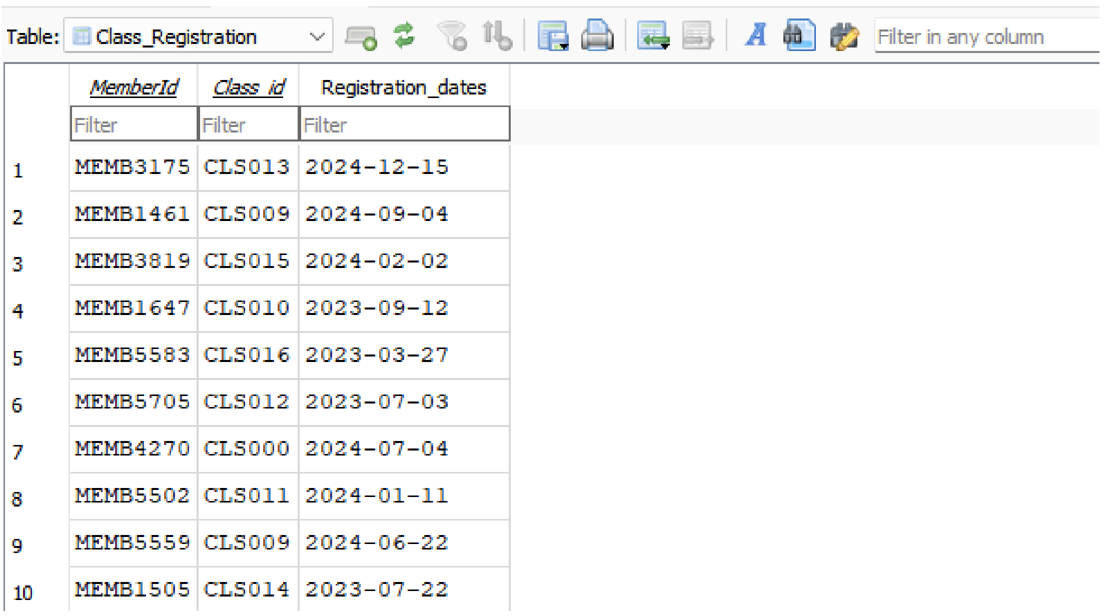


Table: Class_Registration

	MemberId	Class_id	Registration_dates
1	MEMB3175	CLS013	2024-12-15
2	MEMB1461	CLS009	2024-09-04
3	MEMB3819	CLS015	2024-02-02
4	MEMB1647	CLS010	2023-09-12
5	MEMB5583	CLS016	2023-03-27
6	MEMB5705	CLS012	2023-07-03
7	MEMB4270	CLS000	2024-07-04
8	MEMB5502	CLS011	2024-01-11
9	MEMB5559	CLS009	2024-06-22
10	MEMB1505	CLS014	2023-07-22

MemberId & ClassId (Composite Key): Ensures that a member cannot register for the same class multiple times enforcing logical integrity. Both columns together make a unique identity column.

```
CREATE TABLE "Class_Registration" (
    "MemberId"  TEXT,
    "Class_id"  TEXT,
    "Registration_dates"    TEXT DEFAULT 'Date',
    PRIMARY KEY("MemberId","Class_id"),
    FOREIGN KEY("Class_id") REFERENCES "class"("class_id"),
    FOREIGN KEY("MemberId") REFERENCES "Member"("MemberId")
);
```

Registration Date: Tracks when the member signed up for a class.

Representing Datatypes

The attributes in the database can be categorized based on their measurement scales:

Nominal Data (Categorical, No Order): Member names(text), email IDs(text), phone numbers(text), addresses(text), membership types(text), payment methods(text), class names(text), Member ID (text), Class ID (text), and Membership ID (text).
These values are unique identifiers or categories with no inherent ranking.

Ordinal Data (Categorical, Ordered): Access level(text) in membership types (e.g., Basic, Standard, Premium, VIP). These values have a meaningful order but no fixed difference between levels.

Interval Data: Dates (Join Date, Registration Date, Payment Date).
These values can be compared and differenced meaningfully, but they lack a true zero point (e.g., "zero date" has no meaning).

Ratio Data: Monthly and yearly membership fees(float), amount paid(float), max capacity of a class(integer). These values have a meaningful zero and allow for all mathematical operations like multiplication and division.

