Football Relational Database

Database Systems CSC6710-002, Fall 2022

Samuel McDonald
Department of Computer Science
Georgia State University
Atlanta, Georgia, US
smcdonald32@student.gsu.edu

Diana Gambone
Department of Computer Science
Georgia State University
Atlanta, Georgia, US
dgambone2@student.gsu.edu

Tianjing Guo
Department of Computer Science
Georgia State University
Atlanta, Georgia, US
tguo4@student.gsu.edu

Abstract—**Aside from being a vastly popular global sport, entertaining countless fans a year, football provides a vast and multidimensional source of information. Any given league can generate thousands of data points concerning the league's players, games, managers and so on. This information is utilized for a wide variety of applications ranging from internal managerial decisions to fan sports betting. This study primarily focuses on organizing data concerning the top five largest and most popular football leagues from the years 2014 to 2020. Due to the many connections among the sport's characteristics and involved groups, a relational model and database was implemented as the most concise and accurate representation of the data. Overall, issues encountered due to the database design were minimal, with the exception of the model's inability to connect a player with a specific team. Though the database was not hosted on AWS EC2 as initially proposed, the database is successfully and securely hosted using an online app server through Heroku PostgreSQL. The database provides insight on multiple features of the modeled data, such as player, team, and game statistics.**

I. Introduction

The sport of soccer, or football as it is known outside the United States, is a global industry employing and entertaining hundreds of millions of people. A 2021 report from FIFA, the global governing body of organized football, disclosed that the top 20 football clubs in the world generated more than 9 billion Euros in revenue during the 2018-2019 season, an increase of 11% over the previous year [1].

Owing to the global scope of the industry and continuous growth, there are thousands of both long running and newly emerging teams competing across several leagues annually. Each league has diverse player rosters that match up across thousands of games. In particular, the scope and popularity of European football generates an extensive volume of data across many dimensions. This complexity presents many challenges for effectively and efficiently organizing information.

Having a clear and accurate representation of football data has both internal and external practical relevance. Internally, team managers and executive boards utilize this information for making managerial decisions. Externally, industries like gambling and sports media require accurate information for their business operations.

II. Problem Statement

Considering the velocity and cardinality of the data surrounding football, the scope of the analysis was narrowed to the five largest European football leagues, the Premier League of England, Serie A of Italy, Bundesliga of Germany, La Liga of Spain, and Ligue 1 of France, between the years 2014 and 2020. Each of these leagues have between 18 and 20 teams, resulting in hundreds of games played per season. A relational model provides an ideal structure for this data given the many types of interactions and relationships among teams, players, leagues, and managers over time.

The present study looks to design and implement a relational database to organize the dataset ”Football Database - Football and Betting Statistics of the European Top 5 Leagues”, which was retrieved from Kaggle on October 17, 2022 [2]. Additional data was sourced from football statistics websites, such as Soccerbase.com, FBREF.com, and transfermarkt.us [3] [4] [5] to add more complexity and detail to the database. If needed, this relational model can be extended to include data from other football leagues and time periods.

III. Methodology and Implementation

A. Logical Database Design

The relational model implemented in the final database includes the Season, League, Player, Appearance, Manager, Shot, Game, Team, and Venue relations, and the ManagerTeam and GameTeam join tables (see appended figure at end of paper). Each of the tables were normalized to be in Boyce-Codd Normal Form (BCNF) which results in their key attributes being minimally-redundant superkeys. Of note in this normalization process, the need to preserve BCNF and avoid data anomalies led to the decision to not include a match result attribute in the Game table. The game result can be inferred from the homeGoals and awayGoals attributes, which would be a bad functional dependency. Including a result attribute would therefore violate BCNF.

The majority of relationships proposed in the conceptual design stage of this project were one-to-many and therefore did not require restructuring of the model in the logical design phase. The exceptions are the relationship between Manager and Team and between Game and Team, which are both many-to-many. The model needed to allow Managers to coach at more than one Team over time, and vice versa, so a join table was added representing an instance of a manager working for a specific team over a given period. The design challenge for the Game-Team relationship was to allow two teams to play against each other in any configuration (home or away) for any number of times. The GameTeam join table was designed as a three-way join table that links a Game instance and two Team instances, with a notation of which team id's are home and away.

