DATA MINING AND DISCOVERY SQL ASSIGNMENT

Student Name: Praneet Sivakumar
Student ID: 23095964
Topic: Smart Gym Database Management

Introduction:

The Smart Gym Database is a very well-organized system that bestows innumerable advantages on gym management. The **Members, Trainers, Workouts,** and **Sessions** standard tables are the four main parts that keep track of gym activities and operations.

The Members table keeps basic details regarding the gym-goers, such as personal info, subscription plans, and fitness progress. The Trainers table manages the instructors in the gym by documenting their **skills, experiences,** and **remuneration packages**. The Workouts table checks for various exercises, difficulty levels, equipment, and applicable calories burned. This makes the structuring of the workout plan easier. The Sessions table records training sessions, including details on **attendance, trainer assignments, session length,** and **feedback.**

With this structured database, gym managers can handle membership management, monitor progress, assign trainers and analyse workout trends with great efficiency. Active data-driven gyms will create an engagement for their customers, improve existing fitness programs, and ensure smooth running operations, all in all translating into a good experience for both members and trainers.

Data Generation:

Members Data:

- **Member_ID:** Unique identifier for each gym member, sequentially generated.
- **Name:** Full name of the member, generated using Faker.
- **Gender:** Randomly assigned as 'Male' or 'Female'.
- **Age:** Integer value between 18 and 60, representing the member's age.
- **Subscription_Type:** Randomly selected from 'Basic', 'Premium', and 'VIP'.
- **Weight:** Random float value between 50kg and 120kg.
- **Height:** Random float value between 150cm and 200cm.
- **Join_Date:** Randomly assigned within the last two years.

Code:

```
import numpy as np
import pandas as pd
from faker import Faker
import sqlite3
# Initialize Faker to generate realistic names and data
fake = Faker()

# Define the number of gym members to generate
```

```
n_members = 1000

# Create a dictionary to store member data
members_data = {

    'Member_ID': range(1, n_members + 1),  # Unique ID for each member
    'Name': [fake.name() for _ in range(n_members)],  # Randomly generated names
    'Gender': np.random.choice(['Male', 'Female'], n_members),  # Randomly assigned gender
    'Age': np.random.randint(18, 60, n_members),  # Age between 18 and 60
    'Subscription_Type': np.random.choice(['Basic', 'Premium', 'VIP'], n_members),  # Membership type

    'Weight': np.random.uniform(50, 120, n_members),  # Random weight in kg
    'Height': np.random.uniform(150, 200, n_members),  # Random height in cm
    'Join_Date': pd.to_datetime(
        np.random.choice(pd.date_range('2022-01-01', '2024-12-31', periods=n_members).date, n_members)
    )  # Random join date within the last two years
}


# Convert the dictionary into a Pandas DataFrame
members_df = pd.DataFrame(members_data)

# Connect to (or create) the SQLite database
conn = sqlite3.connect('smart_gym.db')

# Save the DataFrame as a table in the database, replacing it if it already exists
members_df.to_sql('members', conn, if_exists='replace', index=False)

# Commit changes and close the database connection
conn.commit()
conn.close()
```

Output:

	Member_ID	Name	Gender	Age	Subscription_Type	Weight	Height	Join_Date
0	1	Tammy Sosa	Male	23	Premium	102.865860	157.069038	2023-05-09
1	2	Kristen Koch	Female	43	Basic	104.765685	174.758138	2022-09-27
2	3	Francisco Mack	Female	55	VIP	50.115568	157.314701	2024-10-08
3	4	Adrienne Smith	Male	28	VIP	54.957042	186.281893	2023-12-23
4	5	Maria King	Male	33	Premium	112.316977	195.313072	2022-05-13
...	...	...	...	...	...	...	...	...
995	996	Christie Shaw	Male	22	VIP	94.897849	199.008669	2024-09-04
996	997	Kathryn Martin	Female	59	Premium	87.765372	155.657081	2024-09-13
997	998	Danielle Hart	Female	59	VIP	110.043956	161.355266	2023-03-13
998	999	Samuel Gibson	Female	56	VIP	70.879842	189.214797	2024-02-24
999	1000	Maria Tyler	Female	44	Premium	77.523964	150.674610	2024-12-11

1000 rows × 8 columns

Trainers Data:

- **Trainer_ID:** Unique identifier for each trainer.
- **Name:** Full name generated using Faker.
- **Specialization:** Randomly chosen from 'Yoga', 'Cardio', 'Weight Lifting', 'Pilates'.
- **Experience_Years:** Random integer between 5 and 20.
- **Salary:** Random float between 30,000 and 70,000.
- **Email:** Randomly generated using Faker.
- **Phone:** Randomly generated phone number.

