Data Engineering Assignment 2

Design and Implementation of Data Architecture and Data Processing Pipelines

David Akkermans 2028316 [d.l.akkermans@tilburguniversity.edu](mailto:d.l.akkermans@tilburguniversity.edu)

Thijs Eekelaar 2027144 [t.f.j.eekelaar@tilburguniversity.edu](mailto:t.f.j.eekelaar@tilburguniversity.edu)

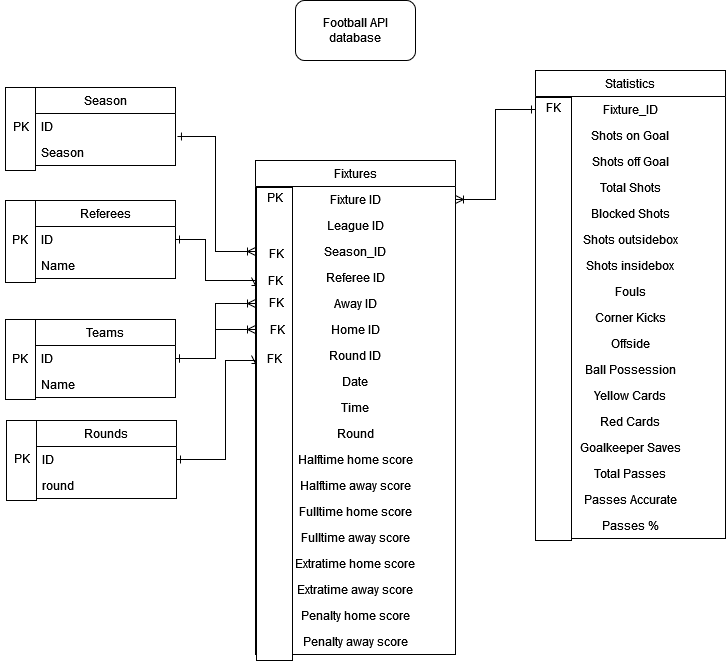
Niels Gaastra 2071137 [n.t.gaastra@tilburguniversity.edu](mailto:n.t.gaastra@tilburguniversity.edu)  
Vincent van Tiel 2109251 [v.l.vantiel@tilburguniversity.edu](mailto:v.l.vantiel@tilburguniversity.edu)

Quinten Tulp 2089837 [q.j.tulp@tilburguniversity.edu](mailto:q.j.tulp@tilburguniversity.edu)

Project GitHub repository: GITHUB REPO

# Overview of the Data Pipelines

Before the development of our data pipelines could get started, a dataset needed to be chosen. We chose to extract football data from the publicly available football API[[1]](#footnote-0). This API offers data about football fixtures from all over the world with their respective statistics, we chose to use only fixtures from the last two World Championships.

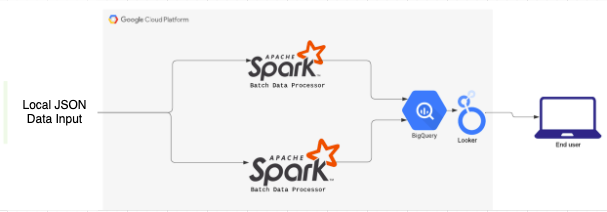


*Figure 1: ER-diagram of Football API database*

The initial choice was to use Google Looker to present our data. However this turned out to be a paid service which led to the decision to switch. Writing a framework to navigate database data is very time consuming, so the next choice was PowerBI. The linking of the BigQuery database to PowerBI seemed hard however. This left little options for a suitable data studio so we resorted to creating some simple diagrams using matplotlib in a notebook.

All scripts of Spark (2 workers, 1 master) and Python (PySpark) are run on a GCP Virtual Machine with Ubuntu 20.04 LTS headless as operating system.

# Design and Implementation of Data Architecture



*Figure 2: Data Implementation overview*

As can be seen in figure 2, the data used within this data architecture is put in locally via an UI which is set up through Flask. This data then gets stored locally on the Google Cloud Platform Virtual Machine (GCP, VM). The data is then transported, read and transformed all done with the use of Spark (PySpark). The particular transformations described here will be elaborated further on in the nest section. After undergoing the transformation, a data frame is stored in BigQuery, the data warehouse, as separate tables.  
  
For the visualisation of our output, the Google tool Looker is used. As this tool is also a product of Google and is integrated with BigQuery, the implementation was smooth. When receiving the respective tables, Looker automatically generates visualisations and the ones that are most fit can be chosen.

# 

# Design and Implementation of Data Pipelines

The data is inputted as a raw JSON file that is directly pulled from a football score API (<https://www.api-football.com>). This data contains two different formats, fixtures and statistics. They both are formatted differently so they were handled differently. The useful features that were chosen to be extracted can be found in the database design. The next step was to conform the data to the database design. This primarily consisted of making pivot tables to reduce data redundancy in the database. This building of pivot tables from existing data was primarily done with the use of windows to convert set values to id’s which could be used later to join the data together. Furthermore the ‘date’ feature included a DateTime format. This DateTime format was converted into a ‘date’ and a ‘time’ element to allow efficient storing in the database.

# Reflection on Design and Implementation of Data Architecture and Data Pipelines

As mentioned in the sections above, we chose to implement two batch data oriented pipelines. This choice was made because our chosen dataset is not particularly suited for stream data pipelines. Most matches/fixtures played are already stored and if there are new games this will be three new ones per day at maximum (the group stages). However, if the goal was to present the user with real-time data and statistics, live betting for example, then a stream pipeline would definitely be the way to go. Here one batch pipeline would be replaced and Kafka should be implemented. Also the data pipeline would need different transformations within Spark as the focus would be on different data configurations. Furthermore, we tried to implement batch upload using a Flask front-end. This seemed hard to implement however so we chose to simply use the notebooks as the data-pipeline. This made the use of Google storage buckets unnecessary so we chose to use local data upload instead.

# 

# Individual Contributions of Students

**Niels** was prominently active with the whole implementation of Spark and the overall configuration of the whole data architecture. **Niels** did also design the Flask UI and thus the data input.

**Thijs** worked together with **Niels** on Spark and its data transformations. **Thijs** was also active with the database design working with BigQuery and designing the ER diagram.

**David** worked together with **Niels** on the data ingestion within Spark and also explored the implementation of Kafka with a possible stream pipeline. **David** did also have a prominent role in setting up the report.

**Quinten** worked on the data extraction out of the API into Google Cloud Storage using buckets. **Quinten** did also work on BigQuery and among other things explored its possibilities.

**Vincent** worked prominently with task distribution and the overall work planning. Whenever needed **Vincent** helped among others with Spark, Looker and BigQuery.

1. https://www.api-football.com/sports [↑](#footnote-ref-0)