This classification ensures appropriate analysis and processing of data while maintaining **constraints and validations**.

Data privacy and ethics

The database involves personal details so it is a key focus to consider data privacy and ethics.

- The data has been created artificially to make it realistic yet anonymized confirming that no real individuals' information was used to prevent privacy concerns and maintain ethical integrity.

  ```
  #Creating fake 1000 names
  names = [fake.name() for _ in range(n)]
  ```

- A certain percentage null / nan values were given to some attributes to reflect real scenarios. However, constraints were enforced to prevent unrealistic data contradiction.

  ```
  postcode_data = np.where(np.random.rand(n) < 0.15, np.nan, postcode_data)

  join_date = np.where(np.random.rand(n) < 0.1, None, join_date)
  ```

- Duplicate values were included to consider the fact that in reality, different individuals might share similar details or mistakenly enter incorrect information.

```
    # Introduce duplicate names
    duplicates = int(0.003 * n)
    duplicate_indices = np.random.choice(n, duplicates, replace=False)
    for i in duplicate_indices:
        names[i] = np.random.choice(names)  # Replace with a random existing name

    # Introduce duplicate phone numbers
    duplicate_indices = np.random.choice(n, duplicates, replace=False)
    for i in duplicate_indices:
        phone[i] = np.random.choice([p for p in phone if p is not None])
```

- To avoid exposing real phone numbers, the middle digits were masked using ****, ensuring that the generated phone numbers remain anonymous while still resembling realistic formatting. Similarly, email addresses were generated with the domain **@gymmail.com** to prevent any unintended matching with real data while maintaining a realistic structure.

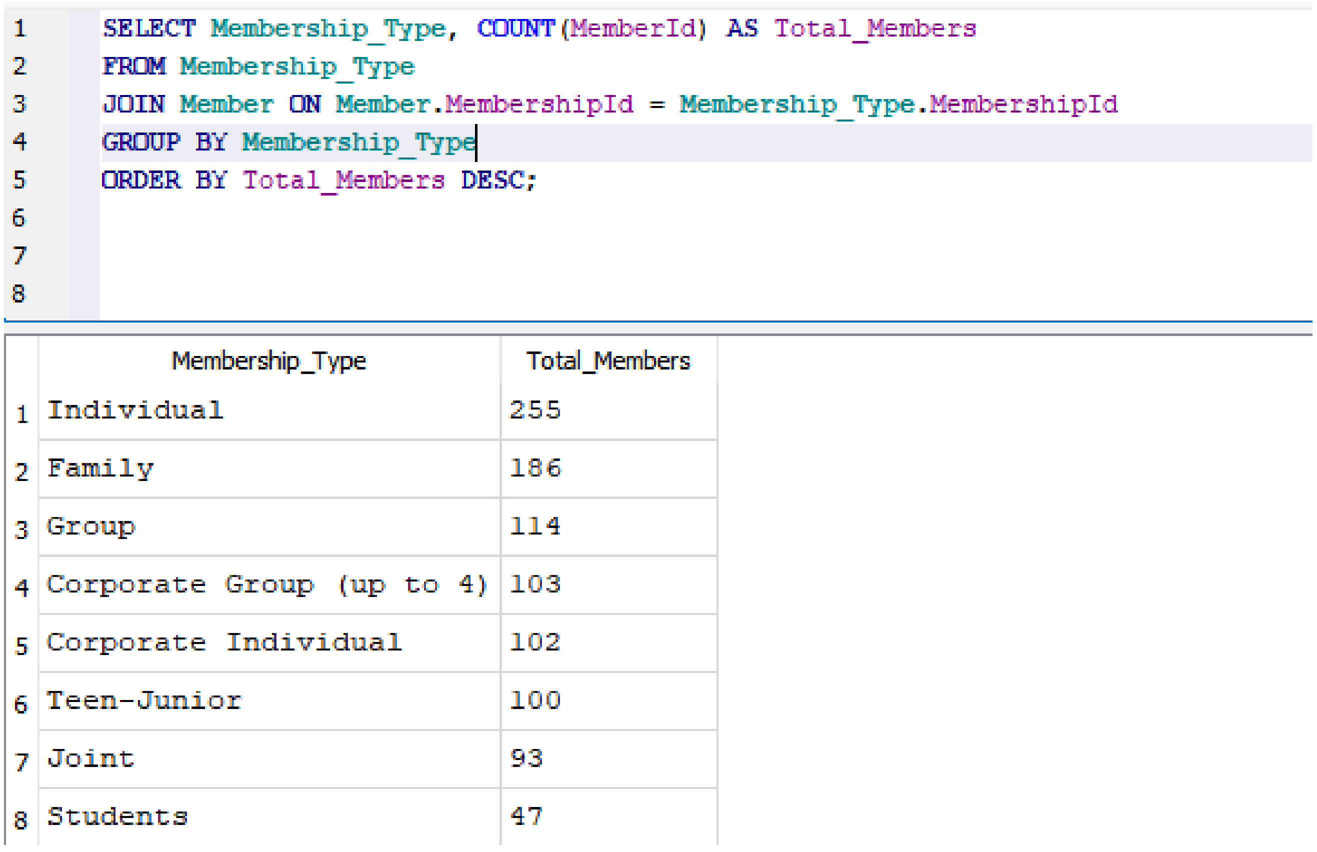
```
email = [f"{name.split()[0].lower()}{name.split()[1].lower() if len(name.split()) > 1 else ''}
        {np.random.randint(100, 999)}@gymmail.com" for name in names]

phone = [f"{np.random.randint(600, 999)}-****-{np.random.randint(100, 999)}" for _ in range(n)]
```

