1 Business Description

1.1 Business Overview

FitFlow Fitness is a multi-location gym and wellness chain that operates 30+ clubs across major cities and offers both in-person and digital fitness services. Members can access gyms, attend group classes, book personal training sessions, and join live or on-demand workouts through a mobile app. The business targets busy professionals, students, and fitness enthusiasts who want flexible access to facilities and coaching across locations.

1.2 Customers

FitFlow's primary customers are individual members who purchase monthly or annual memberships. They are segmented into tiers: 1. Basic: Single-location access; 2. Plus: Multi-location access; 3. Premium: Multi-location + classes + digital content.

Secondary customers include corporate clients who purchase bulk memberships for employees, and personal clients who pay for 1:1 or small-group coaching. Trainers and instructors are important "internal users" of the system, since they rely on scheduling, attendance, and performance.

1.3 Revenue Streams

The main revenue stream is recurring membership fees from subscription plans. Additional revenue comes from:

- Paid add-ons (class packs, personal training sessions, nutrition coaching)
- Drop-in day passes for non-members
- Corporate membership contracts
- Retail sales (merchandise, supplements, drinks at the front desk)
- Digital add-ons such as premium online programs and live-streamed classes

Because of the subscription model and add-ons, the business generates a high volume of transactional data related to membership billing, class bookings, and check-ins.

1.4 Data Needs

Operationally, FitFlow needs to track members, memberships, locations, staff, class schedules, bookings, and payments.

For the **OLTP system**, key data includes member profiles, active subscriptions, check-ins at each location, class sessions (who taught, where, when), member bookings, and payment records (amount, method, status). This data supports daily operations such as verifying access at the front desk, enforcing capacity limits in classes, allocating trainers, and handling billing and failed payments.

From an **analytical (OLAP) perspective**, FitFlow wants to analyze trends across time, locations, and member segments. Example questions include: Which locations have the highest utilization? What plans have the best retention? Which classes drive the most engagement and add-on revenue? This requires aggregating historical transactional data into fact tables (e.g., `fact_membership_billing`, `fact_class_attendance`) with dimensions such as time, location, member, plan, and class type. Overall, the business has high data throughput and clear separation between real-time transactional needs and historical analytics.

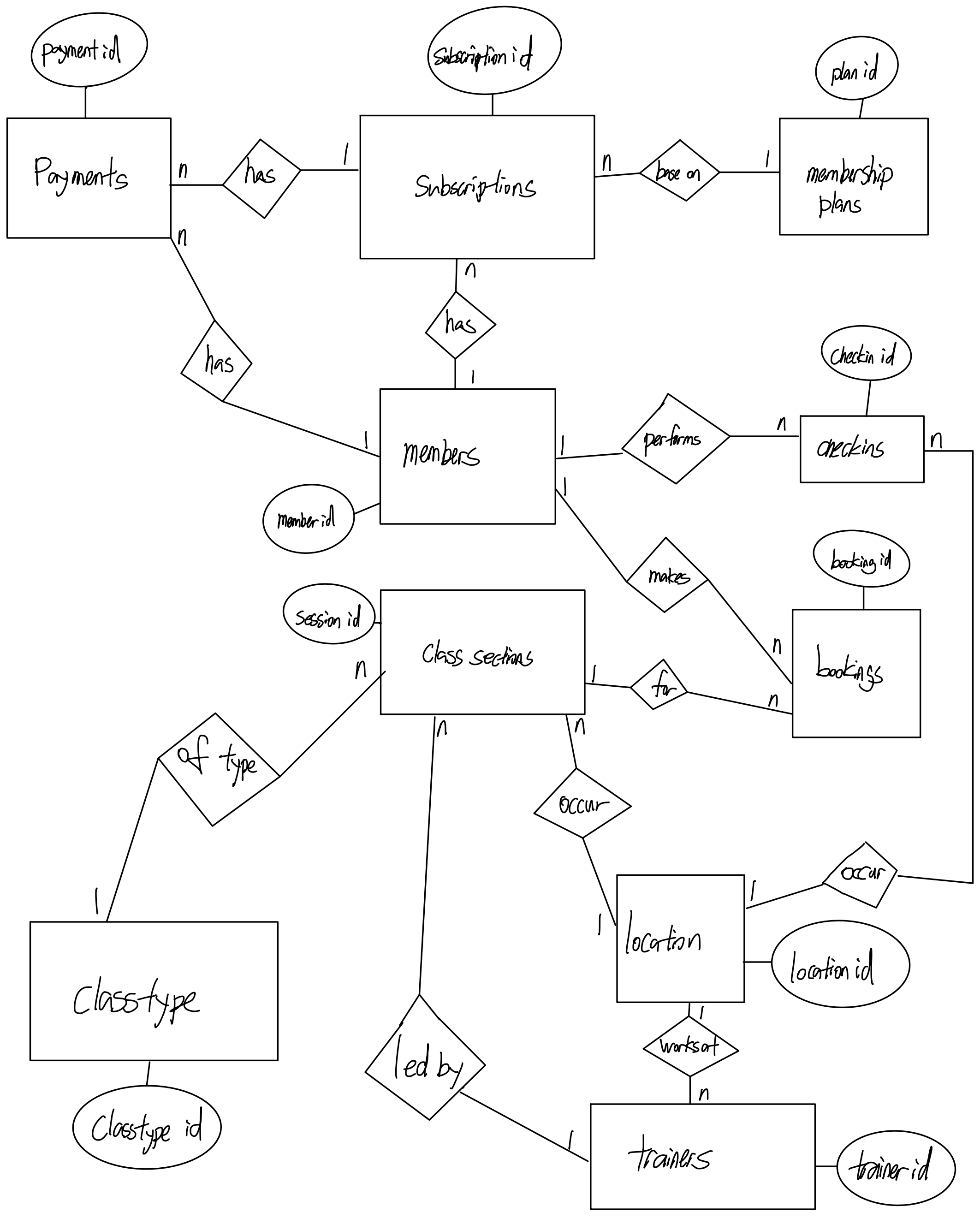


Figure 1: ER Diagram for OLTP Schema (Keys and Partial Attributes)

2 Transactional vs. Archival Data Needs

For FitFlow Fitness, the data architecture is bifurcated into two distinct categories based on operational requirements and analytical goals.

2.1 Transactional Data (OLTP)

The transactional data needs are driven by the daily operations of the 30+ gym locations. This data is characterized by high-frequency writes, real-time availability, and strict consistency (ACID properties). It's mainly used for:

- **Access Control & Check-ins:** The system must verify membership status in real-time when a member scans their badge.
- **Class Bookings:** The system tracks individual session bookings to enforce capacity limits. This requires immediate consistency to prevent overbooking.
- **Billing:** Payment processing generates critical transactional records that link specific members to specific subscription charges and add-on purchases.

2.2 Archival Data (OLAP)

The archival data needs focus on historical analysis to support strategic decision-making of the management. This data is read-heavy and often aggregated to identify trends across time, locations, and member segments.

- **Trend Analysis:** Management requires insights into utilization rates (e.g., "Which locations have the highest utilization?") and financial performance over long periods.
- **Aggregated Metrics:** Unlike the granular transactional view ("John Doe booked Spin Class at 5 PM", for example), the archival view aggregates data into metrics such as "Total Revenue per Location per Month" or "Retention Rate by Plan Tier".

Notes: Separating these systems is important in the context of the project as running heavy analytical queries (e.g., aggregating years of data) on the OLTP system would require extensive read locks. This, however, might inhibit current write operations, possibly causing latency for members trying to check in or book classes in real-time.

3 Relational Schema Design

3.1 OLTP Schema

The OLTP system is designed in 3rd Normal Form (3NF) to minimize redundancy and maintain data integrity. It strictly enforces referential integrity through foreign keys. The schema consists of 10 tables:

- **`members`**: Stores profile information (PK: `member_id`).
- **`locations`**: Stores gym branch details including capacity (PK: `location_id`).
- **`membership_plans`**: Defines membership tiers (Basic, Plus, Premium) and pricing (PK: `plan_id`).
- **`class_types`**: Defines the catalog of available classes (e.g., Yoga, HIIT) and their categories (PK: `class_type_id`).

- **`trainers`**: Stores trainer details and links them to their assigned "home" location (PK: `trainer_id`, FK: `location_id`).
- **`subscriptions`**: Links members to specific plans and tracks status (PK: `subscription_id`, FKs: `member_id`, `plan_id`).
- **`checkins`**: Records individual member visits to a location (PK: `checkin_id`, FKs: `member_id`, `location_id`).
- **`class_sessions`**: Represents specific scheduled class instances with an assigned trainer and location (PK: `session_id`, FKs: `class_type_id`, `location_id`, `trainer_id`).
- **`bookings`**: Records member reservations for specific class sessions (PK: `booking_id`, FKs: `member_id`, `session_id`).
- **`payments`**: Records transactional billing events linked to subscriptions (PK: `payment_id`, FKs: `subscription_id`, `member_id`).

3.2 OLAP Schema (Star Schema)

For the analytical system, we implemented a Star Schema to facilitate high-performance read queries. This involves denormalized dimension tables and centralized fact tables.

3.2.1 Dimension Tables

- **`Dim_Location`**: Denormalized location data (Attributes: City, State, Capacity).
- **`Dim_Plan`**: Plan details (Attributes: Plan Name, Tier, Price).
- **`Dim_Class_Type`**: Class categorization (Attributes: Category, Name).
- **`Dim_Date`**: A comprehensive date dimension for temporal analysis (Attributes: Year, Month, Day, Is_Holiday).

3.2.2 Fact Tables

- **`Fact_Monthly_Revenue`**: Aggregates financial performance.
  - *Granularity*: Per Location, Per Plan, Per Month.
  - *Measures*: Total Revenue, Transaction Count, Active Members.
- **`Fact_Class_Attendance`**: Tracks operational efficiency.
  - *Granularity*: Per Location, Per Class Type, Per Day.
  - *Measures*: Total Bookings, Attendance Count, Utilization Rate (derived from Attendance / Capacity).

4 ETL Process and Implementation

4.1 Implementation Environment

We implemented both the OLTP and OLAP systems using Azure SQL Database. The ETL process was developed using Python, leveraging `pandas` and `SQLAlchemy`. This setup allows for flexible data manipulation while adhering to the logical separation of operational and analytical processing.

4.2 ETL Workflow Description

1. **Extract**
   To minimize the load on the live operational system and prevent potential table locks that could disrupt front-desk check-ins or member bookings, we adhered to two strict extraction rules:

   - **No Joins on OLTP:** We extracted raw data from the `Payments`, `Subscriptions`, and `Bookings` tables individually. We avoided performing any `JOIN` operations or complex aggregations on the OLTP side.
   - **Incremental Extraction:** Although we populated the initial database with synthetic data, our extraction logic is designed to be incremental. By checking the timestamp of the last ETL run, we only extract records that have been created or modified since that time, rather than reloading the entire dataset.

2. **Transform**
   All raw data was loaded into `pandas` DataFrames, which served as our temporary staging area. Processing data here ensures that transformations do not impact the OLTP system.

   - **Schema Mapping & Joins:** As the OLTP schema is normalized, we performed joins in the staging area to enrich the data before aggregation. For example, to generate the `Fact_Class_Attendance` table, we joined `bookings` with `class_sessions` to associate each booking with its specific class type and location. Although `trainers` data exists in OLTP for operational scheduling, it was not moved to OLAP as trainer performance analysis was outside the current scope of our Dimension design.
   - **Data Cleaning & Integrity:** We filtered out invalid records (e.g., failed payments) and standardized date formats. Transaction timestamps were converted into standard date keys (YYYYMMDD) to align with the `Dim_Date` table.

3. **Load**
   The final step involved loading the transformed data into the Azure OLAP system.

