**KPMG Technical Assessment Report – Vijay Dattada**

**1. Understanding the Case Study**

**1.1 Overview**

This case study focuses on building a robust data pipeline and analytical framework for swimming competition results. The primary goal is to extract, process, and analyze swimming competition data to generate valuable insights for stakeholders such as coaches, athletes, and regulatory bodies.

**1.2 Case Study Requirements**

The case requires:

1. **Data Extraction**: Collect raw data from structured sources (CSV files).
2. **Data Processing**: Transform raw data into a structured format while maintaining historical tracking.
3. **Data Storage**: Store cleaned and structured data efficiently for analysis.
4. **Data Analysis & Visualization**: Provide analytical insights through views and reporting.
5. **Infrastructure & Deployment**: Ensure scalability and reproducibility using containerized solutions.

**2. Breaking Down the Problem**

**2.1 Challenges Identified**

* **Data Source Constraints**:
  + The competition website’s robots.txt restricted web scraping.
  + Alternative API solutions were non-compliant with policies.
  + **Solution**: Manually compiled CSV files to serve as the data source.
* **Data Consistency & Quality**:
  + The raw data required validation before processing.
  + Different data sources had format variations.
  + **Solution**: Implemented automated validation checks during extraction.
* **Tracking Historical Changes**:
  + Swimmers' details (e.g., club affiliations, rankings) change over time.
  + **Solution**: Implemented Slowly Changing Dimension (SCD) Type 2 for historical tracking.
* **Scalability & Reproducibility**:
  + Needed an environment that could be easily deployed across systems.
  + **Solution**: Used Docker containers for portability.

**3. Big Picture: Data Pipeline & Architecture**

The data pipeline is structured as follows:

1. **Data Ingestion Layer:**
   * **Reading CSV Files**: You begin by reading raw CSV files containing data, which is the standard method for data ingestion.
   * **Validation and Quality Checks**: Ensuring the data is valid and consistent is crucial for avoiding downstream issues.
2. **Processing Layer (Dimensional Tables):**
   * **Data Cleaning and Structuring**: This involves transforming raw data into a more usable form, ensuring it’s structured according to the dimensional model.
   * **SCD Type 2**: Implementing Slowly Changing Dimensions (SCD) Type 2 allows for tracking historical changes in data, which is important for analytics over time (e.g., swimmer history tracking).
   * **Surrogate Keys**: These are necessary to uniquely identify rows in dimension tables, especially when dealing with SCD Type 2.
3. **Fact Tables Layer:**
   * **Joins and Analytical Tables**: You join your dimension tables with fact tables to create comprehensive analytical tables that will store performance results.
   * **Aggregated Rankings**: Storing summarized metrics, such as rankings and performance results, is essential for querying and reporting purposes.
4. **Optimization Layer:**
   * **PySpark Optimizations**: Techniques like broadcast joins, window functions, and caching help improve performance and scalability in your data processing pipeline, especially when dealing with large datasets.
   * **Temp Tables**: Used for intermediate data processing, reducing redundant computations and enhancing performance.
5. **Storage & Analytics Layer:**
   * **PostgreSQL**: Efficient storage for your data, offering powerful querying capabilities and compatibility with the analytics workload.
   * **Analytical Views**: Creating views for rankings, club performance, and swimmer demographics allows users to easily access analytical insights.

**Airflow Orchestration**

* **Triggering on New Files**: Using Apache Airflow to monitor the raw\_data folder for new files and triggering the processing pipeline ensures an automated, real-time workflow.
* **Task Dependencies and DAGs**: You’ve set up the Directed Acyclic Graph (DAG) in Airflow, specifying the sequence of tasks, from data processing to creating views.
* **Retries and Error Handling**: Implementing retry mechanisms ensures resilience in case of failures, and the error handling setup ensures that any issues are logged and managed properly.

A diagram of a company

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**4. Implementation Details**

**4.1 Data Sources**

* CSV files containing swimmer details, competition events, and rankings.
* Pre-processed datasets ensuring structured data ingestion.

**4.2 Data Extraction Layer**

* Reads CSV files and performs initial validation.
* Maps raw data to a structured schema.

**4.3 Data Processing / Intermediary Layer**

* **Dimensional Modeling & Star Schema**:
  + The data warehouse follows a **Star Schema** to ensure fast query performance and maintain simplicity.
  + Fact tables contain measurable metrics, while dimension tables store descriptive attributes.
* **Slowly Changing Dimensions (SCD) Implementation**:
  + dim\_swimmer follows **SCD Type 2**, tracking historical changes in club affiliations.
  + dim\_club follows a **hybrid SCD** (Type 1 for address, Type 2 for city/country changes).
  + dim\_competition follows **SCD Type 1**, as venue updates don’t require historical tracking.
* **Handling Updates & Historical Data**:
  + Insert new records with a generated surrogate key.
  + Maintain historical accuracy by closing old records when changes occur.
  + Ensure consistency using versioning techniques (valid\_from, valid\_to).
* **Fact and Dimension Tables**:
  + **Dimension Tables**:
    - dim\_swimmer: Contains swimmer details with historical tracking.
    - dim\_event: Stores event information.
    - dim\_club: Tracks club details with mixed SCD implementation.
    - dim\_competition: Holds competition details.
  + **Fact Tables**:
    - fact\_swimming\_performance: Contains competition results, linking swimmers, events, and competitions.
    - fact\_rankings: Aggregates performance rankings, tracking best times and rankings.

