Body BluePrint Fitness Center Report

Introduction

The Body Blueprint Center project aims to create a comprehensive database management system to streamline operations and enhance decision-making for the newly opened fitness center. This system is designed to manage various aspects of the center, including memberships, trainers, equipment, employees, and class schedules. By centralizing this information, the project facilitates efficient tracking of financial performance, member engagement, and resource allocation.

Main Requirements

The primary requirements of the Body Blueprint Center database project include:

1. Body BluePrint offers 8 subscription plans. Each plan enables specific features. From those plans offered, there's a student plan for people less than 30, and a senior plan for people more than 57 years.

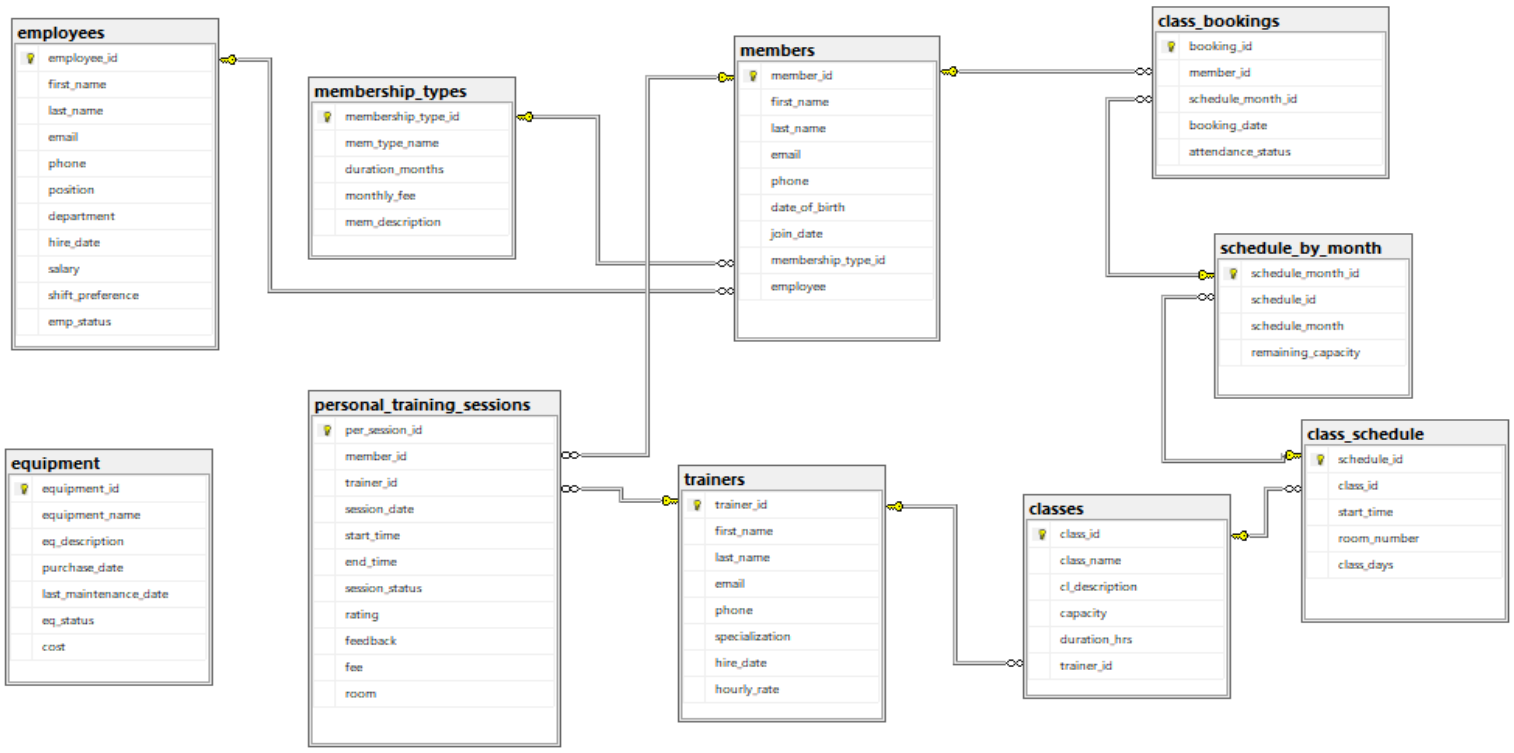
2. Body BluePrint offers 8 fitness classes open to everyone. These classes do not require any additional fee other than the subscription plan.
3. The center has 8 trainers one for each class and employees for front desk, management, maintenance and cleaning staff.

4. These classes are a month long and have a specific capacity. Every month, different classes are open. Those classes are available 2 days a week, all before 2 pm.

5. Booking these classes starts from the 25th till the 31rst of the previous month.

6. The center also gives you the option to book a personal training session with the trainer you want. Those sessions are usually after 4pm. Those sessions require an additional fee other than the subscription plan.

ERD



Tables & Constraints & Triggers:

1. **members:**

   CONSTRAINTS: 1- DF Join date to get current date

```
ALTER TABLE [dbo].[members] ADD  DEFAULT (getdate()) FOR [join_date]
GO
```

   TRIGGERS: 1- manage membership students for people less the 30 years old and membership seniors for people more than 57 years.

```
CREATE TRIGGER [dbo].[membership_management]
ON [dbo].[members]
AFTER INSERT
AS
BEGIN
    DECLARE @memship INT, @dob DATE;
    SELECT @memship = membership_type_id FROM inserted;
    SELECT @dob = date_of_birth FROM inserted;
    IF(@memship=3 and (DATEPART(YEAR, GETDATE())-DATEPART(YEAR,@dob)>30))
    BEGIN
        RAISERROR('This member can not benefit from student plan.',16,1)
        ROLLBACK TRANSACTION
    END
    IF (@memship=4 and (DATEPART(YEAR, GETDATE())-DATEPART(YEAR,@dob)<57))
    BEGIN
        RAISERROR ('This member can not benefit from senior plan.',16,1)
        ROLLBACK TRANSACTION
    END

END;



GO

ALTER TABLE [dbo].[members] ENABLE TRIGGER [membership_management]
GO
```