B. Data Pre-Processing

The data were primarily sourced from a collection of comma-separated values (CSV) files hosted on the Kaggle repository, originally uploaded by the Kaggle user Technika148 [2]. The data set covers the time period between the 2014 and 2020 seasons of the Top 5 European football leagues. Included were details of those leagues, matches played, players who made appearances, shots attempted, and team and individual performance statistics.

To augment the information in the Kaggle data set and present a fuller view of European football, additional data was manually aggregated about player demographics, team managers and their tenures, and the sporting arenas that hosted matches. This data was compiled from a number of commonly referenced football statistics websites, including Soccerbase.com, FBREF.com, and transfermarkt.us [3] [4] [5].

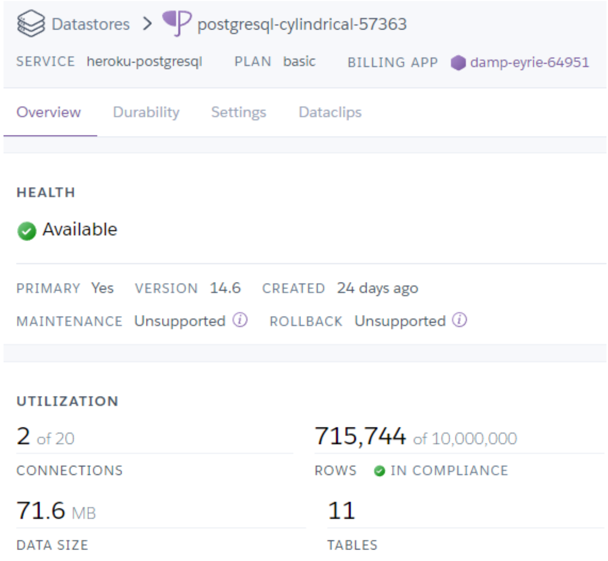
Once the data was gathered, it was loaded into Python and processed using the pandas module. Data from the source CSV's was merged into tables formatted to match the proposed logical schema, player biographic data was matched to the appropriate players, and join tables were created to resolve many-to-many relationships in the conceptual model. Each table was written out to a CSV in preparation for loading to the physical database. The logical schema is represented in the entity-relationship diagram appended to the end of the paper.

C. Physical Database Implementation

Once the data was processed into a set of tables in *.csv format, the database was created and hosted on a cloud-based platform. The Heroku platform was chosen to host the football database because it allows users the flexibility to build a web interface for generating reports and modifying data, on top of the cloud-hosted database. From the Heroku documentation,

> 'Heroku is a platform as a service based on a managed container system, with integrated data services and a powerful ecosystem, for deploying and running modern apps.' [6]

A Heroku app server is the central element of a web-hosted project, with additional on-demand services such as data services, continuous integration, and security services attached to the app. In the present implementation, a Heroku app with the Heroku PostgreSQL add-on was used. The primary advantage of using the Heroku platform is that it's modular - add-ons can be added as appropriate for the use case. Since the football data is not large, Heroku provides sufficient hosting capabilities and it was unnecessary to host the database elsewhere and configure app integration from scratch. Heroku provides a web-based dashboard to monitor different services where details of the PostgreSQL database can be easily accessed (Figure 1). The PostgreSQL add-on also provides a dataclip function that the database owner can use to run SQL queries and directly export the returned result as a *.csv or JSON file. (Figure 2)



SERVICE heroku-postgresql PLAN basic BILLING APP damp-eyrie-64951

HEALTH
Available

PRIMARY Yes VERSION 14.6 CREATED 24 days ago
MAINTENANCE Unsupported ⓘ ROLLBACK Unsupported ⓘ

