SPD Sports Club Database

Darius Parker, Pierina M Logrono & Samantha Reyes

2023

**SPD Sports Club**

**Summary/Goal:**

SPD Sports Club is the coolest gym in town! Each facility offers a wide variety of fitness group classes and there is one facility in each borough. All members have full access to a variety of fitness classes such as Cardio, Strength, Yoga, Pilates, and Cross-fit. These classes are offered throughout the week at different times of day and members have access to the schedule in the user app. SPD Sports Club is new to the market, and it’s looking to expand and open more facilities throughout NYC. With that, investors would like to analyze which locations have the most traffic and which classes are most popular. Since SPD Sports Club has the best and most knowledgeable Group Fitness instructors in the industry, they want to make sure they are making the right decisions with their next investment.

There are two different types of memberships offered in SPD Sports Club:

**Standard** - $50 per month. Standard members are allowed access to the gym. They do not get access to perks such as online training and meal plans.

**Premium -** $100 per month. Includes special perks such as access to online training and meal plans.

To track member activity and keep the facilities running smooth, members can check-in when they present to the gym. The major benefit of check-in is that gym managers can track the popularity of different facilities and track information such as the demographics of members at each facility.

**Stakeholders:**

**-**Owners and decision makers

-Members that like to track their activity and take advantage of the perks offered within their membership

**Business Rules:**

* There are five facilities, one facility for each borough of New York.
* Each facility has three classes per day Monday through Saturday. The class start times are 7:00am, 12:00pm and 6:00pm.
* There are five categories of class types: cardio, strength, yoga, pilates, and crossfit. For each specialty, there are seven different trainers on staff. One for each facility, and two additional trainers for each category to cover for vacation, sick days etc.
* Members can sign up for a standard plan for $50, or a premium plan for $100.
* Members check in at the front desk when they enter the gym. The trainers keep count of attendees and report the number of members who attend the classes.

**Glossary:**

* **Gym member -** Members at the gym.
* **Gym trainer** - The instructors leading the classes offered.
* **Check-in** – In order to access the gym, members must first check in to confirm their membership is active.
* **Classes** – The different exercise sessions offered by the gym at a designated time.
* **Facilities**- The gyms at different locations throughout the city
* **Gym Plans** – Membership types offered to clients at different prices with different benefits/access.

**Entities and Attributes:**

|  |  |  |
| --- | --- | --- |
|  | **Attributes** | **Relationships** |
| **gymclasses** | **gymclass\_id -** INT IDENTITY(1,1) PRIMARY KEY **gymclass\_category -** VARCHAR, NOT NULL **gymclass\_name -** VARCHAR, NOT NULL **gymclass\_trainer\_id** - INT, NOT NULL **gymclass\_facility\_id** -INT, NOT NULL **gymclass\_day\_of\_week -** VARCHAR, NOT NULL **gymclass\_start\_time -** TIME, NOT NULL **gymclass\_end\_time** - TIME, NOT NULL | **gymclass\_facility\_id** - Foreign key, references gymfacilities(gymfacility\_id) **gymclass\_trainer\_id** - Foreign key, references gymtrainers(gymtrainer\_id) |
| **gymclassdates** | **gymclass\_date\_id** - INT IDENTITY(1,1), PRIMARY KEY **gymclass\_date -** DATE, NOT NULL **gymclassdates\_gymclass\_id -** INT, NOT NULL **num\_atendees\_on\_date -** INT, NOT NULL | **gymclassdates\_gymclass\_id** - Foreign key, references gymclasses(gymclass\_id) **CONSTRAINT chk\_future\_date** - A check constraint to ensure that future dates are not added. |
| **gymfacilities** | **gymfacility\_id -** INT, IDENTITY(1,1), PRIMARY KEY **gymfacility\_zipcode -** VARCHAR, NOT NULL **gymfacility\_borough** VARCHAR, NOT NULL |  |
| **gymmembers** | **gymmember\_id -** INT, IDENTITY(1,1) PRIMARY KEY **gymmember\_firstname -** VARCHAR, NOT NULL **gymmember\_lastname -** VARCHAR, NOT NULL **gymmember\_age -** INT, NOT NULL **gymmember\_gender** - VARCHAR **gymmember\_email -** VARCHAR, UNIQUE **gymmember\_plan\_id -** INT, NOT NULL **gymmember\_phonenumber -V**ARCHAR(100) NOT NULL **gymmember\_zipcode -** VARCHAR, NOT NULL | **gymmember\_plan\_id** - Foreign key, references gymplans(gymplan\_id) |
| **gymplans** | **gymplan\_id** - INT, IDENTITY(1,1), PRIMARY KEY **gymplan\_price -** MONEY, NOT NULL **gymplan\_type -** VARCHAR, NOT NULL |  |
| **gymtrainers** | **gymtrainer\_id** - INT IDENTITY(1,1) PRIMARY KEY **gymtrainer\_firstname** - VARCHAR, NOT NULL **gymtrainer\_lastname** VARCHAR, NOT NULL **gymtrainer\_email** - VARCHAR, UNIQUE, NOT NULL **gymtrainer\_specialty** - VARCHAR, NOT NULL |  |
| **checkins** | **checkin\_id** - INT, IDENTITY(1,1), PRIMARY KEY **checkin\_date** - DATE, NOT NULL **checkin\_time -** TIME, NOT NULL **checkin\_gymmember\_id -** INT, NOT NULL **checkin\_facility\_id** - INT, NOT NULL | **checkin\_gymmember\_id** - Foreign key, references gymmembers(gymmember\_id) **checkin\_facility\_id** - Foreign key, references gymfacilities(gymfacility\_id) **CONSTRAINT chk\_checkin\_date\_past -** A check constraint to ensure that future dates are not added. |