2. **Membership-type:**

   NO CONSTRAINTS OR TRIGGERS

3. **Trainers:**

   CONSTRAINTS: 1-DF hire date to current date

```
ALTER TABLE [dbo].[trainers] ADD  DEFAULT (getdate()) FOR [hire_date]
GO
```

   NO TRIGGERS

4. **Equipment:**

CONSTRAINTS: 1-CK equipment status in (Out of Order, Under Maintenance, In Use, Available

```
ALTER TABLE [dbo].[equipment]  WITH CHECK ADD CHECK
((([eq_status]='Out of Order' OR [eq_status]='Under Maintenance' OR [eq_status]='In Use' OR [eq_status]='Available'))
GO
```

NO TRIGGERS

5. **Classes:**
   NO CONSTRAINTS OR TRIGGERS

6. **Class schedule:**
   NO CONSTRAINTS

   TRIGGERS: 1- Manage schedule time before 2 pm

```
CREATE TRIGGER [dbo].[schedule_time_before_2]
ON [dbo].[class_schedule]
AFTER INSERT
AS
BEGIN
    DECLARE @start_time TIME;
    SElect @start_time = start_time FROM inserted;
    IF (DATEPART(HOUR,@start_time)>14)
    BEGIN
        RAISERROR ('Class time should be before 2 pm',16,1)
        ROLLBACK TRANSACTION
    END

END;

GO

ALTER TABLE [dbo].[class_schedule] ENABLE TRIGGER [schedule_time_before_2]
GO
```

7. **Schedule month:**

NO CONSTRAINTS

TRIGGERS: 1-Set by default remaining capacity to class capacity

```
CREATE TRIGGER [dbo].[set_default_remaining_capacity1]
ON [dbo].[schedule_by_month]
AFTER INSERT
AS
BEGIN
    DECLARE @class_id INT;
    DECLARE @capacity INT;
    DECLARE @schedule INT;
    DECLARE @month INT;
    -- Get the class_id from the inserted row
    SELECT @schedule = schedule_id FROM inserted;
    SELECT @month = schedule_month FROM inserted;
    SELECT @class_id = class_id FROM class_schedule WHERE @schedule = schedule_id;

    -- Get the capacity from the classes table
    SELECT @capacity = capacity
    FROM classes
    WHERE class_id = @class_id;

    -- Insert the new row with the default remaining_capacity

    UPDATE schedule_by_month
    SET remaining_capacity=@capacity
    WHERE schedule_id=@schedule and schedule_month=@month;
END;
GO

ALTER TABLE [dbo].[schedule_by_month] ENABLE TRIGGER [set_default_remaining_capacity1]
GO
```

8. **Class Booking:**

CONSTRAINTS: 1- CK booking date day between 25 to 31

```
ALTER TABLE [dbo].[class_bookings]  WITH CHECK ADD CHECK
((datepart(day,[booking_date])>=(25) AND datepart(day,[booking_date])<=(31)))
GO
```

2-DF attendance status is Booked

```
ALTER TABLE [dbo].[class_bookings] ADD  DEFAULT ('Booked') FOR [attendance_status]
GO
```

TRIGGERS: 1-Check availability of class capacity and manage booking date to be after member join date and trainer hire

date and book date month is one month before the schedule month

```
CREATE TRIGGER [dbo].[check_class_capacity_manage_date]
ON [dbo].[class_bookings]
AFTER INSERT
AS
BEGIN
    DECLARE @class_capacity INT;
    DECLARE @current_bookings INT;
    DECLARE @schedule_id INT, @schedule_month_id INT;
    DECLARE @bookdate DATE, @member INT,@join DATE,@hiredate DATE, @class INT;
    DECLARE @schedule_month INT;

    SELECT @schedule_month_id = schedule_month_id FROM inserted;
    SELECT @bookdate= booking_date FROM inserted;
    SELECT @member = member_id FROM inserted;
    SELECT @schedule_id=schedule_id FROM schedule_by_month WHERE schedule_month_id=@schedule_month_id;
    SELECT @class= class_id FROM class_schedule WHERE schedule_id=@schedule_id;
    SELECT @join = join_date FROM members WHERE member_id=@member;
    SELECT @hiredate = hire_date FROM trainers WHERE trainer_id=(SELECT trainer_id FROM classes WHERE class_id=@class);
    SELECT @schedule_month=schedule_month FROM schedule_by_month WHERE schedule_month_id=@schedule_month_id;


    -- Get the capacity of the class
    SELECT @class_capacity = capacity
    FROM classes
    WHERE(class_id=@class);

    -- Get the current number of bookings for the class
    SELECT @current_bookings = remaining_capacity
    FROM schedule_by_month
    WHERE schedule_month_id = @schedule_month_id;

     -- Check if the class is already full
    IF @current_bookings < 0
    BEGIN
        RAISERROR('Class is already full', 16, 1);
        ROLLBACK TRANSACTION;
    END
    IF (@join>@bookdate or @hiredate>@bookdate)
    BEGIN
        RAISERROR('The booking date should be after the member join date or the trainer hire date',16,1)
        ROLLBACK TRANSACTION
    END
    IF ((@schedule_month)!=(DATEPART(month,@bookdate)+1))
    BEGIN
        RAISERROR('The booking date month is not coherent with the schedule month date',16,1)
        ROLLBACK TRANSACTION
    END;
END;
GO

ALTER TABLE [dbo].[class_bookings] ENABLE TRIGGER [check_class_capacity_manage_date]
GO
```

2-Update remaining capacity in class schedule depending on a class booking

```
CREATE TRIGGER [dbo].[update_remaining_capacity]
ON [dbo].[class_bookings]
AFTER INSERT
AS
BEGIN

    UPDATE schedule_by_month
    SET remaining_capacity = remaining_capacity - 1
    WHERE schedule_month_id = (SELECT schedule_month_id FROM inserted)


END;
GO

ALTER TABLE [dbo].[class_bookings] ENABLE TRIGGER [update_remaining_capacity]
GO
```

9. **Personal training sessions:**

CONSTRAINTS: 1-CK Rating between 1 and 5

```
ALTER TABLE [dbo].[personal_training_sessions]  WITH CHECK ADD CHECK
(([rating]>=(1) AND [rating]<=(5)))
GO
```

2-CK session status in (No-show, Cancelled, Completed, Scheduled)

```
ALTER TABLE [dbo].[personal_training_sessions]  WITH CHECK ADD CHECK
(([session_status]='No-Show' OR [session_status]='Cancelled' OR [session_status]='Completed'
OR [session_status]='Scheduled'))
GO
```

