Prepared by:

Nilufar Isakova

Abigail Parra

5-05-17

CS 5318 Database Management Systems

University of Houston- Downtown

Soccer Glimpse Database System

Table of Contents

[Abstract 2](#_Toc481785216)

[Introduction 3](#_Toc481785217)

[Functions 3](#_Toc481785218)

[Administrator Use Cases 3](#_Toc481785219)

[Customer Use Cases 11](#_Toc481785220)

[Entity Relationship Diagram 15](#_Toc481785221)

[Relational Model 16](#_Toc481785223)

[The Competition Relation 16](#_Toc481785224)

[The Team Relation 16](#_Toc481785225)

[The Player Relation 17](#_Toc481785226)

[The Fixture Relation 18](#_Toc481785227)

[The Result Relation 19](#_Toc481785228)

[The League\_table Relation 19](#_Toc481785229)

[The Champions League\_table Relation 20](#_Toc481785230)

[The Manager Relation 21](#_Toc481785231)

[The Stadium Relation 22](#_Toc481785232)

[Database Functionalities 24](#_Toc481785233)

[BCNF Verification 30](#_Toc481785234)

[Prototype’s Technical Specifications 30](#_Toc481785235)

[Time Table 30](#_Toc481785236)

[Stored procedures for all functionalities 31](#_Toc481785237)

[Test Records 49](#_Toc481785238)

[Conclusion 50](#_Toc481785239)

[References 51](#_Toc481785240)

# 

# Abstract

Soccer Glimpse is a web based application soccer fans can use to view scores, fixtures, information about competitions, teams, players, managers, and stadiums. This system is modeled after popular sports websites such as ESPN Sports. The application’s database will be populated using the football-data RESTful API (application programming interface); an open source that serves football (Soccer) data and makes it easy-to-use for free. As a result, the users of Soccer Glimpse will only be able to access information about certain competitions that are available in the football-data API. The system will allow the website administrator to add, modify and delete data from the database. Additionally, the user will be able to search information based on distinct filters as well as to keep up to date with the latest scores and upcoming fixtures.

# Introduction

This document describes the requirements, design, implementation and testing of the Soccer Glimpse. Soccer Glimpse application's database contains data acquired from the football-data RESTful API. Python programming language is used to retrieve football data with API. MAMP is used as a local server and PostgreSQL Database Management System is used as a database, we used PGAdmin4 as an interface for our database. Users will be able to view information about the teams, fixtures, stadiums and managers of the soccer team.

# Functions

## Administrator Use Cases

|  |
| --- |
| **UC1- Add new competition**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “competition” table and the “insert” option. 6. Output: System prompts administrator to enter information in each of the competition table fields. 7. Output: System displays “Table will be updated, click yes to continue”. 8. Input: Administrator clicks on “yes” option. 9. Output: The system checks to verify that the entered information does not already exist in the system. 10. If the data is not a duplicate the new competition is added to the system and is displayed including the generated ID field. |
| **UC2- Add new team**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “team” table and the “insert” option. 6. Output: System prompts administrator to enter information in each of the team table fields. 7. Output: System displays “Table will be updated, click yes to continue”. 8. Input: Administrator clicks on “yes” option.pla 9. Output: The system checks to verify that the entered information does not already exist in the system. 10. If the data is not a duplicate the new team is added to the system and is displayed including the generated ID field. |
| **UC3- Add new player**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “player” table and the “insert” option. 6. Output: System prompts administrator to enter information in each of the player table fields. 7. Output: System displays “Table will be updated, click yes to continue”. 8. Input: Administrator clicks on “yes” option. 9. Output: The system checks to verify that the entered information does not already exist in the system. 10. If the data is not a duplicate the new player is added to the system and is displayed including the generated ID field. |
| **UC4-Add new league\_table entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “league\_table” table and the “insert” option. 6. Output: System outputs “Please enter the name of competition you would like to create league table for.” 7. Input: Administrator enters the name of the competition. 8. Output: The system verifies that there is not an existing league\_table entry for the entered competition. 9. Output: System prompts administrator to enter information in each of the league\_table fields. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: New league\_table is added to the system and is displayed including the generated ID field. |
| **UC5- Add new fixture**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “fixture” table and the “insert” option. 6. Output: System prompts administrator to enter information in each of the fixture table fields. 7. Output: System displays “Table will be updated, click yes to continue”. 8. Input: Administrator clicks on “yes” option. 9. Output: The system checks to verify that the entered information does not already exist in the system. 10. If the data is not a duplicate the new fixture is added to the system and is displayed including the generated ID field. |
| **UC6- Add new stadium**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “stadium” table and the “insert” option. 6. Output: System prompts administrator to enter information in each of the stadium table fields. 7. Output: System displays “Table will be updated, click yes to continue”. 8. Input: Administrator clicks on “yes” option. 9. Output: The system checks to verify that the entered information does not already exist in the system. 10. Output: If the data is not a duplicate the new stadium is added to the system and is displayed including the generated ID field. |
| **UC7- Add new manager**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the “manager” table and the “insert” option. 6. Output: System prompts administrator to enter information in each of the manager table fields. 7. Output: System displays “Table will be updated, click yes to continue”. 8. Input: Administrator clicks on “yes” option. 9. Output: The system checks to verify that the entered information does not already exist in the system. 10. Output: If the data is not a duplicate the new manager is added to the system and is displayed including the generated ID field. |
| **UC8- Modify competition**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the competition table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC9- Modify team**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the team table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC10- Modify player**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the player table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC11- Modify league\_table**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the league\_table table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC12- Modify fixture**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the fixture table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC13- Modify stadium**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the stadium table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC14- Modify manager**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the manager table and the “modify” option. 6. Output: System displays selected table and a search field. 7. Input: The administrator may search for the record to be modified by name or ID number or select it from the table. 8. Output: The record is displayed and the fields become editable. 9. Input: The user modifies the desired fields of the record. 10. Output: System displays “Table will be updated, click yes to continue”. 11. Input: Administrator clicks on “yes” option. 12. Output: The system verifies that the modification does not make the record a duplicate. 13. Output: If the data is not a duplicate the record is updated in the system and is displayed. |
| **UC15- Delete competition entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the name of the competition table and the “delete” option. 6. Output: System displays the competition table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option. 11. Output: Record is deleted from the system and the updated table is displayed. |
| **UC16- Delete team entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the team and the “delete” option. 6. Output: System displays the team table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option.   11. Output: Record is deleted from the system and the updated table is displayed. |
| **UC17- Delete player entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the player table and the “delete” option. 6. Output: System displays player table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option.   11. Output: Record is deleted from the system and the updated table is displayed. |
| **UC18- Delete league\_table entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the name of the league\_table table and the “delete” option. 6. Output: System displays the league\_table table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option.   11. Output: Record is deleted from the system and the updated table is displayed. |
| **UC19- Delete fixture entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the name of the fixture table and the “delete” option. 6. Output: System displays fixture table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option.   11. Output: Record is deleted from the system and the updated table is displayed. |
| **UC20- Delete stadium entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the stadium table and the “delete” option. 6. Output: System displays stadium table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option.   11. Output: Record is deleted from the system and the updated table is displayed. |
| **UC21- Delete manager entry**   1. Input: The administrator logs in the system. 2. Output: The menu options are displayed. 3. Input: Administrator selects “update data” 4. Output: The system displays base table names and “insert”, “delete”, “modify” options. 5. Input: Administrator selects the manager table and the “delete” option. 6. Output: System displays manager table and a search field. 7. Input: The administrator may search for the record to be deleted by name or ID number or select it from the table. 8. Input: The user clicks on “delete” button. 9. Output: System displays “Table will be updated, click yes to continue”. 10. Input: Administrator clicks on “yes” option.   11. Output: Record is deleted from the system and the updated table is displayed. |

## Customer Use Cases

|  |
| --- |
| **UC1- Search for competition by name**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: The user clicks on the search box and types in the desired competition name. 4. Output: If the competition exists in the system, then the options for the competition are displayed. 5. Input: The user selects the menu option of what they want to view about the competition. (eg. Fixtures) 6. Output: The system displays the requested information. |
| **UC2- Search for team by name**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: The user clicks on the search box and types in the desired team name. 4. Output: If the team exists in the system, then the options for the team are displayed. 5. Input: The user selects the menu option of what they want to view about the completion. (eg. players) 6. Output: The system displays the requested information.   **Join operation between team and manager is performed to display the team information along with manager name.** |
| **UC3- Search for player by name**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: The user clicks on the search box and types in the desired player name. 4. Output: If the player exists in the system, then the options for the player’s team are displayed along with a tab for player info. 5. Input: The user selects the menu option of what they want to view about the player. (eg. General Info) 6. Output: The system displays the requested information.   **Join operation between player and team is performed to display the player information along with team name.** |
| **UC6- Search for stadium by name**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: The user clicks on the search box and types in the desired stadium name. 4. Output: If the stadium exists in the system, then the general information about the stadium is displayed   **Join operation between stadium and team is performed to display the stadium information along with team name.** |
| **UC7- Search for manager by name**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: The user clicks on the search box and types in the desired manager name. 4. Output: If the player exists in the system, then the options for the manager’s team are displayed along with a tab for manager info. 5. Input: The user selects the menu option of what they want to view about the manager. (eg. General Info) 6. Output: The system displays the requested information.   **Join operation between manager and team is performed to display the manager information along with team name.** |
| **UC8- View standings of a competition**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User selects the “competition” tab. 4. Output: System shows the league table for the competition, which contains the rank of each team.   **Join operation between league table, competition and team is performed to display the league table information along with the competition name and team name.** |
| **UC9- View teams in a competition**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects competition from menu. 4. Output: Competitions are displayed. 5. Input: The user selects the desired competition and the “teams” option. 6. Output: The system displays the list of teams participating in the competition.   **Join operation between competition and team is performed to display the list of teams where the competition is equal to the searched competition.** |
| **UC10 –View basic stats of a competition**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects competition from menu. 4. Output: The menu options of the competition are displayed. 5. Input: The user selects the “Statistics” option. 6. Output: The system displays the most home goals, most away goals, most wins and most losses*.*   **Aggregate functions max () are performed on the home\_goals, away\_goals, wins and losses fields of the league table view.** |
| **UC11- View fixtures by competition**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects competition from menu. 4. Output: The menu options for the competition are displayed. 5. Input: The user selects the “fixtures & scores” option. 6. Output: The system displays the upcoming matches and results for the competition.   **Join operation between fixtures, fixture\_team, team and competition is performed in order to view fixtures corresponding to a certain competition.** |
| **UC12- View fixtures by week**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User clicks on "fixtures & scores" 4. Input: The user clicks on the week he wants to view the matches or scores for. 5. Output: The system displays the matches and results across competitions for the selected week.   **Join operation between fixtures, fixture\_team, team and competition is performed in order to view fixtures including the team and competition name.** |
| **UC13- View Scores across competitions**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User selects the “fixtures & scores” tab. 4. Output: Shows scores in order from most recent to less recent.   **Join operation between fixtures, fixture\_team, team and competition is performed in order to view fixtures including the team and competition name.** |
| **UC14- View fixtures of a team**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects team from menu. 4. Output: The menu options for the team are displayed. 5. Input: The user selects the “fixtures & scores” option. 6. Output: The system display upcoming matches of the team as well as results of the finished matches.   **Join operation between fixtures, fixture\_team, team and competition is performed in order to view fixtures including the team and competition name.** |
| **UC15- View average manager age and largest capacity of stadiums**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User selects basic stats 4. Output: The system displays youngest manager and largest stadium contained in the database.   **Aggregate functions avg\_age () and max() are performed on the manager dob and stadium capacity attributes respectively.** |
| **UC16- View players of a team**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects team from menu. 4. Output: The menu options for the team are displayed. 5. Input: User selects the “squad” option. 6. Output: The list of players belonging to the team is displayed.   **Join operation between player and team is performed in order to view the players who play for the searched team.** |
| **UC17- View team’s standings in competitions**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects team from menu. 4. Output: The menu options for the team are displayed. 5. Input: User selects the “Standings” option. 6. Output: The rank of the team within the competition(s) it is currently participating in is displayed.   **League Table view is used to display the records where team name is equal to the searched name.** |
| **UC18 –View basic stats of a team**   1. Input: User accesses the system. 2. Output: The homepage is displayed along with the menu tabs. 3. Input: User searches or selects team from menu. 4. Output: The menu options of the team are displayed. 5. Input: The user selects the “Statistics” option. 6. Output: The system displays max player market\_value, youngest player, number of wins, losses and draws.   **Aggregate functions max (), min () are performed on market\_value and dob fields respectively.** |