**Models:**

**A diagram of a diagram

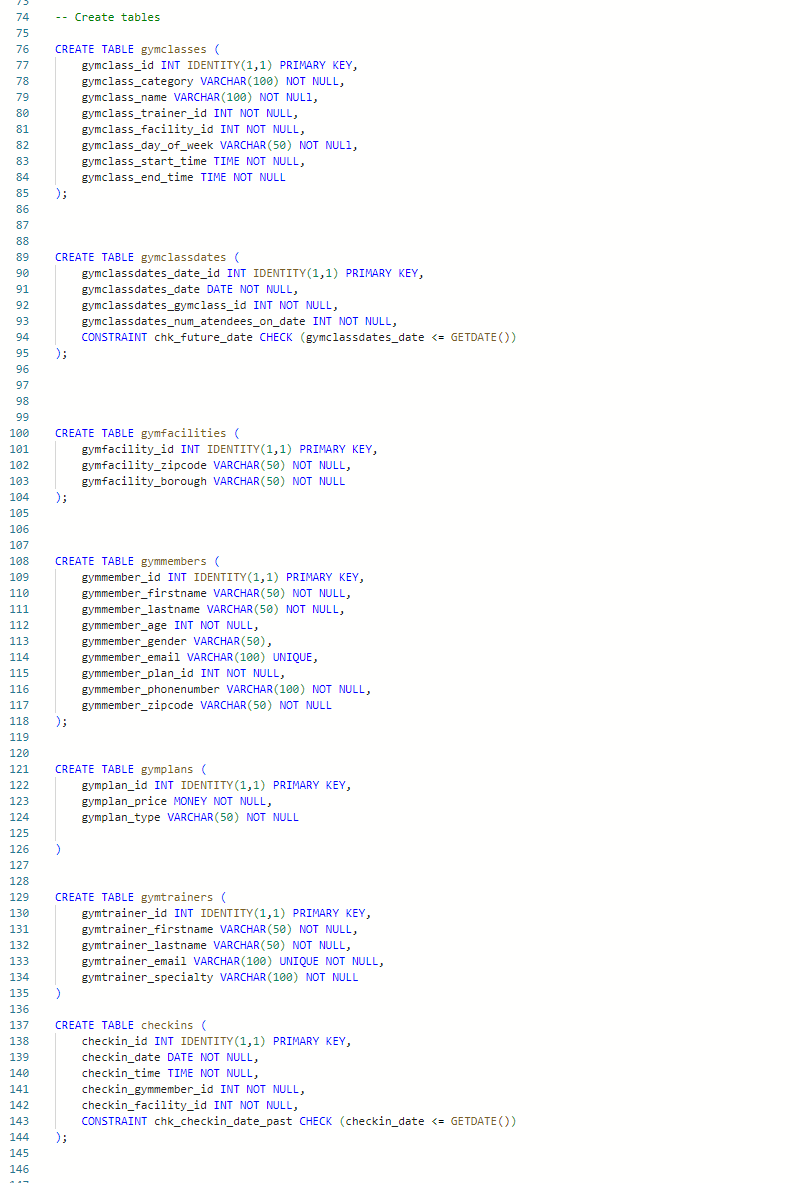
Description automatically generated**

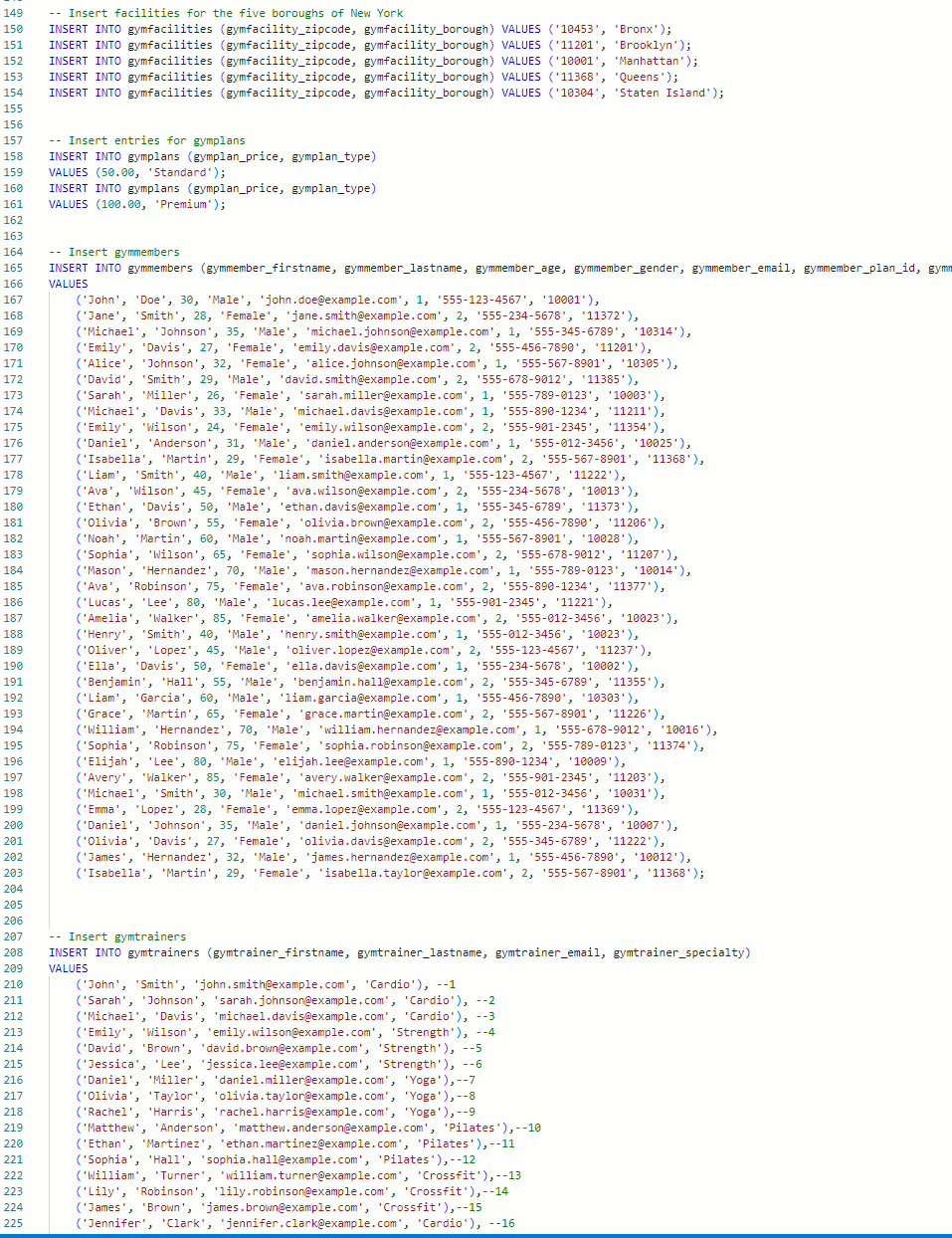
**A diagram of a computer

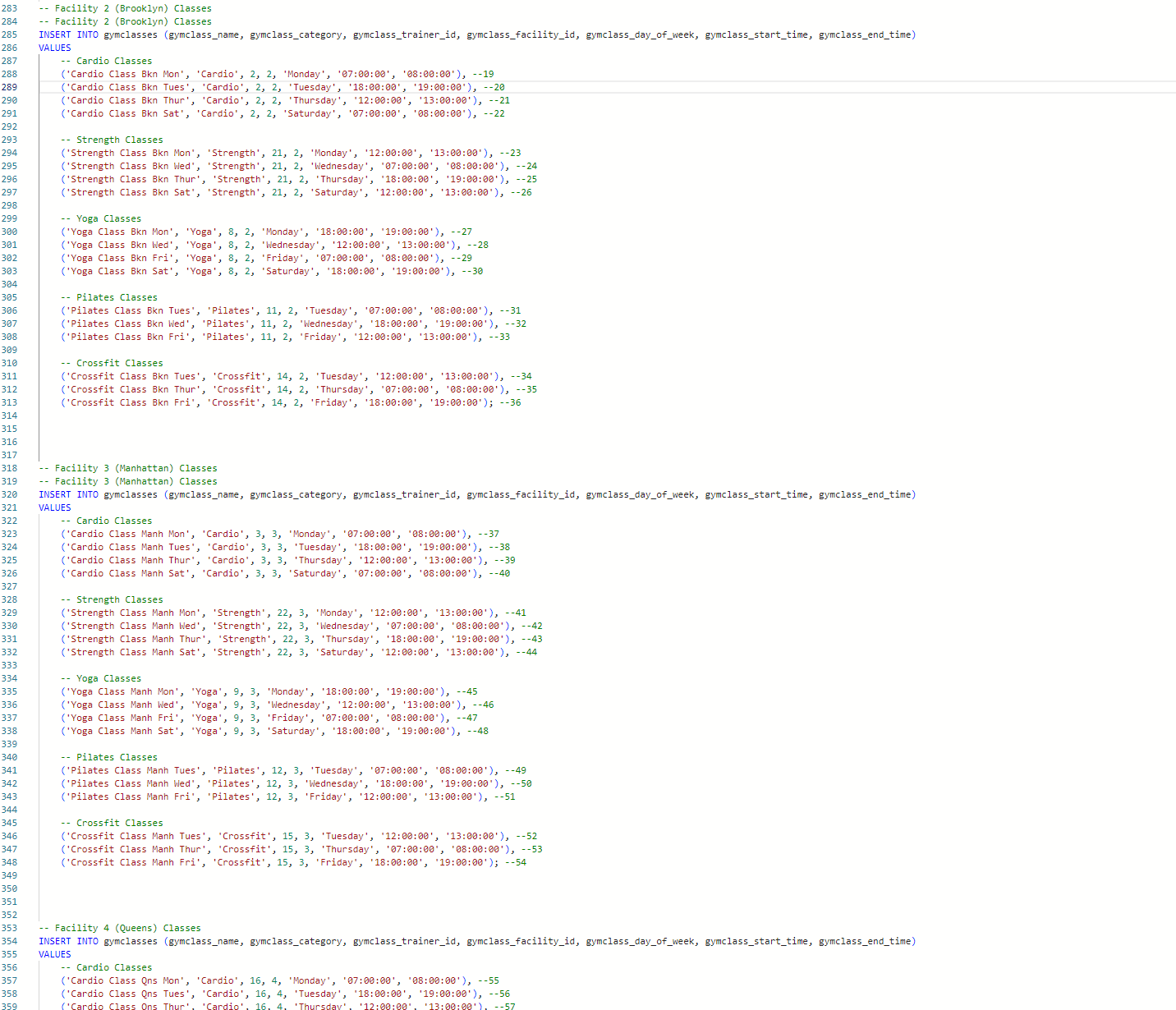
Description automatically generated**

**SQL Code:**

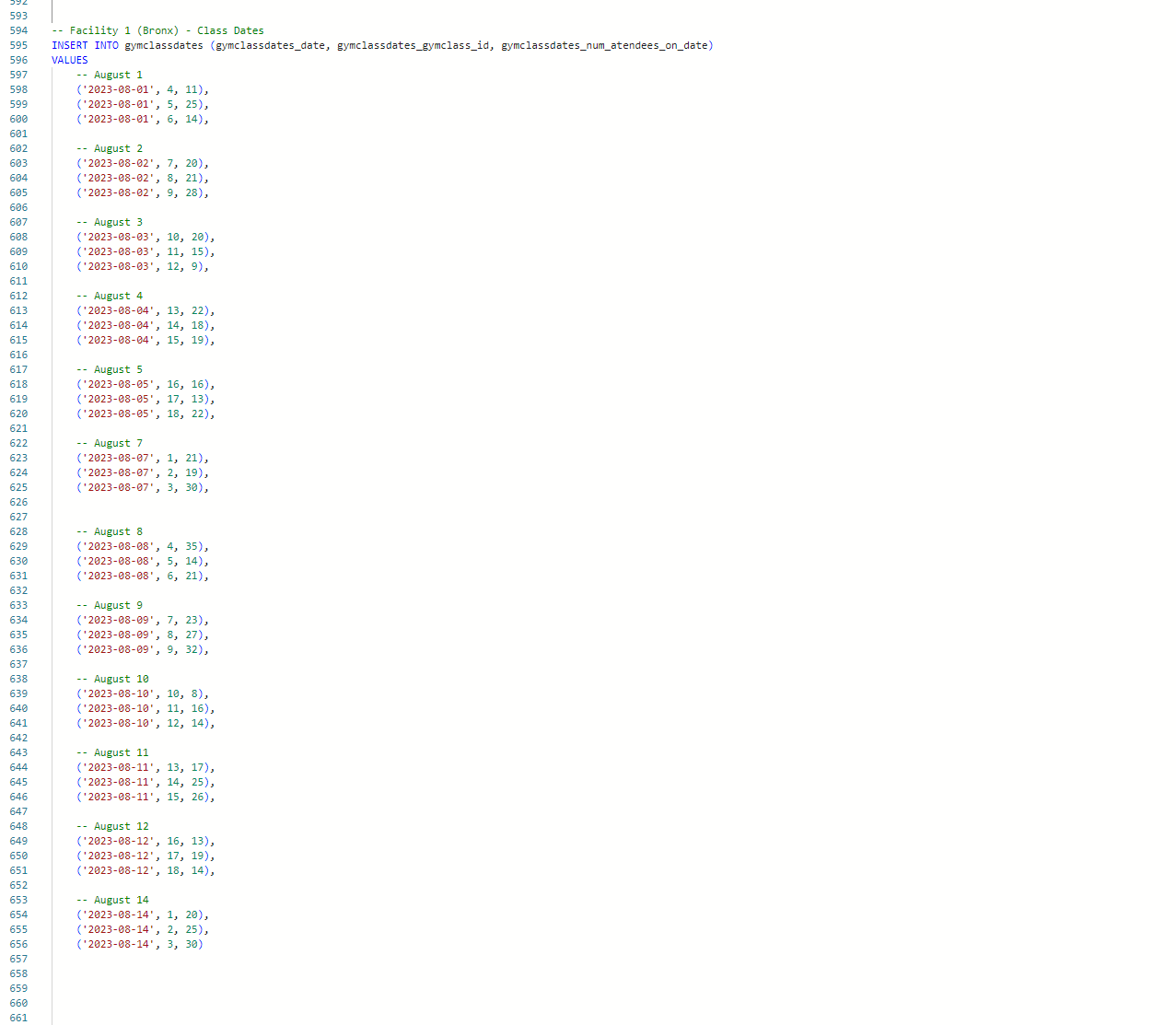














**Screen Layout:** A screenshot of a computer

Description automatically generated

**Logic Questions:**  
  
-- First let's look at the tables  
  
SELECT \* FROM gymfacilities

A screenshot of a computer

Description automatically generated

SELECT \* FROM gymclassdates

A screenshot of a computer

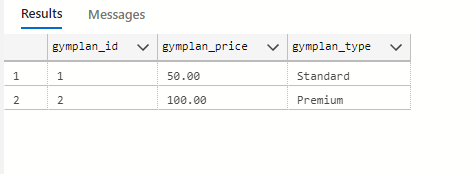