UTILIZATION
2 of 20 CONNECTIONS
715,744 of 10,000,000 ROWS ● IN COMPLIANCE
71.6 MB DATA SIZE
11 TABLES

Fig. 1. Get Heroku PostgreSQL Add-on

To implement the database on Heroku PostgreSQL, the Heroku CLI was installed and used to login to a Heroku account using a system terminal. The detailed install instructions can be found in the official Heroku documentation [7]. The Heroku app server was started by executing the following command from the root of the project repository:

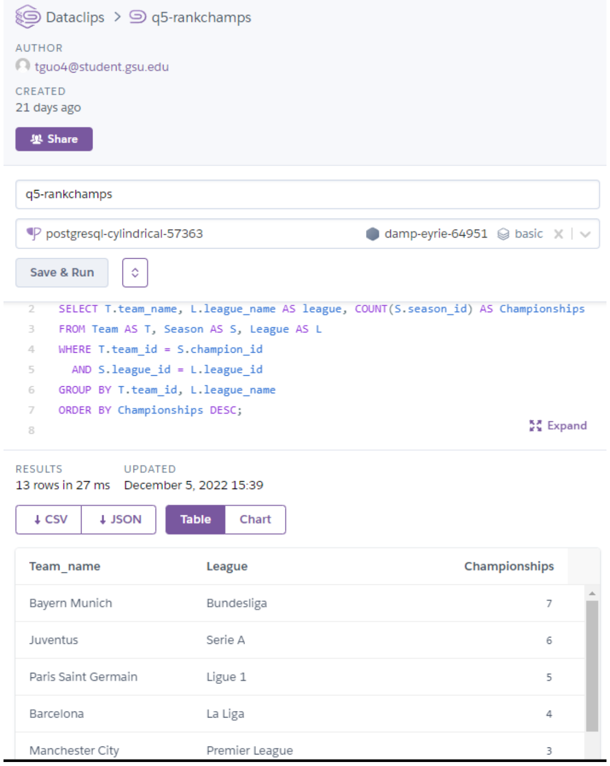
```
heroku create
git push heroku main
```

The Heroku PostgreSQL add-on was added to the new app. Once the corresponding plans were selected, a PostgreSQL instance was created on the app server and made accessible online. The PostgreSQL tool interface was initiated at the terminal by executing:

```
heroku pg:psql
```

Two different methods were utilized while implementing the logical schema of the football database. One way of creating the schema is using the Heroku PostgreSQL CLI. The schema for each table was established with the CREATE SQL command. Using the League table as an example, the CREATE command would be:

```
\CREATE TABLE league
(
  league_id integer NOT NULL,
  league_name character varying(50),
  CONSTRAINT league_pkey
```



```sql
SELECT T.team_name, L.league_name AS league, COUNT(S.season_id) AS Championships
FROM Team AS T, Season AS S, League AS L
WHERE T.team_id = S.champion_id
  AND S.league_id = L.league_id
GROUP BY T.team_id, L.league_name
ORDER BY Championships DESC;
```

RESULTS 13 rows in 27 ms UPDATED December 5, 2022 15:39

Team_name	League	Championships
Bayern Munich	Bundesliga	7
Juventus	Serie A	6
Paris Saint Germain	Ligue 1	5
Barcelona	La Liga	4
Manchester City	Premier League	3

Fig. 2. Running a SQL Query Using Heroku PostgreSQL Dataclip

```
      PRIMARY KEY (league_id)
);
```

The other method was connecting the database to the pgAdmin client and using its graphical interface to implement the schema. The database credentials are located under the settings table of the database dashboard, and the detailed instructions for connecting to the database through pgAdmin are shown in the next section. Once the schema was implemented the COPY command was used in the Heroku CLI to load the processed data to the database:

```
\COPY league
FROM \your\table\location\league.csv
WITH (FORMAT CSV, DELIMITER ',', HEADER true);
```