# Entity Relationship Diagram

# /Users/abbyparra/Documents/Soccer/Documentation/ER Diagram submission 3.pdf

# Relational Model

## The Competition Relation

Each tuple in the competition relation will represent each competition in the Soccer Glimpse system. A tuple of the competition relation will contain the following attributes: name, year, number\_of\_games, number\_of\_teams, and an id which is automatically generated by the database system.

Key constrains

The competition\_id attribute is the primary key of the competition relation. The name attribute is a candidate key since each competition has a unique name.

Referential integrity constrains:

There are no foreign key fields in this relation.

Null constrains:

The id, name, year, number\_of\_games, number\_of\_teams in the Competition relation cannot be Null.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| name | text not null |
| year | integer not null |
| number\_of\_games | integer not null |
| number\_of\_teams | integer not null |

## The Team Relation

Each tuple in the team relation will represent each team. A tuple of the team relation will contain the following attributes: a team\_id, manager\_id, name, code, squad\_market\_value. The team\_id will be generated automatically by the database system.

Key constrains

The team\_id attribute is the primary key of the team relation.

Referential integrity constrains:

The manager\_id attribute in team relation takes values from the manager\_id attribute in the manager relation.

Null constrains:

The team\_id, manageID, name, code attributes in the team relation cannot be Null. The only attribute on the team relation which may contain a Null value is the attribute denoting the squad\_market\_value.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| manager\_id | references Manager(id) not null |
| name | text not null |
| code | text |
| squad\_market\_value | character varying |

## The Player Relation

Each tuple in the player relation will represent each player. A tuple of the player relation will contain the following attributes: a player\_id, team\_id, name, position, jersey\_number, dob, nationality, contracUnit, market\_value. The player\_id will be generated automatically by the database system.

Key constrains

The player\_id attribute is the primary key of the player relation.

Referential integrity constrains:

The team\_id attribute in player relation takes values from the team\_id attribute in the team relation.

Null constrains:

The player\_id, team\_id, and name attributes in the player relation cannot be Null. The only attributes on the team relation which may contain a Null values are the attributes denoting the position, dob, nationality, contract\_until and market\_value.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| team\_id | references Team(id) not null |
| name | text not null |
| position | text |
| jersey\_number | character varying |
| dob | date |
| nationality | text |
| contract\_until | date |
| market\_value | character varying |

## The Fixture Relation

Each tuple in the fixture relation will represent each fixture in the Soccer Glimpse system. A tuple of the fixture relation will contain the following attributes: competition\_id, away\_team, home\_team, result, status, date, matchday, and an id which is automatically generated by the database system.

Key constrains

The fixture\_id attribute is the primary key of the fixture relation.

Referential integrity constrains:

The competiton, away\_team, home\_team and result take values from the competition, team and result relation respectively.

Null constrains:

The fixture\_id, competition\_id, away\_team, home\_team, and status in the fixture relation cannot be Null.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| competition\_id | references Competition(id) not null |
| home\_team | references Team(id) not null |
| away\_team | references Team(id) not null |
| result | references Result(id) |
| status | character varying |
| date | date |
| machday | integer not null |

## The Result Relation

Each tuple in the result relation will represent each result in the Soccer Glimpse system. A tuple of the result relation will contain the following attributes: fixture\_id, goals\_home, goals\_away, odds\_away\_win, odds\_home\_win and an id which is automatically generated by the database system.

Key constrains

The ID attribute is the primary key of the fixture relation.

Referential integrity constrains:

The fixture\_id will take values from the Fixture relation.

Null constrains:

The ID and fixture\_id may in the Result relation cannot be Null.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| fixture\_id | references Fixture(id) not null |
| goals\_home | integer |
| goals\_away | integer |
| odds\_away\_win | float |
| odds\_home\_win | float |

## The League\_table Relation

Each tuple in the league\_table relation will represent each league table entry in the database system. Each competition will have a distinct view of the league table, which only includes the entries belonging to the selected competition. The competitions that will have this version of the league table are: Premier League, Serie A, Primera Division, and Bundesliga. A tuple of the league\_table relation will contain the following attributes: a league\_id, competition, position, games\_played, team, wins, draws, losses, goals, goals\_against, goal\_difference and points. The ID will be automatically generated by the database system.

Key constrains

The ID attribute is the primary key of the league\_table relation.

Referential integrity constrains:

The competition\_id attribute in league\_table relation takes values from the competition\_id attribute in the competition relation.

Null constrains:

None of the attributes in the league\_table relation may be null, all the fields must contain information regarding each team in the competition.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| competition\_id | references Competition(id) not null |
| position | character varying |
| games\_played | integer not null |
| team\_id | references Team(id) not null |
| wins | integer not null |
| draws | integer not null |
| losses | integer not null |
| goals | integer not null |
| goals\_against | integer not null |
| goal\_difference | integer not null |
| points | integer not null |

## The Champions League\_table Relation

Each tuple in the Champions league\_table relation will represent the league table entries belonging the Champions League competition. The league table of this competition has a different set of fields, since participating teams are divided into groups. A tuple of the Champions league\_table relation will contain the following attributes: a league\_id, competition, group, position, team, games\_played, goals, goals\_against, goal\_difference and points. The ID will be automatically generated by the database system.

Key constrains

The ID attribute is the primary key of the Champions league\_table relation.

Referential integrity constrains:

The competition attribute in the Champions league\_table relation takes values from the competition\_id attribute in the competition relation.

Null constrains:

None of the attributes in the Champions league\_table relation may be null, all the fields must contain information regarding each team in the competition.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| competition\_id | references Competition(id) not null |
| position | integer not null |
| group | ENUM (‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘H’) |
| games\_played | integer not null |
| team\_id | references Team(id) not null |
| goals | integer not null |
| goals\_against | integer not null |
| goal\_difference | integer not null |
| points | integer not null |

## The Manager Relation

Each tuple in the Manager relation will represent the manager of a soccer team. A tuple of the Manager relation will contain the following attributes: a manager\_id, team, name, and age. The ID will be automatically generated by the database system.

Key constrains

The ID attribute is the primary key of the Manager relation.

Referential integrity constrains:

The team attribute takes values from the team\_id in the Team Relation.

Null constrains:

The ID, name and team cannot be null in the Manager relation.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| team\_id | references Team(id) not null |
| name | text not null |
| age | integer |

## The Stadium Relation

Each tuple in the Stadium relation will represent the stadium that belongs to a soccer team. A tuple of the Stadium relation will contain the following attributes: a stadium\_id, team, name, location and capacity. The ID will be automatically generated by the database system.

Key constrains

The ID attribute is the primary key of the Stadium relation.

Referential integrity constrains:

The team attribute takes values from the team\_id in the Team Relation.

Null constrains:

The ID, name and team cannot be null in the Stadium relation.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| team\_id | references Team(id) not null |
| name | text not null |
| location | text |
| Capacity | integer |

The Competition\_Team Relation

This relation is meant to associate a competition with all teams participating in it. Some of the teams in the database system may participate in more than one competition. It contains pairs of competition\_id and team\_id attributes.

Key constrains

Both the competition\_id and the team\_id attribute are primary keys in their native tables. Therefore, each tuple is uniquely identifiable using a combination of these two attributes.

Referential integrity constrains

The team\_id attribute in the Competition\_Team relation takes values from the id attribute in the Team relation. The competition\_id attribute takes values from the id in the competition relation.

Null constraints

None of the values in the Competition\_Team relation can be null.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| competition\_id | references Competition(id) |
| team\_id | References Team(id) |

The League\_table\_Team Relation

This relation is meant to associate league\_table tuples with teams. Some of the teams in the database system may participate in more than one competition, therefore they may be present in more than one league\_table. It contains pairs of league\_id and team\_id attributes.

Key constrains

Both the league\_id and the team\_id attributes are primary keys in their native tables. Therefore, each tuple is uniquely identifiable using a combination of these two attributes.

Referential integrity constrains

The team\_id attribute in the League\_table\_Team relation takes values from the id attribute in the Team relation. The league\_id attribute takes values from the id in the competition relation.

Null constraints

None of the values in the League\_table\_Team relation can be null.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| ID | serial primary key |
| league\_id | references League\_table(id) |
| team\_id | References Team(id) |

The Fixture\_Team Relation

This relation is meant to associate teams participating in a match and the corresponding fixture. A tuple of the Fixture\_Team relation contains home\_team, away\_team, and fixture\_id.

Key constrains

The home\_team, away\_team, and the fixture\_id attributes take values from primary keys in their native tables. Therefore, each tuple is uniquely identifiable using a combination of these three attributes.

Referential integrity constrains

The home\_team, away\_team and Fixture\_Team attributes take values from the id attribute in the Team relation and Fixture relation respectively.

