BILKENT UNIVERSITY

SPRING 2017 - CS 353

TERM PROJECT DESIGN REPORT

Football Database Management System

GROUP 13

21301854 Ömer Akgül
21302483 Ali Osman Çetin
21300994 Ali Günes
21401058 Orhun Kar

cs353.github.io

TABLE OF CONTENTS

1-) Revised Entity Relation

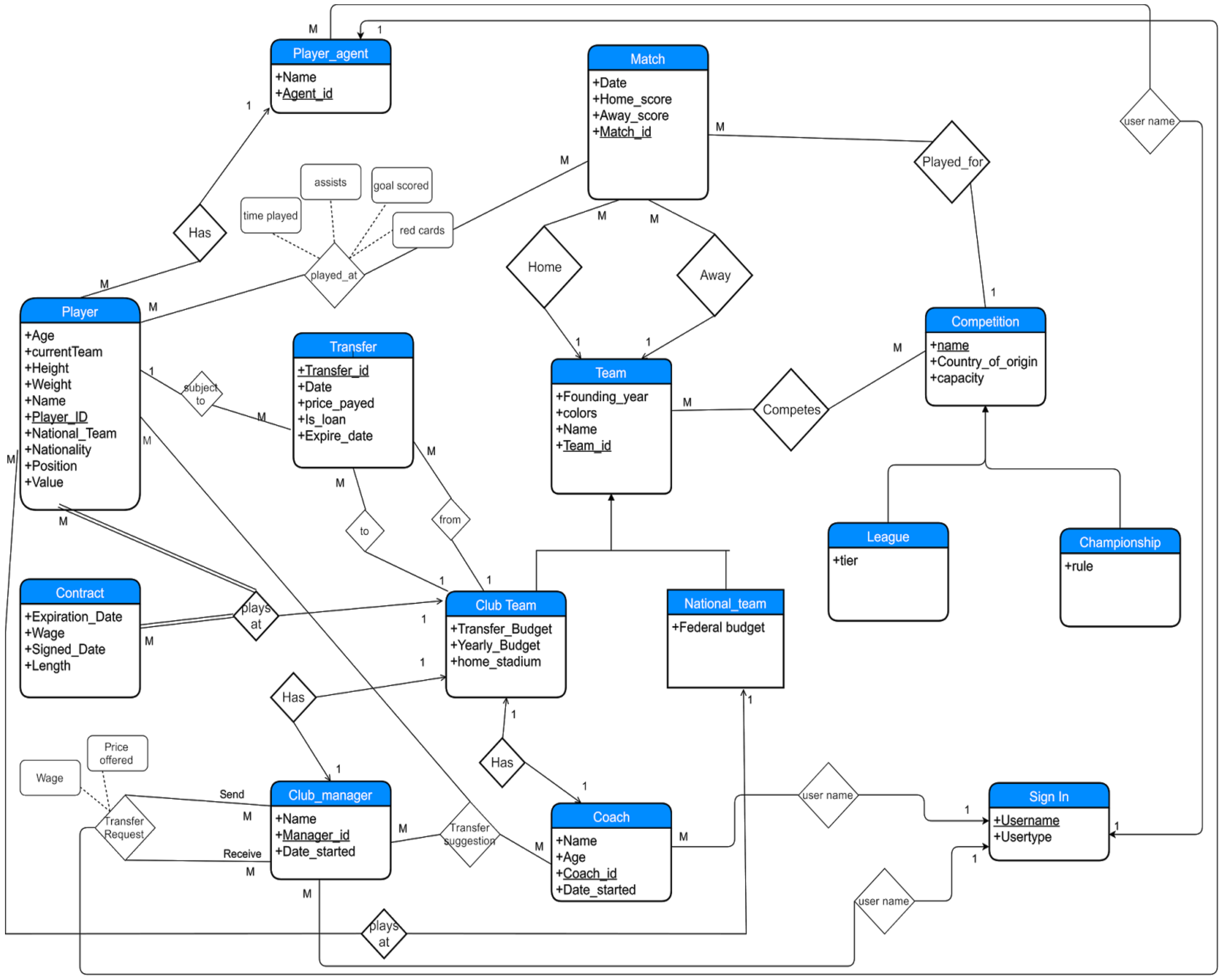


Figure 1: Revised Entity relation model

2-) Table schemas:

2.1-) Player

Relational Model

player (player_id, jersey_number, age, height, weight, name, national_team_id, club_team_id, agent_id, nationality, position, value)

Functional Dependencies:

national_team_id -> nationality

Candidate Keys:

player_id

Foreign Keys:

agent_id, natioal_team_id, club_team_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table player (
 player_id VARCHAR(10) PRIMARY KEY ,
 jersey_number INT ,
 age INT NOT NULL ,
 height INT NOT NULL ,
 weight INT NOT NULL ,
 name VARCHAR(50) NOT NULL ,
 national_team_id VARCHAR(10),
 club_team_id VARCHAR(10),
 agent_id VARCHAR(10),
 nationality VARCHAR(20) NOT NULL ,
 position VARCHAR(10),
 value INT,
 FOREIGN KEY (agent_id) REFERENCES player_agent,
 FOREIGN KEY (national_team_id) REFERENCES national_team,
 FOREIGN KEY (club_team_id) REFERENCES club_team
);
```

2.2-) Player Agent

Relational Model

player_agent(agent_id, name)

Functional Dependencies:

-

Candidate Keys:

agent_id

Foreign Keys:

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table player_agent(
 agent_id VARCHAR(10) PRIMARY KEY ,
 name VARCHAR(20) NOT NULL
);
```

2.3-) Transfer

Relational Model

transfer(transfer_id, player_id, from_team_id, to_team_id, price_payed, date, is_loan, expire_date)

Functional Dependencies:

If is_loan then expire_date exists.

Candidate Keys:

trasnfer_id

Foreign Keys:

player_id, from_team_id, to_team_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table transfer(
 transfer_id VARCHAR(10) PRIMARY KEY ,
 player_id VARCHAR(10),
 from_team_id VARCHAR(10),
 to_team_id VARCHAR(10),
 price_payed INT,
 date TIMESTAMP NOT NULL,
 is_loan TINYINT DEFAULT 0,
 FOREIGN KEY (player_id) REFERENCES player,
 FOREIGN KEY (from_team_id) REFERENCES club_team,
 FOREIGN KEY (to_team_id) REFERENCES club_team
);
```

2.4-) Contract

Relational Model

contract(signed_date, player_id, team_id, expiration_date, wage)

Functional Dependencies:

-

Candidate Keys:

(signed_date, player_id, team_id)

Foreign Keys:

player_id, team_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table contract(
 signed_date TIMESTAMP NOT NULL PRIMARY KEY  ,
 player_id VARCHAR(10) PRIMARY KEY ,
 team_id VARCHAR(10) PRIMARY KEY ,
 expiration_date TIMESTAMP NOT NULL,
 wage INT ,
 FOREIGN KEY (player_id) REFERENCES player,
 FOREIGN KEY (team_id) REFERENCES club_team
);
```

2.5-) Club Team

Relational Model

club_team(team_id, name, colors, coach_id, manager_id, home_stadium, founding_year, transfer_budget, yearly_budget)

Functional Dependencies:

-

Candidate Keys:

team_id

Foreign Keys:

coach_id, manager_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table club_team(
 team_id VARCHAR(10) PRIMARY KEY ,
 name VARCHAR(20) NOT NULL ,
 colors VARCHAR(20) NOT NULL ,
 coach_id VARCHAR(10) NOT NULL ,
 manager_id VARCHAR(10) NOT NULL ,
 home_stadium VARCHAR(20) NOT NULL ,
 founding_year INT NOT NULL ,
 transfer_budget INT,
 yearly_budget INT,
 FOREIGN KEY (coach_id) REFERENCES coach,
 FOREIGN KEY (manager_id) REFERENCES club_manager
);
```

2.6-) National Team

Relational Model

national_team(<u>team_id</u>, name, colors, founding_year, federal_budget)

Functional Dependencies:

name->

Candidate Keys:

team_id

Foreign Keys:

-

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table national_team(
 team_id VARCHAR(10),PRIMARY KEY ,
 name VARCHAR(20) NOT NULL ,
 colors VARCHAR(20) NOT NULL ,
 founding_year INT NOT NULL ,
 federal_budget INT,
);
```

2.7-) Club Manager

Relational Model

club_manager(<u>manager_id</u>, name, date_started)

Functional Dependencies:

-

Candidate Keys:

manager_id

Foreign Keys:

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table club_manager(
 manager_id VARCHAR(10) PRIMARY KEY ,
 name VARCHAR(20) NOT NULL ,
 date_started TIMESTAMP NOT NULL
);
```

2.8-) Coach

Relational Model

coach(name,age,<u>coach_id</u>, team_id date_started)

Functional Dependencies:

-

Candidate Keys:

coach-id

Foreign Keys:

team_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table coach(
 name VARCHAR(20) NOT NULL ,
 age INT,
 coach_id VARCHAR(10) PRIMARY KEY ,
 date_started TIMESTAMP NOT NULL,
 team_id VARCHAR(10),
 FOREIGN KEY (team_id) REFERENCES club_team
);
```

2.9-) League

Relational Model

league(name, capacity, country_of_origin, tier)

Functional Dependencies:

-

Candidate Keys:

name

Foreign Keys:

-

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table league(
 name VARCHAR(20) PRIMARY KEY ,
 capacity INT,
 country_of_origin VARCHAR(20) NOT NULL
);
```

2.10-) Championship

Relational Model

championship(<u>name</u>, capacity, country_of_origin, rule)

Functional Dependencies:

-

Candidate Keys:

name

Foreign Keys:

-

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table championship(
 name VARCHAR(20) PRIMARY KEY ,
 capacity INT,
 country_of_origin VARCHAR(20) NOT NULL,
 rule VARCHAR(200)
);
```

2.11-) Match

Relational Model

match(match_id, home_score, away_score, home_team_id, away_team_id, date,

competition_name)

Functional Dependencies:

-

Candidate Keys:

match_id

Foreign Keys:

home_team_id, away_team_id, competition_name

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table match(
 match_id VARCHAR(10) PRIMARY KEY ,
 home_score INT NOT NULL ,
 away_score INT NOT NULL ,
 home_team_id VARCHAR(10) NOT NULL ,
 away_team_id VARCHAR(10) NOT NULL ,
 date TIMESTAMP NOT NULL ,
 competition_name VARCHAR(20),
 FOREIGN KEY (home_team_id) REFERENCES (club_team,national_team),
 FOREIGN KEY (away_team_id) REFERENCES (club_team,national_team)
);
```

2.12-) Player Played at Match

Relational Model

player_match(player_id, match_id, time_played, goal_scored, assist, red_cards)

Functional Dependencies:

-

Candidate Keys:

{(player_id, match_id)}

Foreign Keys:

Player_id, match_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table player_match(
 player_id VARCHAR(10) PRIMARY KEY ,
 match_id VARCHAR(10) PRIMARY KEY ,
 time_played INT,
 goal_scored INT,
 assist INT,
 red_card TINYINT DEFAULT 0,
 FOREIGN KEY (player_id) REFERENCES player,
 FOREIGN KEY (match_id) REFERENCES match
);
```