TRIGGERS: 1- Automatically calculate the fee of the session knowing the duration of the session and the hourly rate for each trainer

```sql
CREATE TRIGGER [dbo].[CalculateSessionFee]
ON [dbo].[personal_training_sessions]
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @SessionId INT, @TrainerId INT, @StartTime TIME, @EndTime TIME, @Duration INT, @HourlyRate DECIMAL(10,2);

    SELECT @SessionId = per_session_id, @TrainerId = trainer_id, @StartTime = start_time, @EndTime = end_time
    FROM inserted ;

    SELECT @HourlyRate = hourly_rate
    FROM Trainers
    WHERE trainer_id = @TrainerId;

    -- Calculate duration in minutes
    SET @Duration = DATEDIFF(minute, @StartTime, @EndTime);

    -- Calculate fee based on duration and hourly rate
    UPDATE [dbo].[personal_training_sessions]
    SET fee = ((@Duration / 60.0) * @HourlyRate)*1.25
    WHERE per_session_id = @SessionId;
END;
GO

ALTER TABLE [dbo].[personal_training_sessions] ENABLE TRIGGER [CalculateSessionFee]
GO
```

2-Manage start time to be after 4pm and before the end time and manage session date to be after member join date and trainer hire date

```sql
CREATE TRIGGER [dbo].[coherent_start_end_times]
ON [dbo].[personal_training_sessions]
AFTER INSERT
AS
BEGIN
    DECLARE @start TIME, @end TIME, @sessiondate DATE, @member INT, @trainer INT, @joindate DATE, @hiredate DATE;
    SELECT @start = start_time From inserted;
    SELECT @end = end_time From inserted;
    SELECT @sessiondate = session_date FROM inserted;
    SELECT @member = member_id FROM inserted;
    SELECT @trainer = trainer_id FROM inserted;
    SELECT @joindate = join_date FROM members WHERE member_id= @member;
    SELECT @hiredate = hire_date FROM trainers WHERE trainer_id = @trainer;

    IF ((DATEPART(HOUR,@start))> (DATEPART(HOUR,@end)) or (DATEPART(HOUR,@start)<16))
    BEGIN
        RAISERROR ('Make sure the start time is after 4pm and before the end time',16,1)
        ROLLBACK TRANSACTION
    END
    IF (@joindate>@sessiondate or @hiredate>@sessiondate)
    BEGIN
        RAISERROR ('Make sure the session date is after member join date or trainer hire date',16,1)
        ROLLBACK TRANSACTION
    END
END;
GO

ALTER TABLE [dbo].[personal_training_sessions] ENABLE TRIGGER [coherent_start_end_times]
GO
```

10. Employees

CONSTRAINTS: DF employee status (Active)

```
ALTER TABLE [dbo].[employees] ADD  DEFAULT ('Active') FOR [emp_status]
GO
```

NO TRIGGERS

Data Inserted

The database is populated with sample data to facilitate testing and analysis. This includes:

- 8 membership types with different fees and durations.
- 40 members with their specific membership.
- Records of 8 trainers with distinct specializations and rates.
- 20 equipment entries with purchase and maintenance details.
- 15 employee records with salaries and positions.
- 8 classes with their description, duration and capacity.
- Class schedules and 208 bookings reflecting member participation.
- 168 personal training sessions with their details.

Queries Used to Extract Insights

1. A query that calculates the revenue generated by each membership type.

```sql
WITH MembershipRevenue AS (
    SELECT
        mt.mem_type_name,
        COUNT(m.member_id) as member_count,
        SUM(mt.monthly_fee * mt.duration_months) as total_revenue
    FROM members m
    JOIN membership_types mt ON m.membership_type_id = mt.membership_type_id
    GROUP BY mt.mem_type_name
)
SELECT
    mem_type_name,
    member_count,
    total_revenue,
    CAST((member_count * 100.0 / SUM(member_count) OVER()) AS DECIMAL(10, 2))
    AS membership_percentage
FROM MembershipRevenue
ORDER BY member_count DESC, total_revenue DESC;
```

	mem_type_name	member_count	total_revenue	membership_percentage
1	Student	15	2250.00	37.50
2	Basic	8	320.00	20.00
3	Senior	5	100.00	12.50
4	Couple	4	2640.00	10.00
5	Premium	4	720.00	10.00
6	Family	3	2880.00	7.50
7	Off-Peak	1	25.00	2.50

2. Active Sales Employee Revenue and Member Sign-up Analysis

```sql
WITH MemberSignups AS (
    SELECT
        e.employee_id,
        COUNT(m.member_id) as members_signed,
        SUM(mt.monthly_fee * mt.duration_months) as revenue_generated
    FROM employees e
    LEFT JOIN members m ON e.employee_id = m.employee
    LEFT JOIN membership_types mt ON m.membership_type_id = mt.membership_type_id
    GROUP BY e.employee_id,e.department
)
SELECT
    e.first_name + ' ' + e.last_name as employee_name,
    e.position,
    e.department,
    e.shift_preference,
    DATEDIFF(month, e.hire_date, GETDATE()) as months_employed,
    ms.members_signed,
    ms.revenue_generated,
    e.salary,
    ROUND((ms.revenue_generated / e.salary), 2) as revenue_to_salary_ratio
FROM employees e
LEFT JOIN MemberSignups ms ON e.employee_id = ms.employee_id
WHERE e.emp_status = 'Active' and e.department='Sales'
ORDER BY revenue_generated DESC;
```

	employee_name	position	department	shift_preference	months_employed	members_signed	revenue_generated	salary	revenue_to_salary_ratio
1	Kevin Patel	Membership Consultant	Sales	Morning	11	13	3460.00	450.00	7.690000
2	Lisa Chen	Membership Director	Sales	Day	11	14	3265.00	500.00	6.530000
3	Rachel Green	Membership Consultant	Sales	Evening	11	13	2210.00	450.00	4.910000