- Payment information does not include sensitive financial details like card numbers to follow ethical guidelines.
- The dataset structure was designed to avoid discrimination based on personal attributes without favouring any group.

Results of some queries

1. Most popular Membership_Type

Find the membership type with the highest number of members which helps to find which is the most popular membership among the customer so to implement more facilities.

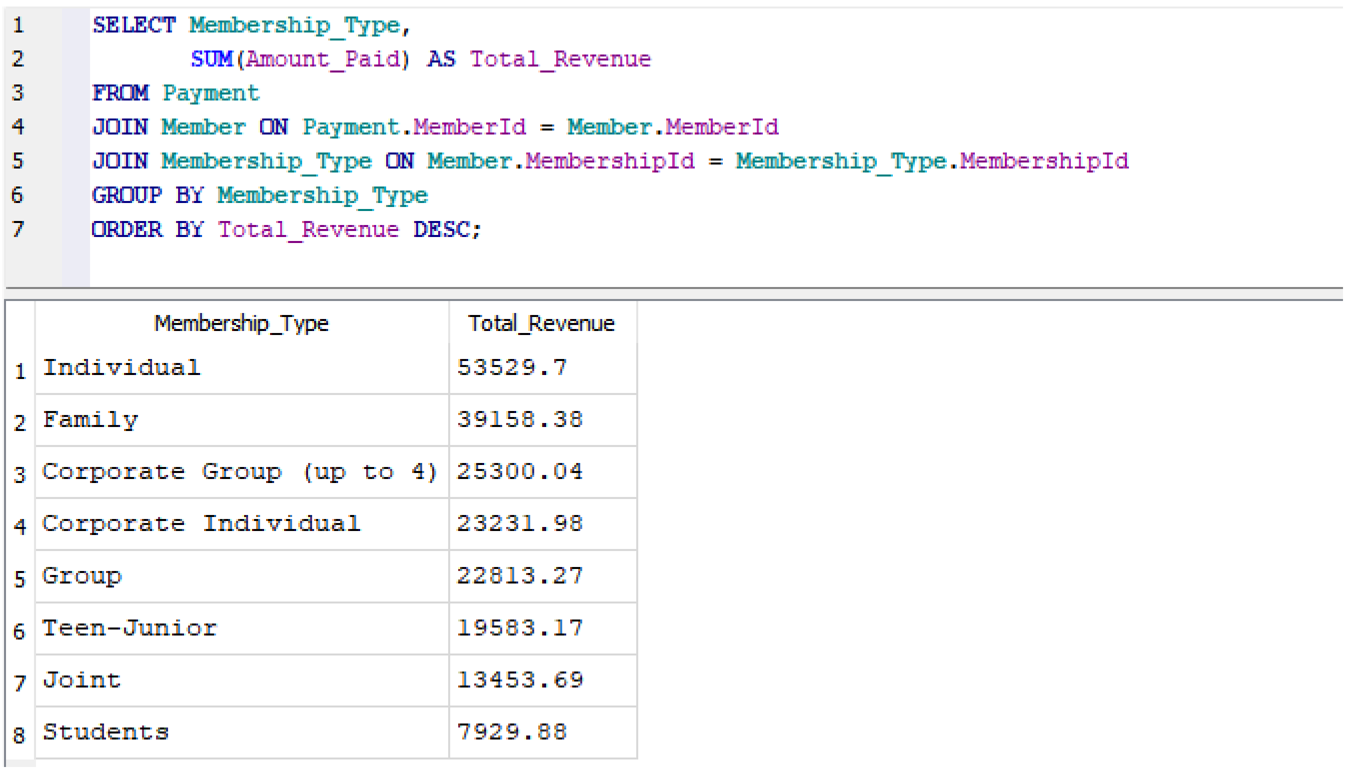


```sql
SELECT Membership_Type, COUNT(MemberId) AS Total_Members
FROM Membership_Type
JOIN Member ON Member.MembershipId = Membership_Type.MembershipId
GROUP BY Membership_Type
ORDER BY Total_Members DESC;
```

