Prepared by:

Nilufar Isakova

Abigail Parra

3-19-17

CS 5310 Database Management Systems

University of Houston- Downtown

formal specification and relational model

# Abstract

# Introduction

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; 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.

# 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. |
| **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. |
| **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 |
| **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. |
| **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. |
| **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. |
| **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*.* |
| **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. |
| **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. |
| **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. |
| **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. |
| **UC15- View youngest manager 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. |
| **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. |
| **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. |
| **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 average player market\_values, youngest player, number of wins, losses and draws. |

# Entity Relationship Diagram

# /Users/abbyparra/Documents/Soccer/Documentation/ER Diagram submission 2.jpg

# 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 | numeric |

## 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 | integer |
| dob | date |
| nationality | text |
| contract\_until | date |
| market\_value | numeric |

## 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 | boolean not null |
| 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 | references Competition(id) not null |
| goals\_away | references Team(id) not null |
| odds\_away\_win | references Team(id) not null |
| odds\_home\_win | references Result(id) |

## 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 | integer not null |
| 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** |
| 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.

INSERT INTO competition(caption, league, year, teams, number\_of\_matches, number\_of\_games, number\_of\_teams)

VALUES ('','','','','','','');

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.

INSERT INTO team(manager\_id, name, code, squad\_market\_value, players)

VALUES ((SELECT manager\_id FROM manager m WHERE m.manager\_id=manager\_id),'','','','');

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.

INSERT INTO player(team\_id, name, position, jersey\_number, dob, nationality, contract\_until, market\_value)

VALUES ((SELECT team\_id FROM team t WHERE t.team\_id=team\_id),'','','','','','','','');

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.

INSERT INTO league\_table(competition\_id, team, pos, played\_games, wins, draws, losses, goals, goals\_aggainst, goal\_difference, points, position)

VALUES ((SELECT competition\_id FROM competition c WHERE c.competition\_id=competition\_id),'','','','','','','','','','','','');

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.

INSERT INTO fixture(competition\_id, status, date, matchday, results, goals\_home, goals\_away, odds\_home\_win, odds\_away\_win)

VALUES ((SELECT competition\_id FROM competition c WHERE c.competition\_id=competition\_id),'','','','','','','','');

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.

INSERT INTO stadium(team\_id, location, capacity)

VALUES

((SELECT team\_id FROM team t WHERE t.team\_id=team\_id), '','');

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.

INSERT INTO manager(team\_id, name, age)

VALUES

((SELECT team\_id FROM team t WHERE t.team\_id=team\_id), '','');

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.

UPDATE competition

SET caption = 'newname' WHERE caption='oldname';

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.

UPDATE team

SET name = 'newname' WHERE name='oldname';

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.

UPDATE player

SET name = 'newname' WHERE name='oldname';

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.

UPDATE league\_table

SET (team, pos) = (team='newteam',pos='newpos' WHERE competition\_id='0943';

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.

UPDATE fixture

SET (status, date) = (status='newstatus', date='newdate') WHERE competition\_id='0944';

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.

UPDATE stadium

SET location = 'newlocation' WHERE location='oldlocation';

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.

UPDATE manager

SET name = 'newname' WHERE name='oldname';

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 FROM competition WHERE year = 'selectedyear';

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 FROM player WHERE name = 'selectedname';

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 FROM fixture WHERE status = 'selectedstatus';

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 FROM stadium WHERE capacity = 'selectedcapacity';

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.

DELETE FROM manager WHERE name = 'selectedname';

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.

SELECT \*

FROM competition

WHERE LOWER(competition.caption) ~ LOWER(competition\_name);

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.

SELECT team.name,

team.code,

team.squad\_market\_value,

manager.name AS manager

FROM team

JOIN manager ON manager.id = team.manager\_id

WHERE LOWER(team.name) ~ LOWER(name);

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.

SELECT \*

FROM player1

WHERE LOWER(player1.name) ~ LOWER(player\_name);

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.

SELECT stadium.name,

stadium.capacity,

stadium.location,

team.name

FROM stadium

JOIN team ON team.id = stadium.team\_id

WHERE LOWER(stadium.name) ~ LOWER(stadium\_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.

SELECT manager.name,

manager.dob,

team.name

FROM manager

JOIN team ON team.id = manager.team\_id

WHERE LOWER(manager.name) ~ LOWER(name);

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.

CREATE OR REPLACE VIEW league\_table1 AS

SELECT league\_table.id,

competition.caption AS competition,

team.name AS team,

league\_table.played\_games,

league\_table.wins,

league\_table.draws,

league\_table.losses,

league\_table.goals,

league\_table.goals\_against,

league\_table.goal\_difference,

league\_table.points,

league.table.position

FROM league\_table

JOIN league\_table\_team ON league\_table\_team.league\_id = league\_table.id

JOIN team ON team.id = league\_table\_team.team\_id

JOIN competition ON competiton.id = league\_table.competition\_id;

SELECT \*

FROM league\_table1

WHERE league\_table1.competition = competition\_name;

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.