3. Member Activity and Engagement Analysis

```sql
WITH MemberActivity AS (
    SELECT
        m.member_id,
        m.first_name + ' ' + m.last_name as member_name,
        m.join_date,
        COUNT(DISTINCT cb.booking_id) as class_bookings,
        COUNT(DISTINCT pts.per_session_id) as personal_sessions,
        MAX(CASE
            WHEN cb.booking_date IS NOT NULL THEN cb.booking_date
            WHEN pts.session_date IS NOT NULL THEN pts.session_date
        END) as last_activity_date
    FROM members m
    LEFT JOIN class_bookings cb ON m.member_id = cb.member_id
    LEFT JOIN personal_training_sessions pts ON m.member_id = pts.member_id
    GROUP BY m.member_id, m.first_name, m.last_name, m.join_date
)
SELECT
    member_id,
    member_name,
    DATEDIFF(month, join_date, GETDATE()) as months_as_member,
    class_bookings,
    personal_sessions,
    DATEDIFF(day, last_activity_date, GETDATE()) as days_since_last_activity,
    CASE
        WHEN DATEDIFF(day, last_activity_date, GETDATE()) > 40 OR DATEDIFF(day, last_activity_date, GETDATE()) IS NULL THEN 'At Risk'
        WHEN DATEDIFF(day, last_activity_date, GETDATE()) > 15 THEN 'Needs Attention'
        ELSE 'Active'
    END as member_status
FROM MemberActivity
ORDER BY member_status, days_since_last_activity ;
```

	member_id	member_name	months_as_member	class_bookings	personal_sessions	days_since_last_activity	member_status
1	39	Gino Brown	1	0	3	3	Active
2	40	Mark Hunter	1	0	2	9	Active
3	36	Zach Eddy	2	0	2	11	Active
4	3	Alice Johnson	11	14	7	12	Active
5	6	David Smith	10	11	7	12	Active
6	24	Mary Earps	5	4	4	12	Active
7	4	Bob Brown	11	12	6	13	Active
8	21	Daxter Doe	6	6	5	13	Active
9	9	Jane Miller	9	8	5	13	Active
10	19	Charlie Davis	6	6	6	14	Active
11	5	Charlie Davis	11	11	9	14	Active
12	22	David Rodman	6	4	4	14	Active
13	8	Jonas Hiller	10	12	5	14	Active
14	2	Jane Smith	11	15	7	15	Active
15	11	David Bronze	9	10	6	15	Active
16	10	Charlie Brown	9	10	4	15	Active
17	38	Alexia Morgan	2	2	0	15	Active
18	37	Leah William...	2	0	0	NULL	At Risk
19	14	John Davis	8	6	9	43	At Risk
20	18	Ella Toone	6	4	5	44	At Risk
21	32	Charlie Puth	4	1	2	44	At Risk
22	31	James Puth	4	1	2	45	At Risk
23	23	Tom Smith	5	1	3	46	At Risk
24	27	Azzi Fudd	5	0	2	56	At Risk
25	28	James Watkins	4	0	2	56	At Risk
26	34	Stephen Jo...	3	0	1	68	At Risk
27	35	Hailey Vanlith	3	1	1	76	At Risk
28	13	Alice Davis	8	4	4	135	At Risk
29	16	Joy Smith	6	2	3	139	At Risk
30	17	Carolina Ha...	6	3	3	16	Needs Attention
31	33	David Smith	4	4	3	16	Needs Attention
32	20	Alice Hampton	6	5	7	16	Needs Attention
33	12	Jane Johnson	9	10	6	16	Needs Attention
34	15	Jake Mills	7	8	5	16	Needs Attention
35	30	Boby Smith	4	5	2	16	Needs Attention
36	25	Elias Roebuck	5	5	3	17	Needs Attention
37	7	Alice Smith	10	9	9	17	Needs Attention
38	1	John Doe	11	12	9	17	Needs Attention
39	26	Paige Bueck...	5	2	2	17	Needs Attention
40	29	Sam Prince	4	0	3	38	Needs Attention

4. Member Services Analysis & Upsell Opportunities

```sql
WITH MemberServices AS (
    SELECT
        m.member_id,
        m.first_name + ' ' + m.last_name as member_name,
        mt.mem_type_name,
        COUNT(DISTINCT cb.booking_id) as class_bookings,
        COUNT(DISTINCT pts.per_session_id) as personal_training_sessions,
        AVG(pts.rating) as avg_training_rating,
        SUM(pts.fee) as total_training_fees
    FROM members m
    LEFT JOIN membership_types mt ON m.membership_type_id = mt.membership_type_id
    LEFT JOIN class_bookings cb ON m.member_id = cb.member_id
    LEFT JOIN personal_training_sessions pts ON m.member_id = pts.member_id
    GROUP BY m.member_id, m.first_name, m.last_name, mt.mem_type_name
)
SELECT
    member_name,
    mem_type_name,
    class_bookings,
    personal_training_sessions,
    avg_training_rating,
    total_training_fees,
    CASE
        WHEN personal_training_sessions = 0 AND class_bookings > 5 THEN 'PT Prospect'
        WHEN personal_training_sessions > 0 AND avg_training_rating >= 4 THEN 'PT Upsell Opportunity'
        WHEN class_bookings = 0 AND personal_training_sessions > 0 THEN 'Class Prospect'
        WHEN mem_type_name NOT LIKE '%Premium%' AND (class_bookings > 8 OR personal_training_sessions > 4) THEN 'Membership Upgrade Prospect'
        ELSE 'Regular Member'
    END as opportunity_category
FROM MemberServices
ORDER BY total_training_fees DESC;
```