Code:

```
    # Define the total number of trainers to generate
total_trainers = 10
# Create a dictionary to store trainer data
trainer_data = {
   'Trainer_ID': range(1, total_trainers + 1),  # Unique ID for each trainer
   'Name': [fake.name() for _ in range(total_trainers)],  # Randomly generated names
   'Specialization': np.random.choice(['Yoga', 'Weight Lifting', 'Cardio', 'Pilates'], total_trainers),  # Trainer
expertise
   'Experience_Years': np.random.randint(5, 20, total_trainers),  # Experience level (between 5 to 20 years)
   'Salary': np.random.uniform(30000, 70000, total_trainers),  # Randomized salary range (in USD)
   'Email': [fake.email() for _ in range(total_trainers)],  # Fake email addresses for contact details
   'Phone': [fake.phone_number() for _ in range(total_trainers)]  # Random phone numbers for trainers
}
# Convert the dictionary into a Pandas DataFrame for structured handling
trainers_df = pd.DataFrame(trainer_data)
# Connect to (or create) the SQLite database
conn = sqlite3.connect('smart_gym.db')
# Store the trainers' data in the database, replacing it if the table already exists
trainers_df.to_sql('trainers', conn, if_exists='replace', index=False)
# Commit changes and close the database connection
conn.commit()
conn.close()
```

Output:

	Trainer_ID	Name	Specialization	Experience_Years	Salary	Email	Phone
0	1	Beth Bishop	Pilates	6	51362.581705	bbailey@example.com	(505)892-2242x8824
1	2	Craig Anderson	Yoga	11	61765.929499	dramos@example.net	471-634-3178x357
2	3	Jonathan Vega	Pilates	12	52334.464976	xbennett@example.com	969.988.1003x806
3	4	Jessica Sharp	Pilates	6	56137.246716	kathryngallagher@example.org	870-311-8592x01293
4	5	Linda Roberts	Pilates	17	38553.593330	ecampbell@example.com	+1-430-748-7664x98587
5	6	Grace Harrison	Weight Lifting	13	64268.893667	mclark@example.net	950.606.0339x3629
6	7	Dustin Crawford	Cardio	6	46266.363053	kristinhopkins@example.org	503-250-5935
7	8	Melody Bowers	Yoga	5	57867.733697	ellisgloria@example.net	+1-931-999-5510x643
8	9	Amy Rice	Pilates	14	54549.775890	underwoodlinda@example.net	001-246-773-3931x60851
9	10	Raymond Snyder	Weight Lifting	7	53546.579037	ryan46@example.com	+1-747-418-4376x29851

Workouts Data:

- **Workout_ID:** Unique identifier for each workout.
- **Workout_Type:** Type of workout (e.g., 'Cardio', 'Yoga').
- **Difficulty_Level:** Ordinal variable (Beginner, Intermediate, Advanced).
- **Duration:** Integer value between 30 and 90 minutes.
- **Calories_Burned:** Estimated calorie expenditure.
- **Equipment_Needed:** Specifies if equipment is needed.
- **Popularity:** Scale from 1 to 10.