2.13-) Club League

Relational Model

club_league(club_id, league_name)

Functional Dependencies:

-

Candidate Keys:

{(club_id, league_name)}

Foreign Keys:

club_id, league_name

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table club_league(
 club_id VARCHAR(10) PRIMARY KEY ,
 name VARCHAR(20) PRIMARY KEY ,
 FOREIGN KEY (club_id) REFERENCES club_team,
 FOREIGN KEY (name) REFERENCES league
);
```

2.14-) Coach to Club Manager Suggest

Relational Model

transfer_suggest(player_id, club_id)

Functional Dependencies:

-

Candidate Keys:

{(player_id, club_id)}

Foreign Keys:

Player_id, club_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table transfer_suggest(
 player_id VARCHAR(10) PRIMARY KEY ,
 club_id VARCHAR(10) PRIMARY KEY ,
 FOREIGN KEY (player_id) REFERENCES player,
 FOREIGN KEY (club_id) REFERENCES club_team
);
```

2.15-) Pending Transfer-Contract Request

Relational Model

pending_transfer_contract(<u>player_id, from_club_id, to_club_id</u>, is_pending, value, wage)

Functional Dependencies:

-

Candidate Keys:

{(player_id, from_club_id, to_club_id)}

Foreign Keys:

player_id, from_club_id, to_club_id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table pending_transfer_contract(
 player_id VARCHAR(10) PRIMARY KEY ,
 from_club_id VARCHAR(10) PRIMARY KEY ,
 to_club_id VARCHAR(10) PRIMARY KEY ,
 is_pending TINYINT DEFAULT 0,
 value INT NOT NULL ,
 wage INT,
 FOREIGN KEY (player_id) REFERENCES player,
 FOREIGN KEY (from_club_id) REFERENCES club_team,
 FOREIGN KEY (to_club_id) REFERENCES club_team
);
```

2.16-) Sign In

Relational Model

signin(<u>username</u>, user_type, id)

Functional Dependencies:

-

Candidate Keys:

{(username)}

Foreign Keys:

id

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table signin(
 usaername VARCHAR(20) PRIMARY KEY,
 user_type VARCHAR(20) NOT NULL ,
 id VARCHAR(10) NOT NULL ,
 FOREIGN KEY (id) REFERENCES club_manager,
 FOREIGN KEY (id) REFERENCES coach,
 FOREIGN KEY (id) REFERENCES player_agent
);
```

3-) Functional Dependencies and Normalization

If there is a functional dependency it is mentioned in the Table Schemas section of this report. Boyce-Codd Normal Form(BCNF) used for the normalization procedure of the relations. In this project it is concluded that there is no need to any decomposition.

4-) Functional Components

4.1-) Use Cases

In this project, there are 4 types of users which are regular user, coach, player agent and club manager. All types of users have different permissions. A regular user can open the site without any registration operation. If a user signs in with his pre-provided (by admin) username and password, this user can reach some extra features as his user type. The features that each user type has will be explained. For preventing complexity, use cases for different user types are drawn in different diagrams.

4.1.1-) Regular User

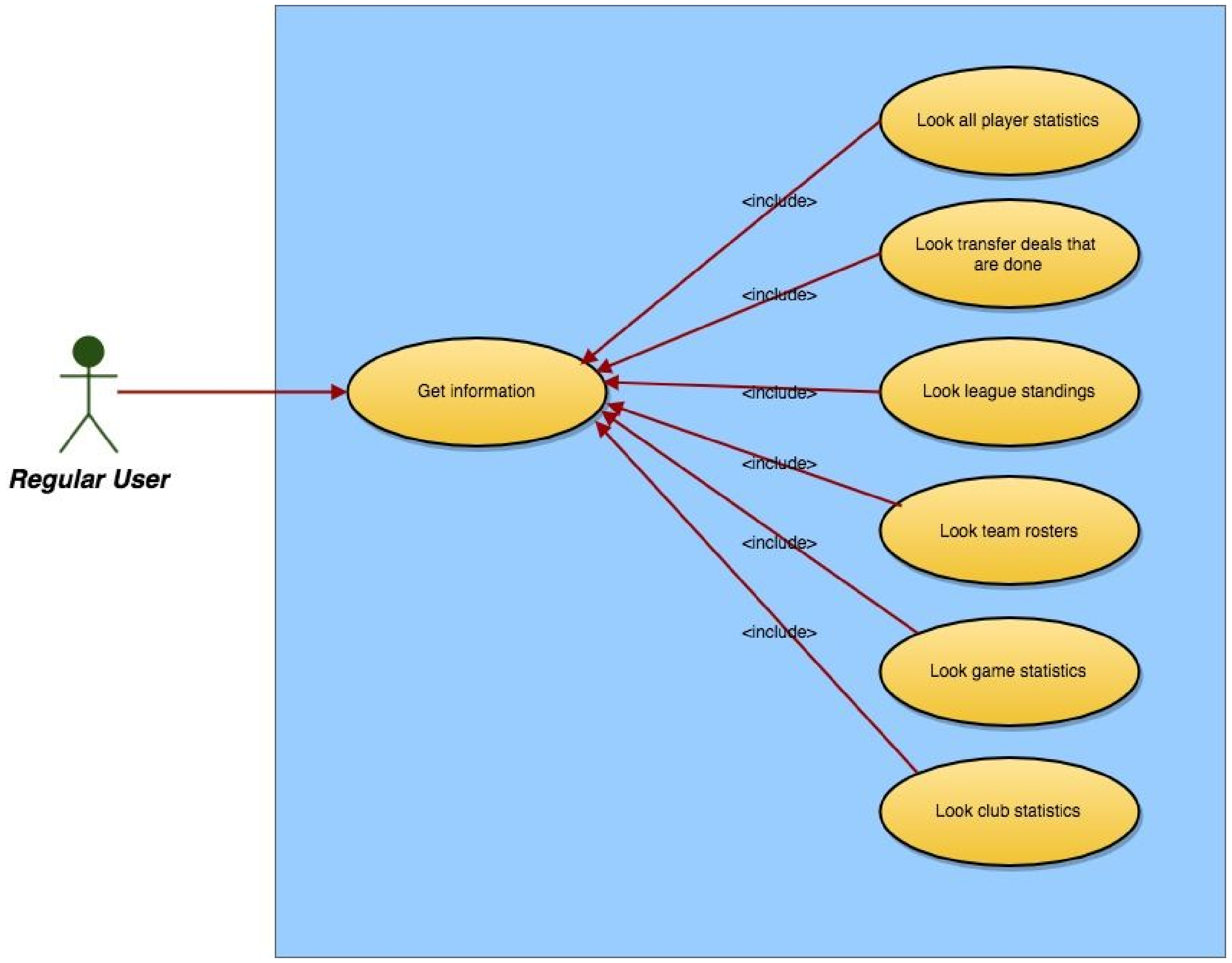


Figure 2: Regular user use case diagram

- Regular users can just get information, they cannot set anything.
- Regular users can look standings of every league in the system. User can select the league name to see league standings and features. Standings are included number of matches won, number of matches lost, number of draws, number of goals for, number of goals against and total points for each team.

- Regular users can look team rosters in the leagues. The user can select the team name to see the team features, the player list of the team and all matches that the team played.
- Regular users can look details of every match. User can see the statistics of game, which are number of goals, number of assists and number of red cards shown. Who scored the goals and who is shown the red card information is also given.
- Regular users can look the player list of the team. When the user select the name of player, player profile and statistics would be shown. Statistics are included number of goals scored, number of assists and number of red cards shown in the entire career of the player.
- Regular users can look done transfer deals and the detailed information about these deals, which are transfer cost, transfer date, transferred by which club and transferred from which club.

4.1.2-) Coach

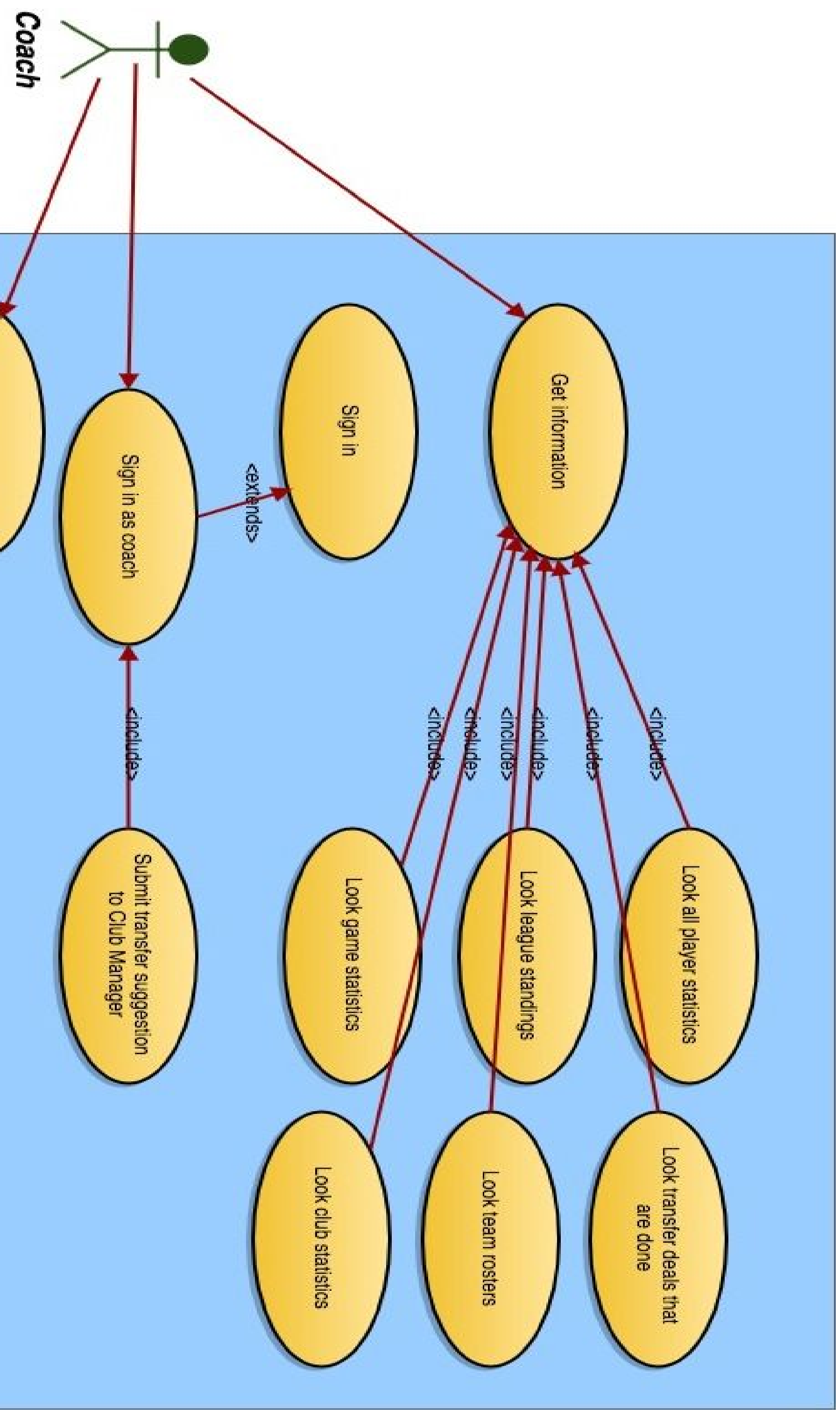


Figure 3: Coach user use case diagram

- Coaches can reach everywhere that regular users reach. For being a coach, you should have pre-provided username and password.
- Coaches can submit transfer suggestion for the players to club manager of coaches' teams. This transfer suggest is only a recommendation, which means the club manager do not have to submit offers to these players.