	Membership_Type	Total_Members
1	Individual	255
2	Family	186
3	Group	114
4	Corporate Group (up to 4)	103
5	Corporate Individual	102
6	Teen-Junior	100
7	Joint	93
8	Students	47

2. Total Revenue

This query helps to get the total income generated from membership payments which provides a financial overview that can be useful for business planning and growth strategies.

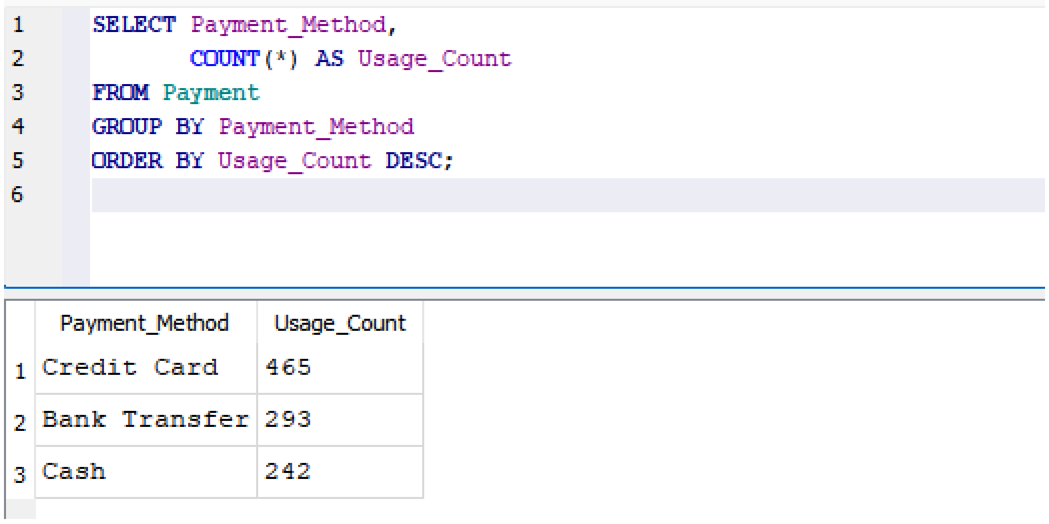


```sql
SELECT Membership_Type,
       SUM(Amount_Paid) AS Total_Revenue
FROM Payment
JOIN Member ON Payment.MemberId = Member.MemberId
JOIN Membership_Type ON Member.MembershipId = Membership_Type.MembershipId
GROUP BY Membership_Type
ORDER BY Total_Revenue DESC;
```

	Membership_Type	Total_Revenue
1	Individual	53529.7
2	Family	39158.38
3	Corporate Group (up to 4)	25300.04
4	Corporate Individual	23231.98
5	Group	22813.27
6	Teen-Junior	19583.17
7	Joint	13453.69
8	Students	7929.88

3. Most Frequently Used Payment Method

Identifying the most commonly used payment method to help in optimizing payment options for members and to give offer discounts or smooth transactions for that method.