Code:

```
        # Define the number of trainers to generate
total_trainers = 10

# Create a dictionary to store trainer details
trainer_data = {

    'Trainer_ID': range(1, total_trainers + 1),  # Unique identifier for each trainer

    'Name': [fake.name() for _ in range(total_trainers)],  # Randomly generated trainer names

    'Specialization': np.random.choice(['Yoga', 'Weight Lifting', 'Cardio', 'Pilates'], total_trainers),

# Assigned area of expertise
    'Experience_Years': np.random.randint(5, 20, total_trainers),  # Years of experience in the fitness industry (between 5 to 20 years)

    'Salary': np.random.uniform(30000, 70000, total_trainers),  # Random salary range (in USD)

    'Email': [fake.email() for _ in range(total_trainers)],  # Fake email addresses for trainers

    'Phone': [fake.phone_number() for _ in range(total_trainers)]  # Randomly generated phone numbers
}

# Convert the dictionary into a Pandas DataFrame for easy handling and visualization
```

```
trainers_df = pd.DataFrame(trainer_data)

# Connect to the SQLite database (creates the database if it doesn’t exist)
conn = sqlite3.connect('smart_gym.db')

# Store the trainers' data in the SQLite database under the 'trainers' table
# If the table already exists, it will be replaced with the new data
trainers_df.to_sql('trainers', conn, if_exists='replace', index=False)

# Commit the changes to save data into the database
conn.commit()

# Close the database connection to prevent data loss or corruption
conn.close()
```

Output:

	Workout_ID	Workout_Type	Difficulty_Level	Duration	Calories_Burned	Equipment_Needed	Popularity
0	1	Pilates	Advanced	41	252	Treadmill	2
1	2	Cardio	Advanced	41	490	Dumbbells	8
2	3	Weight Lifting	Intermediate	45	349	Dumbbells	7
3	4	Weight Lifting	Advanced	33	274	Mat	2
4	5	Yoga	Intermediate	52	233	Barbell	3
5	6	Weight Lifting	Beginner	59	234	Barbell	2
6	7	Yoga	Advanced	55	448	Treadmill	8
7	8	Cardio	Advanced	31	229	Barbell	8
8	9	Cardio	Beginner	74	384	Mat	7
9	10	Weight Lifting	Intermediate	67	404	Mat	2

Sessions Data:

- **Session_ID:** Unique identifier.
- **Member_ID:** Foreign key referencing Members table.
- **Trainer_ID:** Foreign key referencing Trainers table.
- **Workout_ID:** Foreign key referencing Workouts table.
- **Session_Date:** Timestamp.
- **Calories_Burned:** Numeric value.
- **Feedback:** Categorical ('Good', 'Average', 'Poor').
- **Duration:** Workout session duration.

Code:

```
        # Define the total number of workout sessions to generate
total_sessions = 1000

# Create a dictionary to store session details
sessions_data = {
   'Session_ID': range(1, total_sessions + 1),  # Unique identifier for each session
   'Member_ID': np.random.choice(range(1, n_members + 1), total_sessions),  # Assign a random member to the session
   'Trainer_ID': np.random.choice(range(1, total_trainers + 1), total_sessions),  # Assign a random trainer for the session

   'Workout_ID': np.random.choice(range(1, total_workouts + 1), total_sessions),  # Assign a random workout type

   'Session_Date': pd.to_datetime(
      np.random.choice(pd.date_range('2022-01-01', '2024-12-31', periods=total_sessions).date, total_sessions)

   ),
 # Randomly select a session date within the last two years

   'Calories_Burned': np.random.uniform(200, 500, total_sessions),  # Random calories burned per session
   'Feedback': np.random.choice(['Good', 'Average', 'Poor'], total_sessions),  # Random feedback from the session
   'Duration': np.random.randint(30, 90, total_sessions)  # Random session duration between 30 to 90 minutes
}

# Convert the dictionary into a Pandas DataFrame for structured handling
sessions_df = pd.DataFrame(sessions_data)

# Connect to the SQLite database (creates it if it doesn't exist)
conn = sqlite3.connect('smart_gym.db')

# Store the session data in the SQLite database under the 'sessions' table

# If the table already exists, it will be replaced with the new data
sessions_df.to_sql('sessions', conn, if_exists='replace', index=False)


# Commit the changes to ensure data is saved in the database
conn.commit()

# Close the database connection to prevent data corruption
conn.close()
```

Output:

	Session_ID	Member_ID	Trainer_ID	Workout_ID	Session_Date	Calories_Burned	Feedback	Duration
0	1	24	1	1	2023-01-02	366.286970	Average	86
1	2	216	4	9	2023-09-28	231.307878	Poor	42
2	3	262	5	3	2022-10-09	447.464772	Good	86
3	4	121	10	1	2023-10-29	223.067575	Good	61
4	5	28	9	2	2022-09-25	341.862967	Average	52
...	...	...	...	...	...	...	...	...
995	996	665	9	9	2022-04-09	426.511518	Good	56
996	997	441	6	7	2023-02-26	370.234318	Average	86
997	998	453	10	7	2022-12-02	416.134752	Good	88
998	999	207	9	4	2023-10-16	449.841536	Good	69
999	1000	340	8	9	2022-06-04	269.893000	Poor	82

1000 rows × 8 columns

Data Schema:

The Smart Gym database is designed to efficiently store and manage gym-related information, including members, trainers, workout plans, and training sessions. Each table is structured with specific attributes to maintain a well-organized and easily accessible dataset.

Tables (5)

Name	Type	Schema
members		CREATE TABLE "members" ("Member_ID" INTEGER, "Name" TEXT, "Gender" TEXT, "Age" INTEGER, "Subscription_Type" TEXT, "Weight" REAL, "Height" REAL, "Join_Date" TIMESTAMP)
Member_ID	INTEGER	"Member_ID" INTEGER
Name	TEXT	"Name" TEXT
Gender	TEXT	"Gender" TEXT
Age	INTEGER	"Age" INTEGER
Subscription_Type	TEXT	"Subscription_Type" TEXT
Weight	REAL	"Weight" REAL
Height	REAL	"Height" REAL
Join_Date	TIMESTAMP	"Join_Date" TIMESTAMP
sessions		CREATE TABLE "sessions" ("Session_ID" INTEGER, "Member_ID" INTEGER, "Trainer_ID" INTEGER, "Workout_ID" INTEGER, "Session_Date" TIMESTAMP, "Calories_Burned" REAL, "Feedback" TEXT, "Duration" INTEGER)
Session_ID	INTEGER	"Session_ID" INTEGER
Member_ID	INTEGER	"Member_ID" INTEGER
Trainer_ID	INTEGER	"Trainer_ID" INTEGER
Workout_ID	INTEGER	"Workout_ID" INTEGER
Session_Date	TIMESTAMP	"Session_Date" TIMESTAMP
Calories_Burned	REAL	"Calories_Burned" REAL
Feedback	TEXT	"Feedback" TEXT
Duration	INTEGER	"Duration" INTEGER

Members Table:

- **Purpose:** Stores information about gym members.
- **Primary Key: Member_ID**
- **Columns:**
  - **Member_ID:** Unique identifier for each gym member.
  - **Name:** Full name of the member.
  - **Gender:** Gender of the member (Male/Female).
  - **Age:** Age of the member.
  - **Subscription_Type:** Type of membership (Basic/Premium/VIP).
  - **Weight:** Current weight of the member (kg).
  - **Height:** Height of the member (cm).
  - **Join_Date:** Date the member joined the gym.

Trainers Table:

- **Purpose:** Stores details about gym trainers.
- **Primary Key:** Trainer_ID
- **Columns:**
  - **Trainer_ID:** Unique identifier for each trainer.
  - **Name:** Trainer's full name.
  - **Specialization:** Trainer's area of expertise (Yoga, Cardio, etc.).
  - **Experience_Years:** Number of years of experience.
  - **Salary:** Trainer's salary.
  - **Email:** Contact email address.
  - **Phone:** Contact phone number.

Workouts Table:

- **Purpose:** Stores information about different workout routines.
- **Primary K**ey: Workout_ID
- **Columns:**
  - **Workout_ID:** Unique identifier for each workout.
  - **Workout_Type:** Type of workout (Cardio, Weight Lifting, Yoga, etc.).
  - **Difficulty_Level:** Level of difficulty (Beginner, Intermediate, Advanced).
  - **Duration:** Duration of the workout in minutes.
  - **Calories_Burned:** Estimated calories burned per session.
  - **Equipment_Needed:** Equipment required for the workout.
  - **Popularity:** Popularity rating (1-10).

