COMP 533-Final Report

Team Name: Animal Crossing
Database Name: European soccer market system
Team members: Siting Chen(sc122), Jian Fang(jf58)

Abstract

Soccer is one of the most popular sports around the world. Our team focuses on designing a system which helps people acquire and analyze the data in European football market. Firstly, we use PostgreSQL-supported methods to build relationships among players, teams, matches, leagues as well as countries. And then our team filters valid and valuable data we actually need. Finally, our task is to process data for player performance comparisons, player potential analysis, team rating analysis, league comparisons as well as predict future match results and trends.

Problem Description

Motivation

Simply speaking, soccer can be regarded as one of the most popular sports around the world. One regular match could attract thousands of people to purchase tickets. Take European Soccer market as one example, different roles may want to get access to different information. For bosses of clubs, they really want to spend limited money to trade promising players. Besides, in order to find potential players, they need a comprehensive report about the personal values for each player for evaluating. As for players, they want to play in the most comfortable and popular courtiers/leagues . They are eager to become teammates with potential players so that it is easier for them to win the championship. For our fans, we want to take part in some lotteries to predict the winner teams. Therefore, our team builds a multifunctional query system for different demands

Here is the main tasks we are going to solve:

- What is the best relationship for player's height so that they could be more likely to own higher personal values?
- What is the teams rating in one specific season in terms of top players' potential values?
- Which league/teams that scored the most in a given season?
- How the distribution of countries where lead players prefer to play for is like?
- Which team has the highest ratio of total cost as well as players' ability values(that is cost-efficiency)?
- When given a home team and away team, which team is more likely to win the match based on their past records?
- What is the competition trends like being offensive-oriented currently?

Dataset

Our original datasets include two data source. One is about information on teams, players and match records of major leagues in major European countries from 2008 to 2016 and the other one is about the latest (2019) player information in the FIFA database.

Design

Our group firstly concludes the main entities and draw the entity relational diagram:

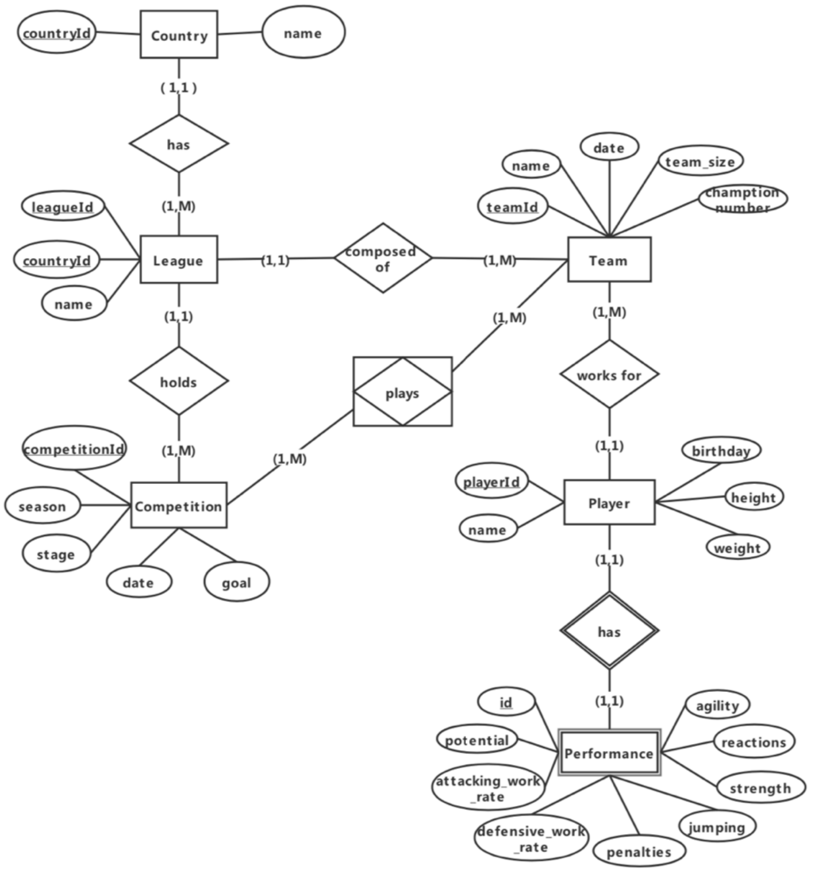


Figure 1. Entity Relational Diagram

- Each *Country* has multiple Leagues. Each *Country* is identified by country_id and has its country_name.
- *Leagues* are tracked through league_id and have league_name. There are multiple teams in one league and *Leagues* also take the responsibility to hold various competitions.
- *Competitions* were held by leagues and teams will compete in *competitions*. Each competition must have a unique competition_id, its holding season, stage, holding date as well as the game score.

- There are multiple *Teams*, each with a unique team_id, team name, establishment date, team_size and the number of champions achieved. *Teams* will participate in different competitions and they are governed by leagues.
- *Players* work for teams. Each player has a unique player_id, name, birthday, height, weight. For different players, their performances are evaluated to judge their values.
- *Performance* is used to track what a player performs in his/her matches. It has performance_id, values for potential, attacking_work_rate, defensive_work_rate, penalties, jumping, strength, reactions as well as agility. *Performance* is entirely dependent on the player.

Methods

Data loaded

Our dataset has two data sources, one is in SQLite format and the other is in csv format. We take different methods for data loading.

- For data in SQLite format, it is a complete database with nine tables in it. Using SQLiteSpy, we can directly export the tables and data into SQL format files, after modifying some incompatible syntax, we can insert them into our PostgreSQL database directly.
- For CSV files, since the data type of their attributes is character, using the import method in PostgreSQL to import data and try to modify the data type of the attribute will report errors. Therefore, we switch to using python, read the records in the csv line by line, then convert the data type according to our needs, and finally execute the insert statement to insert data. This method can make the data type of each attribute convenient for our subsequent processing and analysis.

Data Populated

Our original data is very complete, but many attribute values are not needed for our subsequent analysis, such as the odds of each oddsmaker for each game. Therefore, we have established 7 views to exclude unnecessary redundant data and link the data in multiple tables in the view to facilitate the efficiency of subsequent analysis.

Data Manipulated

After establishing the views, we created 9 functions to obtain and process the data we need. The main functions of these nine functions are: Get the league with the most goals scored in the season we input, get the team with the most goals scored in the season we input, and predict the wins and losses of the two teams in future matches. The specific contents will be described in the next section.

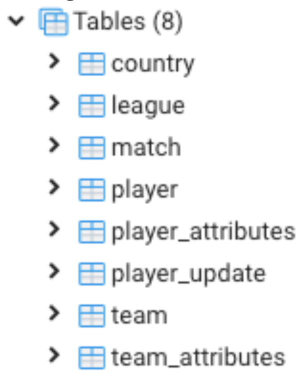
Data Visualizing

After using the function to process data, we visualize the data for analyzing their relationship or comparisons.

Deliverables

Tables

We use declarative SQLs to insert/delete/set/update data into our database. There are eight tables in our database, shown as figure 2:



Tables (8)
- country
- league
- match
- player
- player_attributes
- player_update
- team
- team_attributes

Figure 2. Tables Used in Project

RoadMap

- Jian Fang focuses on importing dataset, creating players views, country views, prediction part as well as analyzing the relationship between height and players ability values.
- Siting Chen is responsible for leagueGoalviews, teamGoalviews and the main ranking functions.

Notes:

source codes are uploaded on https://github.com/AllieChen02/RICE_COMP533_PROJECT

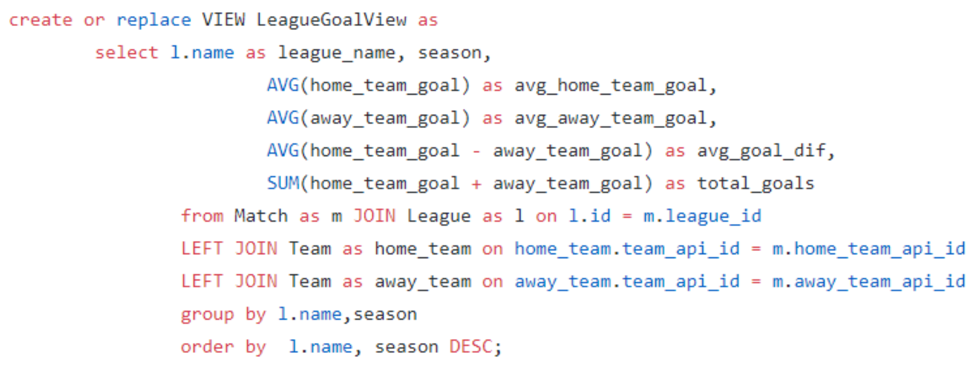
Functions & Views	Purposes
LeagueMatchView.sql	Create the view of LeagueMatchView
countryPlayerView.sql	Create the view of countryPlayerView
playerWithAllAttribute.sql	Create the view of playerWithAllAttributes
playerteamview_V2.sql	Create the view of PlayerTeamView_V2
populate_league_score.sql	Create the view of LeagueGoalView
populate_player_team.sql	Create the view of PlayerTeamView
populate_team_score.sql	Create the views of HomeTeamGoalView and AwayTeamGoalView
getAvgScoreOfLeague.sql	Create and execute function getAvgScoreOfLeague
getRatioOfValueToWage.sql	Create and execute function getRatioOfValueToWage
getRelationBetweenHeightAndPotential.sql	Create and execute function getRelationBetweenHeightAndPotential
getTopScoreLeague.sql	Create and execute function getTopScoreLeague

getTopScoreTeam.sql	Create and execute function getTopTeamOfScore
gettoppotentialteam_v2.sql	Create and execute function getTopPotentialTeam_V2
numberOfTopplayerByCountry_rating.sql	Create and execute function numberOfTopPlayerByCountry_rating
numberOfTopPlayerByCountry_topValue.sql	Create and execute function numberOfTopPlayerByCountry_topValue
predictWinner_V2.sql	Create and execute function predictWinner_V2