   - **"Dimensions First":** We strictly loaded the Dimension tables (`Dim Location`, `Dim Plan`, `Dim Class Type`) first. This ensures that when Fact tables are loaded, the foreign keys (Surrogate Keys) correctly reference existing dimension records.
   - **Fact Table Loading:** Finally, the aggregated transactional data was bulk-loaded into `Fact Monthly Revenue` and `Fact Class Attendance` using the `to_sql` method in append mode.

5 FitFlow Management Dashboard

We developed an interactive dashboard using Python (Dash/Plotly) connected to the Azure OLAP database. The dashboard provides management with real-time insights into revenue and operational efficiency, featuring interactive filtering capabilities.

5.1 Monthly Revenue Trends (Overall)

This visualization tracks the Total Revenue Over Time. When no location is selected, the chart aggregates revenue across all locations to provide an overall business view of the company's financial growth.

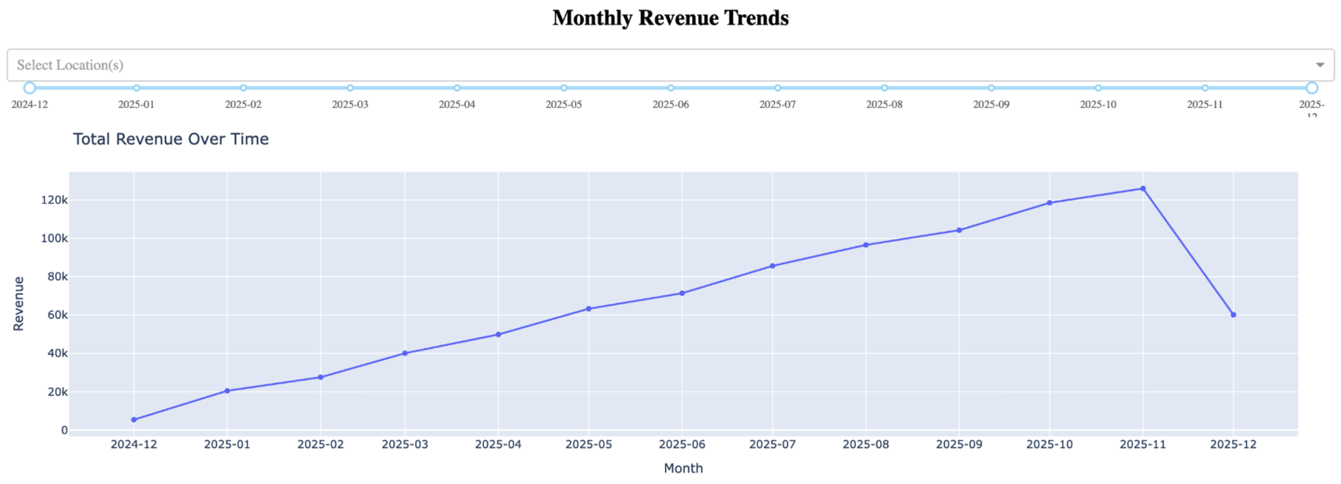


Figure 2: Monthly Revenue Trends (Aggregated)

5.2 Monthly Revenue Trends (Filtered)

This version of the chart shows its interactivity. It is filtered by specific locations and a selected range of months, allowing for granular regional performance comparison.

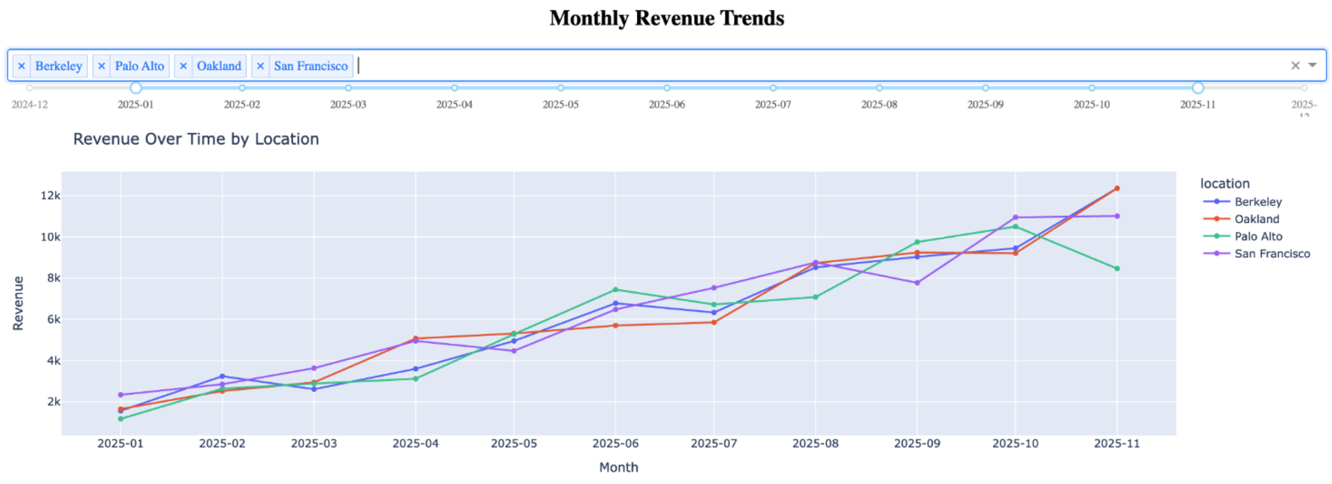


Figure 3: Monthly Revenue Trends (Filtered by Location)

5.3 Total Attendance by Class Type

This bar chart displays Class Popularity by Attendance volume. It helps identify which class types (e.g., HIIT Blast, Pilates Core) are driving the most foot traffic to the gyms.

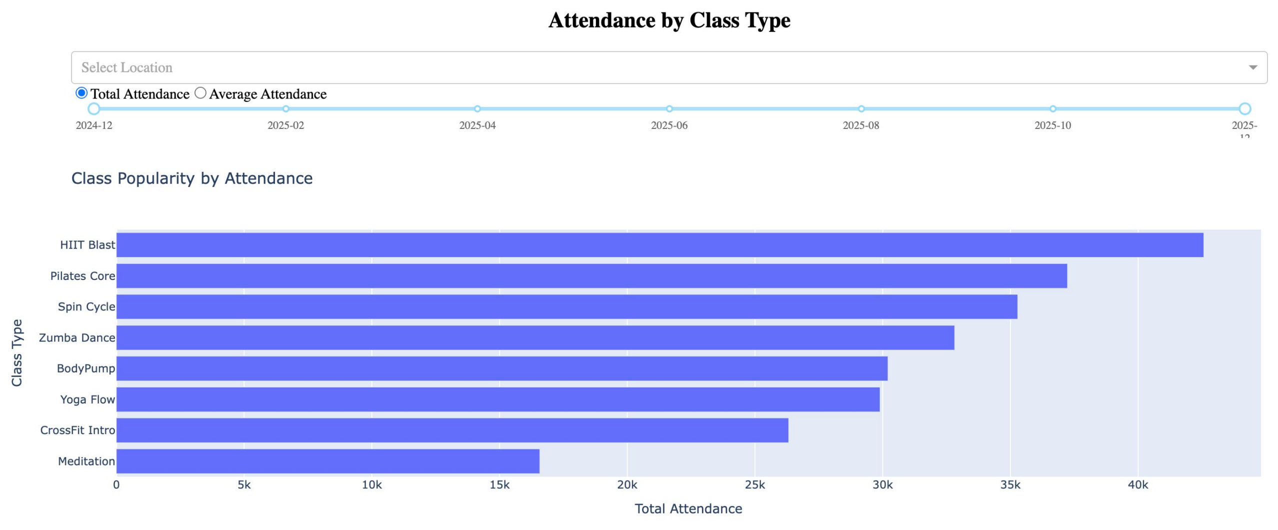


Figure 4: Total Attendance by Class Type

5.4 Average Attendance by Class Type

By toggling the metric to "Average Attendance", management can assess efficiency. This view normalizes the data to show attendance per individual session, highlighting classes that are consistently full even if they are offered less frequently.

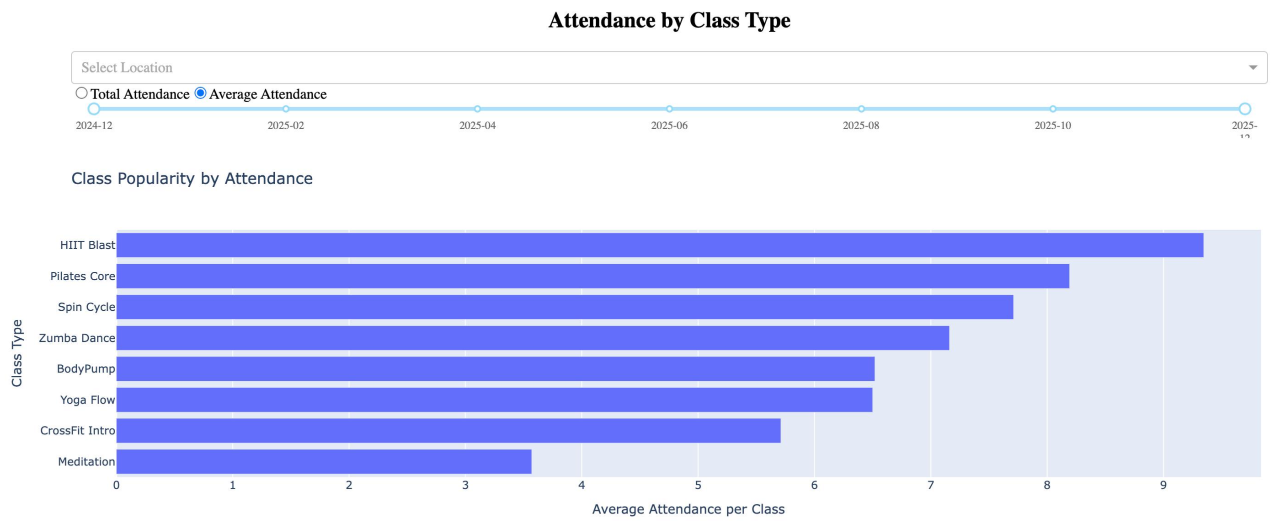


Figure 5: Average Attendance by Class Type

5.5 Attendance by Class Type (Filtered)

This plot shows attendance filtered by a specific location (Fremont) and a specific time range (Q3, 2025). This level of detail aids in local resource planning and schedule optimization for specific branches.

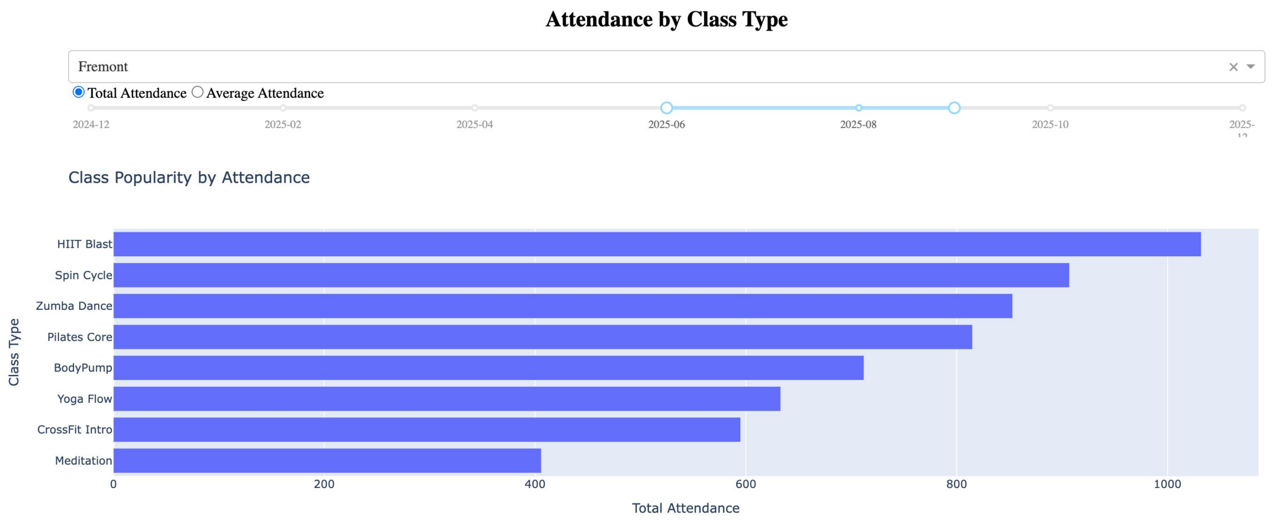


Figure 6: Attendance Filtered by Location (Fremont) and Month

5.6 Seasonality Analysis (Overall Heatmap)

This heatmap visualizes the seasonality in class attendance across all locations. Darker colors indicate higher attendance levels. Distinct patterns, such as "New Year" spikes in January, highlight participation behavior across the network.