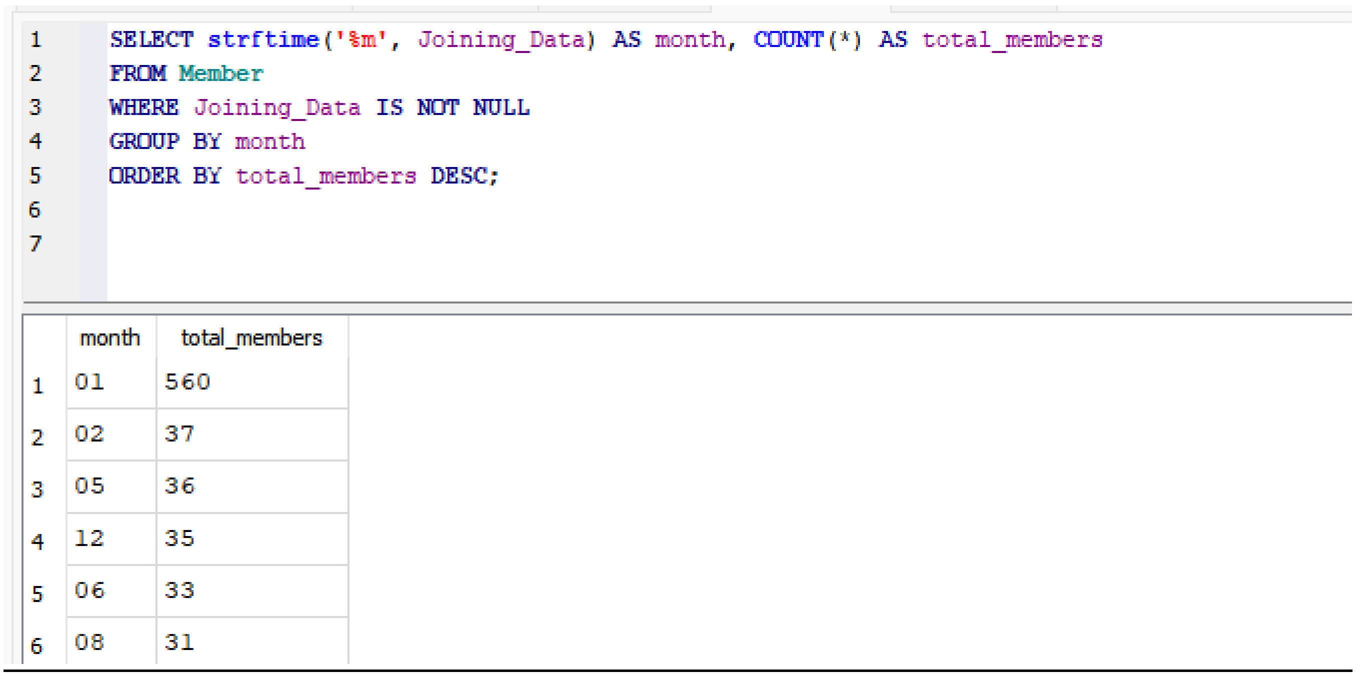


```sql
SELECT Payment_Method,
       COUNT(*) AS Usage_Count
FROM Payment
GROUP BY Payment_Method
ORDER BY Usage_Count DESC;
```

	Payment_Method	Usage_Count
1	Credit Card	465
2	Bank Transfer	293
3	Cash	242

4. Month with most gym registration

This query help to find out which month has the highest number of new registrations. As January has the highest joins, it aligns with New Year's fitness resolutions and seasonal discounts. Understanding peak membership periods allows for better resource allocation, such as increasing staff or offering promotions during slow months.



```
SELECT strftime('%m', Joining_Data) AS month, COUNT(*) AS total_members
FROM Member
WHERE Joining_Data IS NOT NULL
GROUP BY month
ORDER BY total_members DESC;
```

	month	total_members
1	01	560
2	02	37
3	05	36
4	12	35
5	06	33
6	08	31

Conclusion

This project involves all the different aspects of making a realistic database from generating the data using python to building the database in DB browser SQLite. Lastly, through insightful queries, meaningful information has been extracted from the database which demonstrates Its utility for gym management and analytics.