A diagram of a company

AI-generated content may be incorrect.

**4.4 Data Storage Layer**

The **Data Storage Layer** is designed using **PostgreSQL**, following a **star schema** for efficient querying and analytics. The schema consists of **dimension tables** (dim\_swimmer, dim\_event, dim\_competition) and **fact tables** (fact\_swimming\_performance, fact\_rankings), ensuring structured data storage and optimized joins. **Surrogate keys** (\_sk) are implemented as **bigint** values to maintain dimensional stability, improve indexing performance, and enforce **referential integrity** across tables. The **Slowly Changing Dimension (SCD) Type 2** approach is used for dim\_swimmer, allowing historical tracking of club affiliations, while dim\_competition remains static with **SCD Type 1**. The fact tables store **competition results and rankings**, linking to the correct swimmer version at the time of each event. **Indexing and partitioning strategies** are applied to **enhance query speed**, and **materialized views** are used for aggregated competition analytics. The schema aligns with the **Olympic qualification criteria**, ensuring that queries can efficiently filter and evaluate swimmers based on ranking, stroke count, lifetime competitions, and **Olympic A & B cut times**. This structured approach ensures **scalability, data consistency, and optimized analytical processing** for swimming performance analysis.

A diagram of a computer

AI-generated content may be incorrect.

**4.5 Views & Analytics Layer (Specifically mentioned as desired output)**

* **vw\_swimmer\_rankings**: Ranks swimmers based on performance.
* **vw\_club\_region\_performance**: Analyzes regional competition outcomes.
* **vw\_swimmer\_demographics**: Provides insights based on age, club, and region.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**5. Infrastructure & Deployment**

* **Docker-based containers**:
  + **Process Container**: Runs PySpark for ETL processing.
  + **Database Container**: Hosts PostgreSQL for structured storage.
* **Airflow (Optional)**: Automates data pipeline execution.
* **Materialized Views** for faster aggregations in reporting.

### **6. Reports Based on Olympics Qualifying Criteria**

The data solution ensures compliance with **Olympic qualification standards** by generating reports that evaluate swimmer eligibility based on predefined criteria. The reports assess whether swimmers have completed at least **five swims per stroke type** (Butterfly, Freestyle, Backstroke, Breaststroke, and IM) within the past year at the **National, Province, or Local level**. Additionally, the system verifies that each swimmer has participated in a **minimum of 50 lifetime competitions** and has achieved a **ranking between 1st and 5th** in recognized competitions. Performance times are analyzed to ensure they fall within the **22 to 50-second range**, and recent rankings are checked to confirm that a swimmer has placed **1st at least three times in their last five competitions**. The system also enforces the **age requirement** of being **older than 12 years** and tracks **Olympic A and B cut times**, with A cut swimmers given priority.

To support these analyses, **Power BI reports** have been developed and are provided as a **PBIX file**, offering visual insights into swimmer performance and qualification status. The relevant **views and tables** containing this structured data are available in the **database** for further exploration and analysis.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

**Further Recommendations for Implementation:**

This ETL pipeline architecture can be implemented on **Databricks**, **Azure Data Factory (ADF)**, and **Azure SQL** to handle the 4 V’s of data: **velocity**, **variety**, **veracity**, and **volume**.

* **Databricks** provides a highly scalable and optimized environment for data processing. By leveraging **Apache Spark**, I can efficiently process large datasets, ensuring both speed and scalability, no matter the data volume or complexity.
* **Azure Data Factory (ADF)** can be used for orchestration, automating the ETL workflows and triggering tasks like data processing, table creation, and view generation. ADF's GUI offers a visual interface that makes it easy for me to design and manage workflows, much like **Airflow**, while managing task dependencies, retries, and error handling. I can use the **ADF GUI** to implement the orchestration logic, making it a simpler and more intuitive solution compared to writing code.
* **Azure SQL** serves as the efficient storage layer for processed data, providing high-performance storage for both structured and unstructured data. Its integration with other Azure services ensures that my data is secure and easily accessible for analytics.

The architecture I've implemented can easily be scaled by changing configurations for processing and loading data. With the flexibility of Azure tools, I can adapt the pipeline to handle different workloads and complexities, ensuring it performs optimally in any scenario.

This approach leverages cloud-based tools to maximize the scalability, reliability, and efficiency of the ETL pipeline, ensuring it can handle large-scale data processing and analytics effectively.

Dev log details:

Postgres:

User – postgres

Password – postgres

POSTGRES\_DB: swimming\_olympics

Airflow:

Admin – airflow

Password- airflow

Docker command: docker-compose -up –build -d