	member_name	mem_type_name	class_bookings	personal_training_sessions	avg_training_rating	total_training_fees	opportunity_category
1	Alice Johnson	Basic	14	7	3	5162.50	Membership Upgrade Prospect
2	John Doe	Basic	12	9	3	5025.00	Membership Upgrade Prospect
3	Jane Smith	Senior	15	7	3	5015.70	Membership Upgrade Prospect
4	Alice Smith	Family	9	9	3	4275.00	Membership Upgrade Prospect
5	Charlie Davis	Student	11	9	2	4245.34	Membership Upgrade Prospect
6	David Smith	Family	11	7	3	4193.75	Membership Upgrade Prospect
7	Bob Brown	Premium	12	6	4	3712.56	PT Upsell Opportunity
8	Jonas Hiller	Premium	12	5	2	3225.00	Regular Member
9	Jane Johnson	Senior	10	6	3	2875.00	Membership Upgrade Prospect
10	David Bronze	Family	10	6	4	2531.30	PT Upsell Opportunity
11	John Davis	Couple	6	9	2	2512.50	Membership Upgrade Prospect
12	Jane Miller	Senior	8	5	3	2300.00	Membership Upgrade Prospect
13	Charlie Brown	Basic	10	4	3	1687.50	Membership Upgrade Prospect
14	Alice Hampton	Student	5	7	3	1656.25	Membership Upgrade Prospect
15	Jake Mills	Premium	8	5	4	1650.00	PT Upsell Opportunity
16	Charlie Davis	Student	6	6	3	1537.50	Membership Upgrade Prospect
17	Daxter Doe	Senior	6	5	2	1350.00	Membership Upgrade Prospect
18	Alice Davis	Couple	4	4	3	1025.00	Regular Member
19	Ella Toone	Student	4	5	3	925.00	Membership Upgrade Prospect
20	David Rodman	Student	4	4	3	906.24	Regular Member
21	Mary Earps	Student	4	4	3	725.00	Regular Member
22	David Smith	Student	4	3	3	675.00	Regular Member
23	Elias Roebuck	Student	5	3	4	625.00	PT Upsell Opportunity

24	Carolina Ha...	Student	3	3	2	431.25	Regular Member
25	Boby Smith	Student	5	2	4	406.25	PT Upsell Opportunity
26	Joy Smith	Basic	2	3	3	312.50	Regular Member
27	Tom Smith	Basic	1	3	2	175.00	Regular Member
28	Gino Brown	Basic	0	3	2	168.75	Class Prospect
29	Paige Bueckers	Student	2	2	4	162.50	PT Upsell Opportunity
30	Sam Prince	Student	0	3	4	150.00	PT Upsell Opportunity
31	James Watkins	Senior	0	2	2	112.50	Class Prospect
32	Azzi Fudd	Student	0	2	4	100.00	PT Upsell Opportunity
33	Zach Eddy	Premium	0	2	4	100.00	PT Upsell Opportunity
34	James Puth	Couple	1	2	4	93.75	PT Upsell Opportunity
35	Mark Hunter	Basic	0	2	NULL	87.50	Class Prospect
36	Charlie Puth	Couple	1	2	3	81.25	Regular Member
37	Stephen Joh...	Student	0	1	NULL	56.25	Class Prospect
38	Hailey Vanlith	Student	1	1	4	37.50	PT Upsell Opportunity
39	Leah William...	Off-Peak	0	0	NULL	NULL	Regular Member
40	Alexia Morgan	Basic	2	0	NULL	NULL	Regular Member

5. Members workout patterns

```
CREATE OR ALTER FUNCTION fn_GetMemberAttendancePatterns
(
    @monthsBack int = 2
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        m.member_id,
        m.first_name + ' ' + m.last_name as member_name,
        c.class_name,
        cs.class_days,
        cs.start_time,
        COUNT(*) as attendance_count,
        STRING_AGG(CONVERT(varchar, cb.booking_date, 23), ', ') as booking_dates
    FROM members m
    JOIN class_bookings cb ON m.member_id = cb.member_id
    JOIN schedule_by_month sbm ON cb.schedule_month_id = sbm.schedule_month_id
    JOIN class_schedule cs ON sbm.schedule_id = cs.schedule_id
    JOIN classes c ON cs.class_id = c.class_id
    WHERE cb.booking_date >= DATEADD(month, -@monthsBack, GETDATE())
    GROUP BY
        m.member_id,
        m.first_name + ' ' + m.last_name,
        c.class_name,
        cs.class_days,
        cs.start_time
);
```

```sql
WITH MemberPatterns AS (
    SELECT * FROM fn_GetMemberAttendancePatterns(6)
)
SELECT
    member_name,
    COUNT(DISTINCT class_name) as different_classes_attended,
    MAX(attendance_count) as max_class_attendance,
    STRING_AGG(class_name + ' (' + CAST(attendance_count as varchar) + ' times)', '; ') as class_preference
FROM MemberPatterns
GROUP BY member_id, member_name
HAVING COUNT(DISTINCT class_name) > 1
ORDER BY different_classes_attended DESC;
```

	member_name	different_classes_attended	max_class_attendance	class_preference
1	Charlie Brown	6	1	Cardio Training (1 times); CrossFit (1 times); Dance (1 times); Pilates (1 times); Strength Training (1 times); Yoga (1 times)
2	Jake Mills	6	1	Aerobics (1 times); Boxing (1 times); CrossFit (1 times); Dance (1 times); Strength Training (1 times); Yoga (1 times)
3	Bob Brown	5	2	Aerobics (1 times); Boxing (1 times); CrossFit (1 times); Dance (1 times); Yoga (2 times)
4	Alice Hampton	5	1	Cardio Training (1 times); CrossFit (1 times); Pilates (1 times); Strength Training (1 times); Yoga (1 times)
5	Charlie Davis	5	1	Aerobics (1 times); Aerobics (1 times); Boxing (1 times); CrossFit (1 times); Strength Training (1 times); Yoga (1 times)
6	David Bronze	5	1	Aerobics (1 times); Cardio Training (1 times); CrossFit (1 times); Pilates (1 times); Yoga (1 times); Yoga (1 times)
7	David Smith	4	1	Aerobics (1 times); Cardio Training (1 times); Dance (1 times); Yoga (1 times)
8	Daxter Doe	4	3	Boxing (1 times); CrossFit (1 times); Pilates (1 times); Yoga (3 times)
9	Elias Roebuck	4	2	Aerobics (1 times); Boxing (1 times); CrossFit (1 times); Strength Training (2 times)
10	Alice Smith	4	1	Aerobics (1 times); Aerobics (1 times); Boxing (1 times); Dance (1 times); Yoga (1 times)
11	Boby Smith	4	1	Aerobics (1 times); CrossFit (1 times); CrossFit (1 times); Pilates (1 times); Strength Training (1 times)
12	Jane Johnson	4	2	Boxing (2 times); Cardio Training (1 times); Dance (2 times); Yoga (1 times)
13	Jane Miller	4	1	Aerobics (1 times); Aerobics (1 times); CrossFit (1 times); Strength Training (1 times); Yoga (1 times)
14	Jane Smith	4	1	Boxing (1 times); CrossFit (1 times); CrossFit (1 times); Pilates (1 times); Yoga (1 times); Yoga (1 times)
15	John Davis	4	1	Boxing (1 times); Cardio Training (1 times); Dance (1 times); Yoga (1 times)
16	Jonas Hiller	4	2	Aerobics (1 times); Aerobics (2 times); Cardio Training (1 times); CrossFit (1 times); Strength Training (1 times)
17	Mary Earps	4	1	Aerobics (1 times); Cardio Training (1 times); Dance (1 times); Strength Training (1 times)
18	John Doe	3	2	Cardio Training (1 times); CrossFit (2 times); Strength Training (1 times)
19	Carolina Ha…	3	1	Aerobics (1 times); Dance (1 times); Yoga (1 times)
20	Charlie Davis	3	2	Aerobics (2 times); Cardio Training (1 times); Yoga (1 times)
21	Ella Toone	3	1	Aerobics (1 times); Boxing (1 times); Boxing (1 times); CrossFit (1 times)
22	David Rodman	3	2	Aerobics (1 times); CrossFit (1 times); Yoga (2 times)
23	David Smith	3	2	Cardio Training (1 times); Cardio Training (1 times); Pilates (1 times); Yoga (2 times); Yoga (1 times)
24	Alice Davis	2	1	Aerobics (1 times); Yoga (1 times)
25	Alice Johnson	2	2	Aerobics (1 times); Aerobics (2 times); Yoga (2 times); Yoga (1 times)
26	Paige Bueckers	2	1	CrossFit (1 times); Pilates (1 times)
27	Joy Smith	2	1	Aerobics (1 times); Strength Training (1 times)