Functions

- Part1

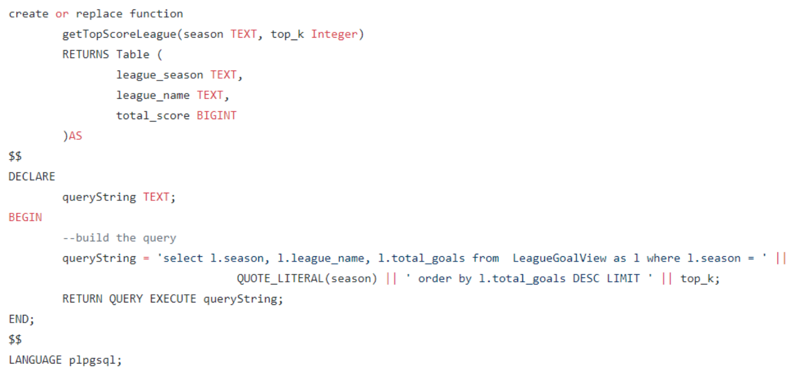
Firstly, we established a view called LeagueGoalView (figure 3). This view join the match table and league table on league_id, and join the team table on team_api_id, and it is grouped by each season, calculating the average goals per game for the home team and the average goals per game for the away team , the average home team goal difference and the total number of goals in the season.



```
create or replace VIEW LeagueGoalView as
        select l.name as league_name, season,
                        AVG(home_team_goal) as avg_home_team_goal,
                        AVG(away_team_goal) as avg_away_team_goal,
                        AVG(home_team_goal - away_team_goal) as avg_goal_dif,
                        SUM(home_team_goal + away_team_goal) as total_goals
                from Match as m JOIN League as l on l.id = m.league_id
                LEFT JOIN Team as home_team on home_team.team_api_id = m.home_team_api_id
                LEFT JOIN Team as away_team on away_team.team_api_id = m.away_team_api_id
                group by l.name,season
                order by  l.name, season DESC;
```

Figure 3. LeagueGoalView View

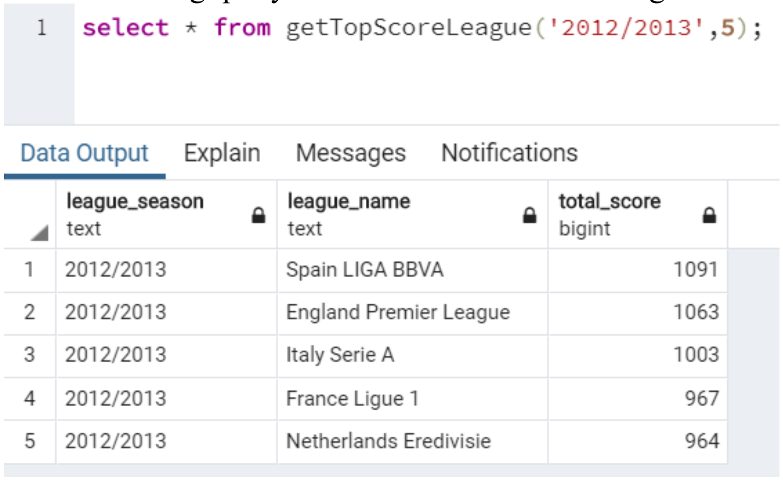
Secondly, we created a function called getTopScoreLeague (figure 4). This function is to obtain the top league which has the highest total goals in a given season. This function requires two parameters, the first parameter is the season, and the second parameter is the number of the top ranking. For example, enter ‘2015/2016’ as the first parameter and 5 as the second parameter to return to the top five leagues in total goals scored for the 2015/2016 season.



```
create or replace function
        getTopScoreLeague(season TEXT, top_k Integer)
        RETURNS Table (
                league_season TEXT,
                league_name TEXT,
                total_score BIGINT
        )AS
$$
DECLARE
        queryString TEXT;
BEGIN
        --build the query
        queryString = 'select l.season, l.league_name, l.total_goals from  LeagueGoalView as l where l.season = ' ||
                                QUOTE_LITERAL(season) || ' order by l.total_goals DESC LIMIT ' || top_k;
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

Figure 4. getTopScoreLeagues Function

Finally, we used the following query to execute the function and gain the result:



```
select * from getTopScoreLeague('2012/2013',5);
```

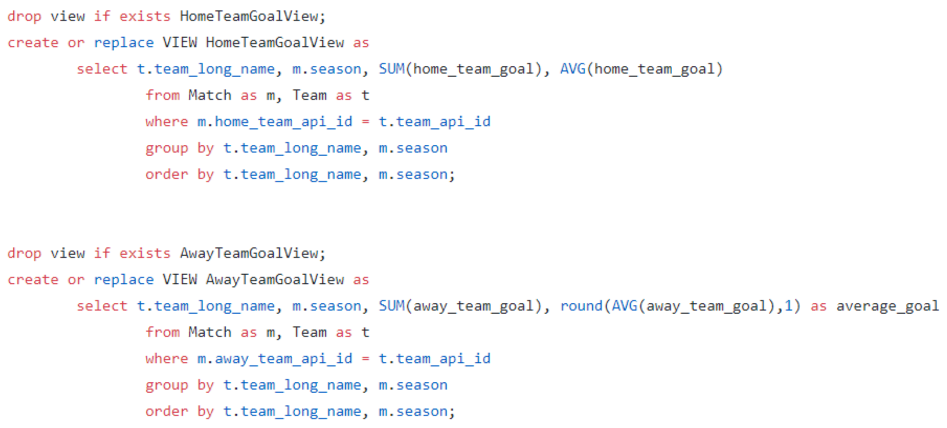
Data Output | Explain | Messages | Notifications

	league_season text	league_name text	total_score bigint
1	2012/2013	Spain LIGA BBVA	1091
2	2012/2013	England Premier League	1063
3	2012/2013	Italy Serie A	1003
4	2012/2013	France Ligue 1	967
5	2012/2013	Netherlands Eredivisie	964

Figure 5. Example of execution of the function

- Part2

Firstly, we established two views HomeTeamGoalView and AwayTeamGoalView (Figure 6). One of the views is to calculate the data of the home team, and the other is to calculate the data of the away team. They all join the team table with the match table on team_api_id and are grouped by the team_name and season. And then calculate total goals and average goals scored by each team as home team or away team each season.

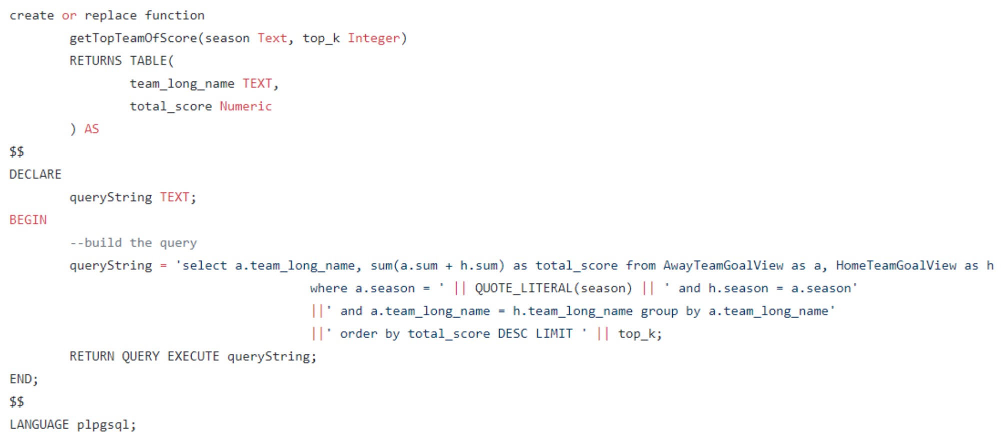


```
drop view if exists HomeTeamGoalView;
create or replace VIEW HomeTeamGoalView as
        select t.team_long_name, m.season, SUM(home_team_goal), AVG(home_team_goal)
                from Match as m, Team as t
                where m.home_team_api_id = t.team_api_id
                group by t.team_long_name, m.season
                order by t.team_long_name, m.season;


drop view if exists AwayTeamGoalView;
create or replace VIEW AwayTeamGoalView as
        select t.team_long_name, m.season, SUM(away_team_goal), round(AVG(away_team_goal),1) as average_goal
                from Match as m, Team as t
                where m.away_team_api_id = t.team_api_id
                group by t.team_long_name, m.season
                order by t.team_long_name, m.season;
```

Figure 6. Views of HomeTeamGoalView and AwayTeamGoalView

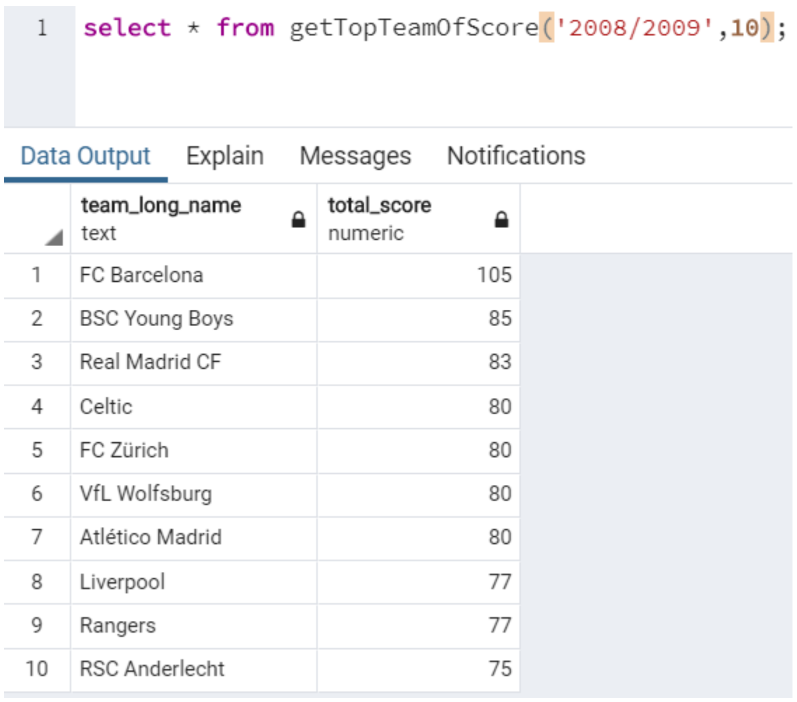
Secondly, a function called getTopTeamOfScore is created (Figure 7). This function is to obtain the top team which has the highest total goals in a given season. This function requires two parameters, the first parameter is the season, and the second parameter is the number of the top ranking. For example, enter '2008/2009'' as the first parameter and 5 as the second parameter to return to the top five teams in total goals scored for the 2008/2009 season.