Description automatically generated

SELECT \* FROM gymmembers

A screenshot of a computer

Description automatically generated

SELECT \* FROM gymplans



SELECT \* FROM gymclasses

A screenshot of a computer

Description automatically generated

SELECT \* FROM gymtrainers

A screenshot of a computer

Description automatically generated  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
SELECT \* FROM checkins

A screenshot of a computer

Description automatically generated

-- 1. Let's see which facility has the most check ins

SELECT

gf.gymfacility\_borough AS Facility\_Borough,

COUNT(ci.checkin\_id) AS Number\_of\_Checkins

FROM

gymfacilities gf

LEFT JOIN

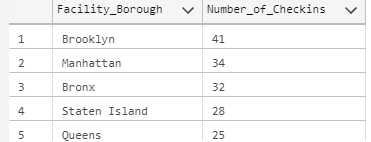
checkins ci ON gf.gymfacility\_id = ci.checkin\_facility\_id

GROUP BY

gf.gymfacility\_borough

ORDER BY

Number\_of\_Checkins DESC;



-- It appears Brooklyn has been the most popular facility, while Queens has had the lowest number of check ins.

-- 2. Let's take a look at the classes and the trainers for the classes. We will order the query by the trainer name so that we can see all of the classes for each trainer.

SELECT

gymclasses.gymclass\_id AS Class\_ID,

gymclasses.gymclass\_category AS Class\_Category,

gymclasses.gymclass\_name AS Class\_Name,

gymclasses.gymclass\_trainer\_id AS Trainer\_ID,

CONCAT(gymtrainers.gymtrainer\_firstname, ' ', gymtrainers.gymtrainer\_lastname) AS Trainer\_Name,

gymfacilities.gymfacility\_borough AS Facility\_Borough,

gymclasses.gymclass\_day\_of\_week AS Day,

gymclasses.gymclass\_start\_time AS Start\_Time,

gymclasses.gymclass\_end\_time AS End\_Time

FROM

gymclasses

JOIN

gymtrainers ON gymclasses.gymclass\_trainer\_id = gymtrainers.gymtrainer\_id

JOIN

gymfacilities ON gymclasses.gymclass\_facility\_id = gymfacilities.gymfacility\_id

ORDER BY

Trainer\_Name;

A screenshot of a computer

Description automatically generated

--3. Let's see which class category is most popular amongst all facilities.