Sessions Table:

- **Purpose:** Records details of workout sessions conducted at the gym.
- **Primary Key:** Session_ID
- **Columns:**
  - **Session_ID:** Unique identifier for each session.
  - **Member_ID:** Reference to the Members table.
  - **Trainer_ID:** Reference to the Trainers table.
  - **Workout_ID:** Reference to the Workouts table.
  - **Session_Date:** Date and time of the session.
  - **Calories_Burned:** Calories burned in the session.
  - **Feedback:** Member's feedback (Good, Average, Poor).
  - **Duration:** Duration of the session in minutes.

Each table is structured to store essential gym-related information while maintaining relationships between members, trainers, and workout sessions. The use of primary and foreign keys ensures data integrity and enables efficient tracking of fitness activities.

QUERIES EXAMPLES:

1. **Retrieve Gym Sessions on a Specific Date:**

   SELECT Session_ID, Member_ID, Trainer_ID, Workout_ID, Calories_Burned
   FROM sessions **WHERE** DATE(Session_Date) = '2024-03-10'

Explanation:

This query retrieves session details for workouts conducted on March 10, 2024. It selects Session_ID, Member_ID, Trainer_ID, Workout_ID, and Calories_Burned, filtering only sessions that took place on the specified date.

	Session_ID	Member_ID	Trainer_ID	Workout_ID	Calories_Burned
1	448	970	8	3	479.511859250753
2	691	856	4	10	438.587112252396
3	968	981	1	3	345.424949518663

```
Execution finished without errors.
Result: 3 rows returned in 9ms
At line 1:
SELECT Session_ID, Member_ID, Trainer_ID, Workout_ID, Calories_Burned
FROM sessions
WHERE DATE(Session_Date) = '2024-03-10';
```

2. **Retrieve Members with a Specific Email Domain:**

   SELECT Member_ID, Name, Email
   FROM members **WHERE** Email LIKE '%@gmail.com'

Explanation:

This query retrieves Member_ID, Name, and Email for all members whose email addresses end with @gmail.com, using the LIKE operator to match the domain.

	Member_ID	Name	Email
1	3	Sarah Spencer	pattersoncynthia@gmail.com
2	6	Christopher Reilly	kmcdowell@gmail.com
3	7	Brent Bennett	vanessasummers@gmail.com
4	9	Andrew Duffy	haasjason@gmail.com
5	11	Daniel Simpson MD	andrew85@gmail.com
6	13	Amy Thompson	hpitts@gmail.com
7	17	John Ray	erikabenton@gmail.com

```
Execution finished without errors.
Result: 339 rows returned in 12ms
At line 1:
SELECT Member_ID, Name, Email
FROM members
WHERE Email LIKE '%@gmail.com';
```

3. **Count Workouts by Difficulty Level**

```
SELECT Difficulty_Level, COUNT(*) AS Workout_Count
FROM workouts GROUP BY Difficulty_Level
ORDER BY Workout_Count DESC
```

Explanation:

This SQL query counts the number of workouts available for each difficulty level (Beginner, Intermediate, Advanced). It groups the data by Difficulty_Level and orders the results in descending order based on the count of workouts.

	Difficulty_Level	Workout_Count
1	Advanced	4
2	Intermediate	3
3	Beginner	3

```
Execution finished without errors.
Result: 3 rows returned in 8ms
At line 1:
SELECT Difficulty_Level, COUNT(*) AS Workout_Count
FROM workouts
GROUP BY Difficulty_Level
ORDER BY Workout_Count DESC;
```

4. **Retrieve Sessions Conducted Within a Specific Date Range**

```
SELECT Session_ID, Member_ID, Trainer_ID, Session_Date
FROM sessions
WHERE Session_Date BETWEEN '2023-01-01' AND '2023-01-31'
```

Explanation:

This SQL query selects Session_ID, Member_ID, Trainer_ID, and Session_Date from the sessions table for sessions held between January 1, 2023, and January 31, 2023. It retrieves data where the Session_Date falls within the specified range using the BETWEEN operator.