```sql
create or replace function
        getTopTeamOfScore(season Text, top_k Integer)
        RETURNS TABLE(
                team_long_name TEXT,
                total_score Numeric
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
        --build the query
        queryString = 'select a.team_long_name, sum(a.sum + h.sum) as total_score from AwayTeamGoalView as a, HomeTeamGoalView as h
                                where a.season = ' || QUOTE_LITERAL(season) || ' and h.season = a.season'
                                ||' and a.team_long_name = h.team_long_name group by a.team_long_name'
                                ||' order by total_score DESC LIMIT ' || top_k;
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

Figure 7. Function of getTopTeamOfScore

Finally, we used the following query to execute the function and gain the result (figure 8). Through the results, we can find that the team with the most goals in the 08/09 season is Barcelona, and those familiar with football know that Barcelona won the Champions League that year. Real Madrid is in third place in the query result.



```sql
select * from getTopTeamOfScore('2008/2009',10);
```

	team_long_name text	total_score numeric
1	FC Barcelona	105
2	BSC Young Boys	85
3	Real Madrid CF	83
4	Celtic	80
5	FC Zürich	80
6	VfL Wolfsburg	80
7	Atlético Madrid	80
8	Liverpool	77
9	Rangers	77
10	RSC Anderlecht	75

Figure 8. Example of execution of the function

- Part3

Firstly, we established a view called VIEW PlayerTeamView_V2(Figure 9). This view will obtain the player's id and name, the name of the club where the player plays, the player's ability rating, potential rating, value and salary (in euros) from the 2019 player attributes table (player_update), and filter out records where the player's club name is empty.

```
CREATE OR REPLACE VIEW PlayerTeamView_V2 as
SELECT player_update.id AS player_fifa_api_id,
    player_update.full_name AS player_name,
    player_update.club_team AS team_name,
    player_update.overall_rating,
    player_update.potential,
    player_update.value_euro,
    player_update.wage_euro
    FROM player_update
  WHERE player_update.club_team IS NOT NULL AND player_update.club_team::text <> player_update.nationality::text
  ORDER BY player_update.id, player_update.full_name;
```

Figure 9. View of PlayerTeamView_V2

Secondly, based on this view we created two functions, getTopPotentialTeam_V2(figure 10) and getRatioOfValueToWage. The getTopPotentialTeam_V2 function requires two parameters. The first parameter is the player's ability rating, and the second is the number of top rankings. We will calculate the average ability ratings of the players whose personal rating is greater than or equal to the first parameter in a team to represent the overall strength of this team, and sort according to this average value in descending order. For example, if the first parameter is 75 and the second parameter is 5, then the top five teams with the average ability ratings of players whose personal rating is greater than 75 will be returned.

```
create or replace function
        getTopPotentialTeam_V2(Greater INTEGER, top_k Integer)
        RETURNS TABLE (
                team_name TEXT,
                avg_over_rating NUMERIC
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
        queryString = 'select p.team_name::text, round(AVG(p.overall_rating),2) as rating  from PlayerTeamView_V2 as p
        where p.overall_rating >= '|| Greater|| ' group by p.team_name order by rating DESC LIMIT ' || top_k;
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

Figure 10. Function of getTopPotentialTeam_V2

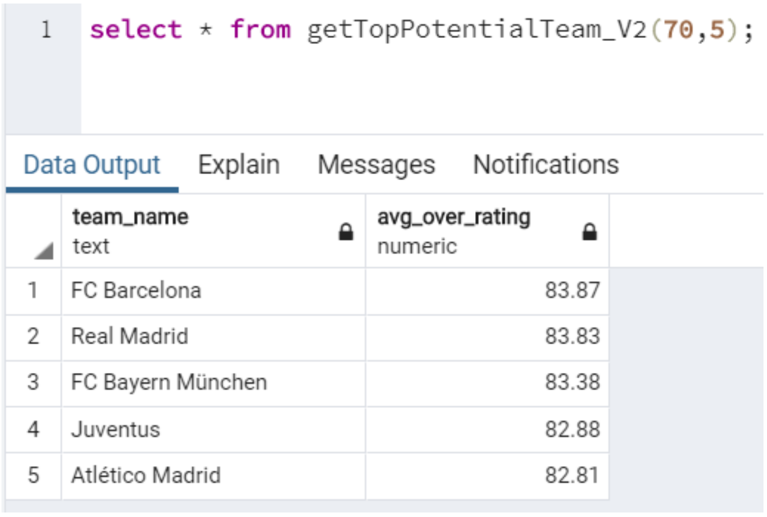
```
create or replace function
        getRatioOfValueToWage(rating INTEGER, top_k Integer)
        RETURNS TABLE (
                team_name TEXT,
                ratio NUMERIC
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
        queryString = 'select p.team_name::text, round(p.value/p.wage,2) as ratio
        from (SELECT team_name,sum(value_euro) as value,sum(wage_euro) as wage FROM PlayerTeamView_V2
        WHERE overall_rating >= '||rating||' AND wage_euro IS NOT NULL AND value_euro IS NOT NULL GROUP BY team_name) as p
        order by ratio LIMIT ' || top_k;
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

Figure 11. Function of getRatioOfValueToWage

The getRatioOfValueToWage function also requires two parameters, the first is the player's ability value, and the second is the number of top rankings. The function will return the ratio of the total value and total salary of the players whose personal rating is greater than the first parameter, in ascending order. Ranked first is the team with the lowest performance / price ratio (low value, high salary).

The above two functions have a parameter (player ability ratings) to filter out some players, because there are generally some reserve players in the team, the ratings of them are very low and cannot represent the overall strength of the team, so they should be filtered.

Finally, we used the following queries to execute functions and gain the results:

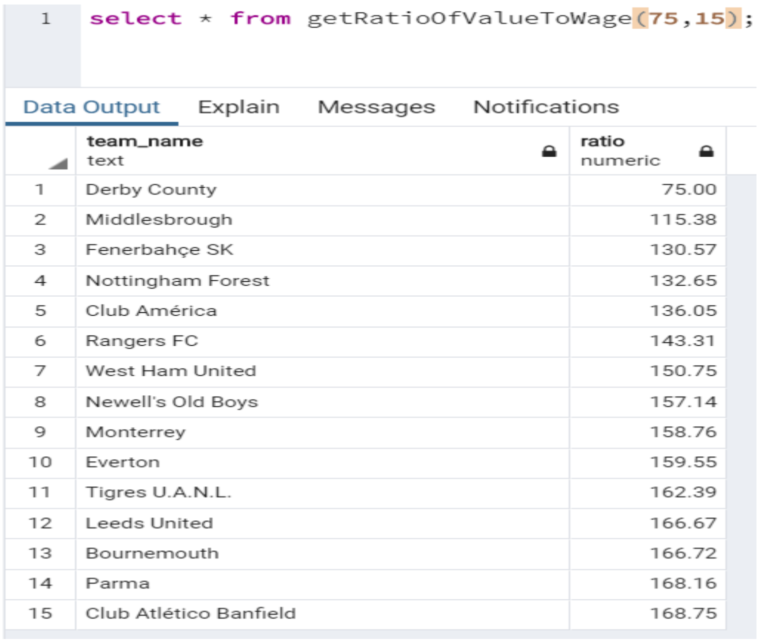


```
select * from getTopPotentialTeam_V2(70,5);
```

Data Output Explain Messages Notifications

	team_name text	avg_over_rating numeric
1	FC Barcelona	83.87
2	Real Madrid	83.83
3	FC Bayern München	83.38
4	Juventus	82.88
5	Atlético Madrid	82.81

Figure 12. Example of function getTopPotentialTeam_V2 execution



```
select * from getRatioOfValueToWage(75,15);
```