SELECT

gymclasses.gymclass\_category AS Class\_Category,

SUM(gymclassdates.gymclassdates\_num\_atendees\_on\_date) AS Total\_Attendees

FROM

gymclasses

JOIN

gymclassdates ON gymclasses.gymclass\_id = gymclassdates.gymclassdates\_gymclass\_id

GROUP BY

gymclasses.gymclass\_category

ORDER BY

Total\_Attendees DESC;

A screenshot of a computer

Description automatically generated

--We see that strength, yoga and cardio are the most popular classes with fairly similar number of attendees.

--4. Let's see if we can find the most popular class categories for each individual facility.

WITH AttendeeCounts AS (

SELECT

gymfacilities.gymfacility\_borough AS Borough,

gymclasses.gymclass\_category AS Class\_Category,

SUM(gymclassdates.gymclassdates\_num\_atendees\_on\_date) AS Total\_Attendees

FROM

gymclasses

JOIN

gymclassdates ON gymclasses.gymclass\_id = gymclassdates.gymclassdates\_gymclass\_id

JOIN

gymfacilities ON gymclasses.gymclass\_facility\_id = gymfacilities.gymfacility\_id

GROUP BY

gymfacilities.gymfacility\_borough,

gymclasses.gymclass\_category

)

SELECT

Borough,

Class\_Category,

Total\_Attendees,

RANK() OVER (PARTITION BY Borough ORDER BY Total\_Attendees DESC) AS Category\_Rank

FROM

AttendeeCounts

ORDER BY

Borough,

Total\_Attendees DESC;

A screenshot of a computer

Description automatically generated

-- It appears that there are differences in the popularity of classes at some facilities. In the Bronx and Brooklyn, cardio are most popular. Yoga and strength are more popular in Manhattan and Staten Island.  
  
  
--5. Let's see if the time a class starts may factor into a class’ popularity. Let's see the total number of attendees grouped by the start of a class time.  
  
SELECT

gymclasses.gymclass\_start\_time AS Class\_Start\_Time,

SUM(gymclassdates.gymclassdates\_num\_atendees\_on\_date) AS Total\_Attendees

FROM

gymclasses

JOIN

gymclassdates ON gymclasses.gymclass\_id = gymclassdates.gymclassdates\_gymclass\_id

GROUP BY

gymclasses.gymclass\_start\_time

ORDER BY

gymclasses.gymclass\_start\_time;

A screenshot of a computer

Description automatically generated

-- It appears that there are nearly an equal amount of attendees for each start time.

--6. Let's take a look at the members. We like to know the demographics of the members across all of our facilities, as well as each individual facility. Let's first find out the average age and the male to female ratio across all facilities.  
  
SELECT

SUM(CASE WHEN gymmember\_gender = 'Male' THEN 1 ELSE 0 END) AS number\_of\_males,

SUM(CASE WHEN gymmember\_gender = 'Female' THEN 1 ELSE 0 END) AS number\_of\_females,

AVG(gymmember\_age) AS average\_age

FROM gymmembers;

A screenshot of a computer

Description automatically generated

-- We see that the number of males to females are about even, and the average age across all gyms is 47.

--7. Let's see the average age based on the check-in's for each facility.

SELECT

f.gymfacility\_borough,

c.checkin\_facility\_id,

AVG(m.gymmember\_age) AS average\_age

FROM checkins c

JOIN gymmembers m ON c.checkin\_gymmember\_id = m.gymmember\_id

JOIN gymfacilities f ON c.checkin\_facility\_id = f.gymfacility\_id

GROUP BY f.gymfacility\_borough, c.checkin\_facility\_id;

A screenshot of a computer

Description automatically generated

-- We see that the age range is similar at each gym.

--8. The manager would like to send an email to all premium members from the Queens facility regarding an upcoming perks for premium members exclusive at the Queens facility.

SELECT DISTINCT

m.gymmember\_firstname,

m.gymmember\_lastname,

m.gymmember\_email,

m.gymmember\_plan\_id

FROM checkins c

JOIN gymmembers m ON c.checkin\_gymmember\_id = m.gymmember\_id

JOIN gymfacilities f ON c.checkin\_facility\_id = f.gymfacility\_id

WHERE f.gymfacility\_borough = 'Queens' AND m.gymmember\_plan\_id = 2;

A screenshot of a computer

Description automatically generated

--9. The gym has hired a database administrator, who would like to improve the company's queries for more efficiency etc.

--Let's try some of the same queries using some of the concepts learned in a later chapter for efficiency etc..

--First, let's create a view function which shows a table of classes and the number of attendees of each class by date.