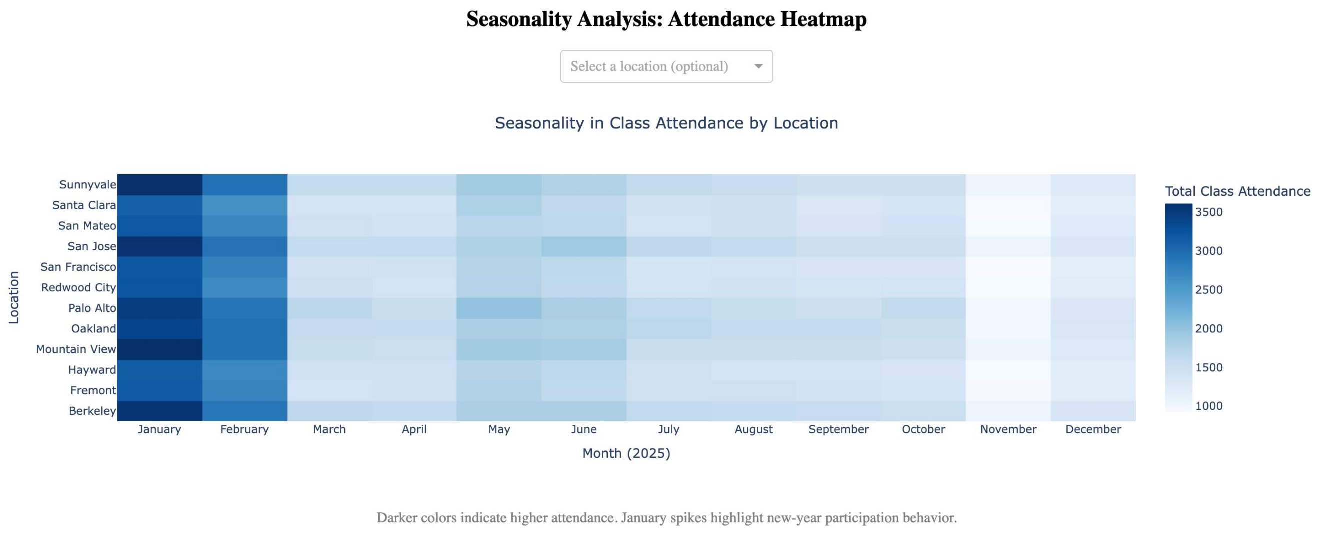


Figure 7: Seasonality Analysis: Attendance Heatmap

5.7 Seasonality Analysis (Filtered)

This heatmap is filtered to show data for a single location (Berkeley). It allows the specific branch manager to analyze their unique seasonal trends independent of the wider network.

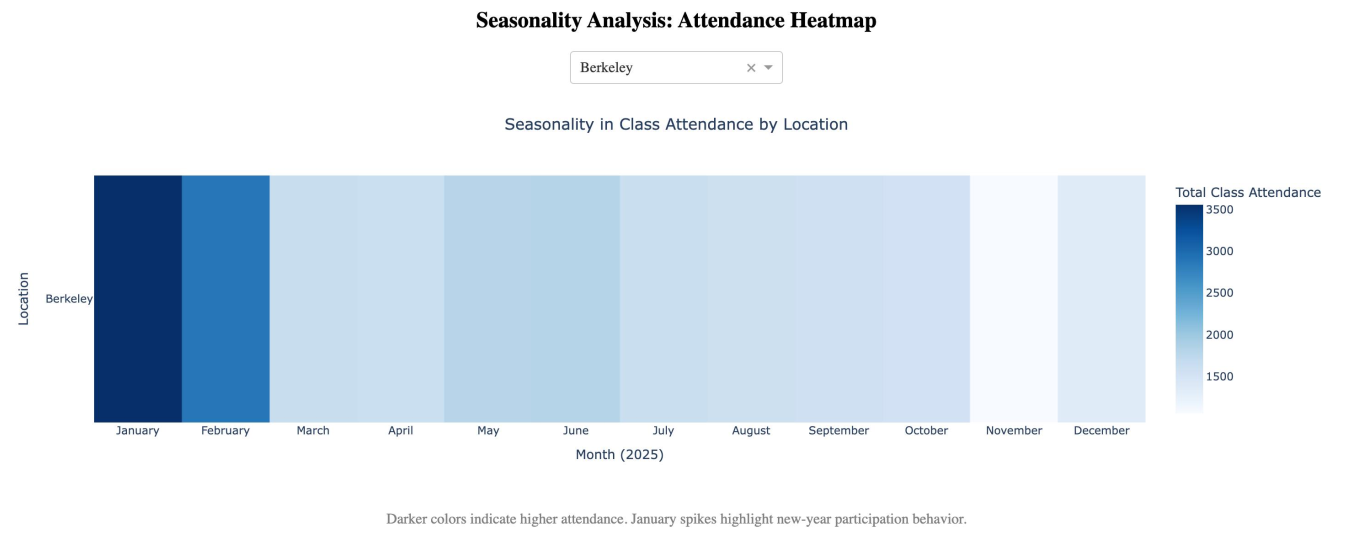

Seasonality Analysis: Attendance Heatmap
Berkeley
Seasonality in Class Attendance by Location
Location
Berkeley
January
February
March
April
May
June
July
August
September
October
November
December
Month (2025)
Total Class Attendance
3500
3000
2500
2000
1500
Darker colors indicate higher attendance. January spikes highlight new-year participation behavior.

Figure 8: Seasonality Analysis Filtered (Berkeley)

6 Informative Analysis

6.1 Monthly Revenue Analysis by Subscription Plan

Objective: The objective of this analytical process is to understand how monthly revenue is distributed across different subscription plans over time. By analyzing revenue at the plan level, this process helps identify which plans contribute most to overall revenue growth and how customer preferences evolve across months.

Data Preparation: This analysis uses data stored in an OLAP-style schema. The main data source is the `Fact_Monthly_Revenue` table, joined with `Dim_Plan` and `Dim_Date`. Using SQL, data is filtered for the period Dec 2024 - Dec 2025. Missing months are explicitly added to ensure a consistent time axis.

Analytical Workflow:

1. Query monthly revenue data by joining fact and dimension tables.
2. Aggregate total revenue by month and subscription plan.
3. Standardize the monthly timeline to ensure continuity.
4. Prepare the data for comparison across plans.

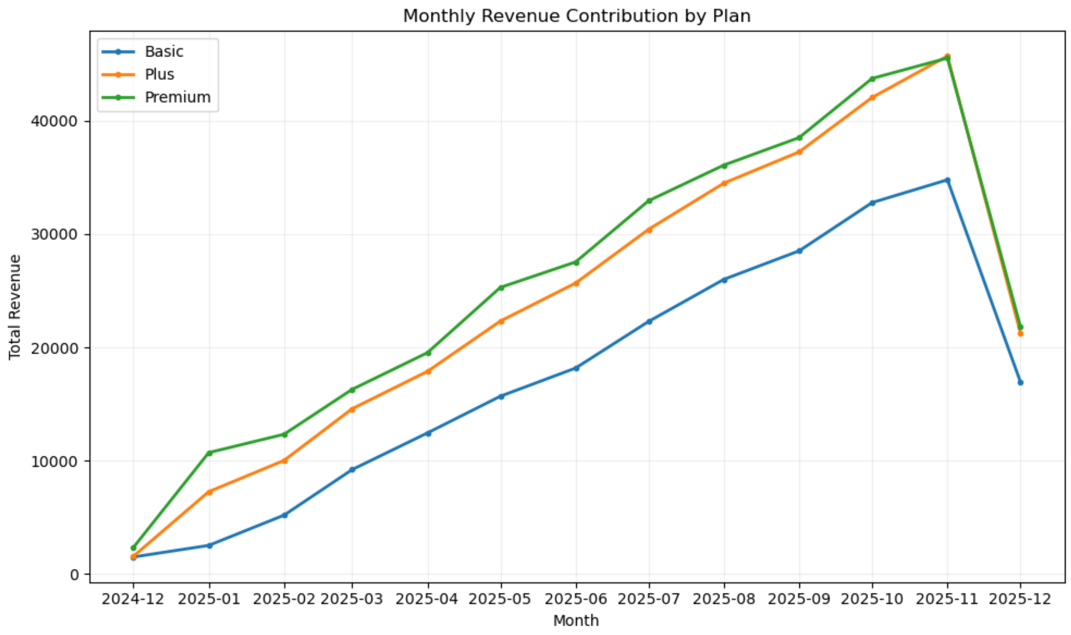


Figure 9: Monthly Revenue Contribution by Plan

6.2 Daily Utilization Rate Analysis and Forecasting

Objective: The objective is to analyze daily utilization rates over time and forecast short-term future utilization trends. Utilization rate reflects how efficiently resources (classes/facilities) are being used.

Data Preparation: Based on `Fact_Class_Attendance`. Missing dates are explicitly added to create a continuous daily time series, and missing utilization values are filled using time-based linear interpolation.

Analytical Workflow:

1. Retrieve daily utilization data using SQL joins.
2. Aggregate utilization rates at the daily level.
3. Enforce continuity in the time dimension and apply interpolation.
4. Train a Seasonal ARIMA (SARIMA) model on the cleaned daily time series.
5. Generate both in-sample fitted values and short-term forecasts.

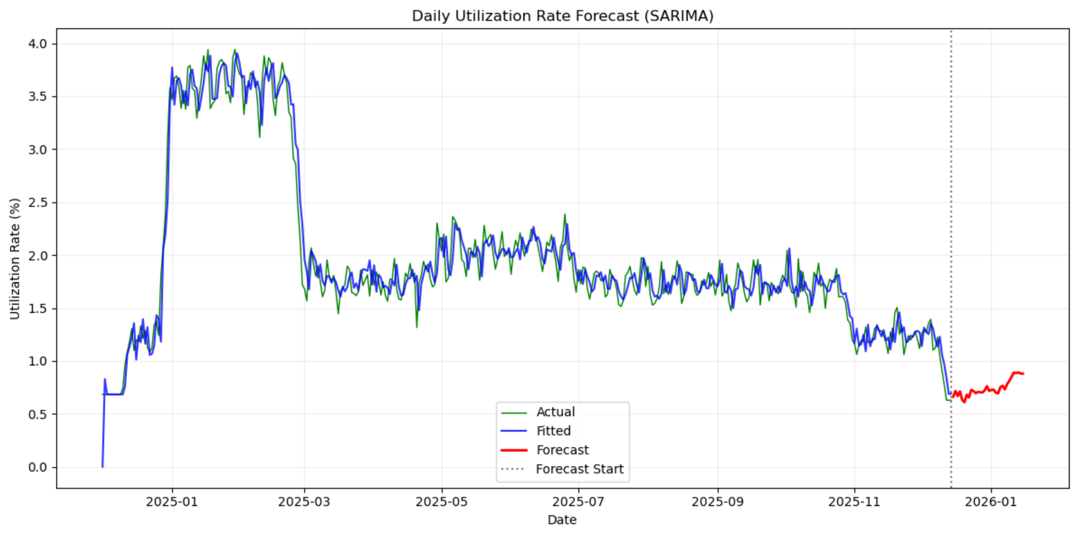


Figure 10: Daily Utilization Rate Forecast (SARIMA)

6.3 Daily Revenue Trend Analysis and Forecasting

Objective: To analyze daily revenue patterns and forecast short-term future revenue trends. Daily revenue is a core business metric that reflects customer activity and overall financial performance.

