FootStats: Project Implementation Outline

Cameron Mouangue & Jacques
M1 Applied Mathematics and Statistics
Institut Polytechnique de Paris / École Polytechnique

February 19, 2026

1 What You Are Building

You are building a PostgreSQL relational database to store and analyze historical football match data, paired with a lightweight Python script to load the data and run statistical queries.

2 Where to Find the Data

You will use the "European Soccer Database" on Kaggle (created by Hugo Mathien). Why this dataset? It is pre-cleaned and contains over 25,000 matches, 10,000 players, and 11 European leagues from 2008 to 2016. The Catch: It is provided on Kaggle as a SQLite file. To meet your course requirements, you will export the tables to CSVs (or read them directly in Python) and ingest them into your Postgres v15 server.

3 The Schema

To demonstrate Schema Refinement and Normal Forms without drowning in tables, restrict your database to these 4 core tables:

- League: league_id, name, country.
- Team: team_id, name, short_name.
- Player: player_id, name, height, weight.
- Match: match_id, league_id, date, home_team_id, away_team_id, home_team_goal, away_team_goal. (This is your main fact table where the heavy lifting happens).

4 Implementation Outline

4.1 Phase 1: Conceptualization and Design

Begin by designing an Entity-Relationship (ER) Model for the football data. Apply principles of Schema Refinement and Normal Forms to ensure the database is structurally sound and avoids redundancies. Document this design visually before writing any code.

4.2 Phase 2: Database Setup and Core SQL

Launch a PostgreSQL v15 server, utilizing either your own laptop or the Polytechnique lab rooms. Write a SQL script containing the Data Definition Language (DDL) to create your tables based on the ER model. Incorporate SQL triggers to handle automated actions, such as updating a team's total points when a new match result is inserted.

4.3 Phase 3: Application Programming and Data Ingestion

Fulfill the Application Programming requirement by writing a Python script. Use this script to connect to your PostgreSQL database and automatically ingest a large dataset (like a CSV file of historical matches). Execute these scripts utilizing the Unix basics covered in your January 30th session.

4.4 Phase 4: Querying and Optimization (The Heavy Lifters)

Draft complex SQL queries that translate Relational Algebra concepts into tangible analytical results. Analyze the performance of these queries using query plans. Demonstrate your understanding of Storage and Indexes by implementing B+trees to optimize slow queries.

5 The Analytics (SQL & Application Programming)

Your applied mathematics and statistics background will shine here. Instead of writing basic SELECT statements, you will write complex SQL queries using Window Functions and Aggregations. The Queries: Calculate rolling win streaks, average goals scored at home vs. away, or a team's historical point accumulation over a season.

The Application: You will write a simple Python script (using psycopg2 or SQLAlchemy) that connects to your database—whether hosted on your laptop or the Polytechnique lab rooms—inserts the Kaggle data, and executes your analytical queries to print the results. This fully covers the Application Programming requirement.

6 The Optimization Feature

This is the most critical part of the project to please your professors. Write a statistically heavy query on the Match table (e.g., querying all matches over a 5-year span for specific teams). Use EXPLAIN ANALYZE to show the Query Plan and prove that the database is doing a slow, inefficient "Sequential Scan". Implement a B+Tree Index on the date or home_team_id columns to fulfill the Storage and Indexes requirement. Run EXPLAIN ANALYZE again to show the massive drop in query cost.

7 Final Delivery Format (Git Repository)

To deliver the project flawlessly, organize your Git repository with a clean, self-explanatory structure. This makes grading incredibly easy for the teaching team. You and Jacques will submit a clean Git repository containing: A folder with your .sql files (table creation, indexes, and queries). Your Python ingestion script. A README.md featuring your ER diagram and a brief explanation of your query optimization results.

Repository Structure

- **README.md:** The most important file. It should contain the project title, team members, a setup guide (how to start the PostgreSQL server and run the app), and a summary of your optimization results.
- **docs/:** A folder containing your ER diagrams as PDFs or images.
- **sql/:** A folder dedicated to your raw SQL files.
  - `01_schema.sql:` Table creations.
  - `02_triggers.sql:` Custom triggers.
  - `03_analytics.sql:` Your complex queries.
  - `04_indexes.sql:` The index creations used for optimization.
- **app/:** Your Python scripts for data ingestion and the user interface.
- **data/:** A small sample CSV file so the teacher can test the database without downloading massive datasets.