CREATE VIEW GymClassAttendance AS

SELECT

gc.gymclass\_id,

gc.gymclass\_name,

gc.gymclass\_category,

gc.gymclass\_trainer\_id,

gt.gymtrainer\_firstname AS trainer\_firstname,

gt.gymtrainer\_lastname AS trainer\_lastname,

gc.gymclass\_day\_of\_week,

gc.gymclass\_start\_time,

gc.gymclass\_end\_time,

gca.gymclassdates\_date,

gca.gymclassdates\_num\_atendees\_on\_date

FROM

gymclasses gc

INNER JOIN

gymclassdates gca ON gc.gymclass\_id = gca.gymclassdates\_gymclass\_id

LEFT JOIN

gymtrainers gt ON gc.gymclass\_trainer\_id = gt.gymtrainer\_id;

GO

SELECT

gymclass\_name,

gymclassdates\_date,

gymclassdates\_num\_atendees\_on\_date

FROM

GymClassAttendance

ORDER BY gymclassdates\_date

A screenshot of a computer

Description automatically generated

-- 10. The database administrator would also like to create a transaction safe procedure which ensures that a class is updated correctly if the trainer changes.

CREATE PROCEDURE UpdateGymClassTrainer

@class\_id INT,

@new\_trainer\_id INT

AS

BEGIN

BEGIN TRY

-- Start a transaction

BEGIN TRANSACTION;

-- Check if the gym class exists

IF NOT EXISTS (SELECT 1 FROM gymclasses WHERE gymclass\_id = @class\_id)

BEGIN

THROW 50001, 'Gym class does not exist.', 1;

END;

-- Check if the new trainer exists

IF NOT EXISTS (SELECT 1 FROM gymtrainers WHERE gymtrainer\_id = @new\_trainer\_id)

BEGIN

THROW 50002, 'New trainer does not exist.', 1;

END;

-- Update the gym class with the new trainer

UPDATE gymclasses

SET gymclass\_trainer\_id = @new\_trainer\_id

WHERE gymclass\_id = @class\_id;

-- Commit the transaction

COMMIT;

PRINT 'Gym class trainer updated successfully.';

END TRY

BEGIN CATCH

-- Rollback the transaction if an error occurs

ROLLBACK;

DECLARE @ErrorMessage NVARCHAR(4000);

DECLARE @ErrorSeverity INT;

DECLARE @ErrorState INT;

SELECT

@ErrorMessage = ERROR\_MESSAGE(),

@ErrorSeverity = ERROR\_SEVERITY(),

@ErrorState = ERROR\_STATE();

THROW @ErrorSeverity, @ErrorMessage, @ErrorState;

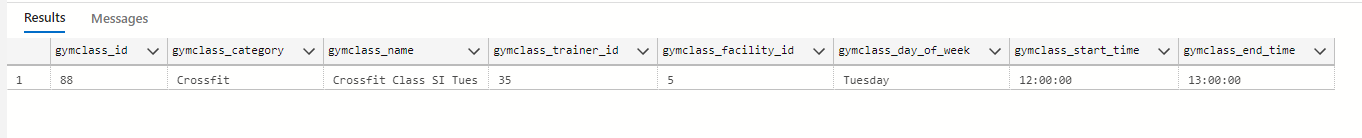
END CATCH;

END;

-- Execute the stored procedure to update the trainer for a gym class

EXEC UpdateGymClassTrainer 88, 35; -- Replace with the appropriate class\_id and new\_trainer\_id

SELECT \* FROM gymclasses WHERE gymclass\_id = 88



--We can see that the gymtrainer\_id is now 35 for this class.

--11. The database administrator would like to use indexing to improve the database performance. Recall the query from earlier where we found the class category popularity by facility. Let's try to improve with indexing.

-- Create an index on gymclass\_id and gymclass\_category columns in the gymclasses table

CREATE INDEX idx\_gymclass\_id\_category ON gymclasses (gymclass\_id, gymclass\_category);

-- Create an index on the gymfacility\_borough column in the gymfacilities table

CREATE INDEX idx\_gymfacility\_borough ON gymfacilities (gymfacility\_borough);

-- Again, let's see which class category is most popular amongst all facilities.

SELECT

gymclasses.gymclass\_category AS Class\_Category,

SUM(gymclassdates.gymclassdates\_num\_atendees\_on\_date) AS Total\_Attendees

FROM

gymclasses

JOIN

gymclassdates ON gymclasses.gymclass\_id = gymclassdates.gymclassdates\_gymclass\_id

GROUP BY

gymclasses.gymclass\_category

ORDER BY

Total\_Attendees DESC;

A screenshot of a computer

Description automatically generated

--12. Lastly, the database administrator would like to create a trigger function to ensure that members are over the age of 18 if they are added to the gymmembers table.