6. Trainers personal training sessions performance over the last 3 months

```sql
SELECT
    t.trainer_id,
    t.first_name + ' ' + t.last_name as trainer_name,
    t.specialization,
    COUNT(pts.per_session_id) as total_sessions,
    COUNT(CASE WHEN pts.session_status = 'Completed' THEN 1 END) as completed_sessions,
    CAST(AVG(pts.rating) as DECIMAL(10,2)) as avg_rating,
    SUM(pts.fee) as total_revenue

FROM trainers t
LEFT JOIN personal_training_sessions pts ON t.trainer_id = pts.trainer_id
WHERE pts.session_date >= DATEADD(month, -3, GETDATE())
GROUP BY t.trainer_id, t.first_name, t.last_name, t.specialization
HAVING COUNT(pts.per_session_id) > 0
ORDER BY total_revenue DESC;
```

	trainer_id	trainer_name	specialization	total_sessions	completed_sessions	avg_rating	total_revenue
1	5	Patricia Jackson	Pilates	10	7	2.00	576.56
2	2	James Anderson	Aerobics	8	6	3.00	506.25
3	7	Sara Davis	Dance	9	4	4.00	393.75
4	4	Chris Moore	Cardio Training	7	6	2.00	350.00
5	1	Anna Taylor	Crossfit	7	4	4.00	306.25
6	6	Mike Brown	Yoga	7	6	2.00	281.25
7	3	Linda Thomas	Boxing	7	7	3.00	262.50
8	8	Tom Wilson	Strength Training	1	1	5.00	62.50

7. Trainers schedule this month

```sql
WITH TrainerSchedule AS (
    SELECT
        t.trainer_id,
        t.first_name + ' ' + t.last_name as trainer_name,
        DATEPART(weekday, pts.session_date) as day_of_week,
        DATEPART(hour, pts.start_time) as hour_of_day,
        COUNT(*) as sessions_booked,
        SUM(pts.fee) as tot_session_fee
    FROM trainers t
    LEFT JOIN personal_training_sessions pts ON t.trainer_id = pts.trainer_id
    WHERE pts.session_date >= '2024-12-01'
    GROUP BY t.trainer_id, t.first_name, t.last_name,
             DATEPART(weekday, pts.session_date),
             DATEPART(hour, pts.start_time)
)
SELECT
    trainer_name,
    CASE day_of_week
        WHEN 1 THEN 'Sunday'
        WHEN 2 THEN 'Monday'
        WHEN 3 THEN 'Tuesday'
        WHEN 4 THEN 'Wednesday'
        WHEN 5 THEN 'Thursday'
        WHEN 6 THEN 'Friday'
        WHEN 7 THEN 'Saturday'
    END as weekday,
    hour_of_day,
    sessions_booked,
    tot_session_fee
FROM TrainerSchedule
ORDER BY trainer_id, day_of_week, hour_of_day;
```

	trainer_name	weekday	hour_of_day	sessions_booked	tot_session_fee
1	Anna Taylor	Sunday	17	1	43.75
2	James Anderson	Wednesday	18	1	56.25
3	James Anderson	Friday	16	1	112.50
4	Linda Thomas	Monday	18	1	37.50
5	Chris Moore	Saturday	18	1	50.00
6	Patricia Jackson	Sunday	17	1	70.31
7	Patricia Jackson	Monday	17	1	56.25
8	Mike Brown	Friday	18	1	56.25
9	Sara Davis	Tuesday	17	1	43.75
10	Sara Davis	Saturday	17	1	43.75

8. Equipment maintenance analysis

```sql
SELECT
    e.equipment_name,
    e.eq_status,
    e.purchase_date,
    e.last_maintenance_date,
    DATEDIFF(DAY, e.last_maintenance_date, CURRENT_TIMESTAMP) AS days_since_maintenance,  -- Calculate days since last maintenance

    CASE
        WHEN DATEDIFF(MONTH, e.last_maintenance_date, CURRENT_TIMESTAMP) > 6 THEN 'Overdue'
        WHEN DATEDIFF(MONTH, e.last_maintenance_date, CURRENT_TIMESTAMP) BETWEEN 5 AND 6 THEN 'Due Soon'
        ELSE 'Maintained'
    END AS maintenance_status  -- Maintenance status categorization
FROM
    equipment e
WHERE
    e.eq_status <> 'Under Maintenance'  -- Filter out retired equipment
ORDER BY
    days_since_maintenance DESC;  -- Order by days since last maintenance
```