4.1.3-) Player Agent

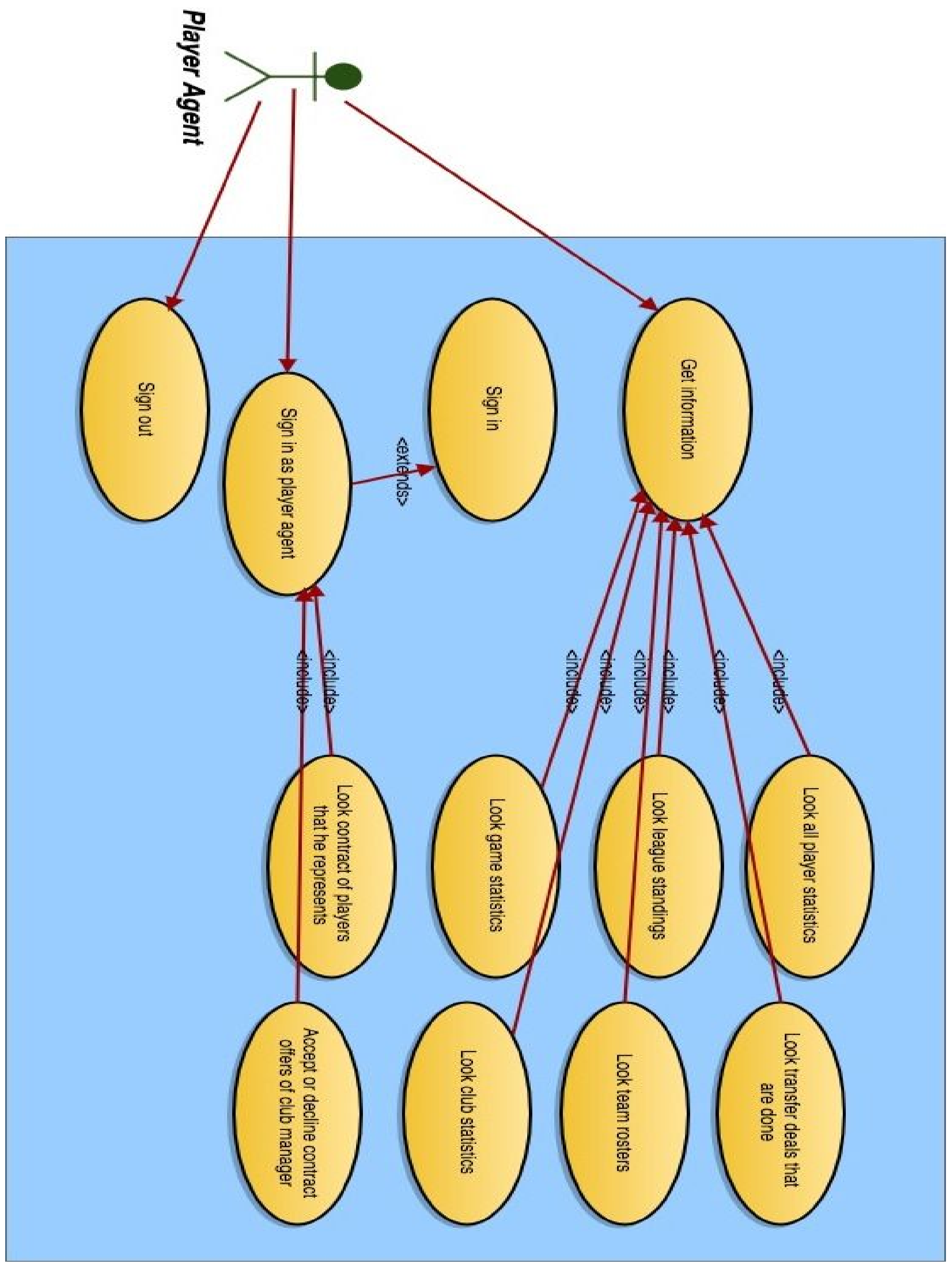


Figure 4: Player agent use case diagram

- Player agents can reach everywhere that regular users reach. For being a player agent, you should have pre-provided username and password.
- Player agents have a permission to look contracts of players that they represents.
- Player agents can accept or decline contract offers which are offered by club managers. If a player plays in a club, only the club manager of this club can offer a contract to player agent.

4.1.4-) Club Manager

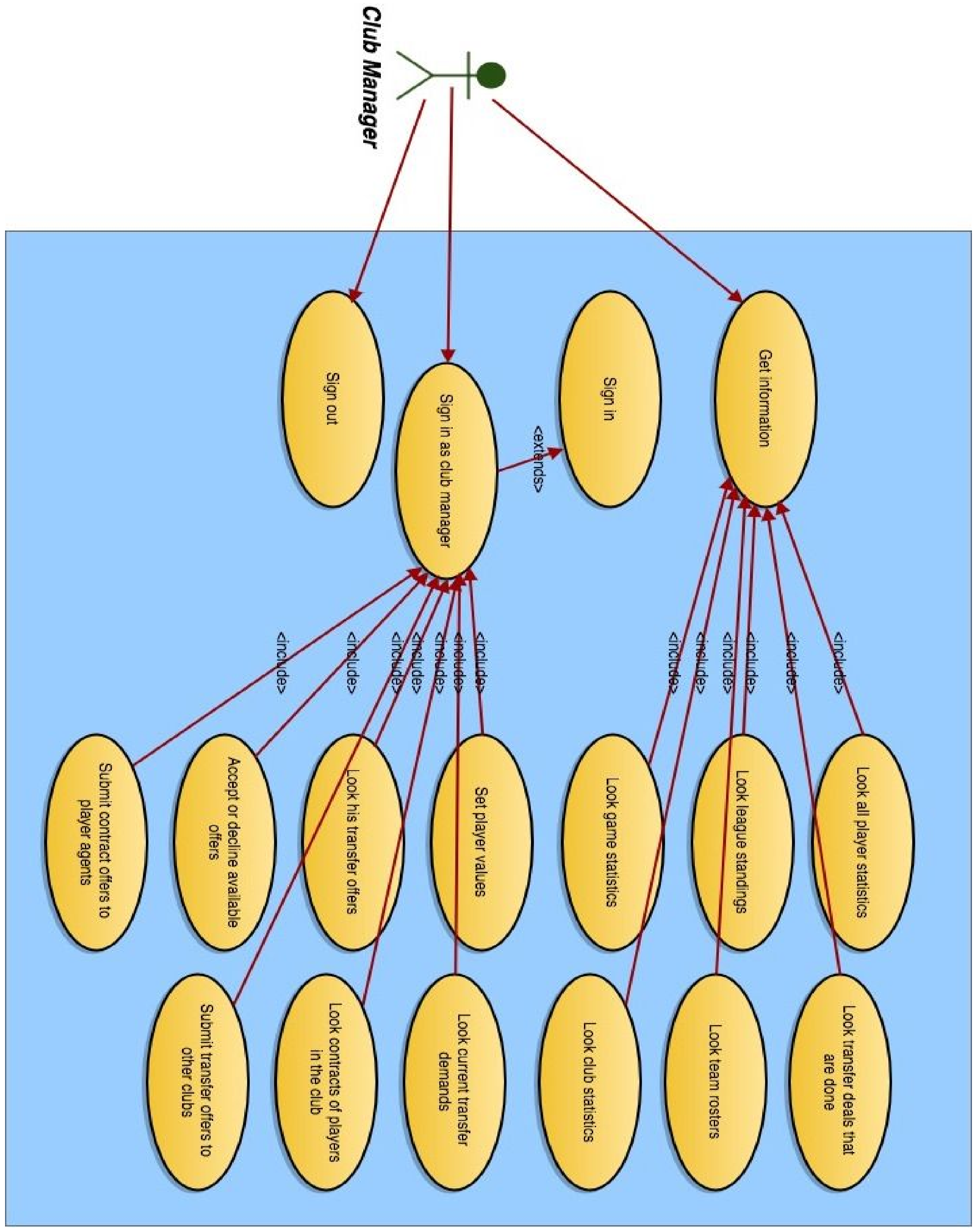


Figure 5: Club manager user case diagram

- Club managers can reach everywhere that regular users reach. For being a club manager, you should have pre-provided username and password.
- Club managers can set a new value to players which plays in the same team with club manager.
- Club managers can look current transfer demands which is provided by team coaches.
- Club managers can submit transfer offers for players to other clubs.
- Club managers can look transfer offers for players done by themselves.
- Club managers can accept or decline available transfer offers.
- Club managers can only look contracts of players who plays in the same club of club manager.
- Club managers can submit contract offers to player agents for players who play in the same club with manager.

4.2-) Algorithms

4.2.1-) Sorting Related Algorithms:

Sorting algorithms will be used when displaying some of the entities of the database. For example players can be selected to be viewed in descending order. Appropriate sorting algorithms will increase the efficiency of the database.

4.2.2-) League Standing Computation Algorithm

An algorithm which uses the Match results(goal numbers of each team) from the Match table constructed to compute total team points.

4.3-) Data Structures

Following data structures will be used

- TIMESTAMP: In this database TIMESTAMP data type will be used to ease the sorting process.
- VARCHAR: for representing text strings
- INT: for holding integer values
- TINYINT: for logical values True and False, instead of using INT for 1 and 0, TINYINT type will increase the performance of the database.
- CURDATE(): this MySQL function will be used to compare date values with current date.