Data Preparation: Based on `Fact_Monthly_Revenue`. Revenue values are aggregated by day. Missing dates are added, and missing revenue values are filled using time-based linear interpolation.

Analytical Workflow:

1. Retrieve daily revenue data using SQL joins.
2. Aggregate revenue at the daily level.
3. Enforce continuity and apply time-based interpolation.
4. Train a Seasonal ARIMA (SARIMA) model to capture trend and weekly seasonality.
5. Generate in-sample fitted values and short-term revenue forecasts.

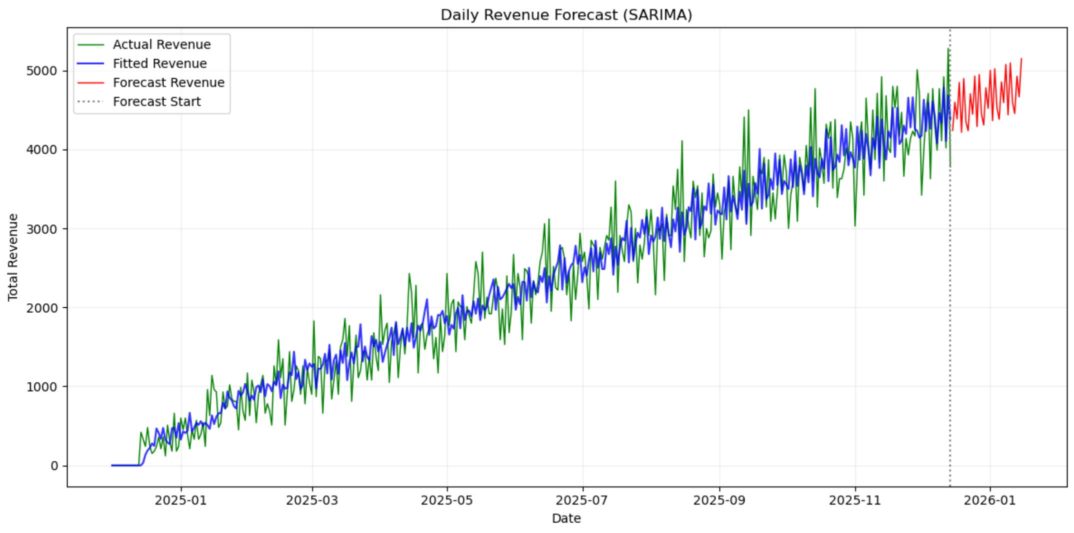


Figure 11: Daily Revenue Forecast (SARIMA)

A Appendix: SQL and Code Implementation

A.1 SQL CREATE Statements (OLTP & OLAP)

```sql
-- 1. Independent Tables
CREATE TABLE locations (
    location_id VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50), state VARCHAR(50), capacity INT
);
CREATE TABLE class_types (
    class_type_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(50), category VARCHAR(20)
);
CREATE TABLE membership_plans (
    plan_id VARCHAR(50) PRIMARY KEY,
    plan_name VARCHAR(50), price DECIMAL(10, 2), tier VARCHAR(20)
);
CREATE TABLE members (
    member_id VARCHAR(50) PRIMARY KEY,
    first_name VARCHAR(50), last_name VARCHAR(50), join_date DATE
);

-- 2. Dependent Tables (Level 1)
CREATE TABLE trainers (
    trainer_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100),
    location_id VARCHAR(50) FOREIGN KEY REFERENCES locations(location_id)
);
CREATE TABLE subscriptions (
    subscription_id VARCHAR(50) PRIMARY KEY,
    member_id VARCHAR(50) FOREIGN KEY REFERENCES members(member_id),
    plan_id VARCHAR(50) FOREIGN KEY REFERENCES membership_plans(plan_id),
    start_date DATE, status VARCHAR(20)
);
CREATE TABLE checkins (
    checkin_id VARCHAR(50) PRIMARY KEY,
    member_id VARCHAR(50) FOREIGN KEY REFERENCES members(member_id),
    location_id VARCHAR(50) FOREIGN KEY REFERENCES locations(location_id),
    checkin_time DATETIME
);

-- 3. Dependent Tables (Level 2 & 3)
CREATE TABLE class_sessions (
    session_id VARCHAR(50) PRIMARY KEY,
    class_type_id VARCHAR(50) FOREIGN KEY REFERENCES class_types(class_type_id),
    location_id VARCHAR(50) FOREIGN KEY REFERENCES locations(location_id),
    trainer_id VARCHAR(50) FOREIGN KEY REFERENCES trainers(trainer_id),
    start_time DATETIME, duration_minutes INT
);
CREATE TABLE bookings (
    booking_id VARCHAR(50) PRIMARY KEY,
    member_id VARCHAR(50) FOREIGN KEY REFERENCES members(member_id),
    session_id VARCHAR(50) FOREIGN KEY REFERENCES class_sessions(session_id),
    booking_date DATETIME, status VARCHAR(20)
);
CREATE TABLE payments (
    payment_id VARCHAR(50) PRIMARY KEY,
    subscription_id VARCHAR(50) FOREIGN KEY REFERENCES subscriptions(subscription_id),
    member_id VARCHAR(50) FOREIGN KEY REFERENCES members(member_id),
    amount DECIMAL(10, 2), payment_date DATETIME, status VARCHAR(20)
);
```

Listing 1: OLTP Schema Creation

```sql
-- Dimension Tables
CREATE TABLE Dim_Location (
    location_key INT IDENTITY(1,1) PRIMARY KEY,
    location_id VARCHAR(50), city VARCHAR(50),
    state VARCHAR(50), capacity INT
);
CREATE TABLE Dim_Plan (
    plan_key INT IDENTITY(1,1) PRIMARY KEY,
    plan_id VARCHAR(50), plan_name VARCHAR(50),
    tier VARCHAR(20), price DECIMAL(10, 2)
);
CREATE TABLE Dim_Class_Type (
    class_type_key INT IDENTITY(1,1) PRIMARY KEY,
    class_type_id VARCHAR(50), name VARCHAR(50), category VARCHAR(20)
);
CREATE TABLE Dim_Date (
    date_key INT PRIMARY KEY, -- YYYYMMDD
    full_date DATE, year INT, month INT,
    day_name VARCHAR(20), is_holiday BIT
);

-- Fact Tables
CREATE TABLE Fact_Monthly_Revenue (
    revenue_key INT IDENTITY(1,1) PRIMARY KEY,
    location_key INT FOREIGN KEY REFERENCES Dim_Location(location_key),
    plan_key INT FOREIGN KEY REFERENCES Dim_Plan(plan_key),
    date_key INT FOREIGN KEY REFERENCES Dim_Date(date_key),
    total_revenue DECIMAL(15, 2),
    transaction_count INT,
    active_members INT
);
CREATE TABLE Fact_Class_Attendance (
    attendance_key INT IDENTITY(1,1) PRIMARY KEY,
    location_key INT FOREIGN KEY REFERENCES Dim_Location(location_key),
    class_type_key INT FOREIGN KEY REFERENCES Dim_Class_Type(class_type_key),
    date_key INT FOREIGN KEY REFERENCES Dim_Date(date_key),
    total_bookings INT,
    attendance_count INT,
    utilization_rate DECIMAL(5, 4)
);
```

Listing 2: OLAP Schema Creation

A.2 Python Code for Synthetic Data Generation

```python
import pandas as pd
import random
from faker import Faker
from sqlalchemy import create_engine
import urllib
import uuid
from datetime import datetime, timedelta

# Configuration (Masked Credentials)
server = 'sql-fitflow-team8-2025.database.windows.net'
database = 'fitflow_oltp'
username = 'fitflowadmin'
password = '*****'

# Connection setup
params = urllib.parse.quote_plus(
    f'Driver={{ODBC Driver 18 for SQL Server}};'
    f'Server=tcp:{server},1433;Database={database};'
    f'Uid={username};Pwd={password};Encrypt=yes;TrustServerCertificate=no;'
)
engine = create_engine(f'mssql+pyodbc:///?odbc_connect={params}')
fake = Faker()

# Data Generation Logic (Simplified for brevity)
# 1. Locations
locations = [{'location_id': f'LOC-{i+1:03d}', 'city': city}
             for i, city in enumerate(['SF', 'Berkeley', 'Oakland', 'SJ', 'Palo Alto'])]
pd.DataFrame(locations).to_sql('locations', engine, if_exists='append', index=False)

# ... (Additional generation logic for all entities) ...
print(Data Generation Complete.)
```

Listing 3: Synthetic Data Generation

A.3 Python Code for ETL Process

```
import pandas as pd
from sqlalchemy import create_engine, text
from datetime import datetime, timedelta
import urllib

# Configuration
server = 'sql-fitflow-team8-2025.database.windows.net'
username = 'fitflowadmin'
password = '*****'

def get_engine(db_name):
    params = urllib.parse.quote_plus(
        f'Driver={{ODBC Driver 18 for SQL Server}};Server=tcp:{server},1433;'
        f'Database={db_name};Uid={username};Pwd={password};Encrypt=yes;'
    )
    return create_engine(f'mssql+pyodbc:///?odbc_connect={params}')

oltp_engine = get_engine('fitflow_oltp')
olap_engine = get_engine('fitflow_olap')

# --- STEP 1: EXTRACT (Incremental & Lock-Free) ---
last_run = datetime.now() - timedelta(days=90)
df_payments = pd.read_sql(text(SELECT * FROM payments WHERE payment_date > :d),
                          oltp_engine, params={d: last_run})
df_subs = pd.read_sql(SELECT * FROM subscriptions, oltp_engine)
df_plans = pd.read_sql(SELECT * FROM membership_plans, oltp_engine)
```

```python

# --- STEP 2: TRANSFORM (Staging Area) ---
# Joins are performed in-memory (Pandas)
df_subs_clean = df_subs[['subscription_id', 'plan_id']]
df_rev = df_payments.merge(df_subs_clean, on='subscription_id', how='left')
df_rev = df_rev.merge(df_plans, on='plan_id', how='left')

df_rev['date_key'] = pd.to_datetime(df_rev['payment_date']).dt.strftime('%Y%m%d').astype(int)

fact_revenue = df_rev.groupby(['location_id', 'plan_id', 'date_key']).agg({
    'amount': 'sum',
    'payment_id': 'count',
    'member_id': 'nunique'
}).rename(columns={'amount': 'total_revenue'}).reset_index()

# --- STEP 3: LOAD (OLAP) ---
fact_revenue.to_sql('Fact_Monthly_Revenue', olap_engine, if_exists='append', index=False)
print(ETL Completed Successfully.)
```