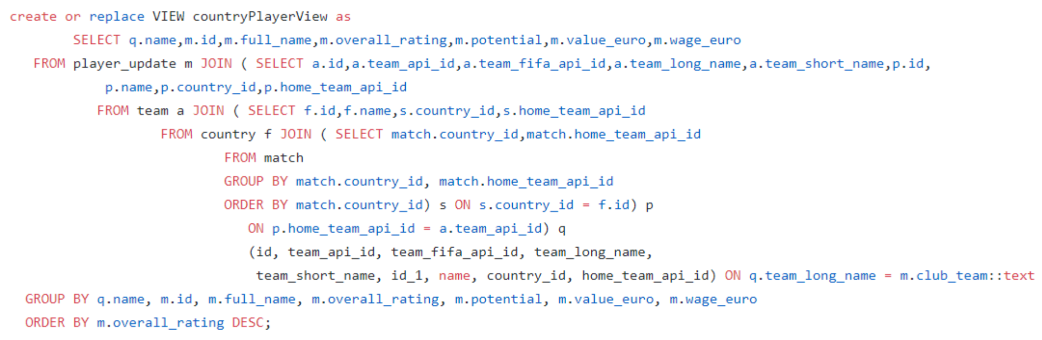
Data Output Explain Messages Notifications

	team_name text	ratio numeric
1	Derby County	75.00
2	Middlesbrough	115.38
3	Fenerbahçe SK	130.57
4	Nottingham Forest	132.65
5	Club América	136.05
6	Rangers FC	143.31
7	West Ham United	150.75
8	Newell's Old Boys	157.14
9	Monterrey	158.76
10	Everton	159.55
11	Tigres U.A.N.L.	162.39
12	Leeds United	166.67
13	Bournemouth	166.72
14	Parma	168.16
15	Club Atlético Banfield	168.75

Figure 13. Example of execution of the function getRatioOfValueToWage

- ## Part4

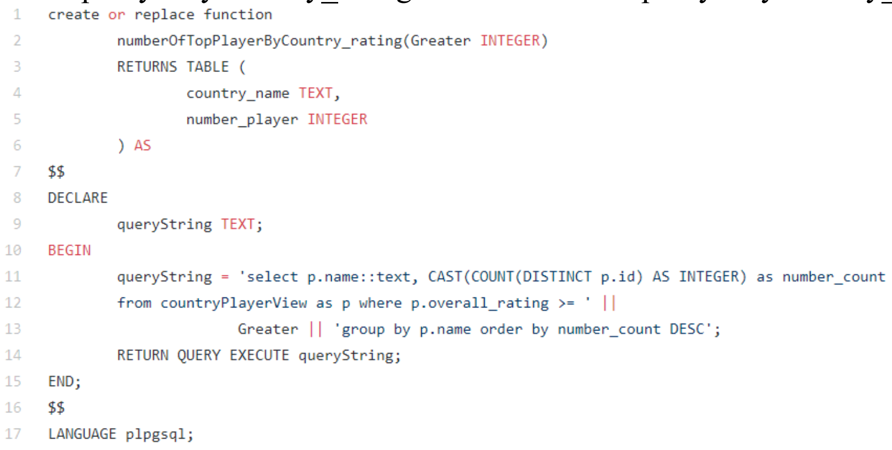
  Firstly, we established a view called countryPlayerView. This view joins the player_update table (player information in 2019), country table, team table, and match table. The final view contains the name of the country where the player plays, the player's id, the player's overall ratings, potential ratings, personal value and wages.



```
create or replace VIEW countryPlayerView as
        SELECT q.name,m.id,m.full_name,m.overall_rating,m.potential,m.value_euro,m.wage_euro
   FROM player_update m JOIN ( SELECT a.id,a.team_api_id,a.team_fifa_api_id,a.team_long_name,a.team_short_name,p.id,
            p.name,p.country_id,p.home_team_api_id
           FROM team a JOIN ( SELECT f.id,f.name,s.country_id,s.home_team_api_id
                   FROM country f JOIN ( SELECT match.country_id,match.home_team_api_id
                           FROM match
                           GROUP BY match.country_id, match.home_team_api_id
                           ORDER BY match.country_id) s ON s.country_id = f.id) p
                              ON p.home_team_api_id = a.team_api_id) q
                              (id, team_api_id, team_fifa_api_id, team_long_name,
                               team_short_name, id_1, name, country_id, home_team_api_id) ON q.team_long_name = m.club_team::text
  GROUP BY q.name, m.id, m.full_name, m.overall_rating, m.potential, m.value_euro, m.wage_euro
  ORDER BY m.overall_rating DESC;
```

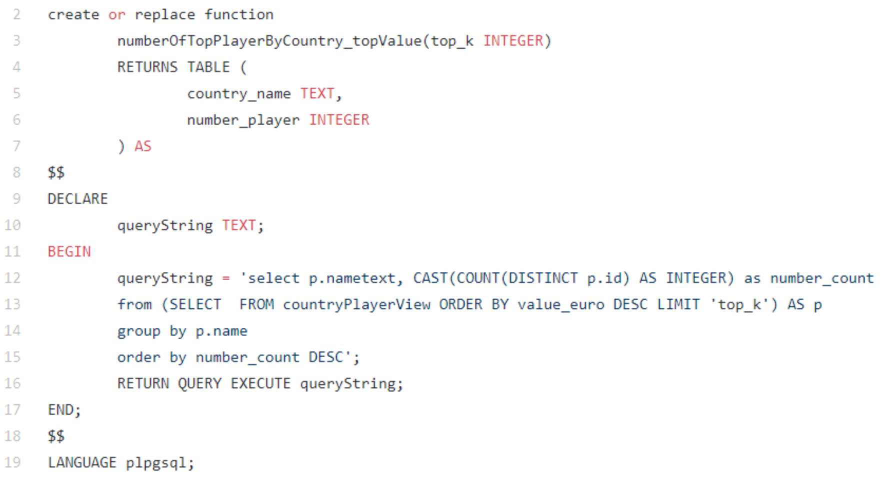
Figure 14. View of countryPlayerView

Secondly, based on the above view, we created two functions, numberOfTopPlayerByCountry_rating and numberOfTopPlayerByCountry_topValue.



```
create or replace function
        numberOfTopPlayerByCountry_rating(Greater INTEGER)
        RETURNS TABLE (
                country_name TEXT,
                number_player INTEGER
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
        queryString = 'select p.name::text, CAST(COUNT(DISTINCT p.id) AS INTEGER) as number_count
        from countryPlayerView as p where p.overall_rating >= ' ||
                       Greater || 'group by p.name order by number_count DESC';
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

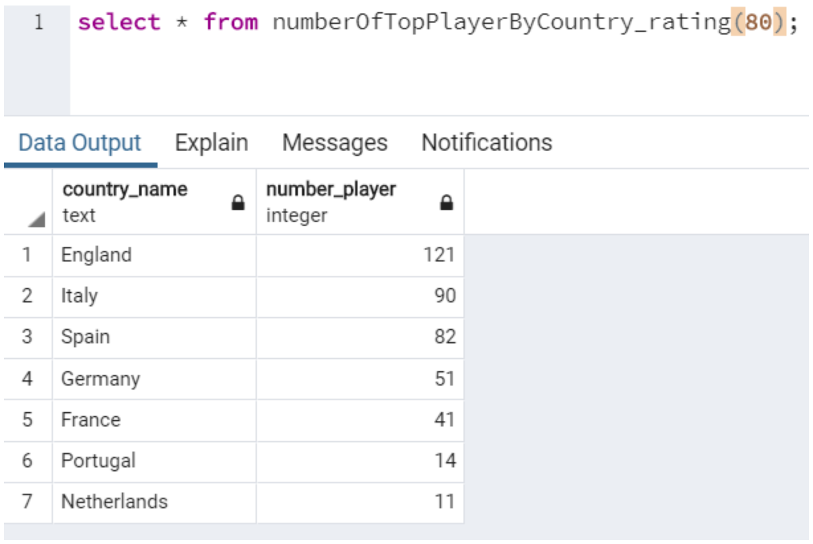
Figure 15. Function of numberOfTopPlayerByCountry_rating



```
create or replace function
        numberOfTopPlayerByCountry_topValue(top_k INTEGER)
        RETURNS TABLE (
                country_name TEXT,
                number_player INTEGER
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
        queryString = 'select p.nametext, CAST(COUNT(DISTINCT p.id) AS INTEGER) as number_count
        from (SELECT  FROM countryPlayerView ORDER BY value_euro DESC LIMIT 'top_k') AS p
        group by p.name
        order by number_count DESC';
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

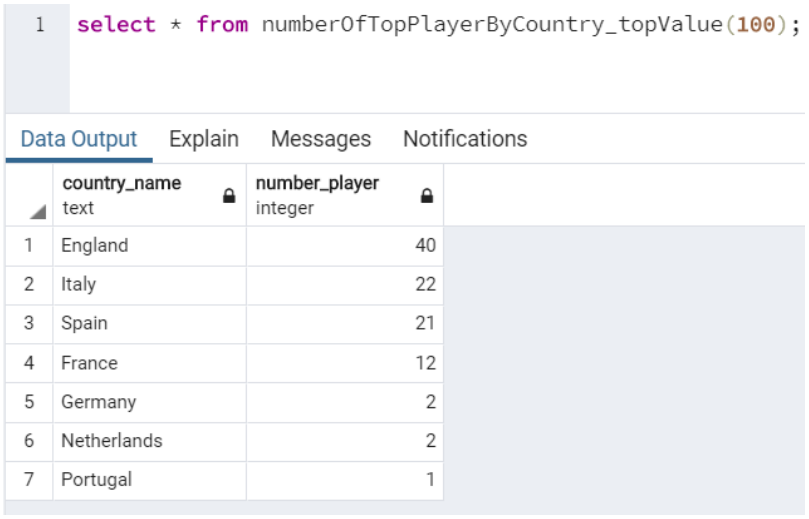
Figure 16. Function of numberOfTopPlayerByCountry_topValue

Finally, we used the following queries to execute functions and gain the results (figure 17, figure 18)


```
select * from numberOfTopPlayerByCountry_rating(80);
```

	country_name text	number_player integer
1	England	121
2	Italy	90
3	Spain	82
4	Germany	51
5	France	41
6	Portugal	14
7	Netherlands	11

Figure 17. Example of numberOfTopPlayerByCountry_rating execution


```
select * from numberOfTopPlayerByCountry_topValue(100);
```

	country_name text	number_player integer
1	England	40
2	Italy	22
3	Spain	21
4	France	12
5	Germany	2
6	Netherlands	2
7	Portugal	1