	Session_ID	Member_ID	Trainer_ID	Session_Date
1	8	447	8	2023-01-13 00:00:00
2	106	534	8	2023-01-07 00:00:00
3	186	987	1	2023-01-19 00:00:00
4	270	674	7	2023-01-03 00:00:00
5	297	335	4	2023-01-29 00:00:00
6	311	856	4	2023-01-16 00:00:00
7	331	358	10	2023-01-27 00:00:00

```
Execution finished without errors.
Result: 21 rows returned in 12ms
At line 1:
SELECT Session_ID, Member_ID, Trainer_ID, Session_Date
FROM sessions
WHERE Session_Date BETWEEN '2023-01-01' AND '2023-01-31';
```

5. **Retrieve Member Names and Subscription Details for Premium Members**

```
SELECT Name, Age, Subscription_Type
FROM members WHERE Subscription_Type = 'Premium'
```

Explanation:

This SQL query retrieves the names, ages, and subscription types of members who are subscribed to the Premium plan. The WHERE clause filters the records to include only those with Subscription_Type = 'Premium'

	Name	Age	Subscription_Type
1	Sarah Spencer	36	Premium
2	Christopher Reilly	20	Premium
3	Brent Bennett	42	Premium
4	Daniel Simpson MD	33	Premium
5	Amy Thompson	52	Premium
6	Cheryl Perry	51	Premium
7	Michael Davis	39	Premium

```
Execution finished without errors.
Result: 342 rows returned in 11ms
At line 1:
SELECT Name, Age, Subscription_Type
FROM members
WHERE Subscription_Type = 'Premium';
```

6. **Retrieve Top 5 Trainers with the Highest Salaries**

   SELECT Name, Specialization, Salary **FROM** trainers **ORDER BY** Salary **DESC LIMIT 5**

Explanation:

This SQL query retrieves the top 5 trainers with the highest salaries by selecting Name, Specialization, and Salary from the trainers table. The results are sorted in descending order by salary, and the LIMIT 5 clause ensures that only the top 5 results are returned.

	Name	Specialization	Salary
1	Jason Gregory	Weight Lifting	71961.7667979874
2	Emma Snyder	Cardio	64955.1118989132
3	Kim Taylor	Cardio	63912.3470140716
4	Robert Johnson	Pilates	61334.3708968801
5	Tyler Perez	Cardio	50401.2102170574

```
Execution finished without errors.
Result: 5 rows returned in 12ms
At line 1:
SELECT Name, Specialization, Salary
FROM trainers
ORDER BY Salary DESC
LIMIT 5;
```

7. **Retrieve Sessions with More Than 400 Calories Burned**

   SELECT Session_ID, Member_ID, Trainer_ID, Calories_Burned

   FROM sessions **WHERE** Calories_Burned > 400 **ORDER BY** Calories_Burned **DESC;**

Explanation:

This SQL query retrieves session details where members burned more than 400 calories in a session. The WHERE clause filters the records based on the Calories_Burned column, and the results are sorted in descending order to show the highest calorie-burning sessions first.

	Session_ID	Member_ID	Trainer_ID	Calories_Burned
1	25	882	4	499.382626064433
2	344	100	1	498.969273736142
3	366	403	7	498.90988135717
4	835	427	9	498.033304380879
5	536	413	4	497.721441903386
6	129	967	1	497.544098942247
7	298	758	1	497.540616979103

```
Execution finished without errors.
Result: 297 rows returned in 11ms
At line 1:
SELECT Session_ID, Member_ID, Trainer_ID, Calories_Burned
FROM sessions
WHERE Calories_Burned > 400
ORDER BY Calories_Burned DESC;
```

Each of these queries helps analyze different aspects of the Smart Gym Database, providing insights into member activities, trainer performance, and workout statistics.

Conclusion :

The Smart Gym Database is a structured and efficient system that streamlines gym management by tracking members, trainers, workouts, and sessions. With well-defined relationships and SQL queries, it enables gym administrators to monitor performance, optimize operations, and enhance member experiences. This data-driven approach ensures better decision-making, improved efficiency, and a more personalized fitness journey for members.