5-) User Interface design and SQL statements

5.1-) Normal User

5.1.1-) Sign in

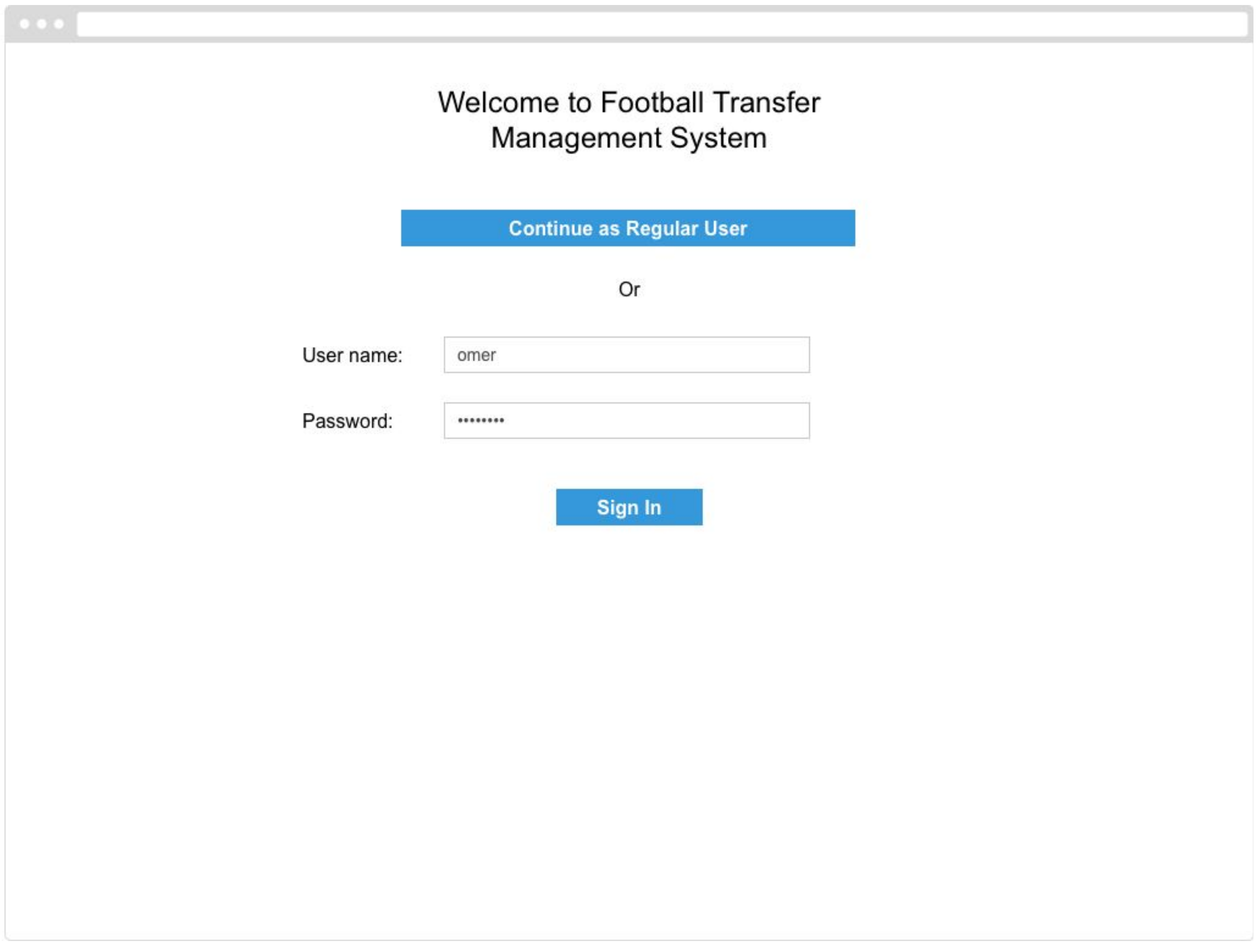


Figure 6: Mockup of sign in page

Functionality: The purpose of this page is to determine who is using the football transfer system. The regular users do not need to sign in since no specific information for a regular user is recorded in the database. However; club coaches, club managers, and player agents need to sign-in in order to be recognized by the system. The sign in also ensures that user information will be confidential.

The following SQL statement will be used to determine the user type if login is attempted. The reason for having such a query is so that we can use external frameworks for a secure login while also determining the user type.

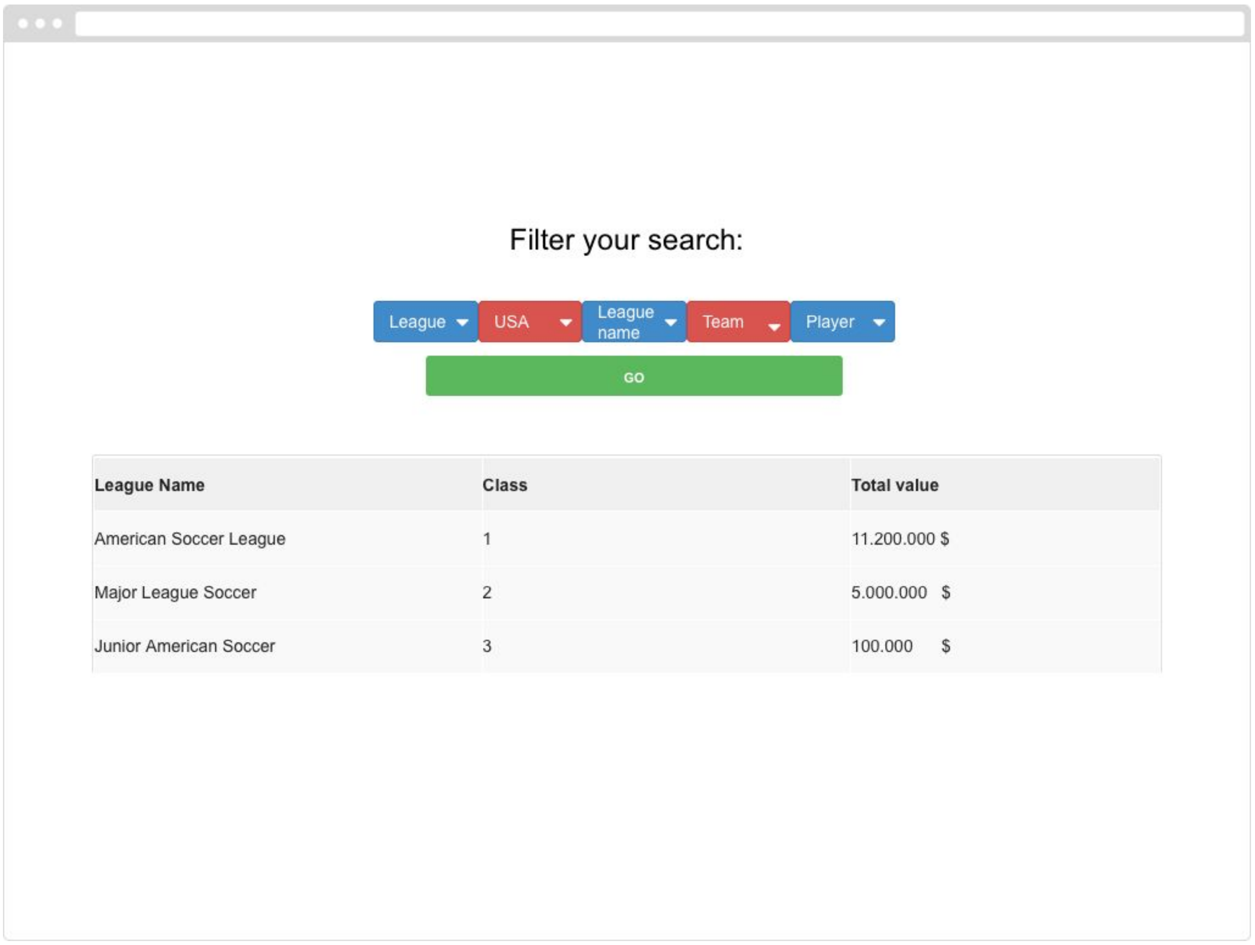
All three statements will be executed and only one will return results therefore we will be able to determine the user type. If no sign in is made or no result is returned the normal user type will be selected as default.

Input : @username -> username used during login

SQL:

```
select user_type, id
from signin, club_manager
where club_manager.manager_id = signin.id and
     signin.username = @username

select user_type, id
from signin, coach
where coach.coach_id = signin.id and
     signin.username = @username

select user_type, id
from signin, player_agent
where player_agent.agent_id = signin.id and
     signin.username = @username
```

5.1.2-) List Leagues



League Name	Class	Total value
American Soccer League	1	11.200.000 $
Major League Soccer	2	5.000.000 $
Junior American Soccer	3	100.000 $

Figure 7: Mockup of listed leagues

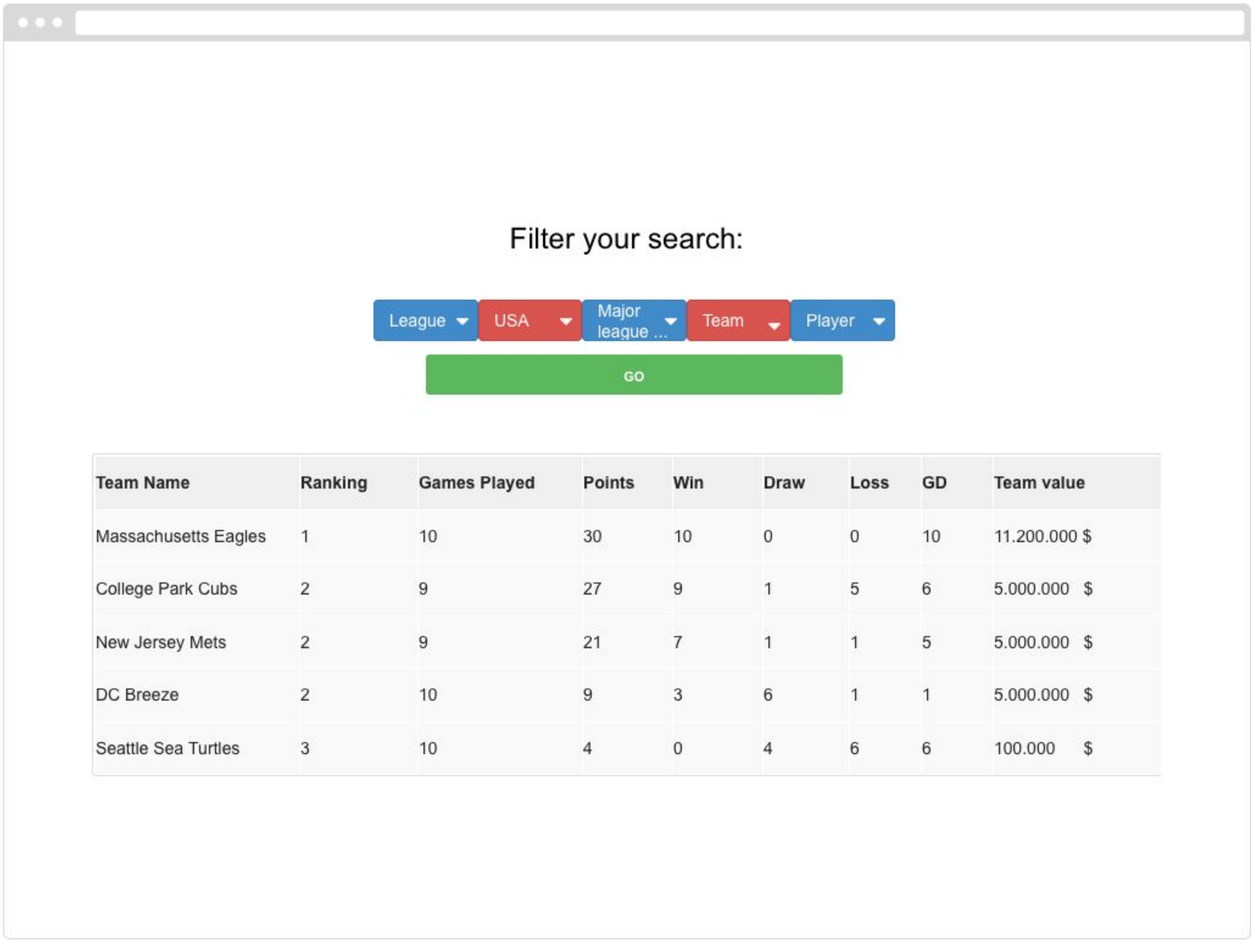
This page is the first version of the search function. The search function works as follows. There are 5 criteria to limit the search from. The scope of each criterion is larger than its left neighbors. For example, league or championship is more general than the country of the competition. Similarly, a team is more general than a player. The most specific option selected will be displayed. For instance in this case country is the most specific criterion that is selected therefore, we see the leagues in that country. More specific search results can be seen in the following sections.