Figure 18. Example of the execution of numberOfTopPlayerByCountry_topValue

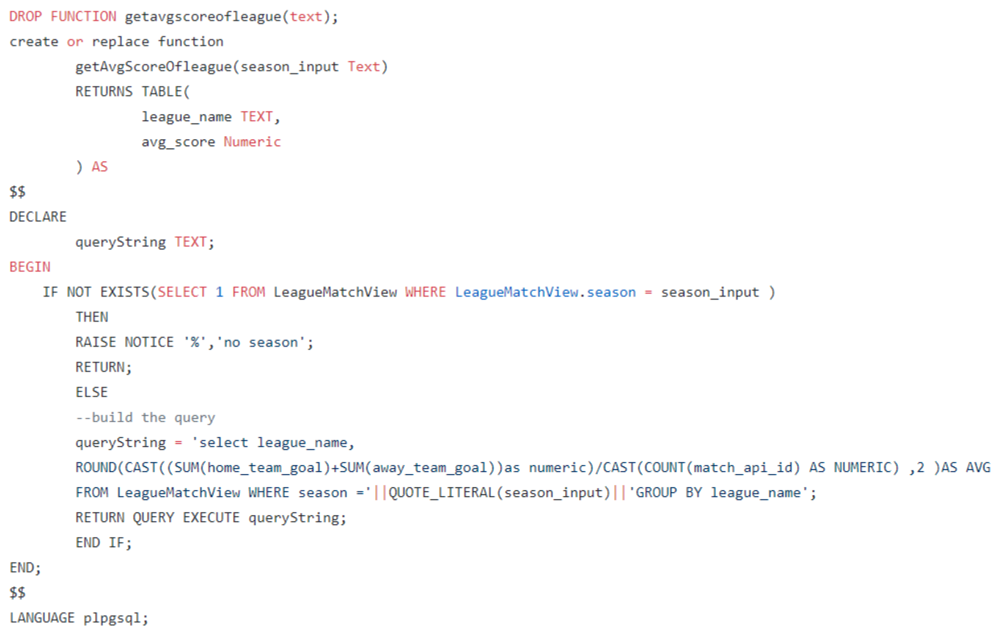
- **Part5**

  Firstly, we established a view called LeagueMatchView. This view joins the match table and the league table. The final view contains the name of the league, game id, date, season, home team id, away team id, home team goal and away team goal.

```
create or replace VIEW LeagueMatchView as
        select l.name as league_name,match_api_id, season,date,home_team_api_id,
        away_team_api_id, home_team_goal,away_team_goal
        from Match as m JOIN League as l on l.id = m.league_id
```

Figure 19. View of LeagueMatchView

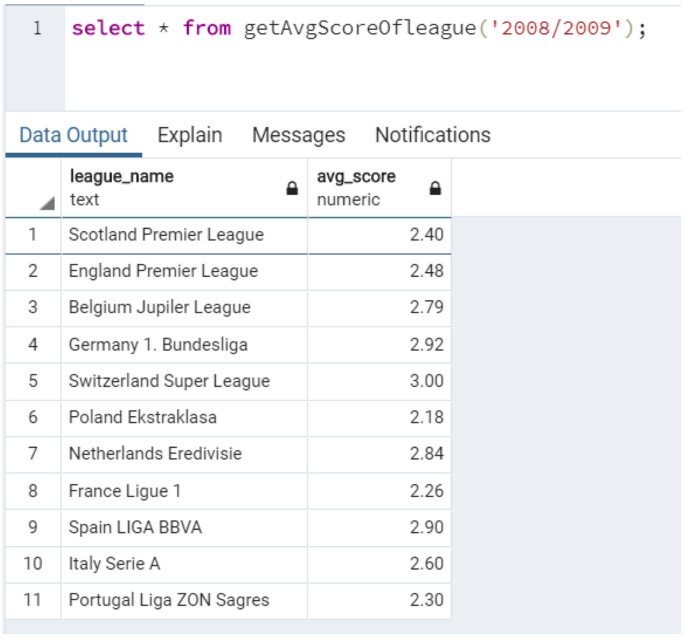
Secondly,We created a function called getAvgScoreOfleague(figure 20). This function has only one input, that is, the season. If there is no data for this season in the database, then the corresponding notice is returned. If there is, then the average of the number of goals in the leagues included in the database during this season will be returned.



```
DROP FUNCTION getavgscoreofleague(text);
create or replace function
        getAvgScoreOfleague(season_input Text)
        RETURNS TABLE(
                league_name TEXT,
                avg_score Numeric
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
    IF NOT EXISTS(SELECT 1 FROM LeagueMatchView WHERE LeagueMatchView.season = season_input )
        THEN
        RAISE NOTICE '%','no season';
        RETURN;
        ELSE
        --build the query
        queryString = 'select league_name,
        ROUND(CAST((SUM(home_team_goal)+SUM(away_team_goal))as numeric)/CAST(COUNT(match_api_id) AS NUMERIC) ,2 )AS AVG
        FROM LeagueMatchView WHERE season ='||QUOTE_LITERAL(season_input)||'GROUP BY league_name';
        RETURN QUERY EXECUTE queryString;
        END IF;
END;
$$
LANGUAGE plpgsql;
```

Figure 20. Function of getAvgScoreOfleague

Finally, we used the following query to execute the function and gain the result (figure 21). The query results are shown in the figure above. By querying the data of multiple seasons and analyzing them using statistical methods, you can find some rules.



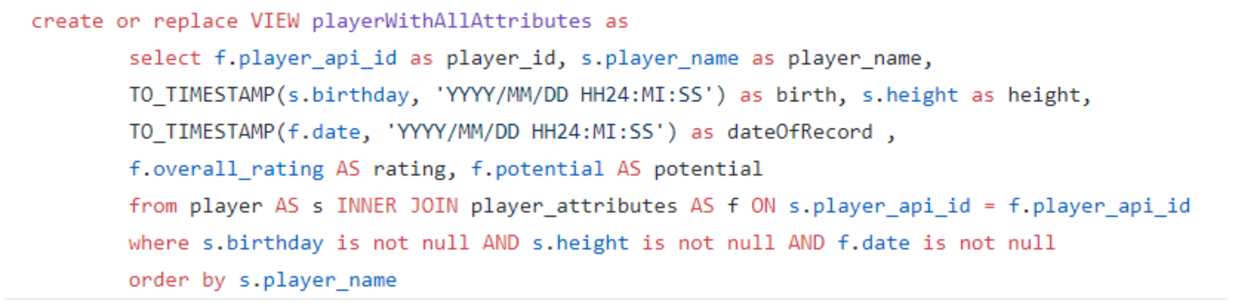
```
select * from getAvgScoreOfleague('2008/2009');
```

	league_name text	avg_score numeric
1	Scotland Premier League	2.40
2	England Premier League	2.48
3	Belgium Jupiler League	2.79
4	Germany 1. Bundesliga	2.92
5	Switzerland Super League	3.00
6	Poland Ekstraklasa	2.18
7	Netherlands Eredivisie	2.84
8	France Ligue 1	2.26
9	Spain LIGA BBVA	2.90
10	Italy Serie A	2.60
11	Portugal Liga ZON Sagres	2.30

Figure 21. The example of the execution of the function

- **Part6**

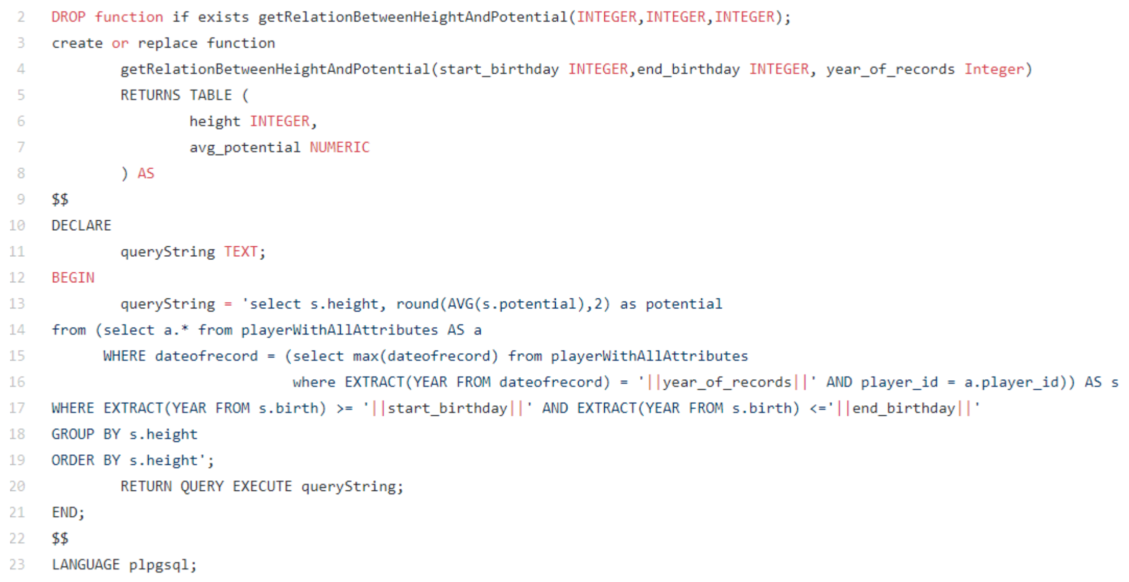
Firstly, we established a view called playerWithAllAttributes. (Figure 22). This view joins the player table and the player_attributes table (contains the data from 2008 to 2016), and converts the player's birthday and record date into a timestamp format, and filters out the data of the birthday, record date and height which are empty.



```
create or replace VIEW playerWithAllAttributes as
        select f.player_api_id as player_id, s.player_name as player_name,
        TO_TIMESTAMP(s.birthday, 'YYYY/MM/DD HH24:MI:SS') as birth, s.height as height,
        TO_TIMESTAMP(f.date, 'YYYY/MM/DD HH24:MI:SS') as dateOfRecord ,
        f.overall_rating AS rating, f.potential AS potential
        from player AS s INNER JOIN player_attributes AS f ON s.player_api_id = f.player_api_id
        where s.birthday is not null AND s.height is not null AND f.date is not null
        order by s.player_name
```

