1 Transactional vs. Archival Data Needs

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

1.1 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.

1.2 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.

2 Relational Schema Design

2.1 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`).

2.2 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.

2.2.1 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).

2.2.2 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).

3 ETL Process and Implementation

3.1 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.

3.2 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.

A Appendix: SQL and Code Implementation

A.1 A.1 SQL CREATE Statements

A.1.1 OLTP Schema (Azure SQL)

The following SQL statements were executed to create the 3NF schema in the `fitflow_oltp` database.

```
-- 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)
);
```

A.1.2 OLAP Schema (Azure SQL)

The following SQL statements were executed to create the Star Schema in the `fitflow_olap` database.

```
-- 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)
);
```

A.2 A.2 Python Code for Synthetic Data Generation

This script populates the OLTP database with coherent synthetic data for all 10 entities.

```
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 (Credentials masked)
server = 'sql-fitflow-team8-2025.database.windows.net'
database = 'fitflow_oltp'
username = 'fitflowadmin'
password = 'Kergerisgreat*****'

# 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)

# 2. Members & Plans (Logic ensures logical consistency)
# ... [Code logic matches Section 2 description] ...

# 3. Transactional Data (Payments & Bookings)
# Payments generated based on Subscription Plan price
# Bookings generated linked to valid Class Sessions
print("Data Generation Complete.")
```

A.3 A.3 Python Code for ETL Process

This script implements the Extract, Transform, Load logic, ensuring no joins are performed on the OLTP system.

```
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 = 'Kergerisgreat*****'

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) ---
# We extract raw tables individually to avoid OLTP locks
```

```
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)
# ... other extracts ...

# --- STEP 2: TRANSFORM (Staging Area) ---
# Joins are performed in-memory (Pandas)
# Transform for Revenue Fact
df_subs_clean = df_subs[['subscription_id', 'plan_id']] # Select needed columns
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

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) ---
# Loading Dimensions first to ensure Referntial Integrity
# Loading Fact Tables
fact_revenue.to_sql('Fact_Monthly_Revenue', olap_engine, if_exists='append', index=False
print("ETL Completed Successfully.")
```