Team number: 18
Design Project Title: EZ Training Database
Date: 11/28/2024
Team members and Responsibilities:
Bhat, Adithya (SDE) ; Deng, Chuyun (Database Administrator);
Deras, Gerson Aaron Morale(Technical lead) ;
Polapragada, SaivenkataNagavyjayanthi (Communications lead);
Qiu, Qi (Emily) (CEO); Zhao, Shiyunyang (Solutions Architect)

TDP 2: Databricks and MongoDB Implementation

PART 1: Databricks Implementation of EzTraining Database Databricks Setup

Created a cluster and installed the required packages and setup Databricks connection and selected the database and collections which have been injected with artificial data.

Query 1: Total Payments by Method and Month Definition: Aggregates total payments by each payment method and groups them by month using MongoDB's aggregation framework.

Purpose: Provides insights into payment trends (e.g., which methods are most used and in which months) to optimize payment processing and identify seasonal trends.

04:57 PM (2s)

```
%sql
SELECT
    p.Member_ID,
    SUM(p.Amount) AS Total_Spent,
    SUM(w.Calories_burnt) AS Total_Calories_Burnt
FROM
    Payment p
LEFT JOIN
    Workout_Log w
ON
    p.Member_ID = w.Member_ID
GROUP BY
    p.Member_ID
ORDER BY
    Total_Spent DESC, Total_Calories_Burnt DESC
LIMIT 5;
```

(3) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Member_ID: integer, Total_Spent: long ... 1 more field]

Table

	Member_ID	Total_Spent	Total_Calories_Burnt
1	165	98	null
2	47	95	null
3	48	94	774
4	58	78	null
5	218	75	null

5 rows | 2.23 seconds runtime

This result is stored as `_sqldf` and can be used in other Python cells.

Query 2: Top 3 Most Frequently Used Equipment and Exercises Definition: Aggregates workout log data to calculate the usage count, average calories burned, and average heart rate for each exercise-equipment combination.

Purpose: Identifies the most frequently used equipment and exercises to optimize gym layout, maintenance schedules, and member experience.

04:56 PM (1s)

```sql
%sql
SELECT
    w.Exercise_ID,
    w.Equipment_ID,
    COUNT(*) AS Usage_Count,
    ROUND(AVG(w.Calories_burnt), 2) AS Avg_Calories_Burnt,
    ROUND(AVG(w.Heart_rate), 2) AS Avg_Heart_Rate
FROM
    Workout_Log w
GROUP BY
    w.Exercise_ID, w.Equipment_ID
ORDER BY
    Usage_Count DESC
LIMIT 3;
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Exercise_ID: integer, Equipment_ID: integer ... 3 more fields]

Table

	Exercise_ID	Equipment_ID	Usage_Count	Avg_Calories_Burnt	Avg_Heart_Rate
1	7	21	1	21	93
2	12	30	1	160	87
3	8	201	1	358	141

Advantages of Databricks Over MySQL for These Queries

Scalability: Handles massive data growth (100x) with distributed computing, ensuring consistent performance for large-scale aggregations and joins.

Performance: Databricks processes queries in parallel across clusters, significantly speeding up operations like sorting, grouping, and calculating averages compared to MySQL.

Flexibility: Supports semi-structured data formats and schema-less operations, making it easier to adapt to changes in data types or structures as the gym expands.

Real-Time Insights: Enables streaming and real-time analytics for dynamic insights whereas MySQL is batch-oriented and less suited for live updates.

Advanced Analytics: Easily integrates with machine learning tools for predictive modeling, providing actionable insights like recommending underutilized equipment or optimizing payment methods.

PART 2: MongoDB Implementation of EzTraining Database

MongoDB Setup

Installed the required packages and setup MongoDB connection and select the database and collections which have been injected with artifical data.

```
! pip install pymongo[srv] pandas
```

```
Requirement already satisfied: pandas in c:\users\adith\onedrive\documents\manalytic
s\project\215 project\.venv\lib\site-packages (2.2.3)
Requirement already satisfied: pymongo[srv] in c:\users\adith\onedrive\documents\man
alytics\project\215 project\.venv\lib\site-packages (4.10.1)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:\users\adith\onedrive\d
ocuments\manalytics\project\215 project\.venv\lib\site-packages (from pymongo[srv])
(2.7.0)
Requirement already satisfied: numpy>=1.23.2 in c:\users\adith\onedrive\documents\ma
nalytics\project\215 project\.venv\lib\site-packages (from pandas) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\adith\onedrive\doc
uments\manalytics\project\215 project\.venv\lib\site-packages (from pandas) (2.9.0.p
ost0)
Requirement already satisfied: pytz>=2020.1 in c:\users\adith\onedrive\documents\man
alytics\project\215 project\.venv\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\adith\onedrive\documents\m
analytics\project\215 project\.venv\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\adith\onedrive\documents\manalyt
ics\project\215 project\.venv\lib\site-packages (from python-dateutil>=2.8.2->panda
s) (1.16.0)
WARNING: pymongo 4.10.1 does not provide the extra 'srv'
```