Null constraints

None of the values in the Fixture\_Team relation can be null.

|  |  |
| --- | --- |
| **Attribute** | **Domain** |
| home\_team | references Team(id) |
| away\_team | references Team(id) |
| fixture\_id | references Fixture(id) |

# Database Functionalities

Add new competition

The administrator may add a new competition to the database system. This functionality is performed by inserting the corresponding data into each attribute of the Competition relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Add new team

The administrator may add a new team to the database system. This functionality is performed by inserting the corresponding data into each attribute of the Team relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Add new player

The administrator may add a new player to the database system. This functionality is performed by inserting the corresponding data into each attribute of the Player relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Add new league\_table entry

The administrator may add a new league\_table entry to the database system. This functionality is performed by inserting the corresponding data into each attribute of the league\_table relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Add new fixture

The administrator may add a new fixture to the database system. This functionality is performed by inserting the corresponding data into each attribute of the fixture relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Add new Stadium

The administrator may add a new stadium to the database system. This functionality is performed by inserting the corresponding data into each attribute of the Stadium relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Add new Manager

The administrator may add a new manager to the database system. This functionality is performed by adding the corresponding data into each attribute of the Manager relation. The new row including an automatically generated ID is only created if it did not previously exist in the system.

Modify competition

The administrator may change a competition's information. This functionality is performed by selecting the row to be updated by competition caption. Then the attributes of the Competition relation are updated with the new inputted values.

Modify team

The administrator may change a team's information. This functionality is performed by selecting the row to be updated by team name. Then the attributes of the Team relation are updated with the new inputted values.

Modify player

The administrator may change a player's information. This functionality is performed by selecting the row to be updated by player name. Then the attributes of the Player relation are updated with the new inputted values.

Modify league\_table

The administrator may change a league table's information. This functionality is performed by selecting the row to be updated by team and position. Then the attributes of the league\_table relation are updated with the new inputted values.

Modify fixture

The administrator may change a fixture's information. This functionality is performed by selecting the row to be updated by status and date. Then the attributes of the Fixture relation are updated with the new inputted values.

Modify stadium

The administrator may change a stadium's information. This functionality is performed by selecting the row to be updated by stadium location. Then the attributes of the Stadium relation are updated with the new inputted values.

Modify manager

The administrator may change a manager's information. This functionality is performed by selecting the row to be updated by manager name. Then the attributes of the Manager relation are updated with the new inputted values.

Delete competition

The administrator may delete a competition from the system. This functionality is performed by selecting the competition to be deleted by name. The selected competition and its league table and fixtures will be deleted from the system.

Delete player

The administrator may delete a player from the system. This functionality is performed by selecting the player to be deleted by name. The selected player will be deleted from the system.

Delete fixture

The administrator may delete a fixture from the system. This functionality is performed by selecting the home team, away team and competition names of the fixture. The selected fixture will be deleted from the system.

Delete stadium

The administrator may delete a stadium from the system. This functionality is performed by selecting the stadium to be deleted by name. The selected stadium will be deleted from the system.

Delete manager

The administrator may delete a manager from the system. This functionality is performed by selecting the manager to be deleted by name. The selected manager will be deleted from the system.

Search for competition by name

The user may search for a competition by name. This functionality is performed by selecting the competition by name. The system displays basic information about the selected competition.

Search for team by name

The user may search for a team by name. This functionality is performed by selecting the team by name and joining the team and manager table. The system displays basic information about the selected team, including the name of the manager.

Search for player by name

The user may search for a player by name. This functionality is performed by selecting the player by name. The system displays basic information about the selected player including their age which is a derived attribute.

Search for stadium by name

The user may search for a stadium by name. The stadium will be selected by name and will be joined with the team table. The system displays information about the selected stadium including the home team name.

Search for manager by name

The user may search for a manager by name. The manager will be selected by name and will be joined with the team table. The system displays information about the selected manager including the name of the team he manages.

View standings of a competition

The user may view the league table of a competition. The league table will be a view created for the selected competition. The league\_table, competition and team base tables will be joined.

View teams in a competition

The user may view the teams participating in a competition. The team and the competition base tables will be joined. The system will display the basic information of the teams participating in the selected competition.

View basic stats of a competition

The user may view basic stats of a competition. The attributes of this table will be derived from attributes contained in base tables. This table will show **most goals, most wins, most losses and max squad market value***.*

Most Home Goals

This function calculates the maximum amount of home goals within a competition. This is performed by using the max () aggregate function on the homeGoals attribute of the league table. The system displays the name of the team with the most goals.

Most Away Goals

This function calculates the maximum amount of away goals within a competition. This is performed by using the max () aggregate function on the awayGoals attribute of the league table. The system displays the name of the team with the most away goals.

Most Wins

This function calculates the maximum amount of wins within a competition. This is performed by using the max () aggregate function on the Wins attribute of the league table. The system displays the name of the team with the most wins.

Most Losses

This function calculates the maximum amount of losses within a competition. This is performed by using the max () aggregate function on the Losses attribute of the league table. The system displays the name of the team with the most losses.

View max squad market value

This function calculates the average squad market value of the teams within a competition. This is performed by calculating the max () aggregate function on the squad\_market\_value attribute of the team table.

View fixtures by competition

The user may view the fixtures and results of the teams participating in a competition. The fixtures are selected by competition name. This functionality is performed by joining the competition, teams and fixtures tables (which is done in a view). The upcoming fixtures and results for the selected competition will be displayed.

View fixtures by date

The user may view fixtures and results of teams by week across competitions. The fixtures are selected by date. This functionality is performed by joining the teams and fixtures tables (which is done in a view). The upcoming fixtures and results for the selected week are displayed.

View Scores across competitions

The user may view the past scores contained in the database system in order from most recent to less recent. This is performed displaying ordering by date in descending order. The results of finished matches will be displayed with the most recent first.

Min number of Games

This function calculates the average number of games in the competitions contained in the database system. This functionality is performed using the min () function on the number\_of\_games attribute of the Competition table.

View fixtures & scores of a team

The user may view the fixtures and scores of a team. The fixtures are selected by team name. This functionality is performed by joining the team and fixture table. The fixtures and results of the selected team will be displayed.

View players of a team

The user may view the squad of a team. The players are selected by team name. This functionality is performed by joining the player and team tables (which was already done by creating view). A list of players that play for the selected team will be displayed.

View team’s standings in competitions

The user may view the rank, wins, losses and draws of a team in all of the competitions it is currently participating in. Essentially the user will view all the league table entries a team appears in. This functionality is performed by joining the league\_table, team and competition base tables (which was already done by creating a view).

View basic stats of a team

The user may view basic stats of a team. The attributes of this table will be derived from attributes contained in base tables. This table will show **Wins, Losses, Draws and max player market\_value.**

Number of wins

This function displays the number of wins of the team.

Number of losses

This function displays the number of losses of the team.

Number of draws

This function displays the number of draws of the team.

Average manager age

This function calculates the minimum age of the managers of teams contained in the database system. This is performed by using the avg\_age () aggregate function on the Age attribute.

Largest Capacity

This function calculates the maximum capacity of the stadiums contained in the database system. This is performed by using the max () aggregate function on the capacity attribute of the stadium table.

Count number of fixtures with each status

This function counts the number of fixtures with a finished or scheduled status. This is performed by using the count () aggregate function on the status attribute of the fixture table.

# BCNF Verification

In our Soccer Glimpse database, we have nine tables, since we used an API to retrieve and organize data for our relations by default the tables are in first, second and third form of normalization, therefore the relations are already in normalized form. The relations are divided by making transitive dependency as a primary key in another table. For example, the "competition\_team" table has "team\_id" as a foreign key which can have relation with "team" table's "id" which is a primary key, making the tables(competition\_team & fixture\_team) as mapping table, shown in the ER diagram. As a result the redundant data is eliminated, and only data related to the attribute is stored within the table.

# Prototype’s Technical Specifications

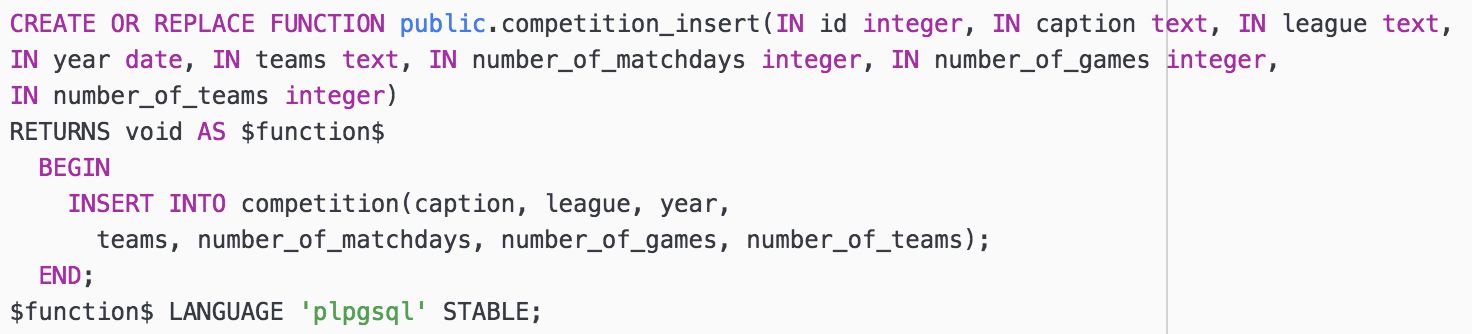
The project will be implemented using the PostgreSQL Database Management System. Also, the application’s source code will be hosted on a remote GitHub repository. Our PostgreSQL database will be populated using the football-data REST API Python Client. Lastly, Python will be used to process JSON objects and insert data into the database system.