SELECT team.id,

team.name,

team.code,

team.squad\_market\_value,

competition.caption AS competition

FROM team

JOIN competition\_team ON competition\_team.team\_id = team.id

JOIN competition ON competition.id = competition\_team.competition\_id

WHERE competition.caption = competition\_name;

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 Home Goals, Most Away Goals, Most Wins and Most Losses and Average 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.

SELECT name, MAX(home\_goals)

FROM league\_table1

WHERE league\_table1.home\_goals = (SELECT MAX(home\_goals) FROM league\_table1)

AND league\_table1.competition = competition\_name;

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.

SELECT name, MAX(away\_goals)

FROM league\_table1

WHERE league\_table1.away\_goals = (SELECT MAX(away\_goals) FROM league\_table1)

AND league\_table1.competition = competition\_name;

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.

SELECT name, MAX(wins)

FROM league\_table1

WHERE league\_table1.wins = (SELECT MAX(wins) FROM league\_table1)

AND league\_table1.competition = competition\_name;

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.

SELECT name, MAX(losses)

FROM league\_table1

WHERE league\_table1.losses = (SELECT MAX(losses) FROM league\_table1)

AND league\_table1.competition = competition\_name;

View average squad market value

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

SELECT AVG(squad\_market\_value)

FROM team

WHERE competiton.name = name;

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.

SELECT \*

FROM fixture1

WHERE fixture1.competition = competition\_name;

View fixtures by week

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.

SELECT \*

FROM fixture1

WHERE fixture1.date = date;

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.

CREATE OR REPLACE VIEW fixture1 AS

SELECT fixture.id,

competition.caption AS competition,

(SELECT name FROM team WHERE team.id = fixture\_team.home\_team) AS home

(SELECT name FROM team WHERE team.id = fixture\_team.away\_team) AS away

fixture.status AS status,

fixture.date AS date,

fixture.matchday,

result.goals\_home,

result.goals\_away,

result.odds\_home\_win,

result.odds\_away\_win

FROM fixture

JOIN fixture\_team ON (fixture\_team.home\_team = fixture.home\_team,

AND fixture\_team.away\_team = fixture.away\_team)

JOIN team ON (team.id = fixture\_team.home\_team OR team.id = fixture\_team.away\_team)

JOIN competition ON competition.id = fixture.competititon\_id

JOIN result ON result.fixture\_id = fixture.id

SELECT

home,

away,

result.goals\_home,

result.goals\_away,

result.odds\_home\_win,

result.odds\_away\_win

FROM fixture1

WHERE fixture1.status = FINISHED

GROUP BY date DESC;

Average 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 avg () function on the number\_of\_games attribute of the Competition table.

SELECT AVG(competiton.number\_of\_games) FROM competition;

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.

SELECT \*

FROM fixtures1

WHERE fixtures1.home = team\_name OR fixtures1.away = team\_name;

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.

SELECT \*

FROM player1

WHERE player1.team = player\_name;

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).

SELECT \*

FROM league\_table1

WHERE league\_table1.team = team\_name;

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, Youngest Player and Average player market\_value.**

Age

This attribute is derived field from the date of birth. This function is performed by using the age () function.

EXTRACT(YEAR FROM (AGE(dob::timestamp))) AS age

Youngest Player

This function calculates the minimum age of the players in a team. This is performed by using the min () aggregate function on the Age attribute.

SELECTname, MIN(age)

FROM player1

WHERE team.name = name;

Number of wins

This function displays the number of wins of the team.

SELECT wins

FROM league\_table1

WHERE team.name = name;

Number of losses

This function displays the number of losses of the team.

SELECT losses

FROM league\_table1 --from league\_table1 view table

WHERE team.name = name;

Number of draws

This function displays the number of draws of the team.

SELECT draws

FROM league\_table1

WHERE team.name = name;

Youngest Manager

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

SELECT MIN(Age) FROM manager1; -- from manager1 view table

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.

SELECT MAX(capacity) FROM stadium;

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.

SELECT status, COUNT(\*)

FROM fixture

GROUP BY status;

# Prototype’s Technical Specifications

The project will be implemented using the Postgres Database Management System. Also, the application’s source code will be hosted on a remote GitHub repository. Our Postgres SQL 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 Postgres server | Nilufar, Abigail | In progress |
| Implement database tables | Nilufar, Abigail | In progress |
| Implement mapping tables | Abigail | Complete |
| Implement database views | Abigail | Complete |
| Implement stored procedures | Nilufar, Abigail | In progress |
| Create GitHub repository | Abigail | Complete |
| Manage version control | Nilufar, Abigail | In progress |
| Complete and revise documentation | Nilufar, Abigail | In progress |
| Perform black box testing / record outcome | Nilufar, Abigail | Pending |
| Write unit tests for critical components | Nilufar, Abigail | Pending |
| Perform BCNF verification | Nilufar, Abigail | Pending |

# BCNF Verification; Stored procedures for all functionalities

# 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.

|  |  |  |
| --- | --- | --- |
| **Name** | **Pass/ Fail** | **Comment** |
| Installation of Postgresql server on macOs | FAIL | The official download installation did not work |
| Installation of Postgresql server on macOs | PASS | Installed the postgresApp 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 |

# Conclusion