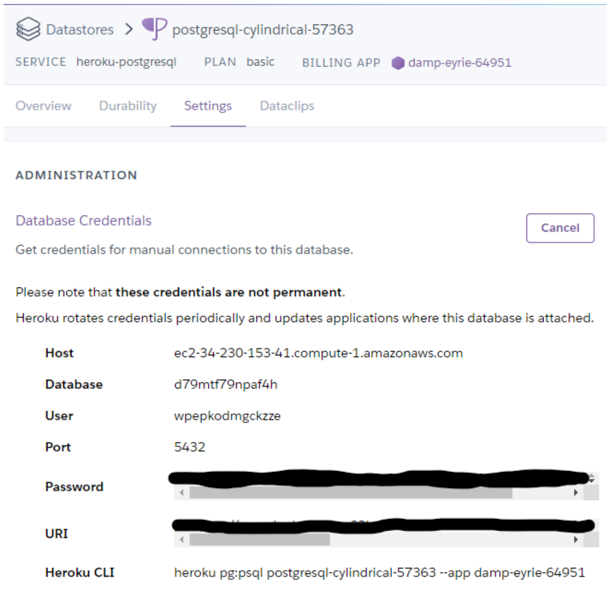
After migrating all data to the Heroku PostgreSQL database, the database setup was complete and ready to use.

D. Using The Database

The owner of the Heroku application can directly access and query the database as highlighted in the previous section (Figure 2). Other users can also query the football database by connecting through pgAdmin. The following information is needed to connect to the database and can be found in the 'Settings' tab of the Heroku web dashboard, see Figure 3:

- Host
- Database Name
- User
- Port

The corresponding location to input this information into pgAdmin's 'register database' wizard is shown in Figure 4. Additionally, the visibility of the database needs to be restricted by adding the database name to the 'DB restriction' item under the advanced tab (see Figure 5). The credentials can be saved to reconnect to the database as needed.


Datastores > postgresql-cylindrical-57363
SERVICE heroku-postgresql
PLAN basic
BILLING APP damp-eyrie-64951
Overview
Durability
Settings
Dataclips
ADMINISTRATION
Cancel

Database Credentials

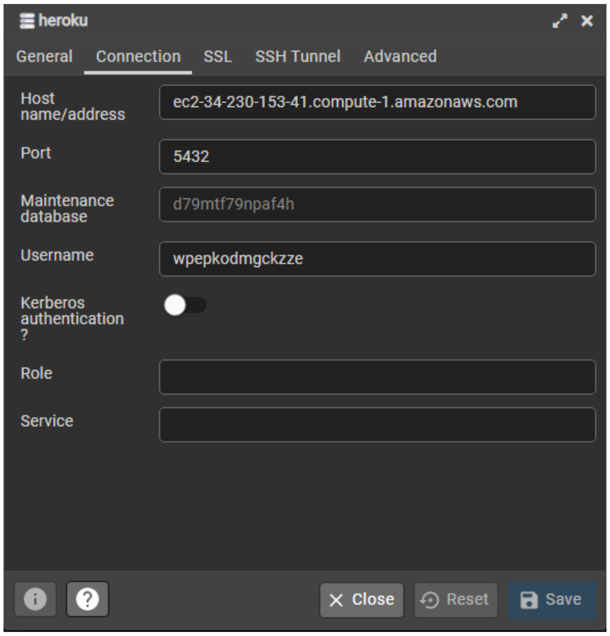
Get credentials for manual connections to this database.

Please note that **these credentials are not permanent.**

Heroku rotates credentials periodically and updates applications where this database is attached.

Host	ec2-34-230-153-41.compute-1.amazonaws.com
Database	d79mtf79npaf4h
User	wpepkodmgckzze
Port	5432
Password	[redacted]
URI	[redacted]
Heroku CLI	heroku pg:psql postgresql-cylindrical-57363 --app damp-eyrie-64951

Fig. 3. PostgreSQL Database Credentials

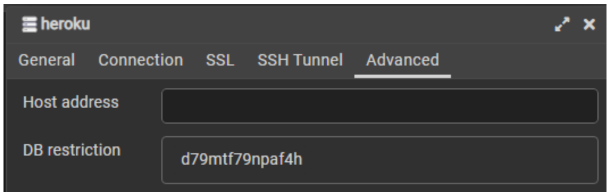


Host name/address	ec2-34-230-153-41.compute-1.amazonaws.com
Port	5432
Maintenance database	d79mtf79npaf4h
Username	wpepkodmgckzze
Kerberos authentication ?	○
Role	
Service	

Fig. 4. The 'Register-Server' Interface in pgAdmin 4.