Listing 4: ETL Process

A.4 Dashboard Implementation Code (Dash)

```python
# =====================================================
# IMPORTS
# =====================================================
import pandas as pd
import urllib
from sqlalchemy import create_engine

import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px

# =====================================================
# DB CONNECTION
# =====================================================
server = 'sql-fitflow-team8-2025.database.windows.net'
database = 'fitflow_olap'
username = 'fitflowadmin'
password = '*****'

params = urllib.parse.quote_plus(
    fDriver={{ODBC Driver 18 for SQL Server}};
    fServer=tcp:{server},1433;
    fDatabase={database};
    fUid={username};Pwd={password};
    fEncrypt=yes;TrustServerCertificate=no;Connection Timeout=30;
)

engine = create_engine(fmssql+pyodbc:///?odbc_connect={params})

# =====================================================
# LOAD DATA  VISUALIZATION 1 (REVENUE)
# =====================================================
query_revenue =
SELECT
    d.year,
    d.month,
    CONCAT(d.year, '-', RIGHT('00' + CAST(d.month AS VARCHAR), 2)) AS year_month,
    l.city AS location,
    SUM(f.total_revenue) AS revenue
FROM Fact_Monthly_Revenue f
JOIN Dim_Date d ON f.date_key = d.date_key
JOIN Dim_Location l ON f.location_key = l.location_key
GROUP BY d.year, d.month, l.city
ORDER BY d.year, d.month;


df_rev = pd.read_sql(query_revenue, engine)
months_rev = sorted(df_rev[year_month].unique())

# =====================================================
# LOAD DATA  VISUALIZATION 2 (CLASS ATTENDANCE)
# =====================================================
query_attendance =
SELECT
    d.year,
    d.month,
    CONCAT(d.year, '-', RIGHT('00' + CAST(d.month AS VARCHAR), 2)) AS year_month,
    l.city AS location,
    c.name AS class_type,
    SUM(f.attendance_count) AS total_attendance,
    AVG(f.attendance_count) AS avg_attendance
FROM Fact_Class_Attendance f
JOIN Dim_Date d ON f.date_key = d.date_key
JOIN Dim_Location l ON f.location_key = l.location_key
JOIN Dim_Class_Type c ON f.class_type_key = c.class_type_key
GROUP BY d.year, d.month, l.city, c.name
ORDER BY d.year, d.month;


df_att = pd.read_sql(query_attendance, engine)
months_att = sorted(df_att[year_month].unique())

# =====================================================
# LOAD DATA  VISUALIZATION 3 (HEATMAP)
# =====================================================
query_heatmap =
SELECT
    DATENAME(month, DATEFROMPARTS(d.year, d.month, 1)) AS month_name,
    l.city AS location,
    SUM(f.attendance_count) AS total_attendance
FROM Fact_Class_Attendance f
JOIN Dim_Date d ON f.date_key = d.date_key
JOIN Dim_Location l ON f.location_key = l.location_key
GROUP BY d.year, d.month, l.city


df_heat = pd.read_sql(query_heatmap, engine)

month_order = [
    January,February,March,April,May,June,
    July,August,September,October,November,December
]

df_heat[month_name] = pd.Categorical(
    df_heat[month_name],
    categories=month_order,
    ordered=True
)

# ====================================================
# DASH APP (ONE APP ONLY)
# ====================================================
app = dash.Dash(__name__)

app.layout = html.Div([

    html.H1(FitFlow Management Dashboard, style={textAlign: center}),

    # ================================================
    # VISUALIZATION 1  REVENUE
    # ================================================
    html.H2(Monthly Revenue Trends),

    dcc.Dropdown(
        id=rev-location-filter,
        options=[{label: i, value: i} for i in sorted(df_rev[location].unique())],
        multi=True,
        placeholder=Select Location(s)
    ),

    dcc.RangeSlider(
        id=rev-time-filter,
        min=0,
        max=len(months_rev) - 1,
        value=[0, len(months_rev) - 1],
        marks={i: months_rev[i] for i in range(len(months_rev))},
        step=1
    ),

    dcc.Graph(id=revenue-line),

    html.Hr(),

    # ================================================
    # VISUALIZATION 2  CLASS ATTENDANCE
    # ================================================
    html.H2(Attendance by Class Type),

    dcc.Dropdown(
        id=att-location-filter,
        options=[{label: i, value: i} for i in sorted(df_att[location].unique())],
        placeholder=Select Location,
        clearable=True
    ),

    dcc.RadioItems(
        id=metric-toggle,
        options=[
            {label: Total Attendance, value: total},
            {label: Average Attendance, value: average}
        ],
        value=total,
        inline=True
    ),

    dcc.RangeSlider(
        id=att-time-filter,
        min=0,
        max=len(months_att) - 1,
        value=[0, len(months_att) - 1],
        marks={i: months_att[i] for i in range(0, len(months_att), max(1, len(months_att)//6))},
        step=1
    ),

    dcc.Graph(id=attendance-bar),

    html.Hr(),

    # =================================================
    # VISUALIZATION 3  HEATMAP
    # =================================================
    html.H2(Seasonality in Class Attendance),

    dcc.Dropdown(
        id=heat-location-filter,
        options=[{label: loc, value: loc} for loc in sorted(df_heat[location].unique())],
        placeholder=Select Location (optional),
        clearable=True,
        style={width: 40%}
    ),

    dcc.Graph(id=attendance-heatmap)
])

# =====================================================
# CALLBACKS
# =====================================================

@app.callback(
    Output(revenue-line, figure),
    Input(rev-location-filter, value),
    Input(rev-time-filter, value)
)
def update_revenue(locations, time_range):
    dff = df_rev.copy()
    dff = dff[dff[year_month].between(
        months_rev[time_range[0]], months_rev[time_range[1]]
    )]

    if locations:
        fig = px.line(dff[dff[location].isin(locations)],
                      x=year_month, y=revenue,
                      color=location, markers=True)
    else:
        dff = dff.groupby(year_month, as_index=False)[revenue].sum()
        fig = px.line(dff, x=year_month, y=revenue, markers=True)

    fig.update_xaxes(categoryorder=array, categoryarray=months_rev)
    fig.update_layout(xaxis_title=Month, yaxis_title=Revenue)
    return fig


@app.callback(
    Output(attendance-bar, figure),
    Input(att-location-filter, value),
    Input(metric-toggle, value),
    Input(att-time-filter, value)
)
def update_attendance(location, metric, time_range):
    dff = df_att.copy()
    dff = dff[dff[year_month].between(
        months_att[time_range[0]], months_att[time_range[1]]
    )]

    if location:
        dff = dff[dff[location] == location]

    if metric == total:
        dff = dff.groupby(class_type, as_index=False)[total_attendance].sum()
        y = total_attendance
    else:
        dff = dff.groupby(class_type, as_index=False)[avg_attendance].mean()
        y = avg_attendance

    fig = px.bar(dff, x=y, y=class_type, orientation=h)
    return fig


@app.callback(
    Output(attendance-heatmap, figure),
    Input(heat-location-filter, value)
)
def update_heatmap(location):
    dff = df_heat.copy()
    if location:
        dff = dff[dff[location] == location]

    fig = px.density_heatmap(
        dff,
        x=month_name,
        y=location,
        z=total_attendance,
        color_continuous_scale=Blues
    )

    fig.update_coloraxes(colorbar_title=Total Class Attendance)
    return fig


# ====================================================
# RUN
# ====================================================
if __name__ == __main__:
    app.run(debug=True)
```

Listing 5: Dash Application Code

A.5 Analytical Processes Code

```
# ====================================================
# Informative Analysis 1: Monthly Revenue by Plan
# ====================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX

jdbc_url = (
    jdbc:sqlserver://sql-fitflow-team8-2025.database.windows.net:1433;
    database=fitflow_olap;
    encrypt=true;
    trustServerCertificate=false;
)

connection_properties = {
    user: fitflowadmin,
    password: *****,
    driver: com.microsoft.sqlserver.jdbc.SQLServerDriver
}

df_mr.createOrReplaceTempView(Fact_Monthly_Revenue)
df_ca.createOrReplaceTempView(Fact_Class_Attendance)
df_date.createOrReplaceTempView(Dim_Date)
df_plan.createOrReplaceTempView(Dim_Plan)

spark_df = spark.sql(
SELECT
    date_format(d.full_date, 'yyyy-MM') AS Date,
    f.total_revenue,
    p.plan_name
FROM Fact_Monthly_Revenue f
JOIN Dim_Plan p
    ON f.plan_key = p.plan_key
RIGHT JOIN Dim_Date d
    ON f.date_key = d.date_key
WHERE d.full_date >= '2024-12-01'
  AND d.full_date <= '2025-12-31'
)

df1 = spark_df.toPandas()
df1 = df1.dropna() # Remove NaNs

total_rev_per_mon = df1.groupby(['Date', 'plan_name'])['total_revenue'].sum().reset_index()
total_rev_per_mon['total_revenue'] = round(total_rev_per_mon['total_revenue'], 2)

total_rev_per_mon[Date] = pd.to_datetime(total_rev_per_mon[Date])
full_months = pd.date_range(start=2024-12-01, end=2025-12-01, freq=MS)

```

```
plans = total_rev_per_mon[plan_name].unique()
full_df = (
    total_rev_per_mon
    .set_index([Date, plan_name])
    .unstack(plan_name)
    .reindex(full_months)
    .stack(plan_name)
    .reset_index()
)
full_df.columns = [Date, plan_name, total_revenue]

plt.figure(figsize=(10,6))
for plan in plans:
    tmp = full_df[full_df[plan_name] == plan]
    plt.plot(
        tmp[Date],
        tmp[total_revenue],
        marker=.,
        linewidth=2,
        label=plan
    )

plt.xticks(full_months, [d.strftime(%Y-%m) for d in full_months])
plt.xlabel(Month)
plt.ylabel(Total Revenue)
plt.title(Monthly Revenue Contribution by Plan)
plt.legend()
plt.grid(alpha=0.2)
plt.tight_layout()
plt.show()

# =====================================================
# Informative Analysis 2: Daily Utilization Forecast
# =====================================================
spark_df2 = spark.sql(
SELECT
    d.full_date AS Date,
    f.utilization_rate
FROM Fact_Class_Attendance f
RIGHT JOIN Dim_Date d
    ON f.date_key = d.date_key
WHERE d.full_date >= '2024-12-01'
  AND d.full_date <= '2025-12-14'
)

df2 = spark_df2.toPandas()
df2['utilization_rate'] = df2['utilization_rate'] * 100
df2[Date] = pd.to_datetime(df2[Date])

df2 = df2.groupby(Date, as_index=False).mean()
df2 = df2.set_index(Date).sort_index()

full_dates = pd.date_range(
    start=df2.index.min(),
    end=df2.index.max(),
    freq=D
)
```

```
df2 = df2.reindex(full_dates)
df2[utilization_rate] = df2[utilization_rate].interpolate(method=time)
df2[utilization_rate] = df2[utilization_rate].ffill().bfill().round(4)
df2 = df2.reset_index().rename(columns={index: Date})

# Modeling
df = df2.copy()
df[Date] = pd.to_datetime(df[Date])
df = df.set_index(Date).sort_index()
y = df[utilization_rate]

model = SARIMAX(
    y,
    order=(1,1,1), # ARIMA part
    seasonal_order=(1, 0, 1, 30), # Monthly seasonality
    enforce_stationarity=False,
    enforce_invertibility=False
)
result = model.fit(disp=False)

# In-sample fitted values
fitted = result.fittedvalues

# Forecast to 2026-01-15
forecast_horizon = pd.date_range(start=2025-12-15, end=2026-01-15, freq=D)
forecast = result.get_forecast(steps=len(forecast_horizon))
forecast_mean = forecast.predicted_mean

plt.figure(figsize=(12,6))
# Actual values
plt.plot(y.index, y, label=Actual, color=green, linewidth=1)
# Fitted values
plt.plot(fitted.index, fitted, label=Fitted, color=blue, linestyle=solid, alpha=0.8)
# Forecast values
plt.plot(forecast_horizon, forecast_mean, label=Forecast, color=red, linewidth=2)

plt.axvline(pd.to_datetime(2025-12-14), color=gray, linestyle=:, label=Forecast Start)
plt.title(Daily Utilization Rate Forecast (SARIMA))
plt.xlabel(Date)
plt.ylabel(Utilization Rate (%))
plt.legend()
plt.grid(alpha=0.2)
plt.tight_layout()
plt.show()

# ======================================================
# Informative Analysis 3: Daily Revenue Forecast
# ======================================================
spark_df_rev = spark.sql(
SELECT
    d.full_date AS Date,
    f.total_revenue
FROM Fact_Monthly_Revenue f
RIGHT JOIN Dim_Date d
    ON f.date_key = d.date_key
WHERE d.full_date >= '2024-12-01'
  AND d.full_date <= '2025-12-14'
```

```python
)

df_rev = spark_df_rev.toPandas()

# 1. Convert Date to datetime
df_rev[Date] = pd.to_datetime(df_rev[Date])
# 2. Aggregate revenue by day
df_rev = df_rev.groupby(Date, as_index=False).sum()
# 3. Set Date as index and sort
df_rev = df_rev.set_index(Date).sort_index()
# 4. Create full daily date range
full_dates = pd.date_range(start=df_rev.index.min(), end=df_rev.index.max(), freq=D)
# 5. Reindex to make dates continuous
df_rev = df_rev.reindex(full_dates)
# 6. Time-based linear interpolation
df_rev[total_revenue] = df_rev[total_revenue].interpolate(method=time)
# 7. Handle edge NaNs
df_rev[total_revenue] = df_rev[total_revenue].ffill().bfill()
# 8. Restore Date column
df_rev = df_rev.reset_index().rename(columns={index: Date})

# Modeling
df = df_rev.copy()
df[Date] = pd.to_datetime(df[Date])
df = df.set_index(Date).sort_index()
y = df[total_revenue]

model = SARIMAX(
    y,
    order=(2,0,2),
    seasonal_order=(1,1,1,7),
    enforce_stationarity=False,
    enforce_invertibility=False
)
result = model.fit(disp=False)

fitted = result.fittedvalues
forecast_horizon = pd.date_range(start=2025-12-15, end=2026-01-15, freq=D)
forecast = result.get_forecast(steps=len(forecast_horizon))
forecast_mean = forecast.predicted_mean

plt.figure(figsize=(12,6))
plt.plot(y.index, y, label=Actual Revenue, color=green, linewidth=1)
plt.plot(fitted.index, fitted, label=Fitted Revenue, color=blue, linestyle=solid, alpha=0.8)
plt.plot(forecast_horizon, forecast_mean, label=Forecast Revenue, color=red, linewidth=1)
plt.axvline(pd.to_datetime(2025-12-14), color=gray, linestyle=:, label=Forecast Start)
plt.title(Daily Revenue Forecast (SARIMA))
plt.xlabel(Date)
plt.ylabel(Total Revenue)
plt.legend()
plt.grid(alpha=0.2)
plt.tight_layout()
plt.show()
```