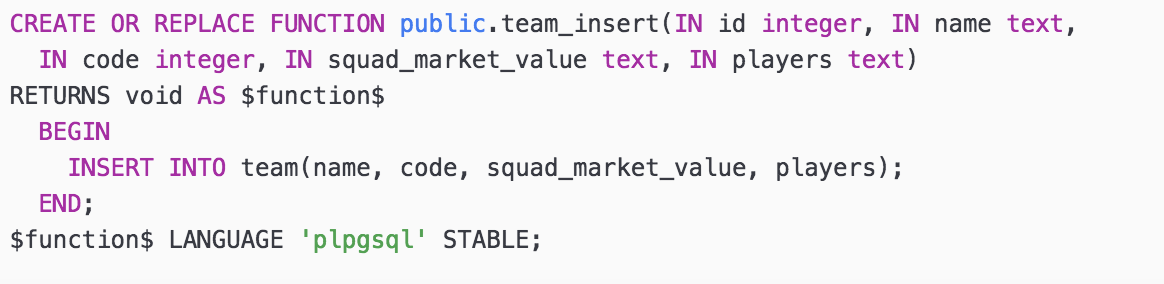
# Time Table

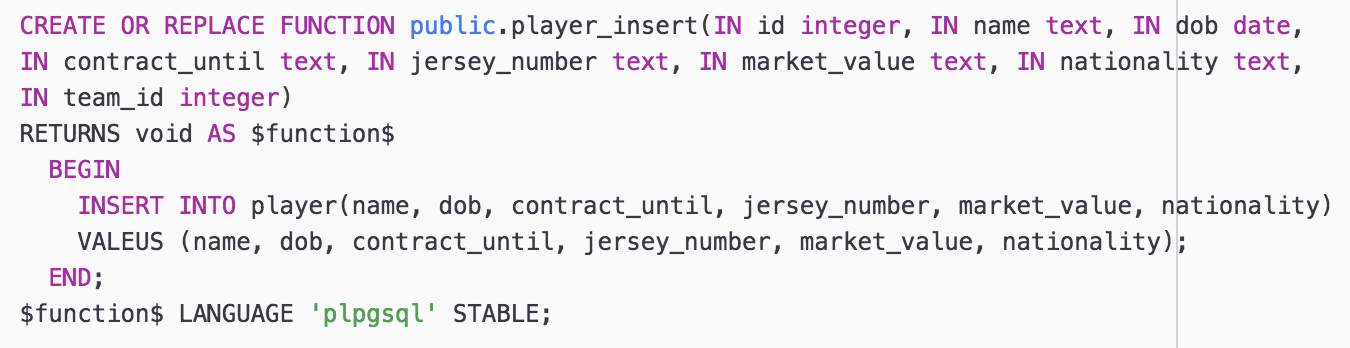
|  |  |  |
| --- | --- | --- |
| **Tasks** | **Completed by** | **Status** |
| Create use cases | Nilufar, Abigail | Complete |
| Complete E/R diagram | Nilufar, Abigail | Complete |
| Formalize relational model | Nilufar, Abigail | Complete |
| Request data from RESTful API | Abigail | Complete |
| Organize JSON data into data frames | Abigail | Complete |
| Acquire data from online sources | Abigail | Complete |
| Create database on PostgreSQL server | Nilufar, Abigail | Complete |
| Implement database tables | Nilufar, Abigail | Complete |
| Implement mapping tables | Abigail | Complete |
| Implement database views | Abigail | Complete |
| Implement stored procedures | Nilufar, Abigail | Complete |
| Create GitHub repository | Abigail | Complete |
| Manage version control | Nilufar, Abigail | Complete |
| Complete and revise documentation | Nilufar, Abigail | Complete |
| Perform black box testing / record outcome | Nilufar, Abigail | Complete |
| Write unit tests for critical components | Nilufar, Abigail | Complete |
| Perform BCNF verification | Nilufar | Complete |

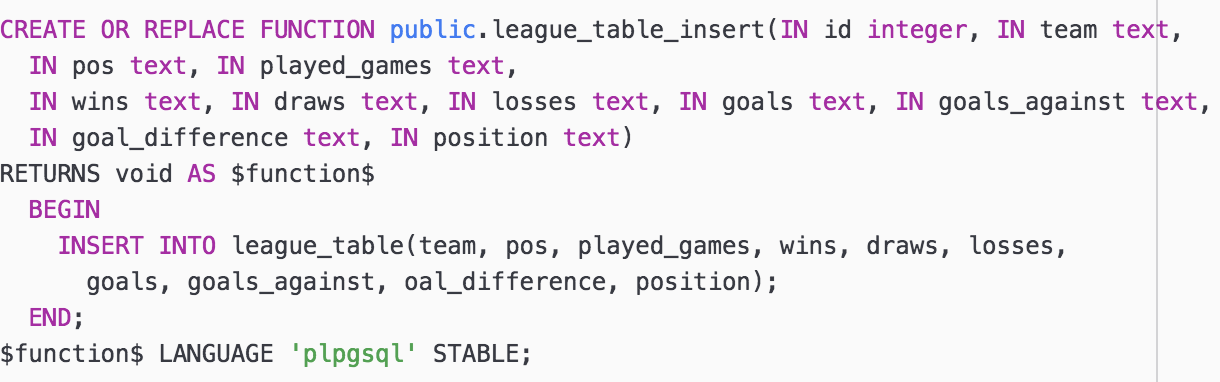
# Stored procedures for all functionalities