The following query returns the exact table shown in figure 6.

SQL:

```
select l.name, l.tier, sum(lp.value) as league_value
from league l, (     select value
                from player p, club_league cl, league l
                where p.club_team_id = cl.club_id
                and cl.league_name = l.name) as lp
group by l.name
```

5.1.3-) League Standings and Match Details



Team Name	Ranking	Games Played	Points	Win	Draw	Loss	GD	Team value
Massachusetts Eagles	1	10	30	10	0	0	10	11.200.000 $
College Park Cubs	2	9	27	9	1	5	6	5.000.000 $
New Jersey Mets	2	9	21	7	1	1	5	5.000.000 $
DC Breeze	2	10	9	3	6	1	1	5.000.000 $
Seattle Sea Turtles	3	10	4	0	4	6	6	100.000 $

Figure 8: Mockup of league standings

The intention of this interface is to inform the user of the current standings as well as the values of each team in the selected league. If we refer to the explanation in 5.1.2 we can say that the league is the most specific criterion specified therefore, we see the league standings as a result. As we can see in figure 7 some team statistics can be seen as well. The rankings are determined by the points. In case of a tie the goal difference is taken into account.

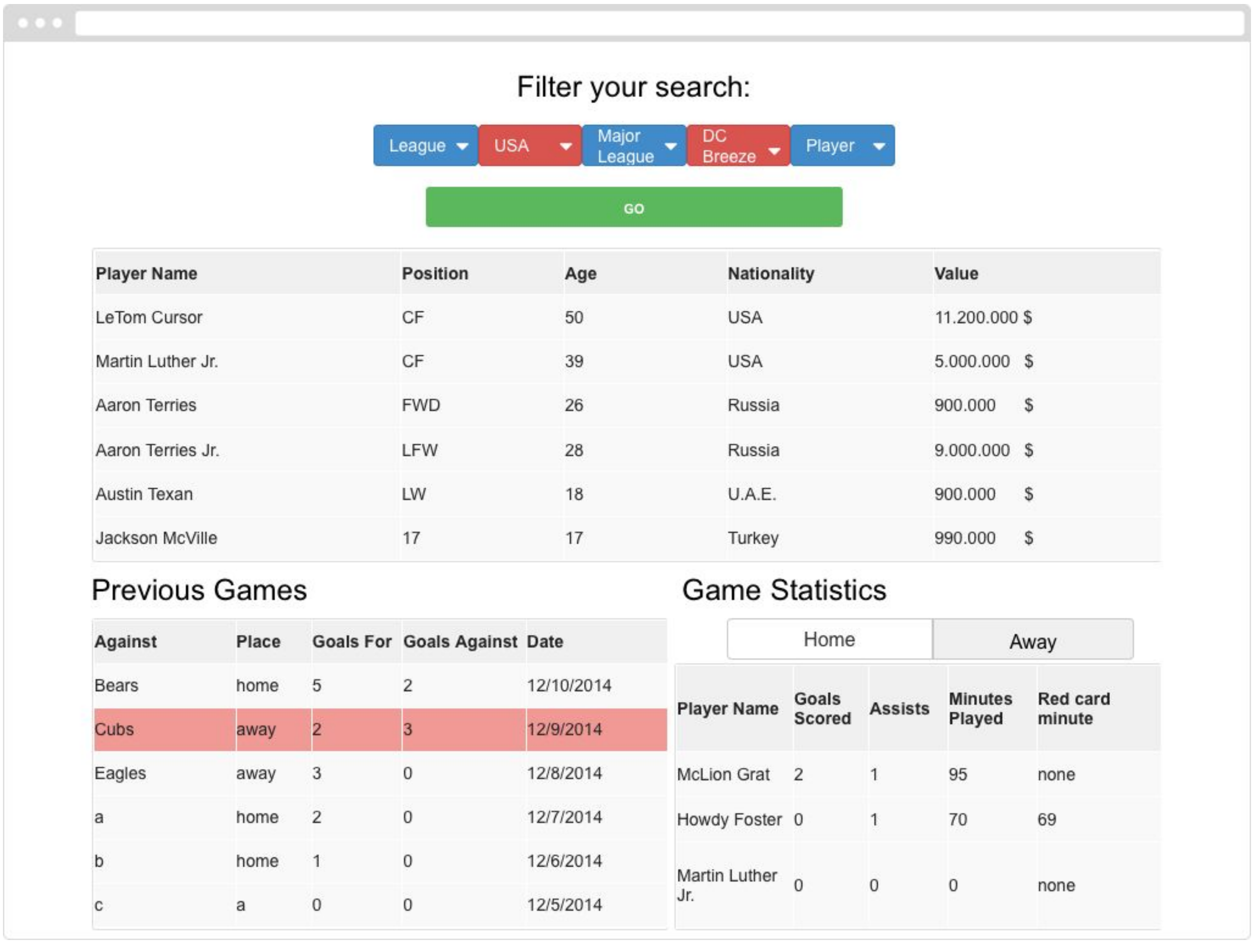
Multiple tables are generated with the 'with' statement and combined in the main query. The query is quite lengthy in order to generate nearly all of the result table. The ranking column however is not generated because it is quite easy to do it with code.

Input: @league_name -> name of a league
SQL:

```
with home_stats(f, a, team_id) as
(select home_score, away_score, club_team.team_id
from match, club_team
where club_team.team_id = match.home_team_id
     and club_team.team_id in (select club_id as team_id
                                      from club_league
                                      where name =
league_name)),

away_stats(f,a,team_id) as
(select away_score, home_score, club_team.team_id
from match, club_team
where club_team.team_id = match.away_team_id
     and club_team.team_id in (select club_id as team_id
                                      from club_league
                                      where name =
league_name)),

all_stats(f,a,team_id) as
(select * from home_stats
union all
select * from home_stats),

team_val(val, team_id) as
(select sum(player.value) as val, teams.team_id
from (select unique team_id from all_stats) teams, player
where teams.team_id = player.club_team_id
group by teams.team_id)

wins(w,team_id) as
(select count(*), team_id
from all_stats
where f > a
group by team_id),

draws(d,team_id) as
(select count(*), team_id
from all_stats
where a = f
group by team_id),

losses(l,team_id) as
(select count(*), team_id
from all_stats
where f < a
```

```
group by team_id),

points(pts, team_id) as
(select isnull(d,0)+isnull(w,0)*3 as pts, team_id
from wins natural join draws),

gd_gp(gd, gp, team_id) as
(select sum(f) - sum(a) as gd, count(*), team_id
from all_stats
group by team_id)

select club_team.name, gp as game_played, points.pts, wins.w as win,
draws.d as draw, losses.l as loss, gd, team_val.val as Team_value
from team_val full join wins full join draws full join losses
     full join points full join gd_dp natural join club_team
order by pts desc
```

5.1.4-) Team Details, Past Matches and Statistics



Player Name	Position	Age	Nationality	Value
LeTom Cursor	CF	50	USA	11.200.000 $
Martin Luther Jr.	CF	39	USA	5.000.000 $
Aaron Terries	FWD	26	Russia	900.000 $
Aaron Terries Jr.	LFW	28	Russia	9.000.000 $
Austin Texan	LW	18	U.A.E.	900.000 $
Jackson McVille	17	17	Turkey	990.000 $

Previous Games

Against	Place	Goals For	Goals Against	Date
Bears	home	5	2	12/10/2014
Cubs	away	2	3	12/9/2014
Eagles	away	3	0	12/8/2014
a	home	2	0	12/7/2014
b	home	1	0	12/6/2014
c	a	0	0	12/5/2014

Game Statistics

Player Name	Goals Scored	Assists	Minutes Played	Red card minute
McLion Grat	2	1	95	none
Howdy Foster	0	1	70	69
Martin Luther Jr.	0	0	0	none

Figure 9: Mockup of team roster, previous games and statistics page

In this view of the system the user is given the information of the teams. The selected team roster is shown as well as the past games in the league. The player statistics can be seen for the game selected as well.