Listing 6: Analytical Processes (Data Prep & SARIMA)

FitFlow Fitness

INDENG 215 Final Project
Team 8

Changxin Shi, Arthur Fang, Da Eun Kim, Eesha Danish, Junqi Huang, Junyu Li

Business Overview

FitFlow Fitness

- Multi-location gym & wellness chain (30+ locations)
- Offers in-person classes, personal training, and digital workouts
- Subscription-based membership model
- Serves busy professionals, students, and fitness enthusiasts

Why This Business?

- High transaction volume
- Strong need for both real-time operations and historical analytics

Customers, Revenue, Data Needs

Customers

- Individual members (Basic, Plus, Premium)
- Corporate memberships
- Trainers & instructors (internal users)

Revenue Streams

- Membership subscriptions
- Class & personal training add-ons
- Retail and digital services

Data Needs

- Operational: check-ins, bookings, billing
- Analytical: revenue trends, utilization, class performance, location comparisons

OLTP vs OLAP: Why Separate?

Transactional (OLTP)

- High-frequency writes
- Real-time consistency (ACID)
- Supports check-ins, bookings, and billing

Analytical (OLAP)

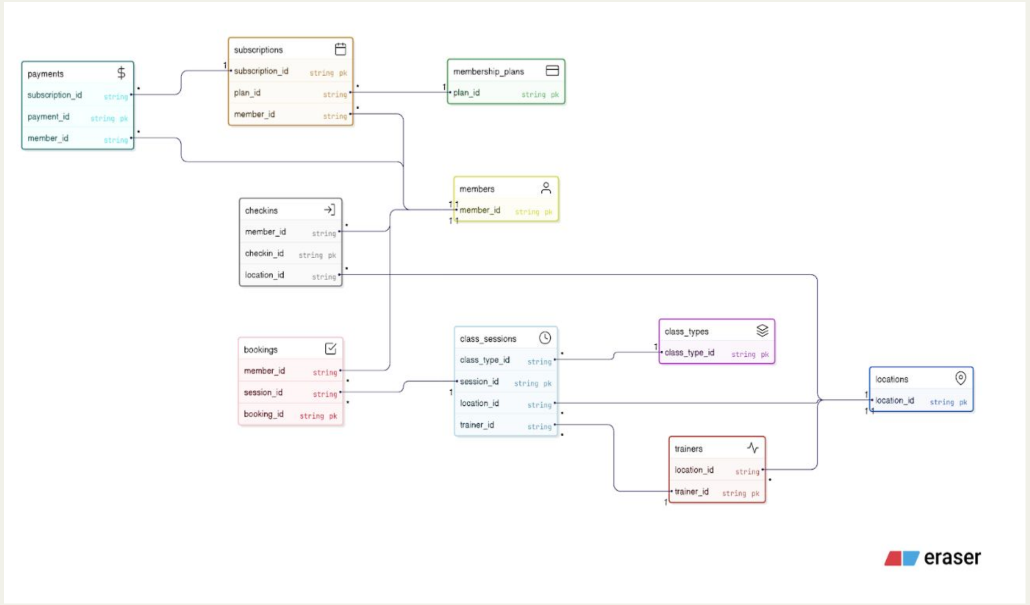
- Read-heavy, historical data
- Aggregated across time, location, and plan
- Used for trend analysis and strategic decisions

Key Reason: Prevents analytical queries from slowing down live operations

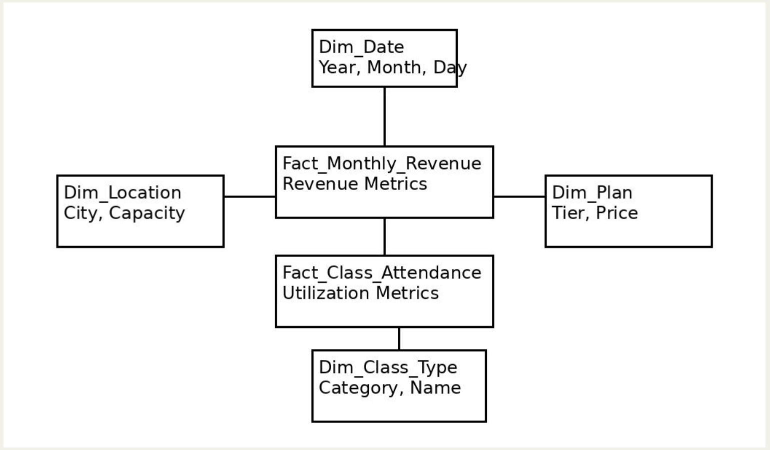
OLTP Schema Design

Core Entities

- Members, Locations, Membership Plans
- Subscriptions, Payments
- Class Sessions, Bookings, Check-ins, Trainers



OLAP Schema Design (Star Schema)



Dimension Tables

- Dim Date (Year, Month, Day, Holiday flag)
- Dim Location (City, State, Capacity)
- Dim Plan (Plan name, tier, price)
- Dim Class Type (Category, name)

Fact Table & Granularity

Fact Monthly Revenue

- Granularity: Location × Plan × Month
- Measures:
    - Total Revenue
    - Transaction Count
    - Active Members

Fact Class Attendance

- Granularity: Location × Class Type × Day
- Measures:
    - Total Bookings
    - Attendance Count
    - Utilization Rate (Attendance / Capacity)

ETL Process Overview + Workflow

Design Principles

- No joins on OLTP system
- Transformations performed in a staging layer
- Incremental extraction logic supported

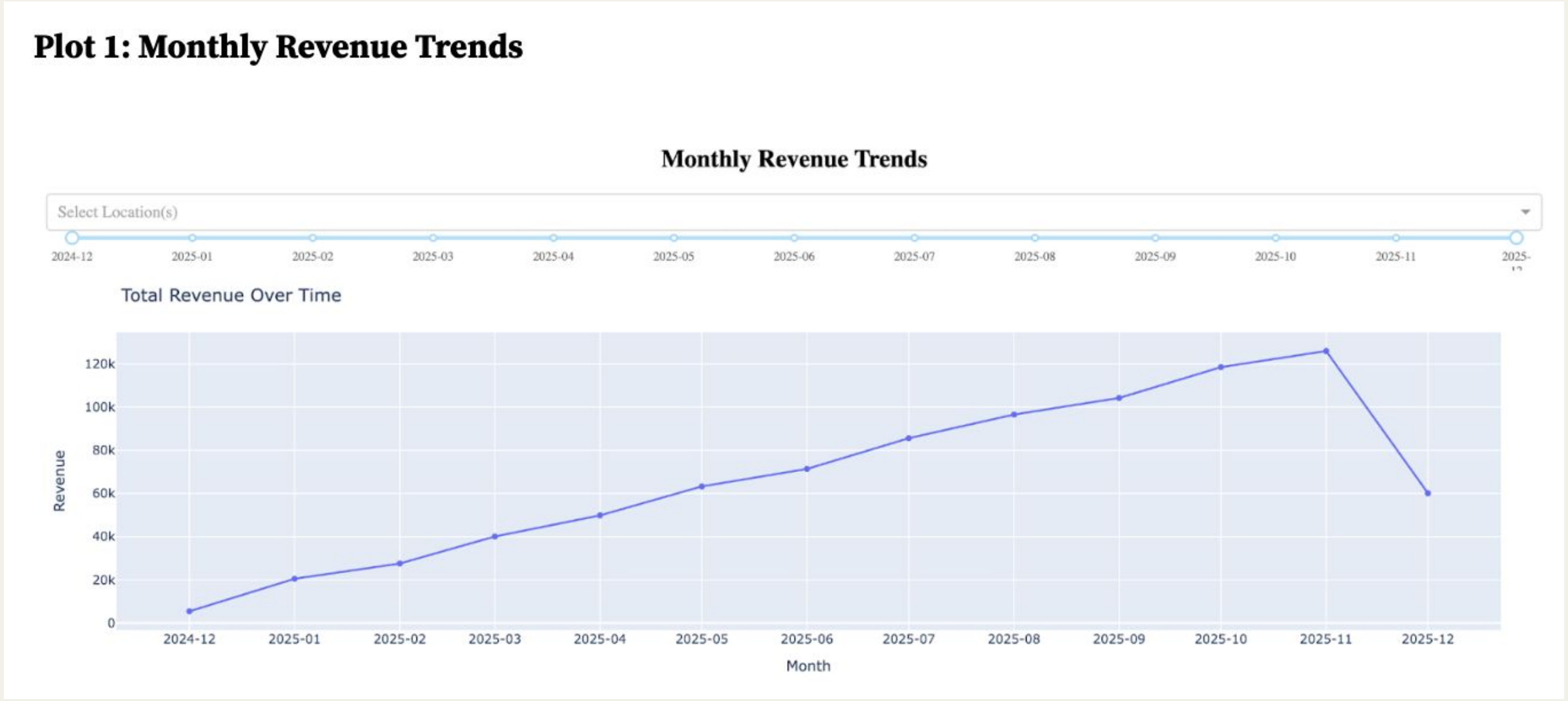
Why This Matters

- Prevents locks on live operational tables
- Preserves OLTP performance

ETL Workflow

1. Extract
   - Raw tables extracted individually
   - No joins or aggregations on OLTP
   - Supports incremental loads
2. Transform
   - Joins and cleaning performed in pandas
   - Invalid records filtered
   - Dates converted into surrogate date keys
3. Load
   - Dimension tables loaded first
   - Fact tables loaded after to ensure referential integrity