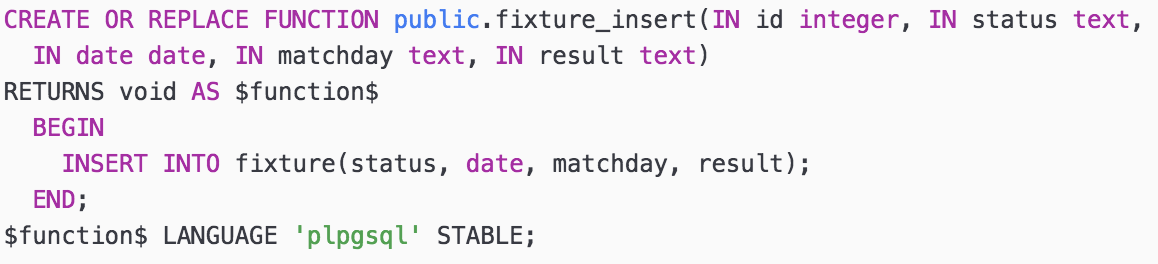
Add new competition



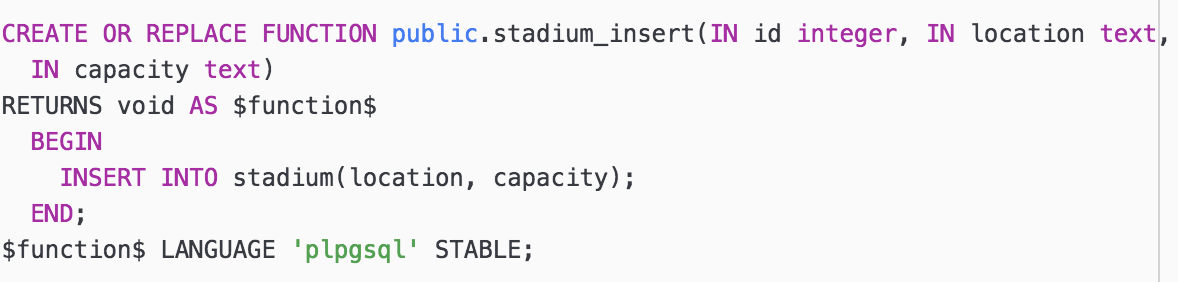
Add new team

Add new player

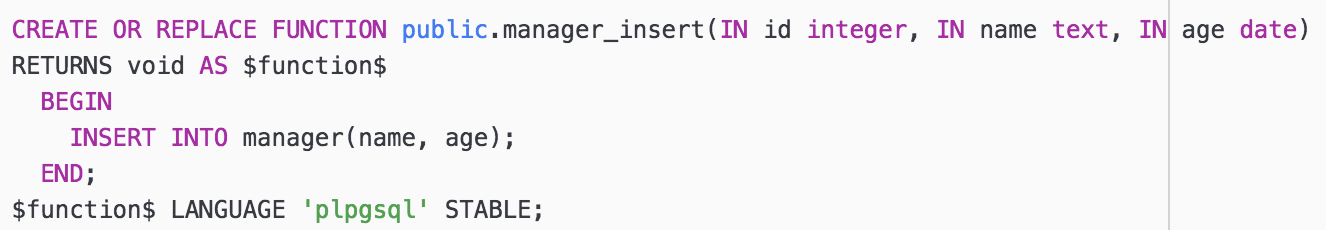
Add new league\_table entry

Add new fixture

Add new Stadium



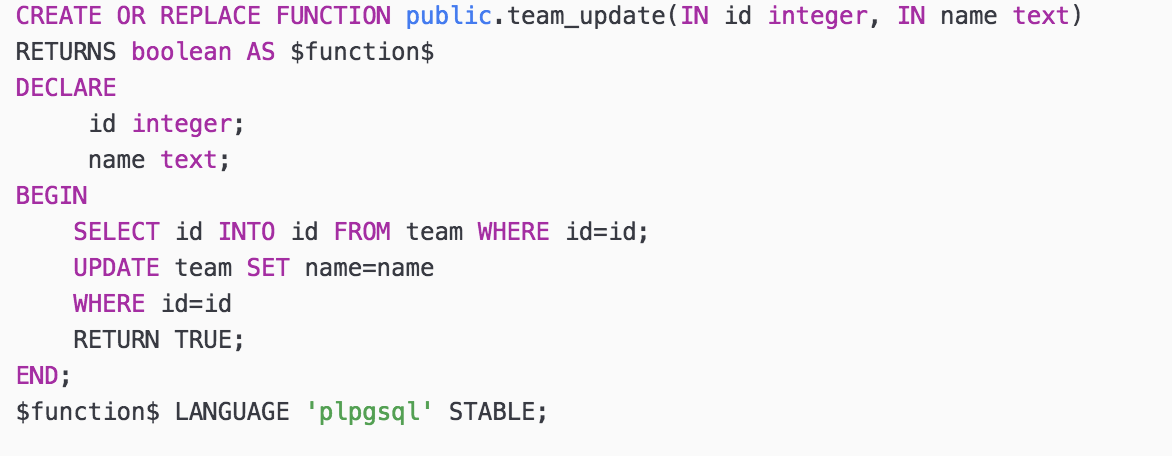
Add new manager



Modify competition



Modify team

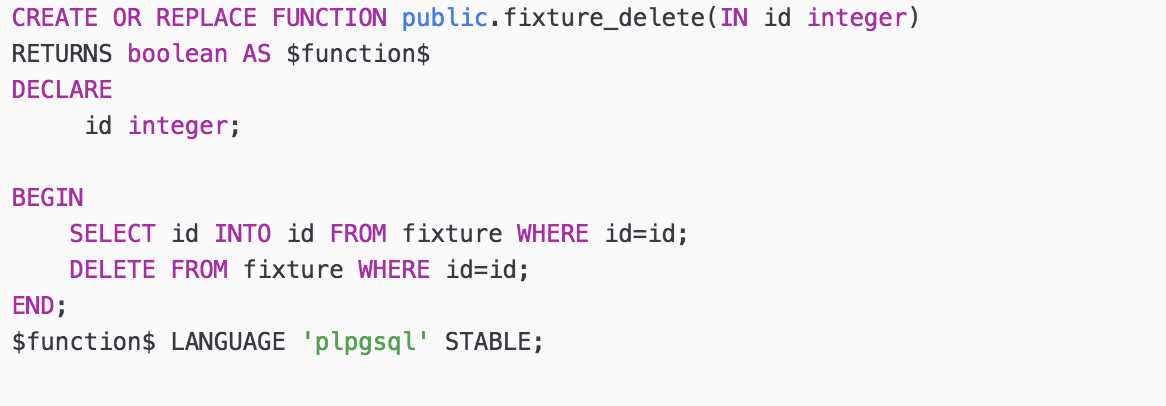


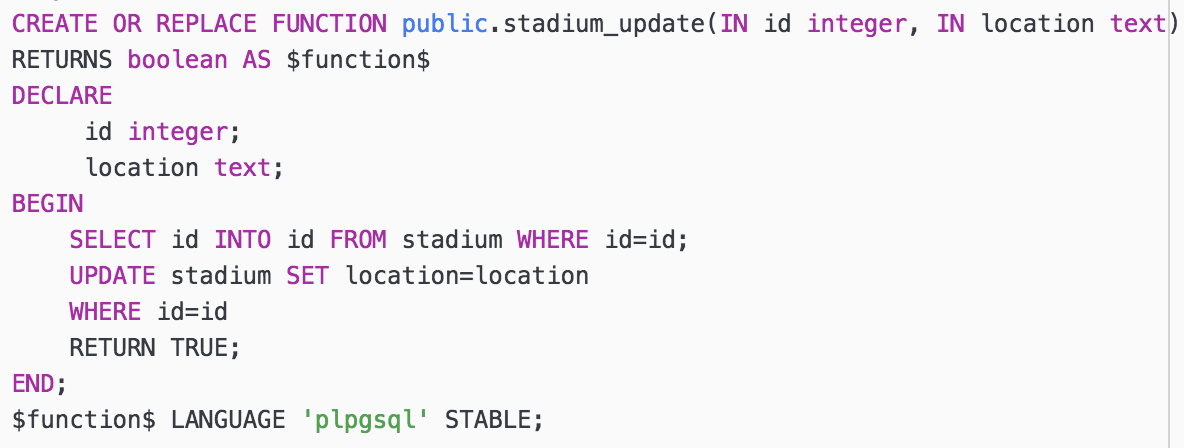
Modify player



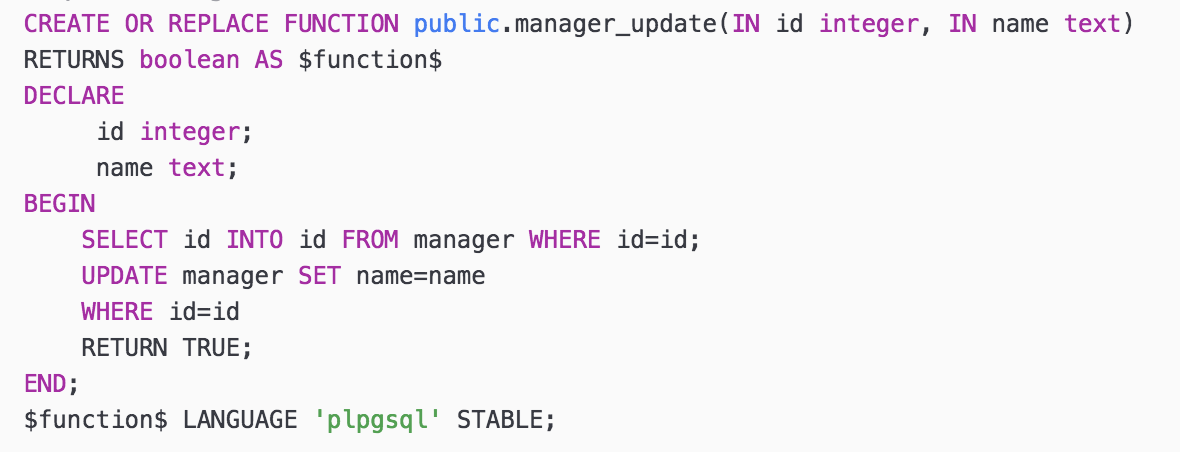
Modify league\_table



Modify fixture

Modify stadium

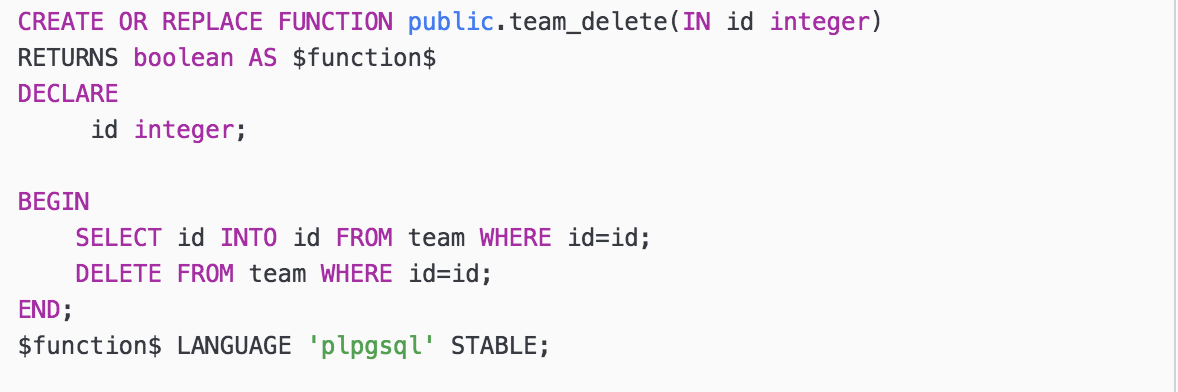
Modify manager



Delete competition



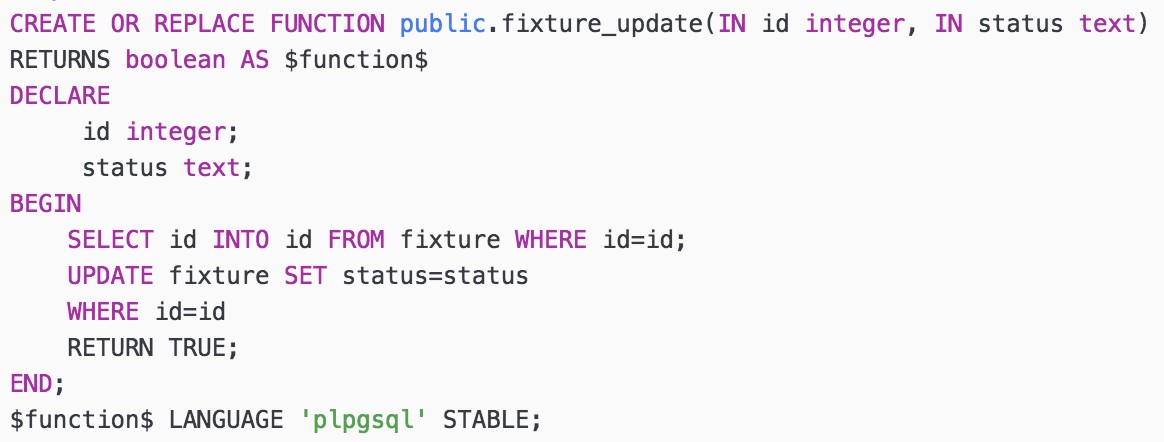
Delete team



Delete player



Delete fixture



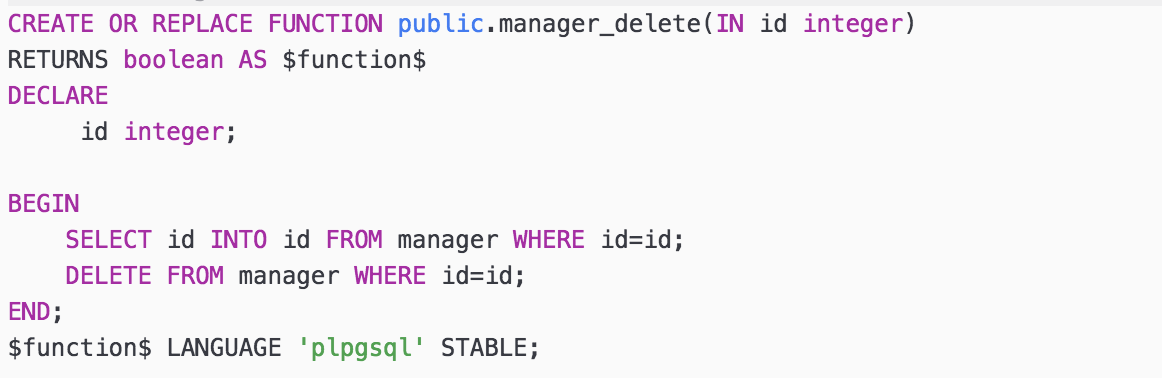
Delete league table



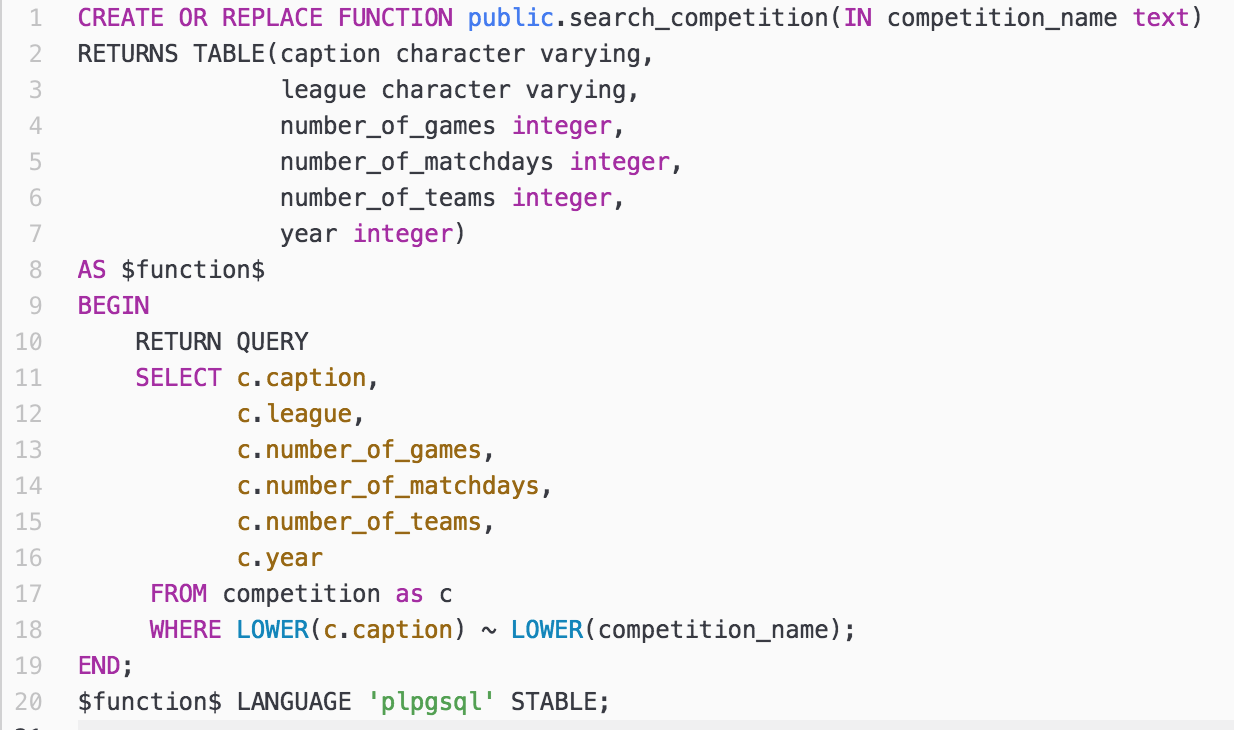
Delete stadium



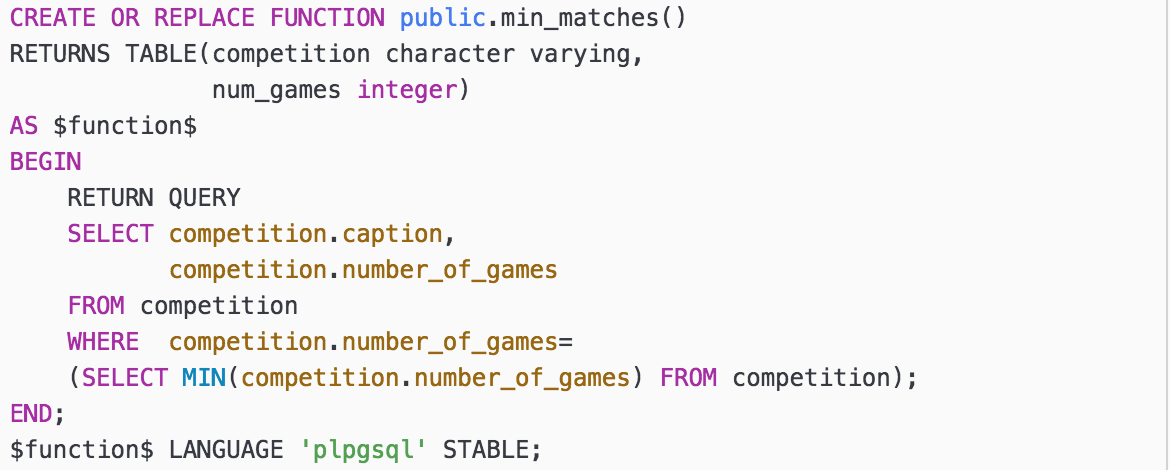
Delete manager



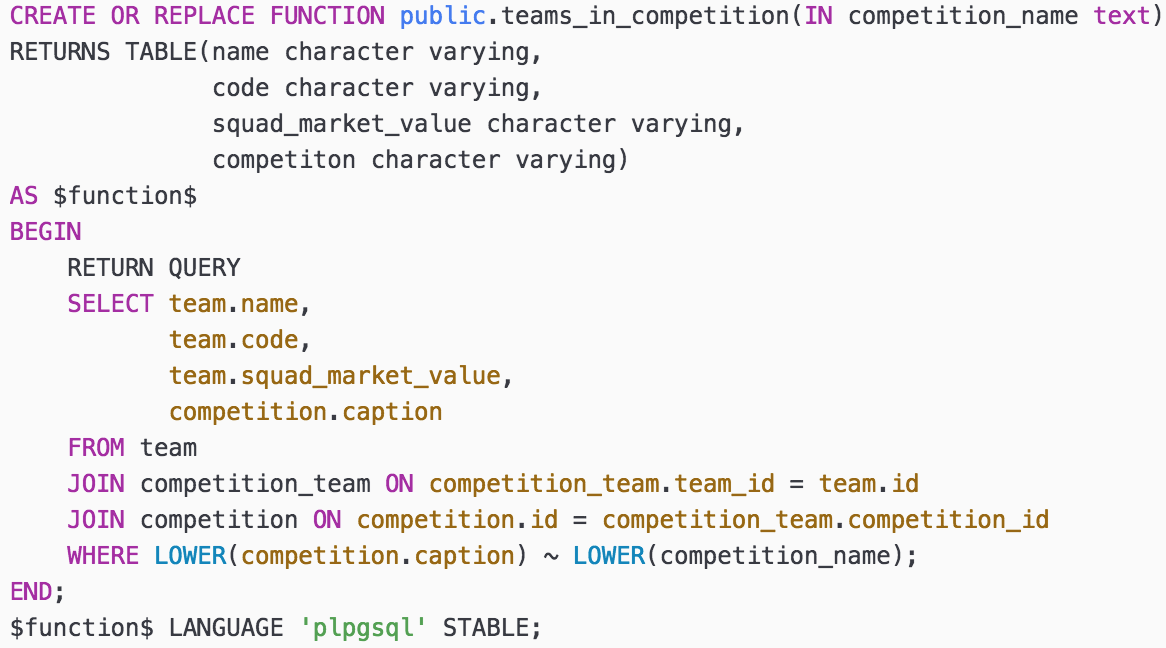
Search competition by name



competition minimum matches



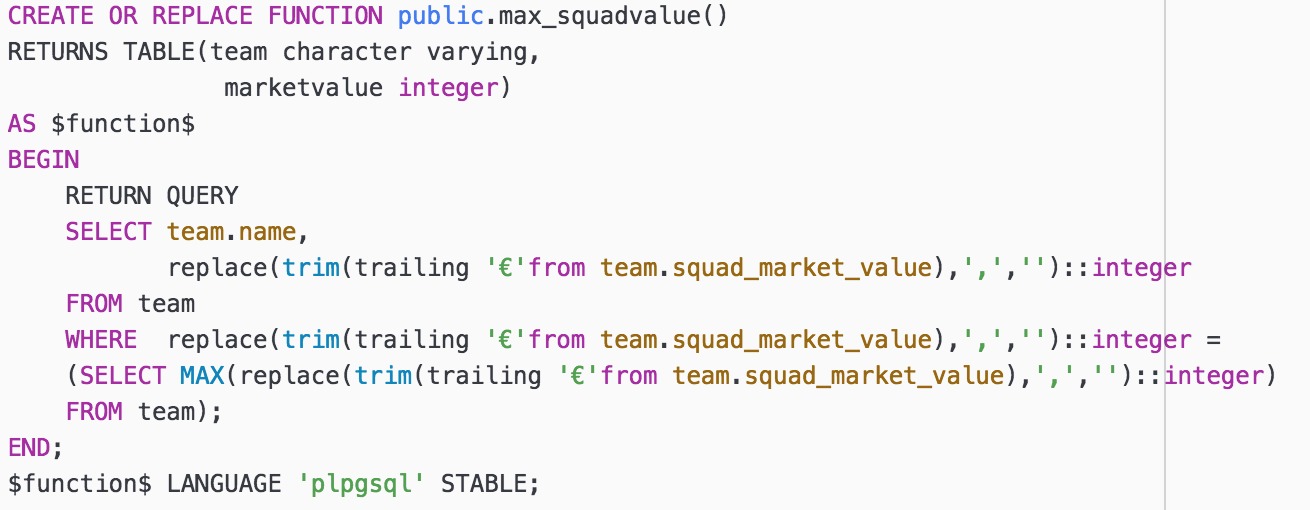
List teams in competition



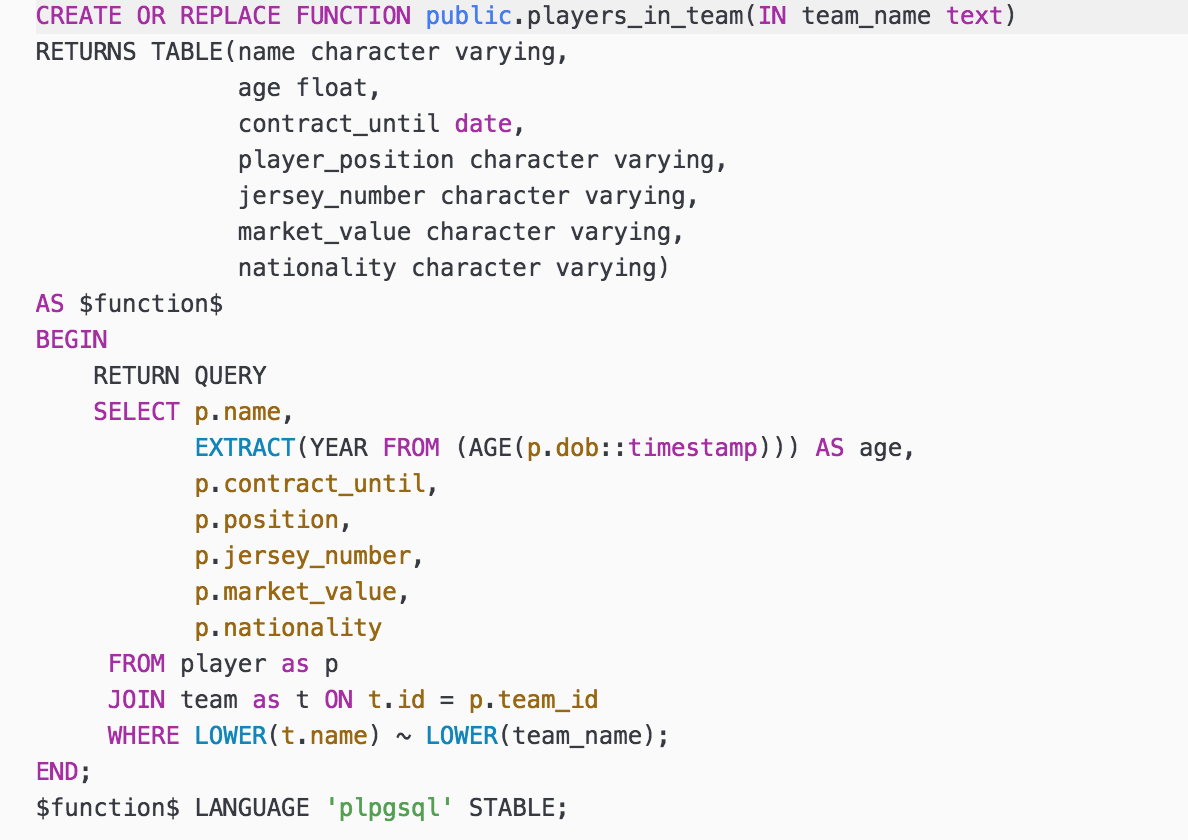
Search team by name



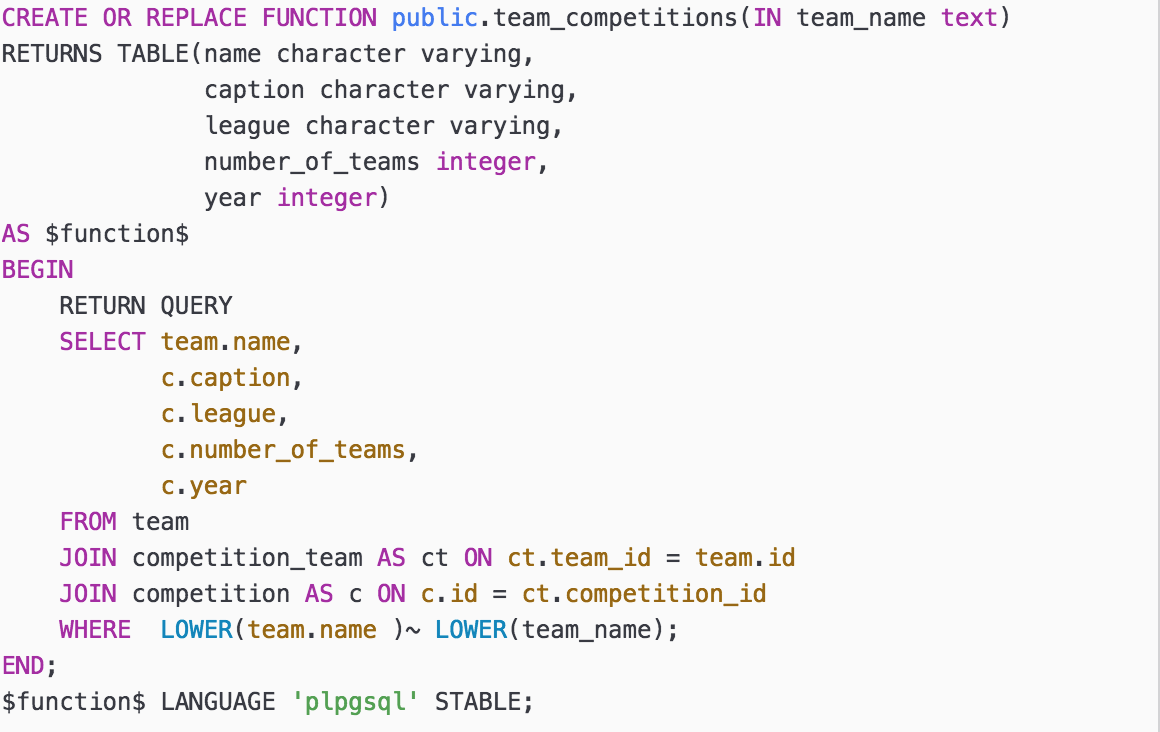
Team max squad market value



List players in team



List competitions team participates in



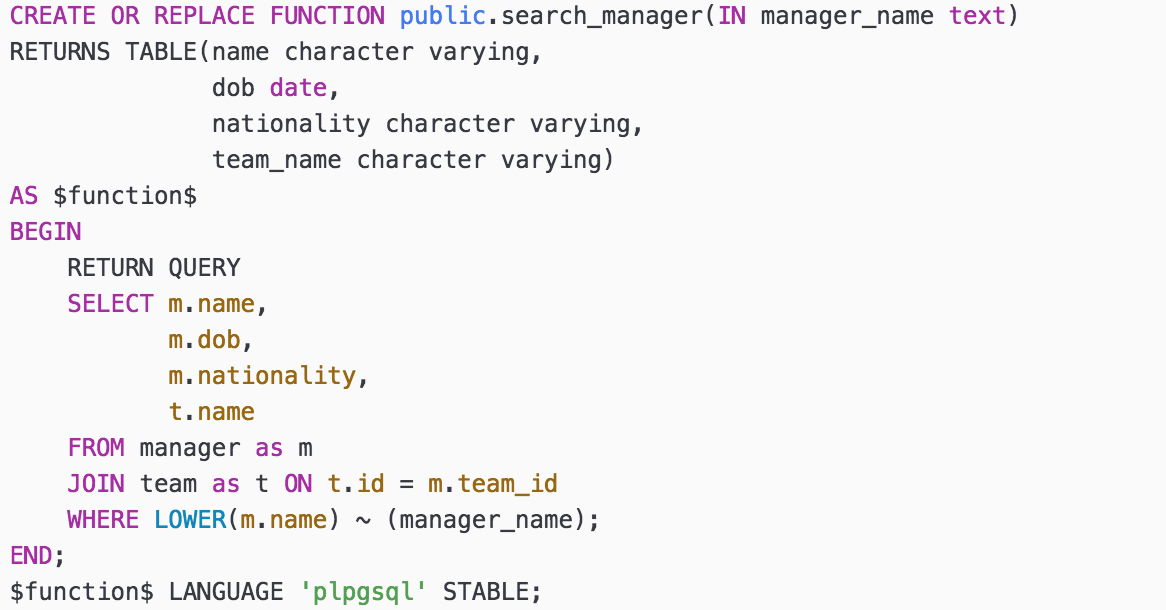
Search player by name



player max market value



Search manager by name



average manager age



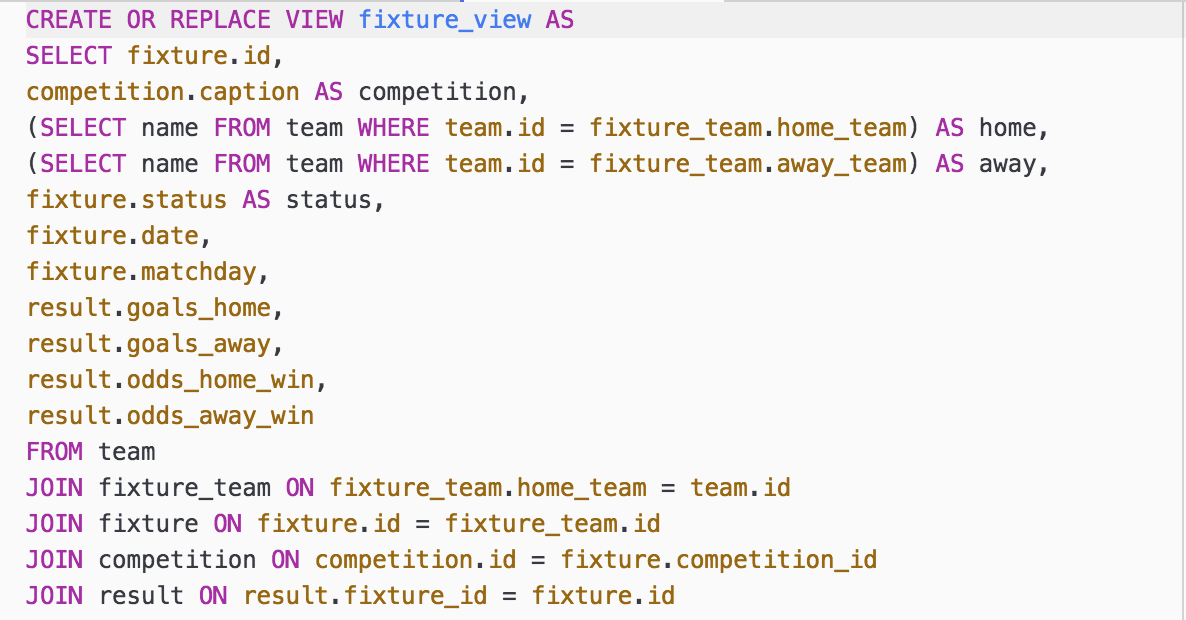
Search stadium by name



stadium max capacity



Fixtures view



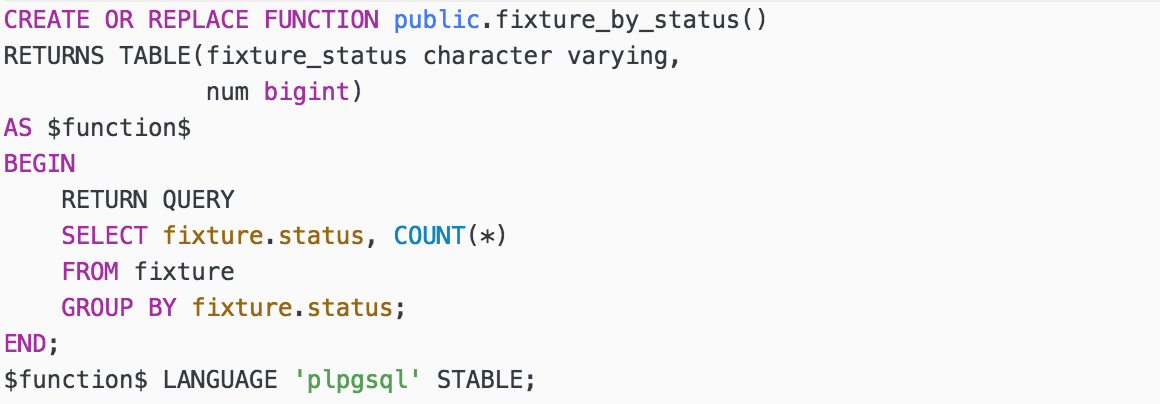
Search fixture by team



Search fixture by date



Fixtures by status