	equipment_name	eq_status	purchase_date	last_maintenance_date	days_since_maintenance	maintenance_status
1	Treadmill Elite X1	Available	2024-01-05	2024-04-01	255	Overdue
2	Power Rack Pro	Available	2024-01-08	2024-04-05	251	Overdue
3	Spin Bike S2000	In Use	2024-01-12	2024-04-10	246	Overdue
4	Chest Press Machine	Available	2024-01-15	2024-04-12	244	Overdue
5	Lat Pulldown Station	Available	2024-01-25	2024-04-18	238	Overdue
6	Kettlebell Set Pro	In Use	2024-01-28	2024-04-20	236	Overdue
7	Leg Extension Machine	Available	2024-02-03	2024-06-21	174	Due Soon
8	Assisted Pull-up Machine	Available	2024-02-07	2024-06-22	173	Due Soon
9	Battle Ropes Set	In Use	2024-02-10	2024-06-23	172	Due Soon
10	Hack Squat Machine	Available	2024-02-14	2024-06-24	171	Due Soon
11	Olympic Weight Set	Available	2024-02-22	2024-06-26	169	Due Soon
12	Functional Trainer	Available	2024-02-25	2024-06-27	168	Due Soon
13	Dumbbell Rack Set	Available	2024-03-20	2024-08-01	133	Maintained
14	Stair Climber SC100	Available	2024-03-02	2024-08-28	106	Maintained
15	Seated Row Machine	In Use	2024-03-05	2024-08-29	105	Maintained
16	Ab Crunch Machine	Available	2024-03-10	2024-08-30	104	Maintained
17	Smith Machine Elite	Available	2024-03-15	2024-08-31	103	Maintained

9. Schedules popularity

```sql
WITH TimeSlots AS (
    SELECT
        cs.start_time,
        c.class_name,
        c.capacity,
        cs.room_number,
        cs.class_days,
        COUNT(cb.booking_id) AS total_bookings,
        DATEPART(HOUR, cs.start_time) AS hour_of_day
    FROM
        class_schedule cs
    JOIN
        classes c ON cs.class_id = c.class_id
    JOIN
        schedule_by_month sbm ON cs.schedule_id = sbm.schedule_id
    LEFT JOIN
        class_bookings cb ON sbm.schedule_month_id = cb.schedule_month_id
    GROUP BY
        cs.start_time, c.class_name, c.capacity, cs.room_number, cs.class_days
)
SELECT
    hour_of_day,
    COUNT(DISTINCT class_name) AS number_of_classes,
    SUM(total_bookings) AS total_bookings,
    SUM(capacity) AS total_capacity,
    COUNT(DISTINCT class_days) as available_schedule_days
FROM
    TimeSlots
GROUP BY
    hour_of_day
ORDER BY
    hour_of_day ;
```

	hour_of_day	number_of_classes	total_bookings	total_capacity	available_schedule_days
1	10	6	76	46	4
2	11	4	81	33	1
3	12	2	32	18	2
4	13	3	19	23	2

10. Classes bookings analysis

```sql
CREATE FUNCTION dbo.GetClassOpenCount
(
    @ClassID INT  -- Input parameter for the class ID
)
RETURNS INT  -- Return type
AS
BEGIN
    DECLARE @Result INT;

    SELECT
        @Result = COUNT( sbm.schedule_id)  -- Count distinct schedule IDs
    FROM
        classes c
    JOIN
        class_schedule s ON c.class_id = s.class_id  -- Join on the schedule table
    JOIN
        schedule_by_month sbm ON s.schedule_id = sbm.schedule_id  -- Join on the schedule_by_month table
    WHERE
        c.class_id = @ClassID;  -- Filter by class ID

    RETURN @Result;  -- Return the count
END;
```

```sql
SELECT
    c.class_name,
    c.capacity AS max_capacity,
    COUNT(cb.booking_id) AS total_bookings,
    COUNT(DISTINCT cb.member_id) AS unique_members,
    t.first_name + ' ' + t.last_name AS trainer_name,
    dbo.GetClassOpenCount(c.class_id) AS TimesOpened,

    COUNT(CASE
        WHEN schm.schedule_month = 11
        THEN cb.booking_id
    END) AS last_month_bookings,
    ROUND(
        CAST(COUNT(cb.booking_id) AS FLOAT) /
        NULLIF(dbo.GetClassOpenCount(c.class_id), 0),
        2
    ) AS avg_bookings_per_schedule
FROM classes AS c

LEFT JOIN trainers AS t ON c.trainer_id = t.trainer_id
LEFT JOIN class_schedule AS sch ON c.class_id = sch.class_id
LEFT JOIN schedule_by_month AS schm ON sch.schedule_id = schm.schedule_id
LEFT JOIN class_bookings AS cb ON schm.schedule_month_id = cb.schedule_month_id

GROUP BY c.class_id, c.class_name, c.capacity, t.first_name + ' ' + t.last_name

HAVING
    COUNT(cb.booking_id) > 0
ORDER BY
    total_bookings DESC;
```

	class_name	max_capacity	total_bookings	unique_members	trainer_name	TimesOpened	last_month_bookings	avg_bookings_per_schedule
1	Yoga	10	43	22	Mike Brown	9	3	4.78
2	Aerobics	8	35	21	James Anderson	7	4	5
3	CrossFit	8	29	20	Anna Taylor	5	6	5.8
4	Boxing	5	25	18	Linda Thomas	7	5	3.57
5	Cardio Training	10	21	16	Chris Moore	4	6	5.25
6	Pilates	10	21	16	Patricia Jackson	5	0	4.2
7	Dance	8	17	14	Sara Davis	3	5	5.67
8	Strength Training	5	17	13	Tom Wilson	4	0	4.25