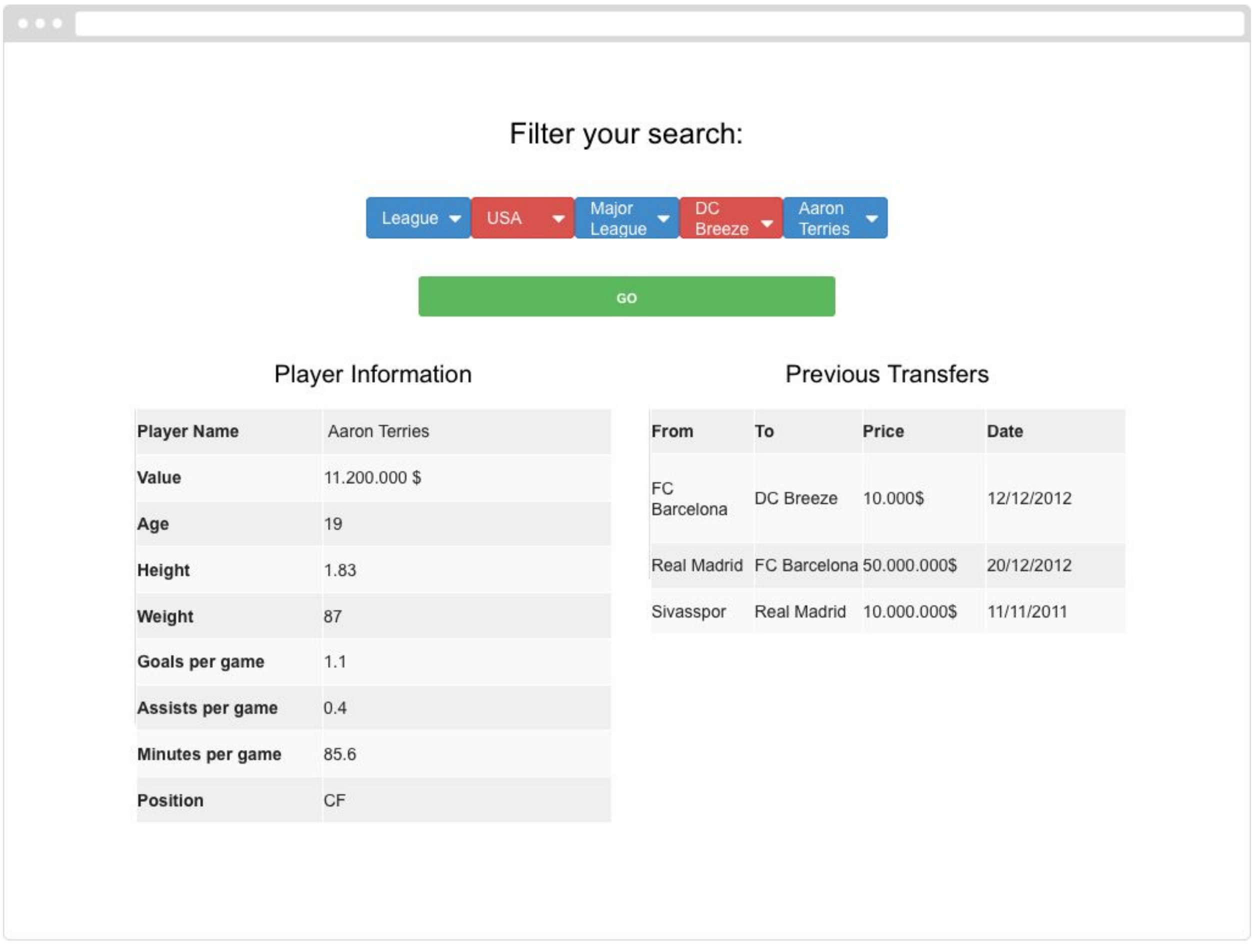
INPUT: @team_name -> name of a team

SQL:

```
select name, position, age, nationality, value
from player
where club_team_id = @team_name

with team_id(id) as (select team_id
                     from club_team
```

```
                    where name = @team_name),

previous_games(against, place, goals_for, goals_against, date) as
(select c.name, ‘home’ as place, sum(pm.goal_scored) as total_goal,
h.date
from club_team c, player_match pm, ( select match_id, date,
                                     away_team_id
                                     from match, team_id
                                     where home_team_id = id)
                                     as home_match h,
where pm.match_id = h.match_id and c.team_id = h.away_team_id
group by h.match_id
natural join
select c.name, ‘away’ as place, sum(pm.goal_scored) as total_goal,
h.date
from club_team c, player_match pm, ( select match_id, date,
                                     home_team_id
                                     from match, team_id
                                     where away_team_id = id)
                                     as away_match a,
where pm.match_id = h.match_id and c.team_id = a.home_team_id
group by h.match_id)

select *
from previous_games
order by time desc
select p.name, pm.goal_scored, pm.assist, pm.time_played,
pm.red_cards
from player p, player_match pm
where p.player_id = pm.player_id
```

5.1.5-) Player Details and Previous Transfers



Player Information

Player Name	Aaron Terries
Value	11.200.000 $
Age	19
Height	1.83
Weight	87
Goals per game	1.1
Assists per game	0.4
Minutes per game	85.6
Position	CF

Previous Transfers

From	To	Price	Date
FC Barcelona	DC Breeze	10.000$	12/12/2012
Real Madrid	FC Barcelona	50.000.000$	20/12/2012
Sivasspor	Real Madrid	10.000.000$	11/11/2011

Figure 10: Mockup of player information and previous transfers page.

This page shows the most specific information available in the system for a normal user. The player information together with some statistics are given. The previous transfers of the selected player can also be seen by the user.

Following are the queries in order to generate the tables in figure 9.

VALUE: @player_name-> name of the player,

@club_name-> name of the club

SQL:

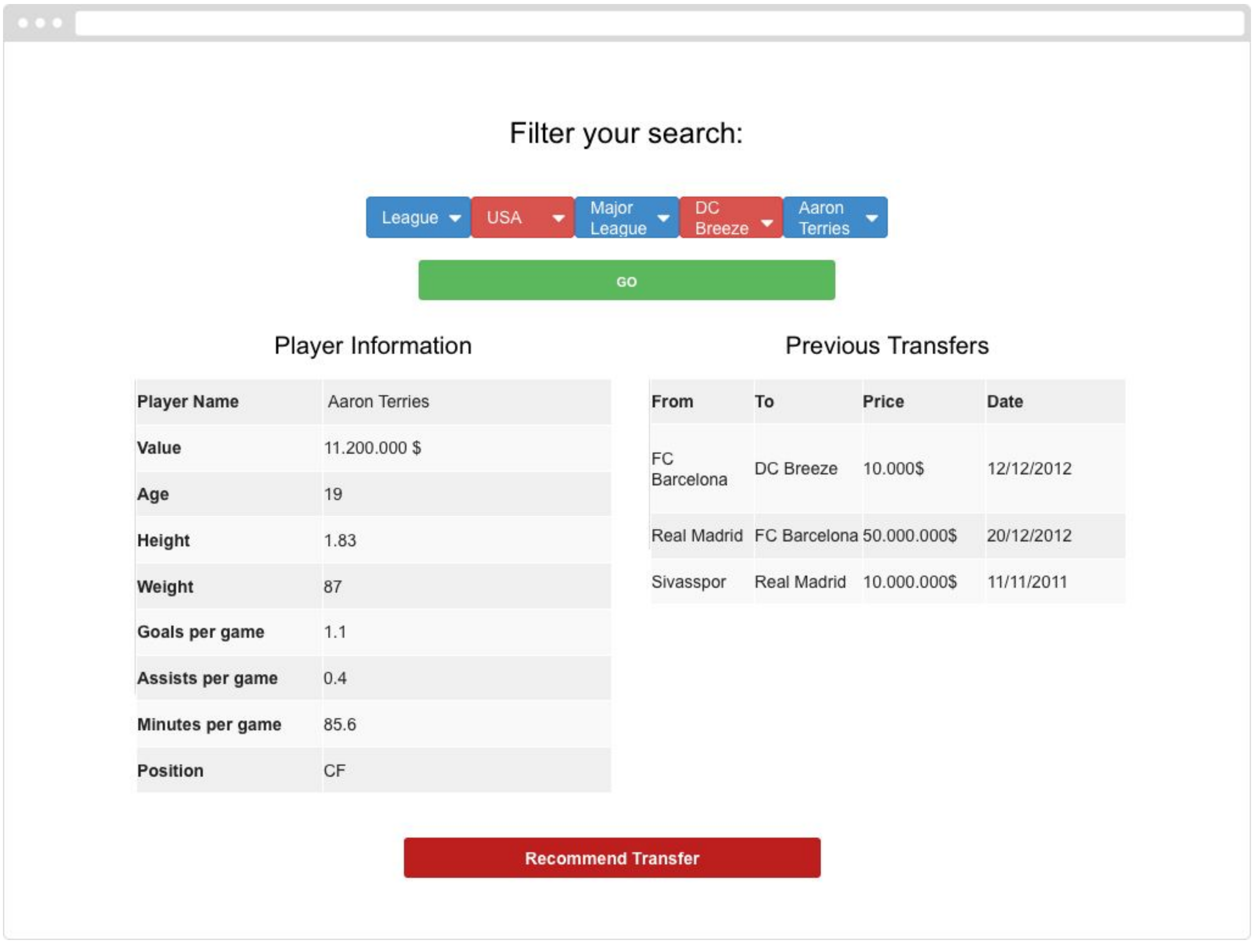
```
with pid(player_id) as (   select player_id
                           from player p, club c
                           where p.name = @player_name
```

```
                          and c.name = @club_name
                          and c.team_id = p.club_team_id)

select p.name, p.value, p.age, p.height, p.weight,
avg(pm.goal_scored), avg(pm.assist), avg(pm.time_played), p.position
from player p, player_match pm, pid pid
where pid.player_id = p.player_id and pid.player_id = pm.player_id

with from_club_name(name, player_id) as
                                (select c.name
                                from club_team c, transfer t
                                where c.team_id = t.from_team_id),
to_club_name(name, player_id) as
                           (select c.name, t.player_id
                           from club_team c, transfer t
                           where c.team_id = t.to_team_id)

select fc.name, tc.name, t.price_payed, t.date
from from_club_name fc, to_club_name tc, transfer t, pid p
where fc.player_id = p.player_id
and tc.player_id = p.player_id
and t.player_id = p.player_id
```

5.2-) Club Coach

All of the user interfaces and the interactions are same with the regular user except the following.

5.2.1-) Player Details, Previous Transfers and Transfer Requests



Player Information

Player Name	Aaron Terries
Value	11.200.000 $
Age	19
Height	1.83
Weight	87
Goals per game	1.1
Assists per game	0.4
Minutes per game	85.6
Position	CF

Previous Transfers

From	**To**	**Price**	**Date**
FC Barcelona	DC Breeze	10.000$	12/12/2012
Real Madrid	FC Barcelona	50.000.000$	20/12/2012
Sivasspor	Real Madrid	10.000.000$	11/11/2011

Figure 11: Mockup of player information and previous transfers page for coaches

This page is unique for the coach users. The player information and previous transfers are given however a new user interface element is also introduced. The red button seen in figure 11 is there so that coaches can make recommendations to their club managers.

The only difference in queries from section 5.1.5 is given below. This SQL statement is for inserting a new recommendation in the recommendation table.