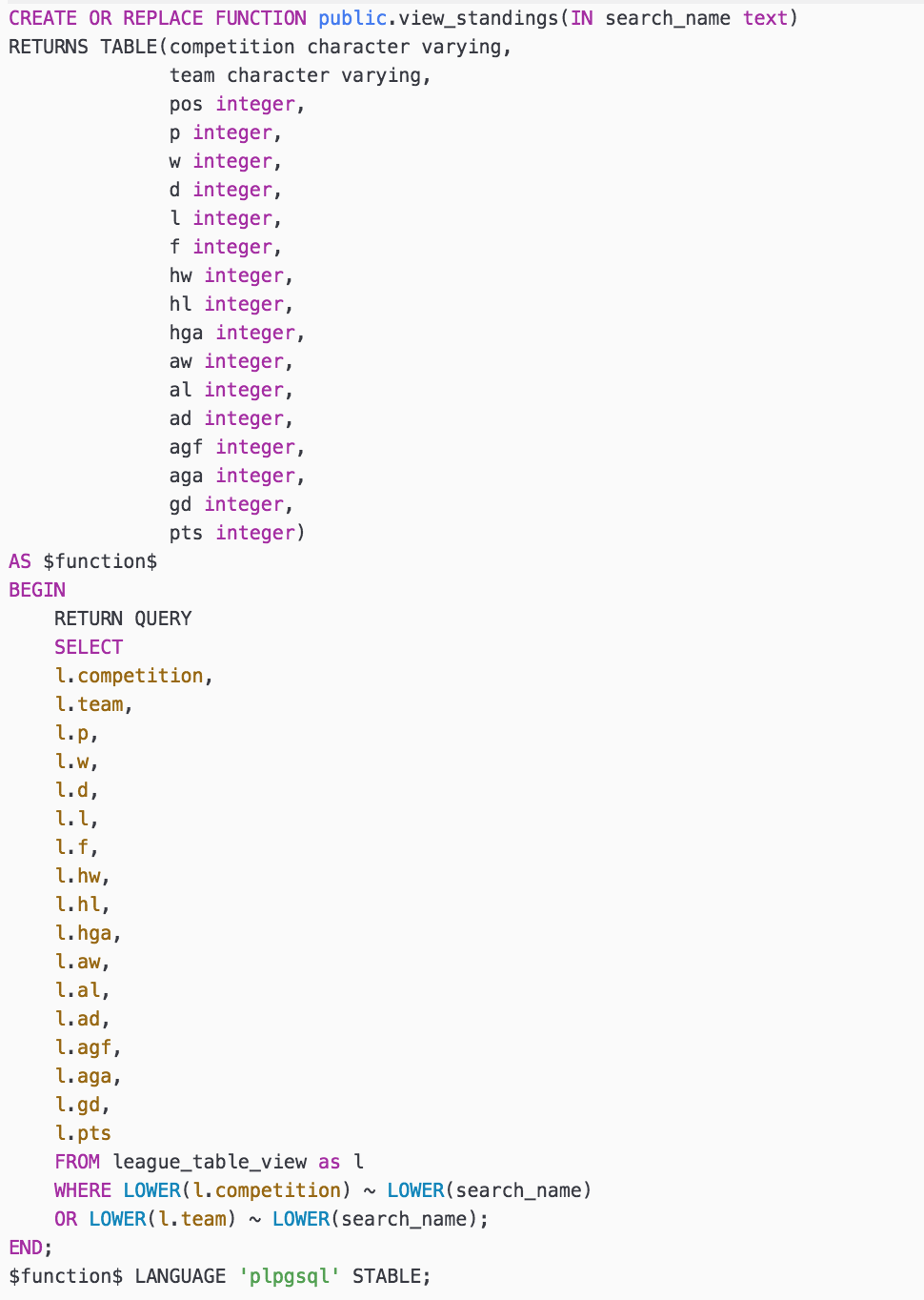
League Table view



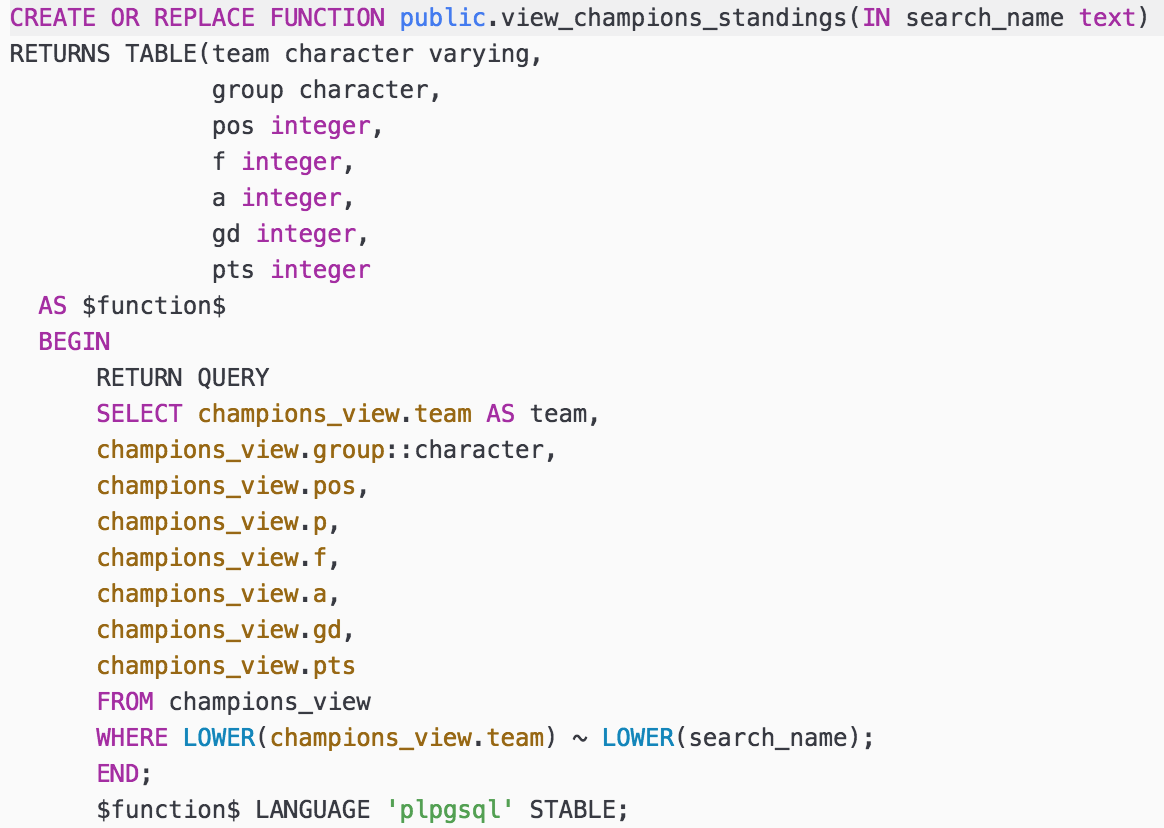
Champions League Table view



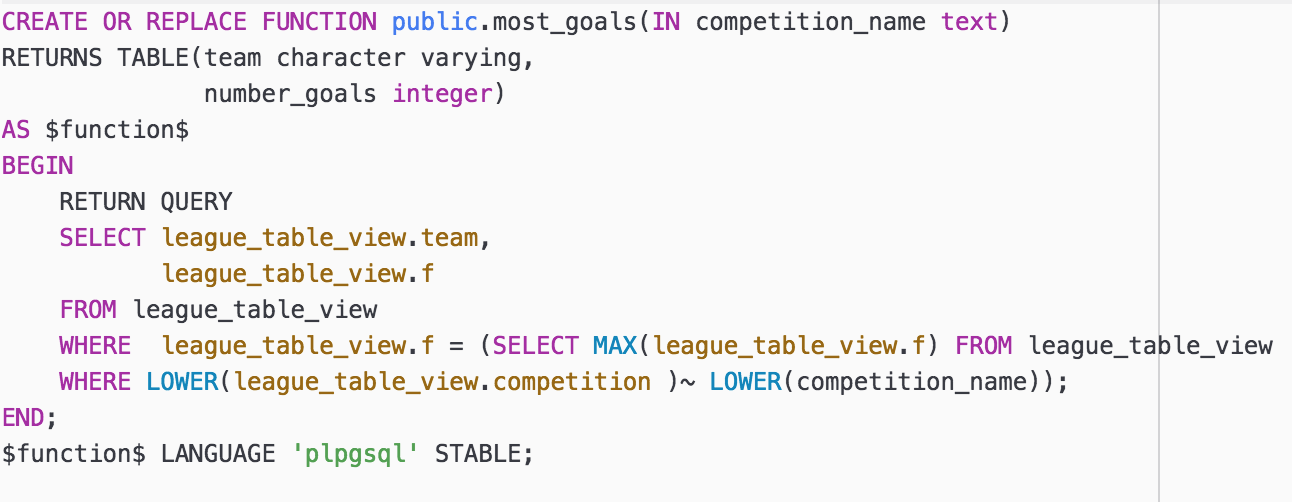
View standings in competition / of a team



View champions league standings



Most goals in competition



# Test Records

Test Plan

Testing is conducted to determine if the product requirements and specifications are met. The testing phase consisted primarily of glass box testing of the code that parsed JSON objects into dataframes (tables). Additionally, we tested the code that requested the data from the Football-data API and code that populated the database. Finally, we conducted black box testing to ensure that the stored procedures met the product requirements.

|  |  |  |
| --- | --- | --- |
| **Task** | **Pass/ Fail** | **Comment** |
| Installation of PostgreSQLql server on macOs | FAIL | The official download installation did not work |
| Installation of PostgreSQLql server on macOs | PASS | Installed the PostgreSQLApp instead of the regular install |
| Github repository setup | PASS | Setup successful on both systems |
| Request data competitions from the football-data API | PASS | Successfully requested data and saved JSON format |
| Request data competitions from the football-data API | FAIL | Exceeded number of allowed requests, API key in incorrect field |
| Request data competitions from the football-data API | FAIL | Exceeded the number of requests per second |
| Request data competitions from the football-data API | PASS | Successfully requested and saved data in JSON format |