Figure 22. The view of playerWithAllAttributes

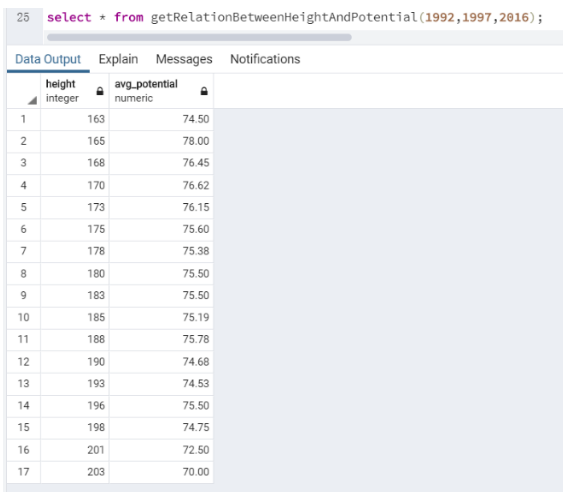
Secondly, we created a function called getRelationBetweenHeightAndPotential (figure 23). This function has three parameters, the first is the lower limit of the players' birthday, the second is the upper limit of the players' birthday, and the third is the year of the record. We will select the data for the year entered. If a player still has multiple data in that year, then the latest one will be selected by default. Then select the players whose birthday is within the input range, and finally group them by height to calculate the average potential ratings of these players.



```
DROP function if exists getRelationBetweenHeightAndPotential(INTEGER,INTEGER,INTEGER);
create or replace function
        getRelationBetweenHeightAndPotential(start_birthday INTEGER,end_birthday INTEGER, year_of_records Integer)
        RETURNS TABLE (
                height INTEGER,
                avg_potential NUMERIC
        ) AS
$$
DECLARE
        queryString TEXT;
BEGIN
        queryString = 'select s.height, round(AVG(s.potential),2) as potential
from (select a.* from playerWithAllAttributes AS a
      WHERE dateofrecord = (select max(dateofrecord) from playerWithAllAttributes
                            where EXTRACT(YEAR FROM dateofrecord) = '||year_of_records||' AND player_id = a.player_id)) AS s
WHERE EXTRACT(YEAR FROM s.birth) >= '||start_birthday||' AND EXTRACT(YEAR FROM s.birth) <='||end_birthday||'
GROUP BY s.height
ORDER BY s.height';
        RETURN QUERY EXECUTE queryString;
END;
$$
LANGUAGE plpgsql;
```

Figure 23. The function of getRelationBetweenHeightAndPotential

Finally, we used the following query to execute the function and gain the result. This query returns the relationship between height and average potential ratings for players whose birthdays were between 1992 and 1997 based on the data in 2016.



```
select * from getRelationBetweenHeightAndPotential(1992,1997,2016);
```

Data Output | Explain | Messages | Notifications

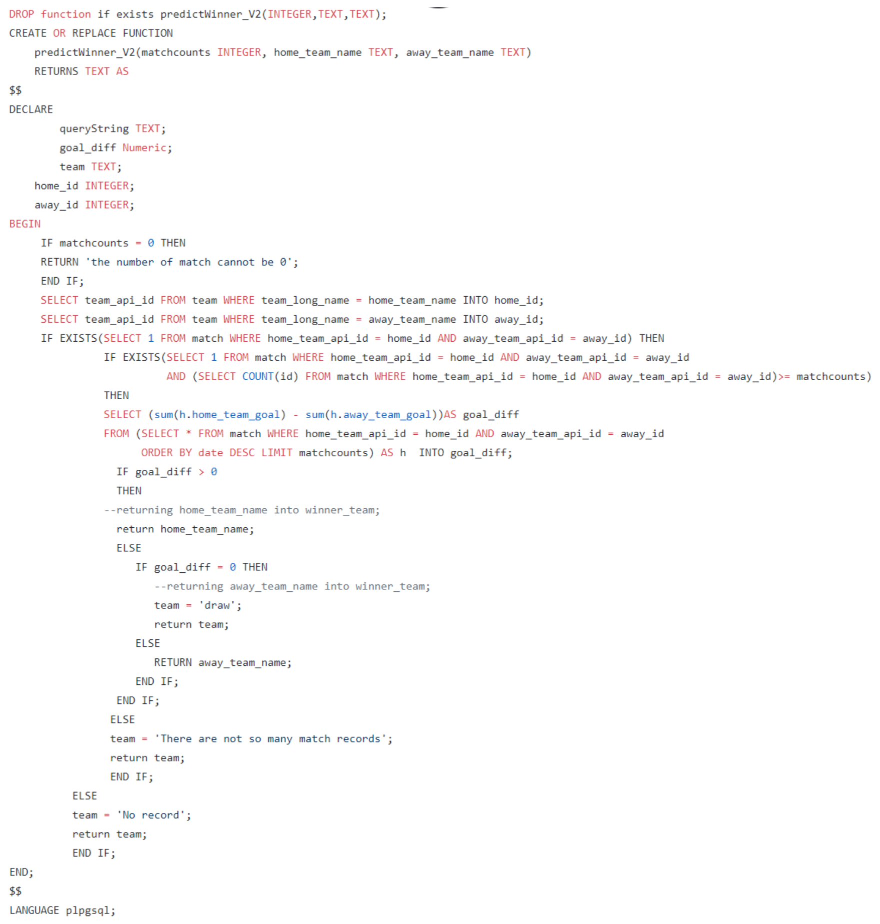
	height integer	avg_potential numeric
1	163	74.50
2	165	78.00
3	168	76.45
4	170	76.62
5	173	76.15
6	175	75.60
7	178	75.38
8	180	75.50
9	183	75.50
10	185	75.19
11	188	75.78
12	190	74.68
13	193	74.53
14	196	75.50
15	198	74.75
16	201	72.50
17	203	70.00

Figure 24. The example of the execution of the function

- Part7

In this part, we do not establish new views, we directly use the match table.

Firstly, we created a function called predictWinner_V2 (figure 25)

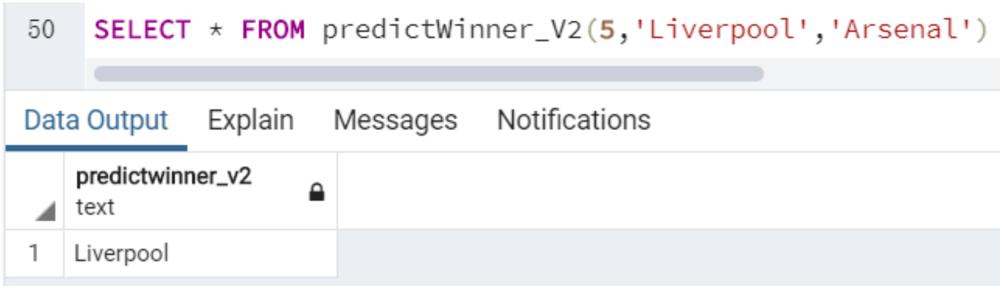


```
DROP function if exists predictWinner_V2(INTEGER,TEXT,TEXT);
CREATE OR REPLACE FUNCTION
    predictWinner_V2(matchcounts INTEGER, home_team_name TEXT, away_team_name TEXT)
    RETURNS TEXT AS
$$
DECLARE
        queryString TEXT;
        goal_diff Numeric;
        team TEXT;
    home_id INTEGER;
    away_id INTEGER;
BEGIN
     IF matchcounts = 0 THEN
     RETURN 'the number of match cannot be 0';
     END IF;
     SELECT team_api_id FROM team WHERE team_long_name = home_team_name INTO home_id;
     SELECT team_api_id FROM team WHERE team_long_name = away_team_name INTO away_id;
     IF EXISTS(SELECT 1 FROM match WHERE home_team_api_id = home_id AND away_team_api_id = away_id) THEN
               IF EXISTS(SELECT 1 FROM match WHERE home_team_api_id = home_id AND away_team_api_id = away_id
                         AND (SELECT COUNT(id) FROM match WHERE home_team_api_id = home_id AND away_team_api_id = away_id)>= matchcounts)
               THEN
               SELECT (sum(h.home_team_goal) - sum(h.away_team_goal))AS goal_diff
               FROM (SELECT * FROM match WHERE home_team_api_id = home_id AND away_team_api_id = away_id
                     ORDER BY date DESC LIMIT matchcounts) AS h  INTO goal_diff;
                 IF goal_diff > 0
                 THEN
               --returning home_team_name into winner_team;
                 return home_team_name;
                 ELSE
                    IF goal_diff = 0 THEN
                       --returning away_team_name into winner_team;
                       team = 'draw';
                       return team;
                    ELSE
                       RETURN away_team_name;
                    END IF;
                 END IF;
                ELSE
                team = 'There are not so many match records';
                return team;
                END IF;
          ELSE
          team = 'No record';
          return team;
          END IF;
END;
$$
LANGUAGE plpgsql;
```

Figure 25. The function of predictWinner_V2

This function needs three parameters, the first parameter is the number of past games referenced (marked as N), the second is the full name of the home team, and the third is the full length of the away team. The function will match the home team and the away team's match records from 08 to 16, and select the most recent N match records, calculate the home team's goals and the away team's goals, we will predict the team who scored more goals to be the winner. if the total goals are equal, then we will predict a draw. If there is no corresponding match record or N records are not reached, the corresponding notice will also be returned.