11. Financial analysis: Profit

```sql
WITH Revenue AS (
    SELECT
        SUM(mt.monthly_fee * mt.duration_months) AS total_membership_revenue,
        (SELECT SUM(p.fee/1.25)
         FROM dbo.personal_training_sessions p
         WHERE p.session_status = 'Completed') AS total_training_revenue
    FROM dbo.members m
    JOIN dbo.membership_types mt ON m.membership_type_id = mt.membership_type_id
),

TrainerCosts AS (
    SELECT SUM(t.hourly_rate * 8) AS total_trainer_expenses
    FROM dbo.trainers t
),

EmployeeCosts AS (
    SELECT SUM(e.salary) AS total_employee_expenses
    FROM dbo.employees e
)
SELECT
    r.total_membership_revenue + r.total_training_revenue AS total_revenue,
    tc.total_trainer_expenses + emc.total_employee_expenses AS total_expenses,
    (r.total_membership_revenue + r.total_training_revenue -
     (tc.total_trainer_expenses + emc.total_employee_expenses)) AS total_profit
FROM
    Revenue r,
    TrainerCosts tc,
    EmployeeCosts emc;
```

	total_revenue	total_expenses	total_profit
1	13791.26	11380.00	2411.26

12. VIEW: personal training sessions receipt

```sql
CREATE VIEW [dbo].[vw_PersonalTrainingReceipt]
AS
SELECT
    pts.per_session_id AS 'Session ID',
    m.first_name + ' ' + m.last_name AS 'Member Name',
    t.first_name + ' ' + t.last_name AS 'Trainer Name',
    CONVERT(VARCHAR(10), pts.session_date, 101) AS 'Session Date',
    CONVERT(VARCHAR(5), pts.start_time, 108) + ' - ' + CONVERT(VARCHAR(5), pts.end_time, 108) AS 'Session Time',
    pts.room AS 'Room',
    pts.fee AS 'Session Fee'
FROM
    [dbo].[personal_training_sessions] pts
JOIN
    [dbo].[members] m ON pts.member_id = m.member_id
JOIN
    [dbo].[trainers] t ON pts.trainer_id = t.trainer_id
WHERE
    pts.session_status = 'Completed';
GO
```

	Session ID	Member Name	Trainer Name	Session Date	Session Time	Room	Session Fee
1	1	John Doe	Anna Taylor	02/01/2024	16:00 - 17:00	R1	43.75
2	2	Alice Johnson	Anna Taylor	02/09/2024	18:00 - 19:00	R1	43.75
3	3	Jane Smith	James Anderson	02/09/2024	18:30 - 20:00	R2	84.38
4	6	John Doe	Anna Taylor	02/18/2024	18:00 - 19:00	R1	43.75
5	7	Bob Brown	James Anderson	02/21/2024	17:30 - 19:00	R2	84.38
6	8	Alice Johnson	James Anderson	02/27/2024	19:00 - 21:00	R2	112.50
7	10	Alice Johnson	Anna Taylor	03/02/2024	16:00 - 17:00	R1	43.75
8	11	David Smith	Patricia Jackson	03/07/2024	18:00 - 19:00	R5	56.25
9	12	Alice Smith	Patricia Jackson	03/07/2024	18:00 - 19:00	R5	56.25
10	13	Jane Smith	Linda Thomas	03/10/2024	20:00 - 21:00	R3	37.50
11	14	Jonas Hiller	Mike Brown	03/14/2024	16:00 - 18:00	R6	75.00
12	15	John Doe	Chris Moore	03/19/2024	19:00 - 19:45	R4	37.50
13	16	Charlie Davis	Patricia Jackson	03/19/2024	19:00 - 19:45	R4	42.19
14	19	David Smith	Patricia Jackson	03/30/2024	20:00 - 21:00	R5	56.25
15	20	Alice Johnson	Anna Taylor	04/01/2024	17:00 - 18:00	R1	43.75
16	21	Charlie Brown	Anna Taylor	04/02/2024	17:00 - 18:00	R1	43.75
17	24	Jane Smith	Linda Thomas	04/05/2024	17:30 - 18:30	R3	37.50
18	25	Jonas Hiller	Linda Thomas	04/06/2024	18:00 - 19:00	R3	37.50
19	27	Bob Brown	Chris Moore	04/08/2024	18:00 - 19:00	R4	50.00
20	28	Alice Smith	Patricia Jackson	04/09/2024	17:15 - 18:15	R5	56.25
21	30	David Bronze	Mike Brown	04/21/2024	17:00 - 18:15	R6	46.88
22	32	Alice Smith	James Anderson	05/01/2024	17:45 - 18:45	R2	56.25
23	33	Jane Johnson	Anna Taylor	05/02/2024	18:00 - 19:00	R1	43.75

13. VIEW: Class Attendance Completed Summary

```sql
CREATE VIEW [dbo].[vw_ClassAttendanceCompleteSummary]
AS
SELECT
    c.class_name AS 'Class Name',
    cs.class_days AS 'Class Days',
    CONVERT(VARCHAR(5), cs.start_time, 108) AS 'Start Time',
    COUNT( cb.booking_id) AS 'Total_Bookings',
    SUM(CASE WHEN cb.attendance_status = 'Completed' THEN 1 ELSE 0 END) AS 'Completed'

FROM
    [dbo].[classes] c
JOIN
    [dbo].[class_schedule] cs ON c.class_id = cs.class_id
LEFT JOIN
    [dbo].[schedule_by_month] sbm ON cs.schedule_id = sbm.schedule_id
LEFT JOIN
    [dbo].[class_bookings] cb ON sbm.schedule_month_id = cb.schedule_month_id
GROUP BY
    c.class_name, cs.class_days, cs.start_time;
GO
```

	Class Name	Class Days	Start Time	Total_Bookings	Completed
1	Aerobics	Tuesday , Thursday	10:30	15	15
2	Aerobics	Wednesday , Friday	12:00	20	17
3	Boxing	Monday , Wednesday	11:00	16	16
4	Boxing	Thursday, Sataurday	10:00	9	9
5	Cardio Training	Tuesday , Thursday	12:00	12	12
6	Cardio Training	Wednesday , Fiday	10:30	9	9
7	CrossFit	Monday , Wednesday	10:00	19	19
8	CrossFit	Tuesday , Thursday	13:00	10	10
9	Dance	Monday , Wednesday	11:00	17	11
10	Dance	Thursday, Saturday	12:00	0	0
11	Pilates	Monday , Wednesday	11:30	17	9
12	Pilates	Tuesday , Thursday	13:00	4	4
13	Strength Tra...	Tuesday , Thursday	10:30	12	7
14	Strength Tra...	Wednesday , Friday	13:00	5	5
15	Yoga	Monday , Wednesday	11:30	31	27
16	Yoga	Tuesday , Thursday	10:00	12	12

Conclusion

The Body Blueprint Center project successfully establishes a robust database system that meets the operational needs of the fitness center. Through effective data management and insightful queries, the center can enhance its financial performance and improve member engagement.