-- Create the trigger function

CREATE TRIGGER CheckNewMemberAgeOver18

ON gymmembers

AFTER INSERT

AS

BEGIN

DECLARE @member\_id INT, @member\_age INT;

-- Get the member ID and age of the inserted member

SELECT @member\_id = i.gymmember\_id,

@member\_age = i.gymmember\_age

FROM inserted i;

-- Check if the member's age is less than or equal to 18

IF @member\_age <= 18

BEGIN

-- Member is under 18; roll back the transaction

ROLLBACK TRANSACTION;

END;

END;

A screenshot of a computer

Description automatically generated

-- Enable the trigger

ENABLE TRIGGER CheckNewMemberAgeOver18 ON gymmembers;

--Now let's see if the trigger works. First we insert a member over the age of 18.

INSERT INTO gymmembers (gymmember\_firstname, gymmember\_lastname, gymmember\_age, gymmember\_gender, gymmember\_email, gymmember\_plan\_id, gymmember\_phonenumber, gymmember\_zipcode)

VALUES ('Randy', 'Watson', 25, 'Male', '[mrrandywatson@soulglo.com](mailto:mrrandywatson@soulglo.com)', 1, '123-456-7890', '12345');

--We can see that this member was inserted successfully.

A screenshot of a computer

Description automatically generated

-- Insert a new member under the age of 18

INSERT INTO gymmembers (gymmember\_firstname, gymmember\_lastname, gymmember\_age, gymmember\_gender, gymmember\_email, gymmember\_plan\_id, gymmember\_phonenumber, gymmember\_zipcode)

VALUES ('Billy', 'Jean', 10, 'Female', '[billy.jean@example.com](mailto:billy.jean@example.com)', 1, '987-654-3210', '54321');

--As we expected, this insert statement was not allowed because the member is under the age of 18.

A white paper with red text

Description automatically generated

**Summary:**

We were able to gain an understanding of the fundamentals of how to design, code and query a database. For this database, we were able to successfully get an understanding of class attendance at the different facilities as well information regarding the members who visit each facility. We were able to create a design in 3NF which reduced redundant tables etc. With our database design, managers can quickly query information as it relates to the operations of the gym.

In conclusion, we suggest that investors focus on opening a new facility in Brooklyn. Based on our findings, it seems that although Strength is most popular in all of NYC, we found that in Brooklyn, the ranking changes, which shows diversity. With that, we recommend offering a good combination of Cardio, Yoga, and Strength classes. We should also continue to offer classes scheduled for 7am, 12pm and 6pm. However, we should consider offering multiple classes in one time slot, 12pm being that is most popular.

Our promotional efforts should have a heavy focus on targeting younger age groups between 18-30 since our Average age group in NYC is 47 and Average age group in Brooklyn is 40.

**Reflection:**

Creating this database gave us all the opportunity to collaborate as a group and really dive into the topics and concepts. We found it interesting how we had one business scenario in mind and as we progressed, we changed certain details based on what we learned and our ability with the time given. Some of the details we would’ve love to add if we had more time was creating different check-in methods to make the fitness centers available for members that prefer not to take classes and expand on the premium benefits such as nutrition and tracking online classes.

**Team Log:**

|  |  |  |
| --- | --- | --- |
| Name | Activity | Date |
| Samantha, Pierina, Darius | Team Meeting - Discuss Project Ideas | 7/18/2023 |
| Samantha, Pierina, Darius | Team Meeting - Finalize Project Idea | 7/28/2023 |
| Samantha, Pierina, Darius | Team Meeting - Discuss Conceptual Model | 8/1/2023 |
| Samantha, Pierina, Darius | Team Meeting - Discuss SQL code and next steps | 8/8/2023 |
| Samantha, Pierina, Darius | Team Meeting - Run sample code discuss next queries to include | 8/15/2023 |
| Samantha, Pierina, Darius | Team Meeting - Run sample code, discuss next queries to include | 8/22/2023 |
| Samantha, Pierina, Darius | Team Meeting - Discuss next steps to keep the project moving | 8/29/2023 |
| Samantha, Pierina, Darius | Team Meeting - finalize business questions and begin write up | 9/5/2023 |
| Samantha, Pierina, Darius | Team Meeting - Go over mock application | 9/12/2023 |
| Samantha, Pierina, Darius | Team Meeting -Work on presentation | 9/13/2023 |
| Samantha, Pierina, Darius | Team Meeting - Finalized report and submit assignment | 9/17/2023 |

**References:**

The materials from the course were primarily used for this project. ChatGPT was used to assist with generating mock data for the tables and used to assist with querying the database.