Secondly, we used the following query to execute the function and gain the result:



```
SELECT * FROM predictWinner_V2(5,'Liverpool','Arsenal')
```

```
  predictwinner_v2
  text
1 Liverpool
```

Figure 26. The example of execution of the function

Results

The final part is to visualize the results:

- Acquire K leagues ranked according to their scores in one specific season
  E.g. Take K= 5 and season = 2012/2013, and the best league is LIGA BBVA.

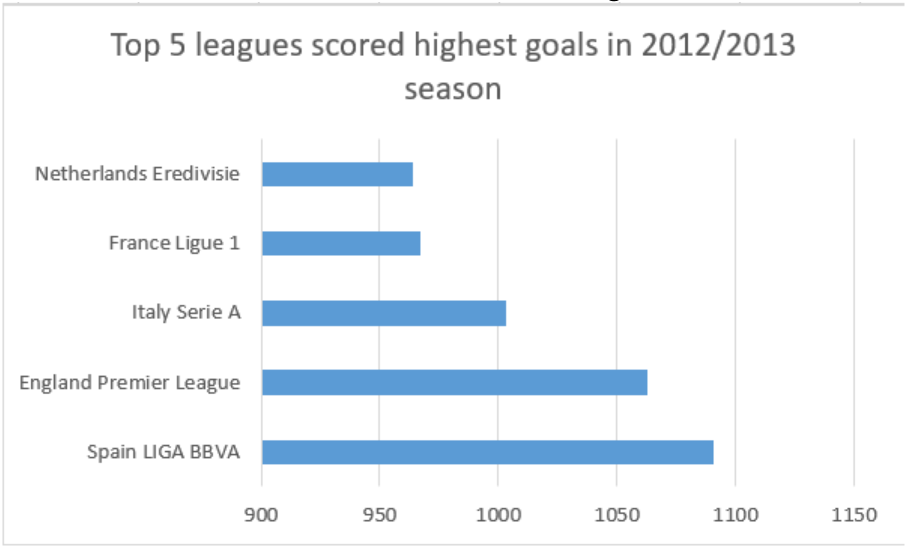


Figure 27. Leagues ranking according to scores in 2012/2013

- Acquire K teams ranked according to their scores in one specific season
  E.g. Take K= 20 and season = 2008/2009, and the best team is FC Barcelona.

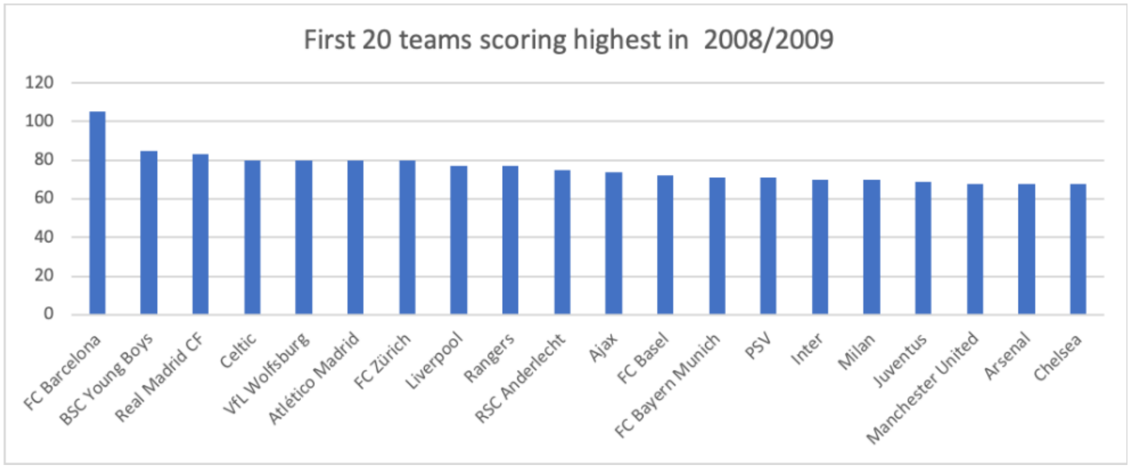


Figure 28. Teams ranking according to scores in 2008/2009

- Rank the best K teams according to the sum of lead player's ability values in the team E.g. Take K= 5 and the best team is FC Barcelona (Based on the player attributes data in 2019)

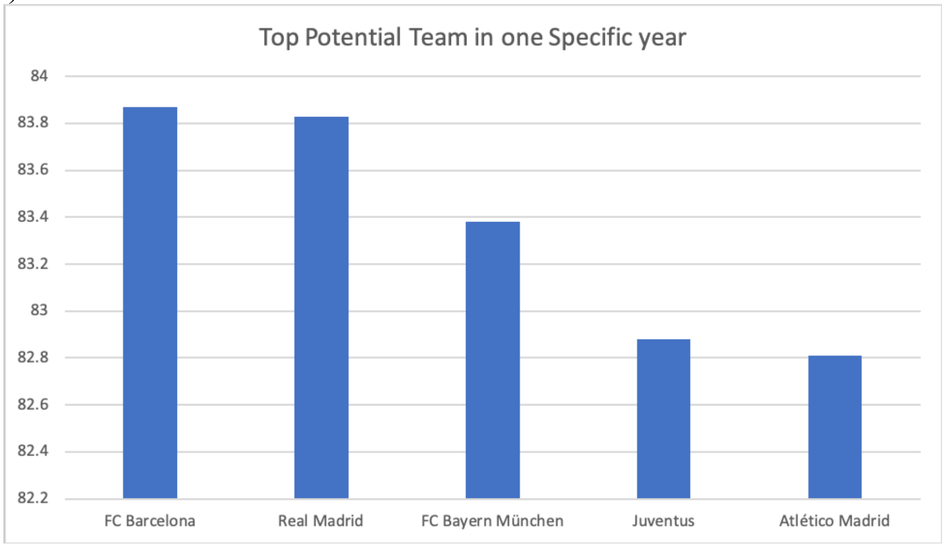


Figure 29. Teams ranking according to top players potential value

- Acquire the distribution of countries where players whose personal value is top 100 prefer to play.
  As is shown, England is the most ideal country where top players prefer to play.

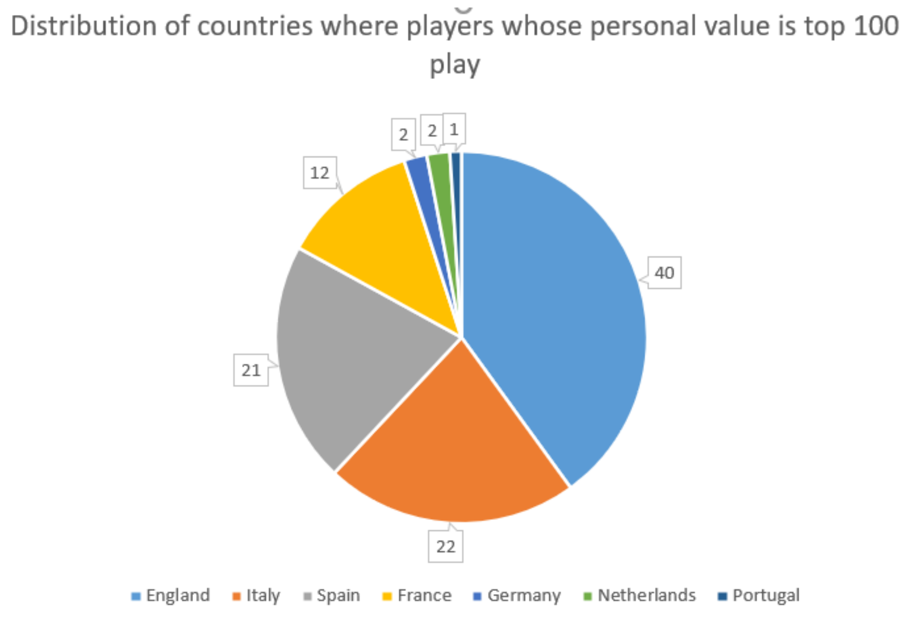


Figure 30. Distributions of countries where top players prefer to play

- Rank teams by their ratio of sum of lead player's personal value and the sum of their corresponding wages
  E.g. the higher the ratio is, the more cost-efficient the team is. As is shown, the team named Derby County spend much money in introducing players, but their performance cannot match their price.

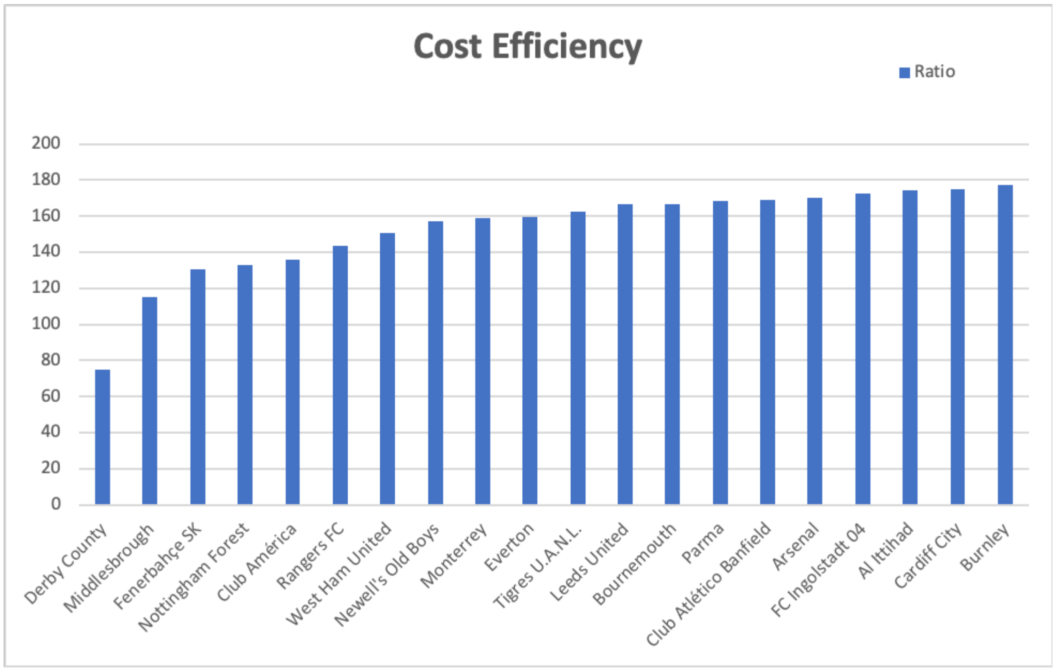


Figure 31. Distributions of countries where top players prefer to play

- Analyze the mode of play of 11 leagues in the past few years
  The average number of goals per game in the Dutch League is first in most seasons.