```python
from pymongo import MongoClient
import pandas as pd
from configparser import ConfigParser

# Read the configuration file
config = ConfigParser()
config.read('config.ini')

# Access the configuration values
password = config.get('mongodb', 'password')
username = config.get('mongodb', 'username')
host = config.get('mongodb', 'host')
app_name = config.get('mongodb', 'app_name')


uri = f"mongodb+srv://{username}:{password}@{host}/?retryWrites=true&w=majority&app
client = MongoClient(uri)

# Select the database and collections
db = client["ez_training"]
payments = db["payments"]
workout_logs = db["workout_logs"]
```

Query 1: Nested Aggregation for Revenue Insights

Purpose: Calculate total revenue grouped by payment_method and further break it down by months. MongoDB's pipeline for advanced analytics, avoiding complex nested SQL queries and improving maintainability as datasets grow.

```
query1 = payments.aggregate([
    {
        "$addFields": {
            "parsed_date": {
                "$dateFromString": {
                    "dateString": "$payment_date",
                    "format": "%m/%d/%Y"
                }
            }
        }
    },
    {
        "$group": {
            "_id": {
                "payment_method": "$payment_method",
                "month": {"$dateToString": {"format": "%m", "date": "$parsed_date"}}
            },
            "total_amount": {"$sum": "$amount"}
        }
    },
    {"$sort": {"_id.month": 1}}  # Sort results by month
])

query1_result = list(query1)
# Flatten the `_id` field
for item in query1_result:
    item.update(item.pop('_id'))

# Convert to DataFrame
df = pd.DataFrame(query1_result)

# Display the DataFrame
print(df.head(5))
```

```
   total_amount payment_method month
0            37          Check    01
1            78         PayPal    02
2            69          Check    03
3            61     Debit Card    03
4            59         PayPal    03
```

Query 2: Hierarchical Data Relationship Join

Purpose: Join workout_logs and workout_sessions collections to find the top 3 workout programs (e.g., "HIIT", "Yoga") where members burned the most calories. This highlights MongoDB's $lookup functionality, which allows you to relate collections dynamically without rigid schema constraints. This is ideal for scaling data relationships without redesigning schemas, a common challenge in MySQL

```
query2 = workout_logs.aggregate([
    {
        "$lookup": {
            "from": "workout_sessions",
            "localField": "log_id",
            "foreignField": "Workout_Log_ID",
            "as": "session_details"
        }
    },
    {"$unwind": "$session_details"},  # Decompose arrays into individual documents
    {
        "$group": {
            "_id": "$session_details.Program_type",
            "total_calories": {"$sum": "$calories_burnt"}
        }
    },
    {"$sort": {"total_calories": -1}},  # Sort by total calories burned (descending
    {"$limit": 3}  # Top 3 programs
])

query2_result = list(query2)


# Convert to DataFrame
df2 = pd.DataFrame(query2_result, columns=["_id", "total_calories"])
df2.rename(columns={"_id": "Program Type"}, inplace=True)
# Display the DataFrame
print(df2.head())
```

```
       Program Type  total_calories
0           Pilates             194
1  Strength Training             160
2            Cardio              51
```

Advantages in scaling with MongoDB over MySQL:

Future-Proofing the System: MongoDB is designed for distributed systems and can handle high-growth scenarios seamlessly. A 100x increase in data is more manageable with MongoDB's sharding and replication capabilities.

Real-Time Analytics: MongoDB's aggregation framework and fast lookups make it ideal for real-time insights, such as identifying top-performing workout programs or analyzing monthly payment trends.

Lower Operational Overhead: MongoDB's schema-less design reduces the need for time consuming migrations, making it easier to adapt to changing business requirements as the client grows.

Flexibility for Evolving Data Models: As the gym expands, new features (e.g., adding VR environments to sessions or new payment methods) can be added with minimal impact on existing collections.

Cost Efficiency: Scaling MySQL often involves adding more powerful hardware to support increasing data loads. MongoDB, with its horizontal scaling capabilities, can add cheaper commodity servers to the cluster.