Dashboard View 1: Monthly Revenue by Plan

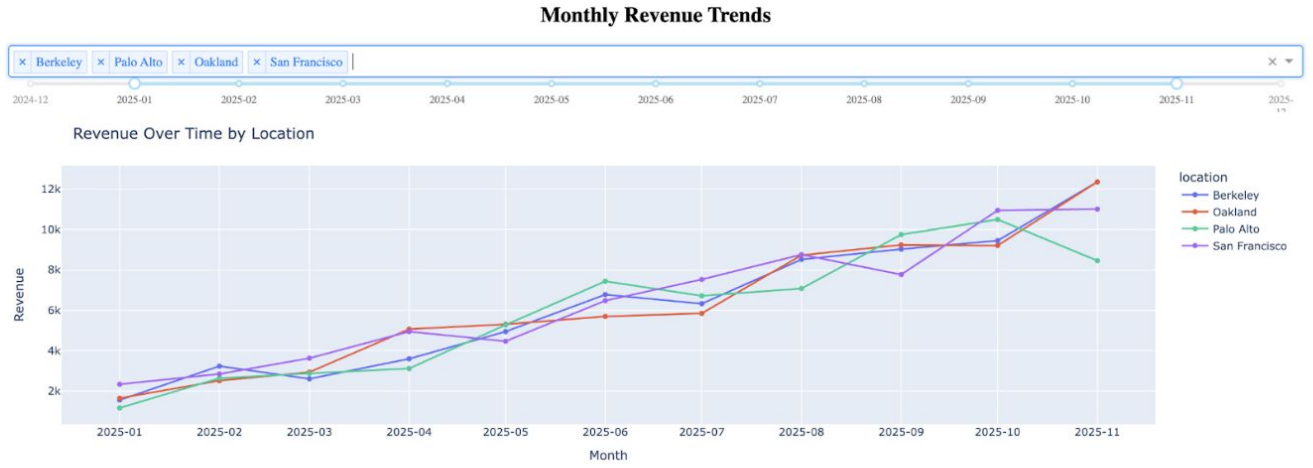


Plot 1: Monthly Revenue Trends

Monthly Revenue Trends

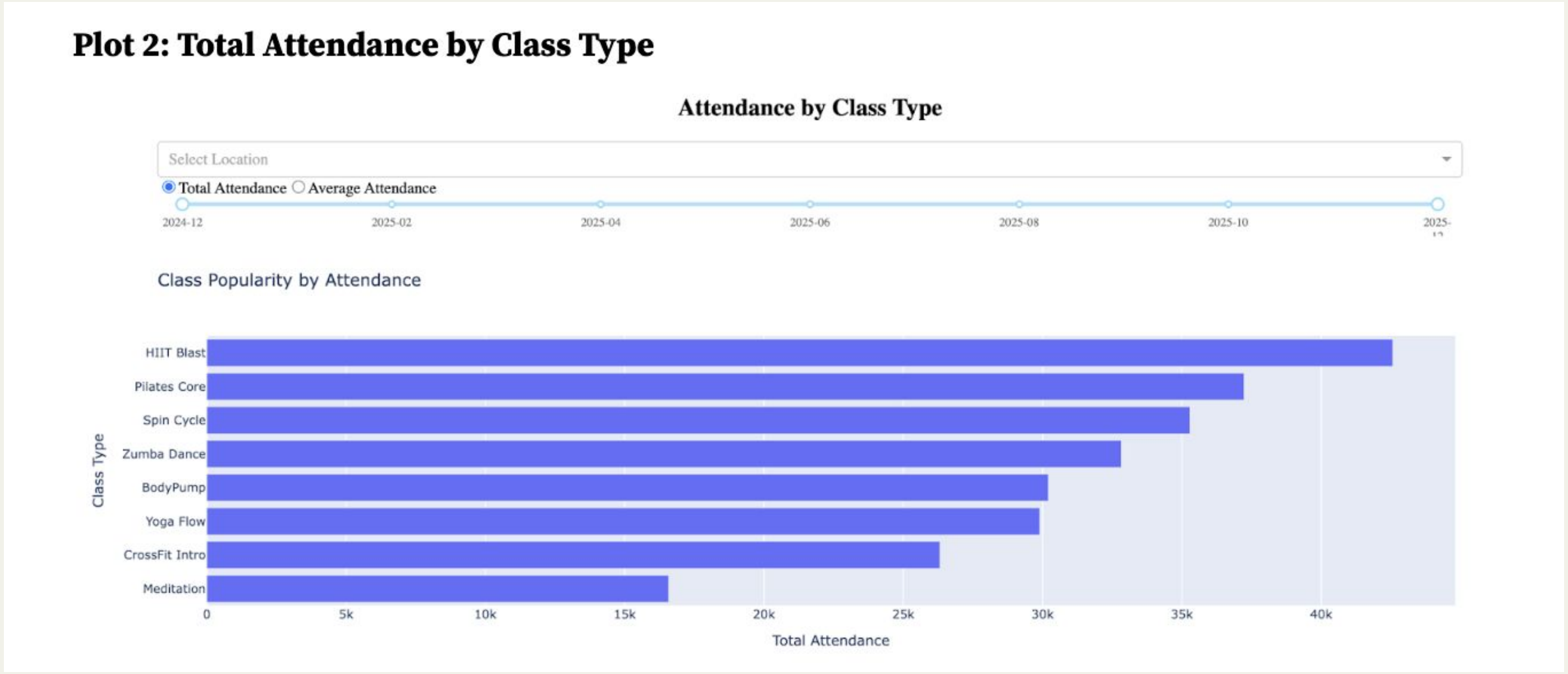
Dashboard View 1: Monthly Revenue by Plan

Plot 1.1: Monthly Revenue Trends *filtered by locations (Berkeley, Palo Alto, Oakland, and San Francisco) and range of months*



Note: When no location is selected, the chart aggregates revenue across all locations to provide an overall business view.

Dashboard View 2: Class Attendance & Utilization


Plot 2: Total Attendance by Class Type

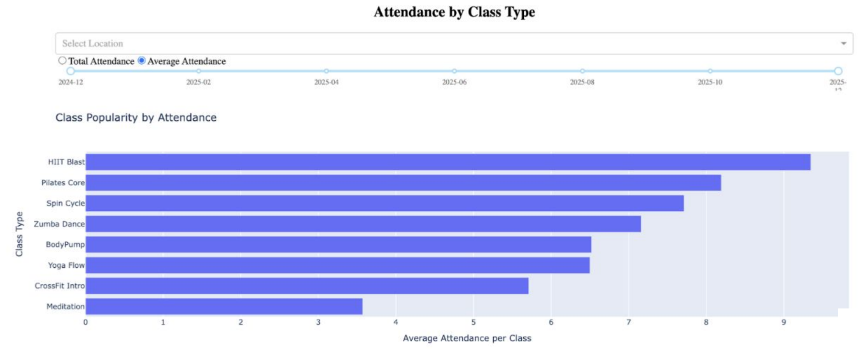
Attendance by Class Type

Select Location

[✓ Total Attendance] Average Attendance

Dashboard View 2: Class Attendance & Utilization

Plot 2.1: Average Attendance by Class Type

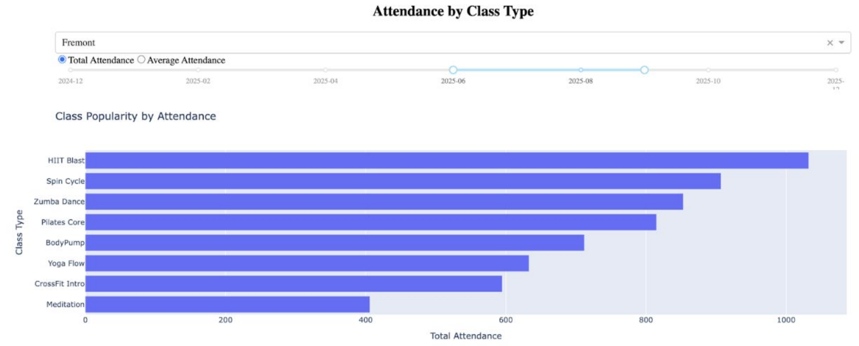


Attendance by Class Type

Select Location

Total Attendance [✓ Average Attendance]

Plot 2.2: Total Attendance by Class Type *filtered by location (Fremont) and months (Q3, 2025)*