IV. Discussion

Due to careful consideration during during the design phases of the project, little needed to be adjusted for the final database. For example, much of the data sourced from the Kaggle data set was already organized in a way that aligned with the original intent of our design. Furthermore, the model was well-



Host address

DB restriction d79mtf79npaf4h

Fig. 5. Limit Visibility To The Football Database Only

normalized in early drafts, so little time had to be delegated to re-structuring the data during the later stages of the project. Establishing a well-normalized model early in the project timeline also prevented functionality issues and unexpected behaviors while preparing queries to demonstrate the database. These queries, combined with the high dimensionality of the data, enabled us to make meaningful discoveries about the data, such as how the COVID-19 pandemic affected the amount of games per league, which teams had the highest rankings, and which players scored the most points per game on average.

One area of difficulty we encountered was during the data processing step. To add additional dimensionality to the data and more details about each Player, information was collected from external sources to include each player's nationality and year they were born, and add team managers to the database [3]. This data was aggregated and temporarily held in a playerDetails table for processing. The playerDetails table consisted of a player's name, three-character country code, and year born. Before combining the Players and playerDetails tables, duplicate player names were removed from the playerDetails table, and an additional table [4] was used to match the country code in playerDetails with the full name of the country, for data presentation reasons. The player name in playerDetails was used as the joining key to merge with the Players table. Many of the player names in the Players table included special characters that were not UTF-8, which caused encoding-related issues when merging the two tables. There were multiple instances where two datasets would spell player names with and without accents or other special characters, so when the data was merged an exact match for a player's name wasn't always found in the playerDetails table. This led to a Players table where many records did not have a nationality or birth year, even though certain players did exist in the playerDetails table but with different encoding which wasn't recognized as an exact match. To remedy this, both data sets were converted to Latin-1 in Pandas before merging the two tables and the special characters were manually replaced within Excel with a UTF-8 version of the character. For example, the character ”é” was replaced with an ”e”, using Excel's ”replace all” feature. This adjustment in the data found matches for a larger portion of players, but since the data was aggregated from two different sources, there were still many players in the Players table that did not exist in the playerDetails table. Considering that the educational nature of this project allowed for relaxing the veracity of the data, the remaining player records that did not have a nationality or birth year value were assigned one randomly from the countries and birth years in the playerDetails table.

Another difficulty encountered during data processing was collecting and adding Manager information to the database. When it was originally decided to add this information to the database, it seemed best to manually source and insert all the team managers for the database's time period. Since there were only about 150 teams and 7 seasons, the task didn't appear to be too complex or time-consuming. Further, considering the lack of data mining expertise among the project team, it seemed that developing an alternative approach to aggregating this data would be more costly, outside the scope of the database systems course, and not conducive to meeting the project deadlines. What was not anticipated was the frequency and complexity of team manager changes, which led to this data gathering task being much more time-intensive than expected. In retrospect, it would have been beneficial to develop another, more efficient way to collect this information in the database.

While deciding what platform would be best to host the database, access issues were encountered when connecting to a server with Amazon Web Services Relational Database Service (AWS RDS). Originally, the database was proposed to be hosted on an AWS Elastic Compute Cloud (EC2) instance, but AWS RDS seemed much more performant while still being free. However, while attempting to connect the RDS instance, non-owner users weren't given read and write access unless all IPv4 clients were allowed to gain full access to the instance, imposing a data security risk. Heroku's online server instance, on the contrary, has a more straightforward approach to access control, as discussed in the Methodology section. The app-centric Heroku platform provided services to help with quick prototyping. A Flask-based web user interface was created to provide easy data manipulation utilizing the Python Flask-SQLalchemy module to directly query the database. For the limited amount of project time, the web-based user interface only has the capability to add new Manager records to the database, but additions and deletions are consistent for all tables.