| Request data soccer teams from the football- data API | PASS | Successfully requested and saved data in JSON format |
| Request data players from the football- data API | PASS | Successfully requested and saved data in JSON format |
| Request data fixtures from the football- data API | PASS | Successfully requested and saved data in JSON format |
| Request data league table from the football- data API | PASS | Successfully requested and saved data in JSON format |
| Load JSON files to python | FAIL | JSON objects are not in the correct format |
| Load JSON files to python | PASS | JSON objects were reformatted |
| Parse JSON files to data frames | FAIL | Data frame cells contain lists instead of atomic values |
| Parse JSON files to data frames | FAIL | Data frames were not collated into one correctly |
| Parse JSON files to data frames | PASS | Data frames successfully created for each table |
| Create / insert into competition table | FAIL | The data type not compatible |
| Create / insert into competition table | PASS | Competition table successfully |
| Create / insert into team table | FAIL | squad market value has a currency symbol, not compatible with integer |
| Create/ insert into tables on Nilu’s database | FAIL | python version is not compatible |
| Create/ insert into tables on Nilu’s database | PASS | created an virtual environment with python 2.7 installed |
| Create / insert into team table | PASS | Set squad market value as character varying. |
| Create / insert into team\_competition table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into fixtures table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into fixture\_team table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into league\_table table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into champions\_table table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into fixtures table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into manager table | PASS | Table successfully exported to PostgreSQL database |
| Create / insert into manager table | PASS | Table successfully exported to PostgreSQL database |

# Conclusion

Soccer Glimpse was created to keep soccer fans informed about competitions, teams, players, fixtures & scores and league table standings. The data contained in the system was obtained from Football-data RESTful API and Wikipedia. Data was requested from the API using Python’s request library. Once JSON objects were returned from the request they were parsed into pandas data frames and later exported to the PostgreSQL database using Python’s sqlalchemy library. Finally, stored procedures were implemented in order to satisfy all of the system’s specifications and requirements. All of the Python scripts, SQL code and accompanying documentation are available on <https://github.com/abbyyy23/Soccer> .

# References

Connolly, T., & Begg, C. (n.d.). *Database Systems—A Practical Approach to Design, Implementation, and Management.* (Sixth ed.).

Freitag, D. (n.d.). Retrieved March 05, 2017, from <http://api.football-data.org/documentation>

Main Page. (2017, April 15). Retrieved April 15, 2017, from <https://www.wikipedia.org/>

Manuals. (n.d.). Retrieved March 05, 2017, from https://www.postgresql.org/docs/manuals/