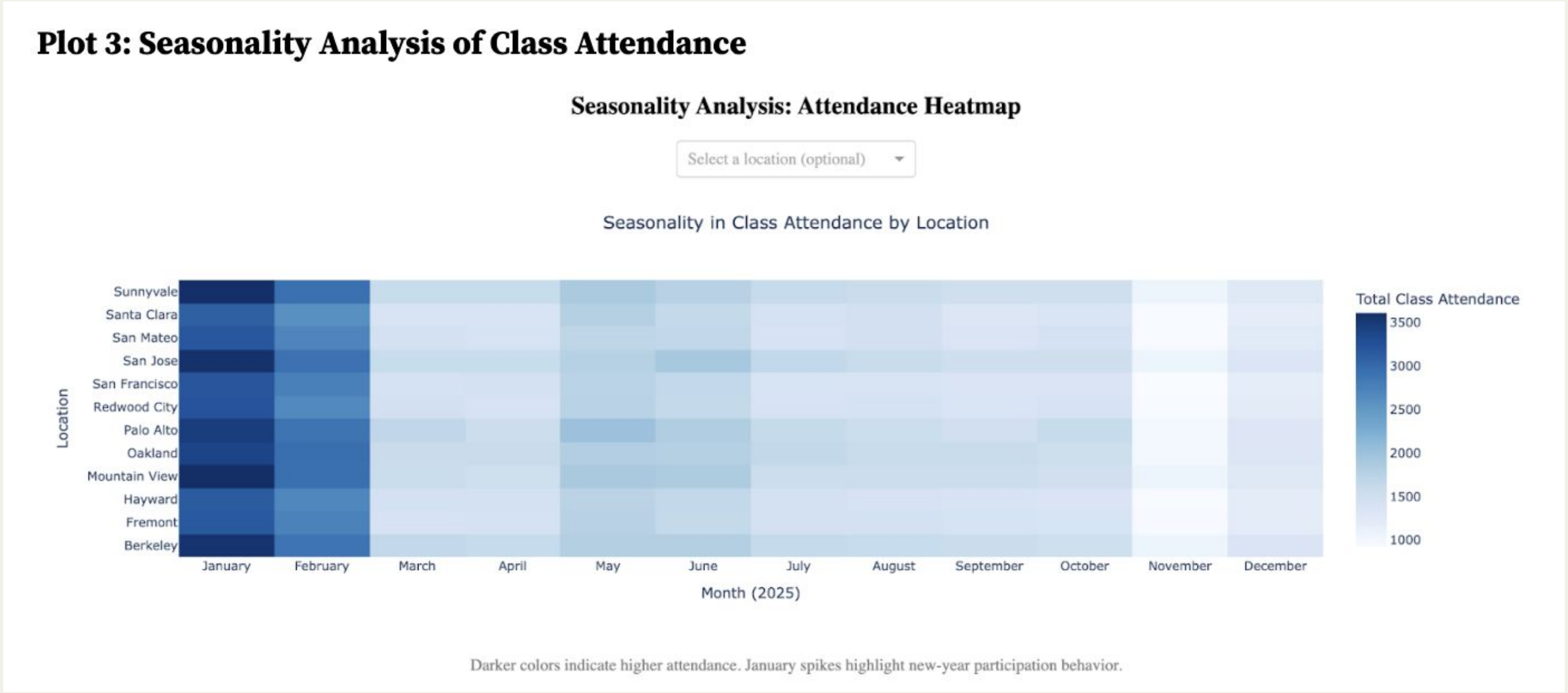


Attendance by Class Type

Fremont

[✓ Total Attendance] Average Attendance

Dashboard View 3: Seasonality Analysis

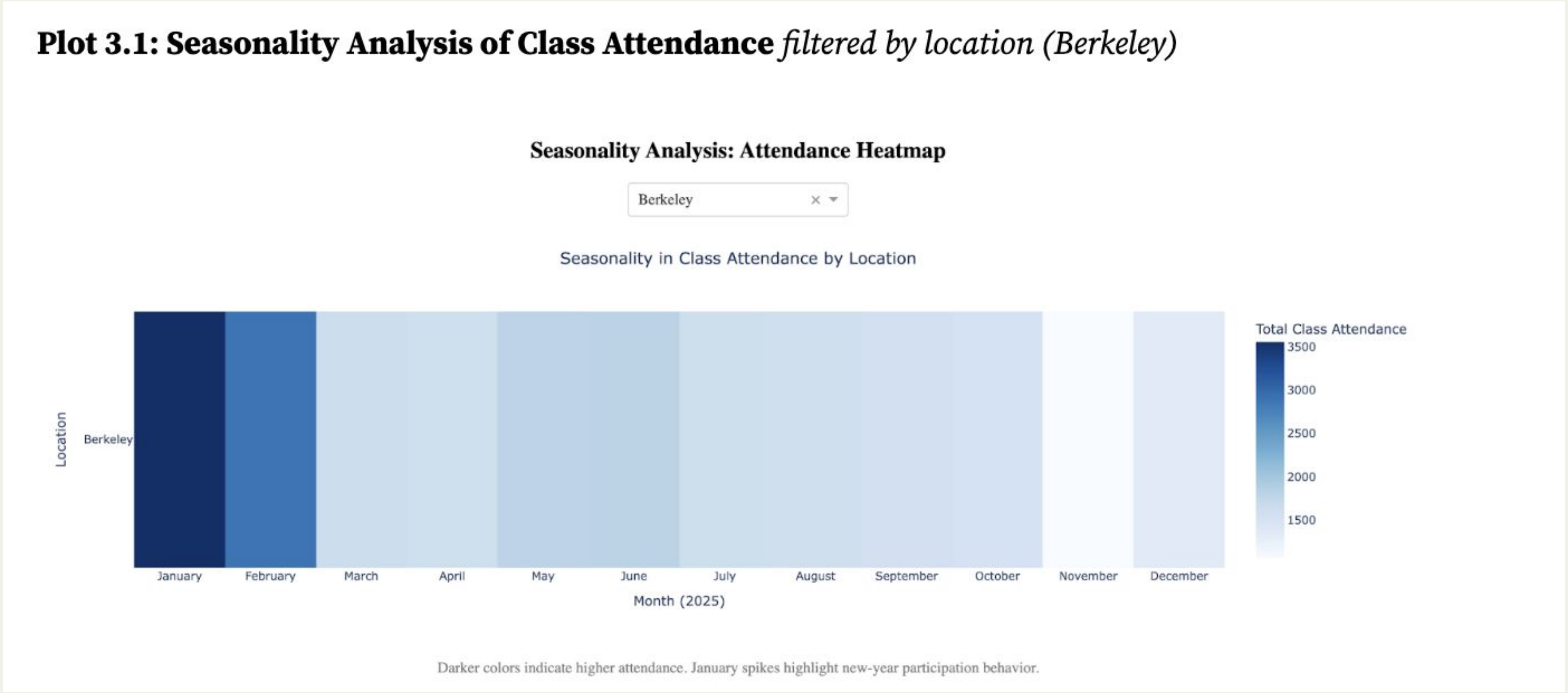


Plot 3: Seasonality Analysis of Class Attendance

Seasonality Analysis: Attendance Heatmap

Darker colors indicate higher attendance. January spikes highlight new-year participation behavior.

Dashboard View 3: Seasonality Analysis



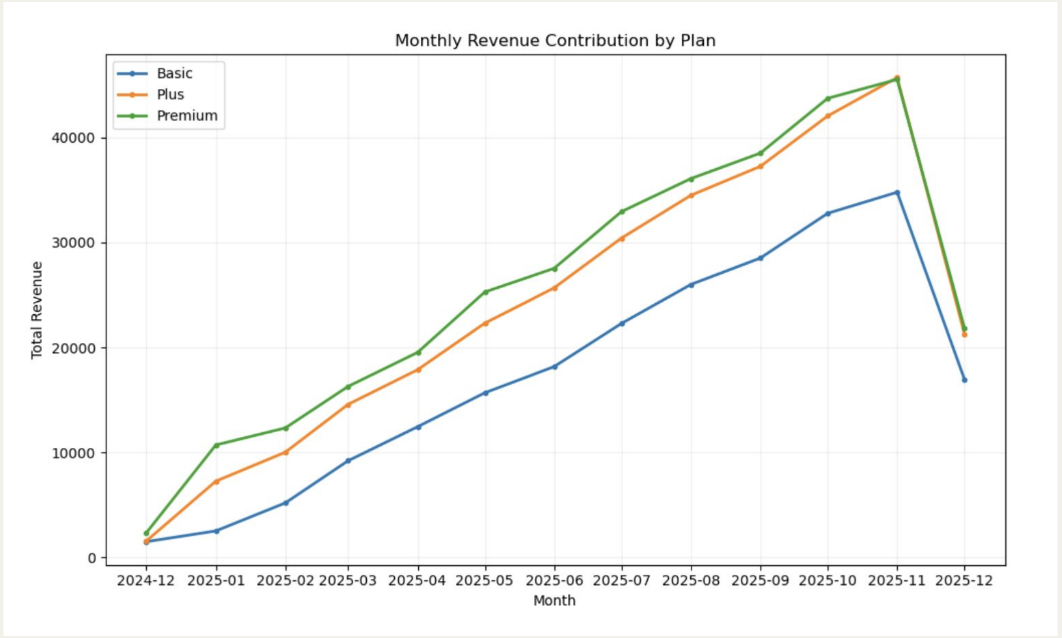
Plot 3.1: Seasonality Analysis of Class Attendance *filtered by location (Berkeley)*

Seasonality Analysis: Attendance Heatmap

Berkeley

Darker colors indicate higher attendance. January spikes highlight new-year participation behavior.

Analytical Process 1: Monthly Revenue by Plan

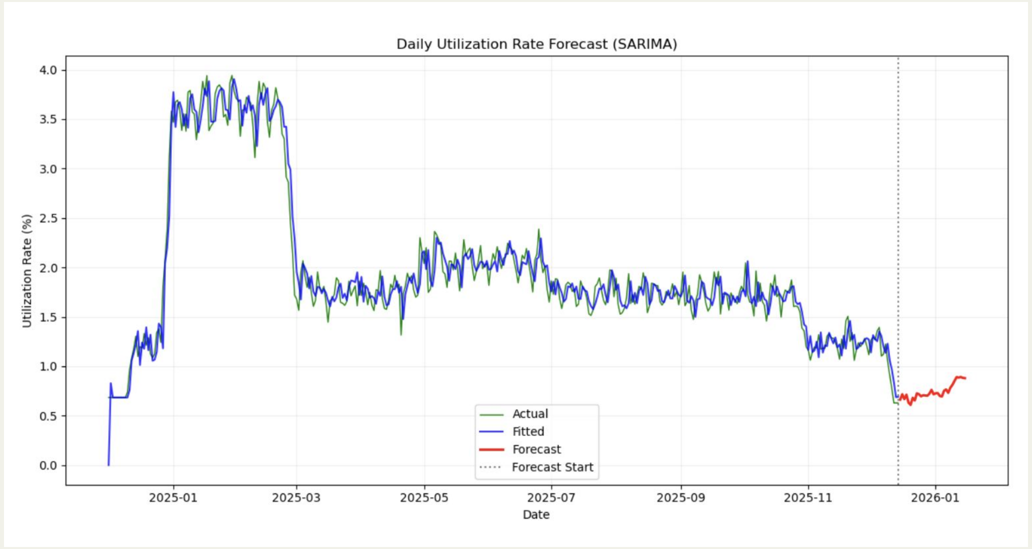


- This plot forecasts short-term revenue trends based on historical revenue patterns.

Why it matters

- Helps management anticipate future revenue fluctuations and plan budgets accordingly.
- Useful for short-term financial planning and performance monitoring.

Analytical Process 2: Utilization Forecasting



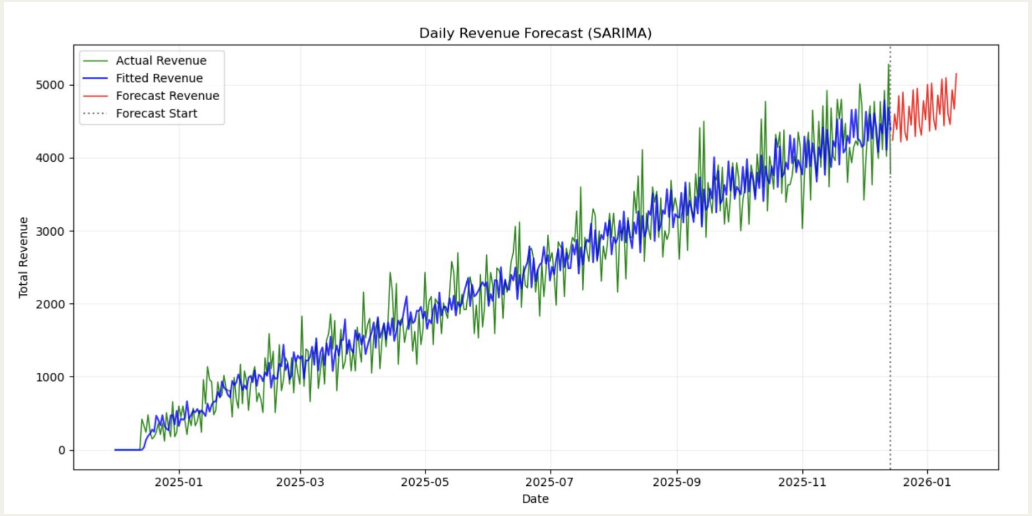
- This plot forecasts class attendance and utilization over time.

Why it matters

- Helps identify periods of high or low demand.

- Supports staffing decisions, class scheduling, and capacity planning.

Analytical Process 3: Daily Revenue Trend & Forecasting



- This plot analyzes historical daily total revenue and forecasts short-term future revenue trends using time-series modeling.

Why it matters

- Daily revenue reflects real-time customer activity and overall financial performance.
- Helps management monitor trends and anticipate short-term revenue changes.

Conclusion & Key Takeaways

What We Built

- Designed a production-style OLTP system to support real-time gym operations
- Implemented a star-schema OLAP system for scalable analytics
- Built a Python-based ETL pipeline that preserves OLTP performance
- Delivered an interactive management dashboard connected to the OLAP database

Why It matters

- Separating transactional and analytical workloads prevents performance bottlenecks
- Star schema enables fast, intuitive reporting across time, location, and plans
- Dashboard allows managers to make data-driven decisions without writing SQL

Project Outcome

- Demonstrates a complete data lifecycle: operations → ETL → analytics (dashboard & forecasting) → decision support