The most surprising aspect of the project was how many issues would arise solely from the actual data, and not how the data was modeled. For example, adding Player nationality and birth year didn't seem to be nearly as troublesome of a task as it ended up being. It was unforeseen that adding this information would be as difficult as it was, specifically with encountering multiple encoding and parsing errors and anomalies. These errors were encountered multiple times not only while merging playerDetails with the Players table but also while loading the data into the database through Heroku and PG Admin.

Another surprising issue was encountered during the later stages of project development where it was realized that a Player could not be traced to a specific Team at any given time. This was mostly due to the nature of the football industry and the source data, in which it is very common for players to be traded, switching team affiliation multiple times throughout a season. Thus, even finding a team roster for a specific season would not be sufficient information since data for this aspect of the sport is not static. Roster information would have to be collected for each team for each game, or each team and the duration each player had played for each team, which would have amounted to thousands of data points. In the initial stages of this project, it was not recognized that the database wouldn't be able to report this information and was an unexpected discovery encountered while drafting queries for the database. As a seemingly essential and principal data point that would be very reasonable information for a user to analyze, this issue proved to be a major shortcoming of the data and the resulting relational model. Given this deficiency was found during the late stages of project development and the approaching deadline, it was decided to leave the data as reported from the Kaggle source, and not model player team information in the database. Considering the number of issues uncovered while adding two attributes to the Players table, it was certain that attempting to add this player team data at such a late stage in the project would be more problematic than the problem it would seek to address. If this issue was recognized in the initial stages of the project, the database may have been designed differently and with additional external data so the database would be able to temporally join each Player to their respective Team.

The level of complexity needed to connect a Season to a Player, or Season to a Team was another unexpected reality of the database. To join either of these two tables, it would be necessary to join through the Games table as well. This was recognized as a trade-off between the normalization of the data, and the database's ease of use. Though this was not a large issue and didn't largely affect the database's practicality, it is something that could have been addressed in the earlier stages of project development and possibly avoided if the data was structured differently.

Another unanticipated realization during project development was how complex it would be to present Team rankings for a Season based on match points. The database holds information for each Season's champion, but an interesting and commonly used statistic of the sport is to rank teams based on match points acquired throughout the Season. It was decided that this information would be beneficial to analyze, thus this data was chosen as an example query for the database. Though this data was available in the database, the query proved to be more complicated than anticipated based on the amount of aggregation needed to generate the rankings.

References

[1] "The Football Landscape Vision Report 2021", FIFA, "https:// publications.fifa.com /en/vision-report-2021 /the-football-landscape/", 2021.

[2] "Football Database - Football and Betting Statistics of the European Top5 Leagues", Technika148, "https:// www.kaggle.com/ datasets/ technika148/ football-database", Last Time Updated 27 August 2021.

[3] "Online football betting," The Football Site. [Online]. Available: https:// www.soccerbase.com/, 2022.

[4] "2022-2023 big 5 European Leagues Nationalities," FBref.com. [Online]. Available: https://fbref.com/en/comps/Big5/nations/Big-5- European -Leagues-Nationalities, 2022.

[5] "Football Transfers, Rumours, Market Values and News," Transfermarkt. [Online]. Available: https://www.transfermarkt.us/, 2022.

[6] "Heroku Platform", Salesforce, [Online], "https://www.heroku.com/ platform", 2022.

[7] "Heroku Documentation", Salesforce, [Online], "https://devcenter .heroku.com /categories/reference", 2022.

[8] "Amazon Elastic Compute Cloud: User Guide for Linux Instances", Amazon Web Services, Inc., "https:// docs.aws.amazon.com/ pdfs/ AWSEC2/latest/UserGuide/ec2-ug.pdf", 2022.