VALUES:	@coach_id -> id of the coach
	@club_name -> name of the team
	@player_name -> name of the player

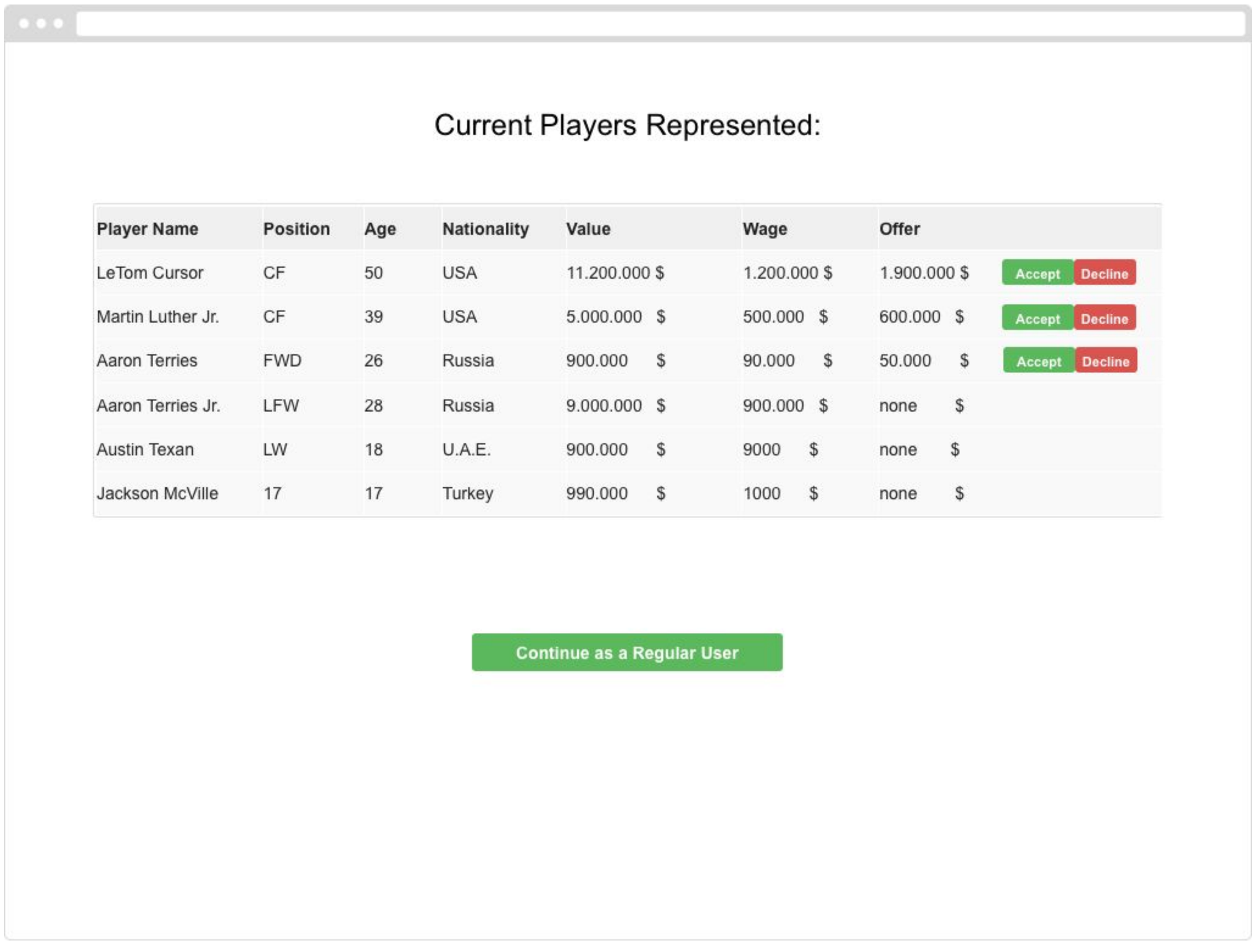
SQL:

```
select p.player_id
from player p, club_team c
where c.coach_id = @coach_id and c.name = @club_name
and c.team_id = p.club_team_id



with pcid(player_id, club_id)
as (  select p.player_id, c.team_id
      from player p, club c
      where p.name = @player_name
      and c.name = @club_name
      and c.team_id = p.club_team_id)

insert into transfer_suggest
values (pcid.player_id, pcid.club_id)
```

5.3-) Player Agent

All of the user interfaces and the interactions are same with the regular user except the following.

5.3.1-) Home page



Current Players Represented:

Player Name	Position	Age	Nationality	Value	Wage	Offer	
LeTom Cursor	CF	50	USA	11.200.000 $	1.200.000 $	1.900.000 $	Accept Decline
Martin Luther Jr.	CF	39	USA	5.000.000 $	500.000 $	600.000 $	Accept Decline
Aaron Terries	FWD	26	Russia	900.000 $	90.000 $	50.000 $	Accept Decline
Aaron Terries Jr.	LFW	28	Russia	9.000.000 $	900.000 $	none $	
Austin Texan	LW	18	U.A.E.	900.000 $	9000 $	none $	
Jackson McVille	17	17	Turkey	990.000 $	1000 $	none $	

Figure 12: Mockup of home page of player agent

This user interface is the homepage of player agents. The player agents can see all the players they represent in this window. The values and the wages of each player are given. Players that have a current offer are shown at the top of the list. The player agent can either accept the offer -in which case the transfer happens or the renewal of the contract happens- or can decline. In case of a declination negotiation is done by the renewal of the offer by the club manager of interest.

The following SQL statements are for the declination and acceptance cases of the offer.

VALUES: @agent_id -> id of the player agent

SQL:

```
select p.name, p.position, p.age, p.nationality, p.value, c.wage,
ptc.wage
from player p, contract c, pending_transfer_request ptc
where p.agent_id = @agent_id and c.player_id = p.player_id
and c.expiration_date > getdate() and ptc.player_id = p.player_id
and ptc.is_pending = 0
order by ptc.wage desc
natural join
select p.name, p.position, p.age, p.nationality, p.value, c.wage,
'none'
from player p, contract c, pending_transfer_request ptc
where p.agent_id = @agent_id and c.player_id = p.player_id
and c.expiration_date > getdate() and ptc.player_id != p.player_id
and ptc.is_pending = 0
```

If an offer is accepted or declined:

NEW_VALUES: @player_id -> id of the selected player

```
with cid(club_id) as (     select club_team_id
                           from player
                           and player_id = @player_id)
```

If accepted:

```
with wage(wage) as (  select wage
                      from pending_transfer_request
                      where player_id = @player_id)

insert into contract
values(getdate(), @player_id, cid.club_id, getdate() + 1, wage.wage)
```

If declined

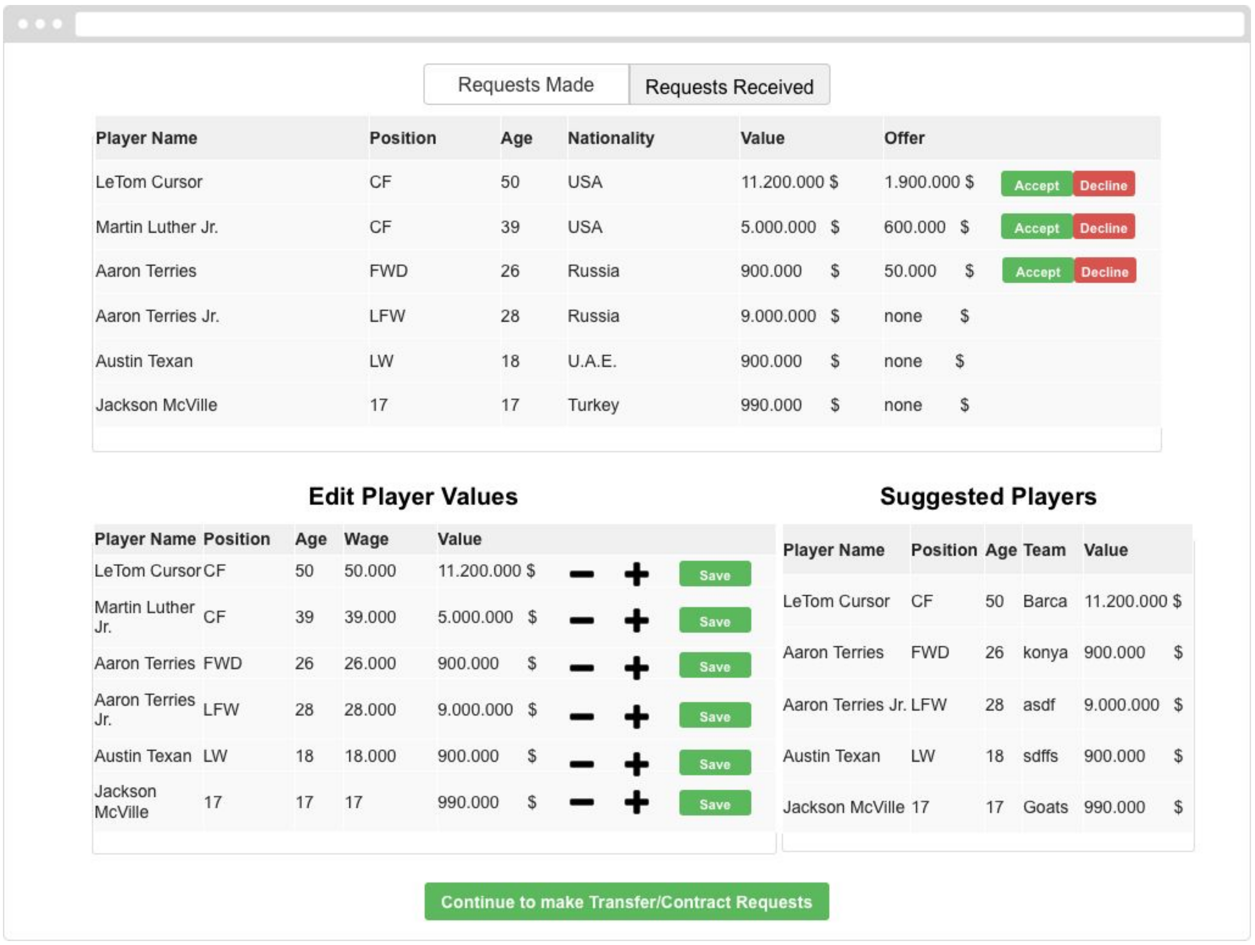
```
delete from pending_transfer_request
```

```
where player_id = @player_id
```

5.4-) Club Manager

All of the user interfaces and the interactions are same with the regular user except the following.

5.4.1-) Home page



Player Name	Position	Age	Nationality	Value	Offer	
LeTom Cursor	CF	50	USA	11.200.000 $	1.900.000 $	Accept Decline
Martin Luther Jr.	CF	39	USA	5.000.000 $	600.000 $	Accept Decline
Aaron Terries	FWD	26	Russia	900.000 $	50.000 $	Accept Decline
Aaron Terries Jr.	LFW	28	Russia	9.000.000 $	none $	
Austin Texan	LW	18	U.A.E.	900.000 $	none $	
Jackson McVille	17	17	Turkey	990.000 $	none $	

Edit Player Values

Player Name	Position	Age	Wage	Value			
LeTom Cursor	CF	50	50.000	11.200.000 $	−	+	Save
Martin Luther Jr.	CF	39	39.000	5.000.000 $	−	+	Save
Aaron Terries	FWD	26	26.000	900.000 $	−	+	Save
Aaron Terries Jr.	LFW	28	28.000	9.000.000 $	−	+	Save
Austin Texan	LW	18	18.000	900.000 $	−	+	Save
Jackson McVille	17	17	17	990.000 $	−	+	Save

Suggested Players

Player Name	Position	Age	Team	Value
LeTom Cursor	CF	50	Barca	11.200.000 $
Aaron Terries	FWD	26	konya	900.000 $
Aaron Terries Jr.	LFW	28	asdf	9.000.000 $
Austin Texan	LW	18	sdffs	900.000 $
Jackson McVille	17	17	Goats	990.000 $

Figure 13: Mockup of home page of club manager

The most sophisticated user interface is depicted in figure 13. This UI allows a club manager to see his sent request status, respond to the received requests, edit current player values, and see the transfer suggestions made by the team coach. If the manager wishes to make a transfer request s/he simply has to click on the green button in order to be directed to the player search screen.

The following sql statements are for the upper table the lower right table and the lower left table respectively.