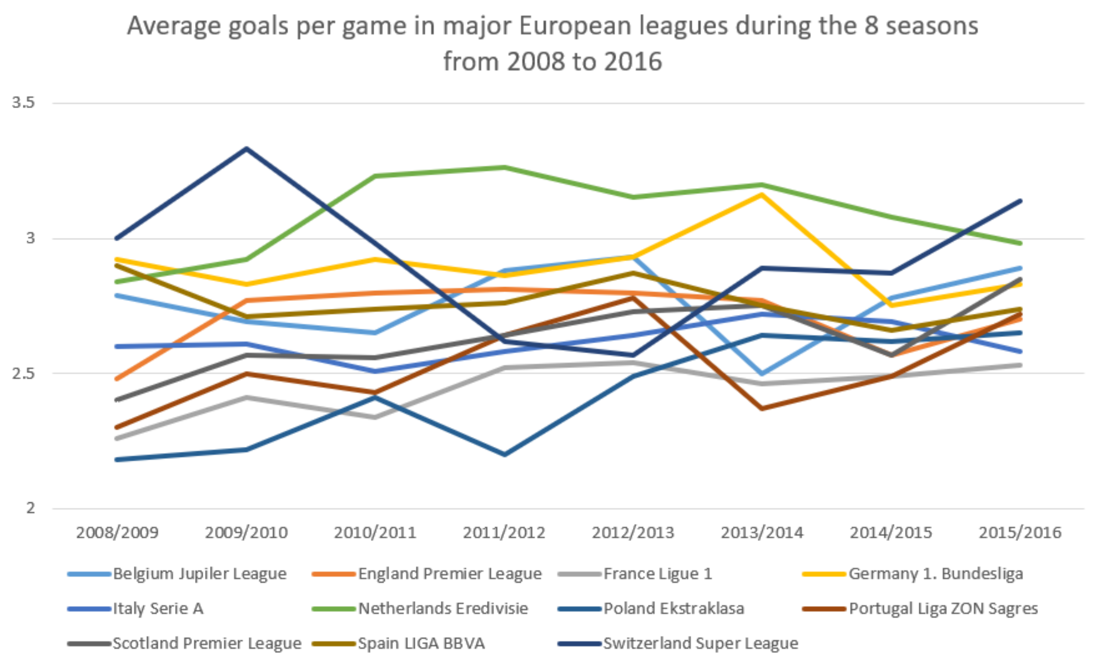


Figure 32. Mode of play

- Analyze the relationship between player personal value and height

  E.g. by analyzing the relationship between the height and potential ratings of players whose birthdays are between 1993 and 1998.(Based on the 2016 record), We can find that there are two peaks in the height range of 168 to 173 and 193 to 198, indicating that players in these two height ranges have more potential. This is also realistic, because most football players need to be agile, so their height cannot be too high. However, some positions, such as center and guard, require tall players, so there are also high-potential players.

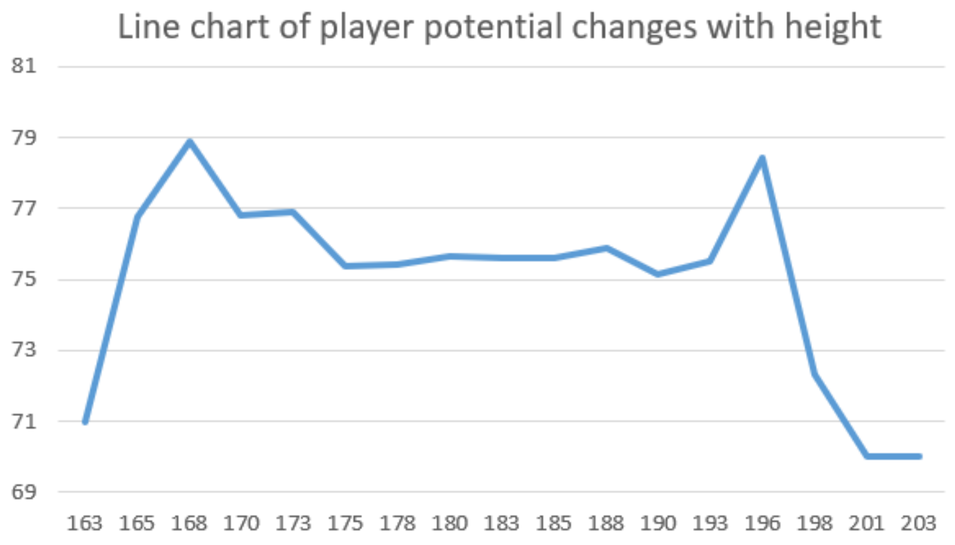
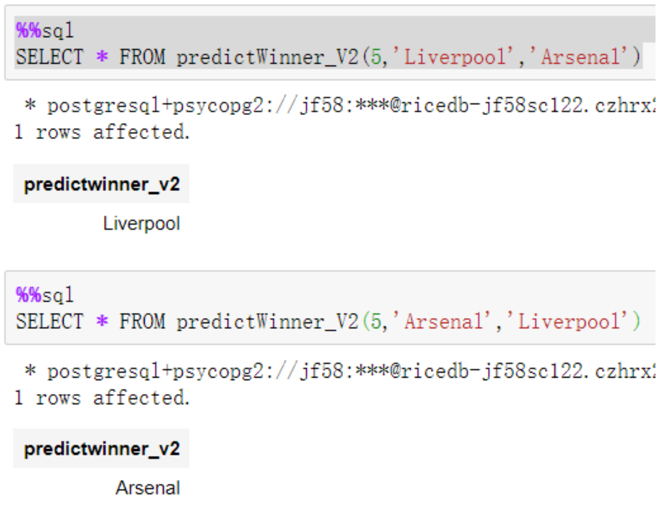


Figure 33. Relationship between height and personal value

- Predict the result of one future match by analyzing their recent K match records.

E.g. by analyzing the recent 5 matches between Liverpool and Arsenal, our team believe that Liverpool will be the winner as a home team. On the contrary, referring to the past 5 games, if Arsenal are the home team, then we predict that Arsenal will be the winner.



```
%%sql
SELECT * FROM predictWinner_V2(5,'Liverpool','Arsenal')

 * postgresql+psycopg2://jf58:***@ricedb-jf58sc122.czhrx
1 rows affected.

predictwinner_v2
       Liverpool

%%sql
SELECT * FROM predictWinner_V2(5,'Arsenal','Liverpool')

 * postgresql+psycopg2://jf58:***@ricedb-jf58sc122.czhrx
1 rows affected.

predictwinner_v2
         Arsenal
```

Figure 34. Prediction of winner team

Discussion

- Challenges

- When importing tables of player attributes, pgAdmin always prompts the error of losing connection with the database due to large amounts of data (180000+ rows). In this case, our team use the SQL shell(pgsql) to import data, first connect to our database and then use the \i syntax to execute inserting in the SQL file, which will be more stable
- When designing the model for predicting the future match, our team firstly just counts total goals of all the past records for the home team as well the away team. However, the method was quite imprecise because matches in five years ago almost have little effects on the present and the records with other teams are also meaningless. Therefore, we must come up with another idea to do prediction. We just make use of the recent match records of the two teams to do prediction.

- Changes from Original Plan

The original dataset cannot form a relational database management system. In the beginning, our group is going to analyze the relationship between US household income and Median price per Sqft in different cities. However, the number of entities is limited so it is hard to find multiple relations in the two datasets.

- Next steps

- Improve query efficiency. For example, when we want to do an analysis of the relationship between player's height and weight, it takes much time for querying. The main reason is that for each year, there will be many records in different months for one player. Our query should start with querying from month first and then select the latest record for analysis, which will take a large amount of time.

- Improve predictive accuracy. In our project, we predict the result for one specific match by analyzing the recent K matches between the home team and away team. In fact, we have to take the player movement, player form,etc. Into consideration to enhance predictive accuracy.

Conclusion

Our team successfully established a system for users to query the information for European football market. For players, they could clearly see the distribution of leagues where top players prefer to play for. For bosses of clubs, they could use this system to check the cost-efficiency of their team to adjust the team structure, avoiding spending much money on less potential players. Besides, for fans, they could easily query the most potential players in that season or with help of the system, they can participate in lotteries because they could use the system analysis to predict which team could be the winner.

Therefore, our project has practical significance and users of different roles could benefit from it. In the future, our team will pay more attention to the query efficiency to improve the system.

Reference

1. https://www.kaggle.com/hugomathien/soccer
2. https://data.world/raghav333/fifa-players/workspace/file?filename=fifa_cleaned.csv
3. https://www.kaggle.com/dimarudov/data-analysis-using-sql