VALUES: @manager_id -> id of the player agent

SQL:

```
select p.name, p.position, p.age, p.nationality, p.value, ptc.value
from player p, pending_transfer_request ptc
where p.manager_id = @manager_id and ptc.player_id = p.player_id
and ptc.is_pending = 1
order by ptc.value desc
natural join
select p.name, p.position, p.age, p.nationality, p.value, 'none'
from player p, pending_transfer_request ptc
where p.manager_id = @manager_id and ptc.player_id != p.player_id
and ptc.is_pending = 1
```

If an offer is accepted or declined:

NEW_VALUES: @player_id -> id of the selected player

```
with cid(club_id) as (     select club_team_id
                           from player
                           and player_id = @player_id)
```

If accepted:

```
with value(value) as (select value
                      from pending_transfer_request
                      where player_id = @player_id)

insert into transfer
values(getdate(), @player_id, cid.club_id, getdate() + 1,
value.value)
```

If declined

```
delete from pending_transfer_request
where player_id = @player_id
```

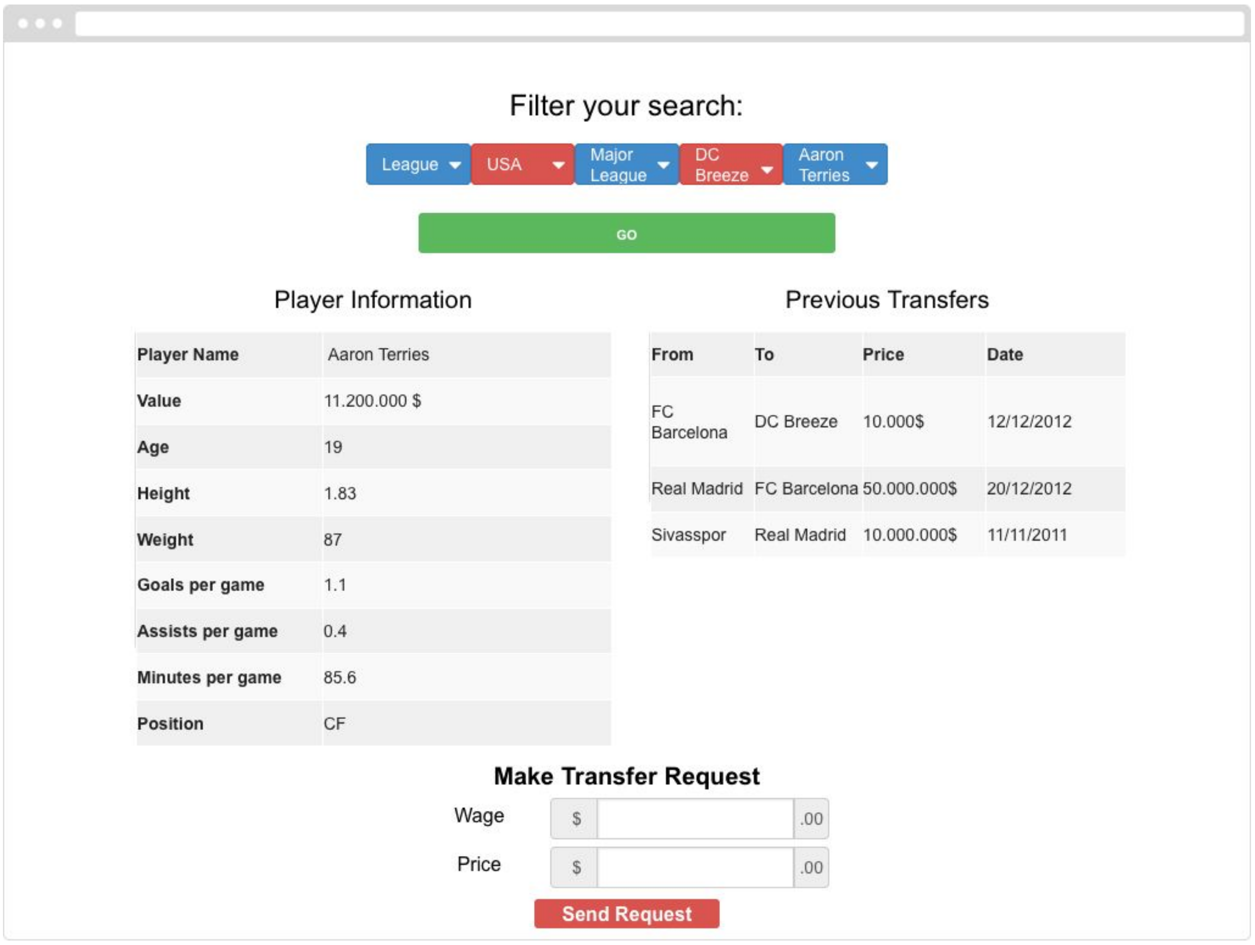
Edit player values query
Input: @value

```
select p.name, p.position, p.age, c.wage, p.value
from player p, contract c
where c.player_id = p.player_id

update player
set value = @value
```

Suggested players query
Input: @manager_id

```
select player.name, player.position, player.age, club_team.name,
player.value
from club_team natural join transfer_suggest, player
where player.player_id = transfer_suggest.player_id
     and @manager_id in (select manager_id
                         from club_team
                         where club_team.team_id =
                         transfer_suggest.club_id)
```

5.4.2-) Player Details, Previous Transfers and Sending Transfer Requests



Player Information

Player Name	Aaron Terries
Value	11.200.000 $
Age	19
Height	1.83
Weight	87
Goals per game	1.1
Assists per game	0.4
Minutes per game	85.6
Position	CF

Previous Transfers

From	**To**	**Price**	**Date**
FC Barcelona	DC Breeze	10.000$	12/12/2012
Real Madrid	FC Barcelona	50.000.000$	20/12/2012
Sivasspor	Real Madrid	10.000.000$	11/11/2011

Make Transfer Request

Wage $ .00

Price $ .00

Send Request

Figure 14: Mockup of player information and previous transfers page for club managers.

This is the user interface for the club manager to make a transfer request to any player. The price and wage to be payed is inserted in the fields and the request is made. The rest of the page is the same as in part 5.1.5.

INPUT: @player_id -> unique id of the player to send a request for.
@club_team_id> the team id of the current club manager
@value-> the value entered in the ui
@wage-> the wage entered in the ui

SQL:

```
insert into pending_transfer_contract(player_id, from_club_id,
to_club_id is_pending, value, wage)
select player_id, club_team.team_id, club_team.team_id, 1, @value,
@wage
from player, club_team
where player.player_id = @player_id
     and player.club_team_id = club_team.team_id
```

6-) Advanced Database Components

6.1-) Views

6.1.1-) Club Manager’s Transfer Suggestion View

A club manager can only view the transfer suggestions made by his/her own team’s coaches. But s/he can not see other team’s coaches transfer suggestions. So this can be handled by the following view.

```
create view all_suggestions as
        select value, position, name, age, team_name
        from player natural join team
        where Manager_id = @m_id
```

@m_id: manager who is viewing this list

6.1.2-) Club Manager’s Player Value View

A club manager can view the values of his/her club’s players. With this view s/he can change this values as s/he desires.

```
create view player_value as
        select value,wage, name, age, position
        from Player natural join Contract
        where Manager_id = @m_id
```

@m_id: manager who is viewing this list

6.1.3-) Regular User’s Previous Transfers View

This view will be available to all users since it is a View for Regular Users. Users can view the past transfers of a Player. For example following view will be used.

```
create view past_transfers as
        select from, to, price_payed, date
        from Club Team natural join Transfer natural join Player
        where Coach_id = @c_id
        order by date
```

@c_id: user who is viewing this list

6.1.4-) Player Agent’s Player View

In this view, player agent will see brief information about his/her players and have an option to accept or decline the transfer offers send to their players.

```
create view
        select name, position, age, nationality, value, wage, offer
        from Player natural join Contract join Transfer Suggestion natural join Club Manager
natural join Transfer Request
        where Agent_id = @a_id
```

@a_id: player agent who is viewing this list

6.2-) Stored Procedures

6.2.1-) Calculating Total Team Points in Leagues

In this database standard procedure will be used for total point calculation. *Home_score* and *Away_score* variables will be collected with unique *match_id.* Each teams points will be calculated with comparing the goal amounts. If the difference is greater than 0, 3 point will be added, if it is 0, 1 point will be added. These procedures will applied to all teams. So this will be a stored procedure.

6.2.2-) Generating Random Unique Transfer IDs

Transfer_id will be generated by random selection to distinguish the transfers easily. For example:

```
select transID(
        round (rand() * 1000000000),
        round (rand() * 1000000000),
        round (rand() * 1000000000),
        round (rand() * 1000000000),
        round (rand() * 1000000000),
        round (rand() * 1000000000)
) as Trasnsfer_id;
```

6.3-) Reports

6.3.1-) Total Number of Goals, Assists and Minutes Played by Player

goals_per_game, assists_per_game, avg_minutes_played attributes will be calculated by first calculating the total amount of them first.

```
select *
from previous_games
order by time desc
```

```
select p.name, pm.goal_scored, pm.assist, pm.time_played, pm.red_cards

from player p, player_match pm

where p.player_id = pm.player_id
```

6.3.2-) Finding the Most Valuable Players

This report will show the most valuable 50 players in the transfer database by simply comparing the *value* attribute in the Player.

```
select player_id, name, value, position, age

from Player

order by value

desc limit 50
```

6.3.3-) Finding the Total Values of Leagues

```
select l.name, l.tier, sum(lp.value) as league_value

from league l, (        select value

                        from player p, club_league cl, league l

                        where p.club_team_id = cl.club_id

                        and cl.league_name = l.name) as lp

group by l.name
```

6.4-) Triggers

6.4.1-) Conflict Check

- Each time when a coach assigned to a team a trigger will check if there is already a coach for that team.
- A player will only have one player agent so a trigger will control that when an agent assigned to a player
- A team coach will suggest a player to Club Manager only once. A trigger will check that and will drop the request from the manager's suggestion list.

6.4.2-) Information Update

- After each match relevant attributes of a Player will be updated. Such as goal_per_game, avg_time_played etc.
- League standings will be updated according to results after each match.

6.4.3-) Contract Cancellation

- A trigger will be constructed for players so that it will check whether contract is expired or not by comparing the *current_date* and *Expiration_date*

6.5-) Constraints

- The system can only be used(Users who are capable of making changes in the database) with a valid username and password. To only browse through the database there is no need to sign in.
- None of the integer values can be less than zero in the database.
- For each match there should be two teams.
- In order to complete the a transfer a Player agent needs to accept the offer proposed by Club Managers .
- A player can only play in one team with exception of national teams.
- Every player in the database only have one player agent. But player agents can have more than one players.
- A submitted transfer request cannot be withdrawn nor modified.
- Negotiations are done through renewals of declined offers. There is no special negotiation system.
- National Teams do not participate in transfer relations.
- Superior and inferior league relations does not exists.

7-) Implementation Plan

For the database implementation SQLite database management system decided to be used. For the Web development part Django framework will be used. This framework allows us to write python code in order to handle both frontend and backend web development paradigms. The written code will be deployed for use using pythonanywhere.