CS353 - Database Systems

Project Design Report

Group 17

BetterBettor

Social Betting Platform

Yasin Balcancı

Nurefşan Müsevitoğlu

Faruk Şimşekli

Burak Yaşar

25.11.2018

1.Revised E/R Model

1.1 What is changed in E/R Model?

According to feedback that we have received we develop our E/R model into more integrated one.

We delete who_liked and who_shared attributes from Post entity. Instead of that we add two more relation ‘like’ and ‘create’ to the E/R model and we changed Post from weak entity to strong entity. Similarly we changed Comment from weak entity to strong entity and remove who_liked attribute, instead we add ‘like’ relation between User and Comment as well. ‘write’ relation between User and Comment has been changed to ternary relation and Post is connected to this relation as well. Cardinalities of these relations are fixed.

We remove follower relation and simply changed the name of relation ‘following’ to follow. We provide follower and followed by adding inputs to the lines of this relation.
We did Coupon entity strong instead of weak and remove the unnecessary attributes total_odd, min_bet(FK to Match), system. Since we plan to provide recommended coupons in our system, not every coupon has to join do relation.

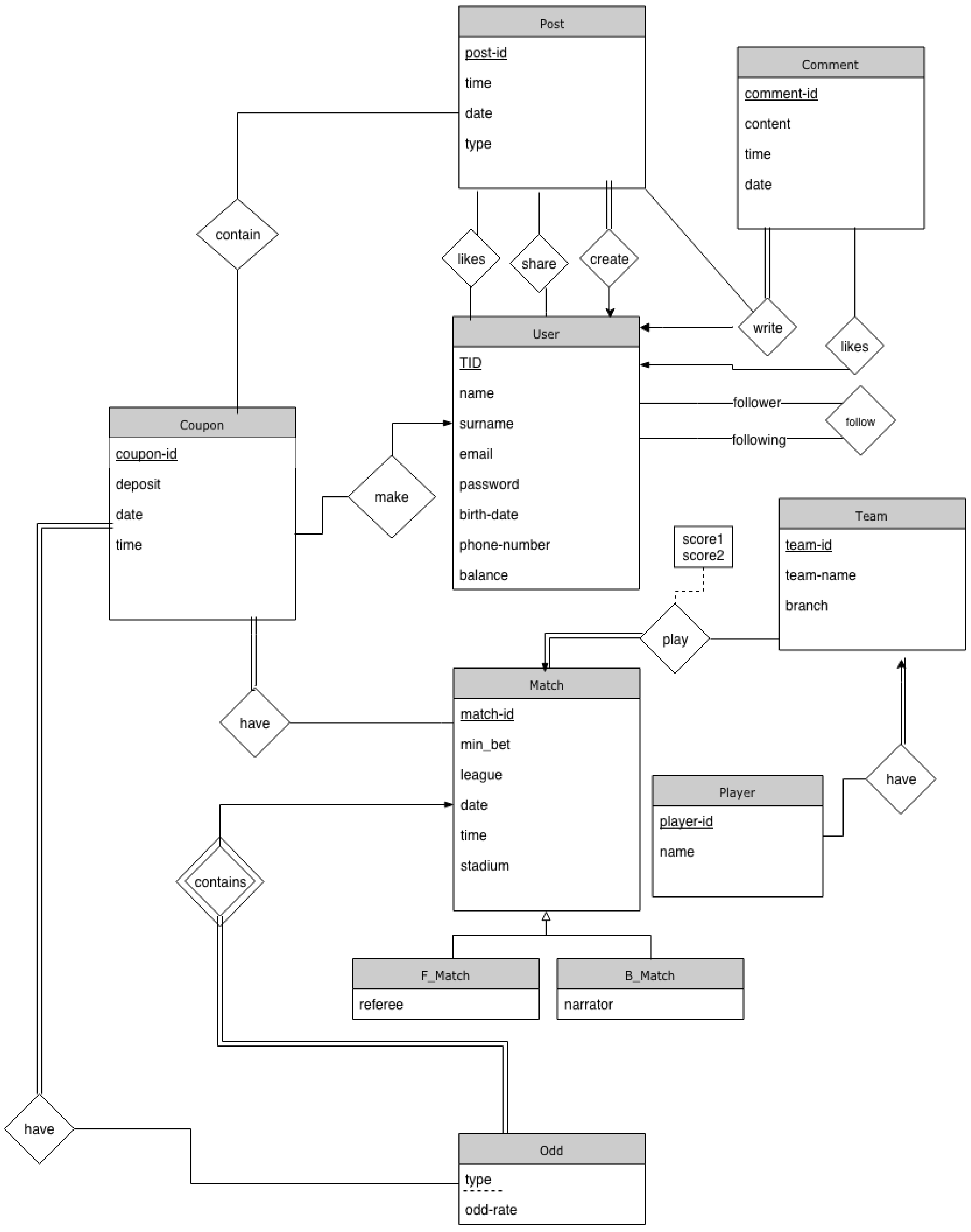
We remove F_Team and B_Team entities instead we add branch attribute to the Team, we change the name last_matches to statistics for clarification. Cardinalities of play relation (former name ‘do’) which is between Match and Team have been changed as every match should has Teams but not every Team should play the match. Also we added Player entity and have relation under the Team entity. We think that every Player should have at most one team and every Team should have at least one Player. We removed last_matches since we can derive this information and we added team-name attribute tı the Team entity.

We removed f_match from F_match entity and b_match from B_match those are unnecessary since we have match-id in Match entity. Therefore we add the attributes referee to F_match and narrator to B_match to distinguish themselves from each other. Also, entities F_Odds and B_Odds have been deleted and these odd types have been shown in the attribute ‘type’ in the Odd entity which is a weak entity. Odd and Match have weak relation ‘contains’ which is added to revised E/R diagram. In Odd entity we also have odd-rate attribute which specifies the rate of the each odd type for example ft1 is the type and its odd is 1.35.

Also, we added have relation between Coupon and Odd and finally we removed the Admin entity from the diagram.

1.2 E/R Model

Revised Entity Relation Model is given in the next page.


Post
post-id
time
date
type
Comment
comment-id
content
time
date
contain
likes
share
create
write
likes
User
TID
name
surname
email
password
birth-date
phone-number
balance
follower
following
follow
Coupon
coupon-id
deposit
date
time
make
score1
score2
play
Team
team-id
team-name
branch
have
Match
match-id
min_bet
league
date
time
stadium
Player
player-id
name
have
contains
F_Match
referee
B_Match
narrator
have
Odd
type
odd-rate

2. Relation, Schemas, FDs and Normalization

2.1 User

Relational Model:

User(<u>TID</u>, name, surname, email, password, birth-date, phone-number, balance)

Functional Dependencies:

TID -> name, surname, email, password, birth-date, phone-number, balance

Candidate Keys: {(TID), (email)}

Normal Form: BCNF

Table Definition:

```
create table User(
        TID                 char(11),
        name                varchar(255) not null,
        surname             varchar(255) not null,
        email               varchar(30) not null,
        password            varchar(30) not null,
        birth-date          varchar(10) not null,
        phone-number        varchar(14) not null,
        balance             int not null,
        primary key(TID)
);
```

2.2 Post_Create

Relational Model: Post_Create(<u>post-id</u>, date, time, description, TID)

Functional Dependencies: post-id -> date, time, description, TID

Candidate Keys: {(post-id)}

Normal Form: BCNF

Table Definition:

```
create table Post(
          post-id                  char(8),
          date                     varchar(10) not null,
          time                     varchar(5) not null,
          description              varchar(255) not null,
          TID                      char(11) not null,
          primary key(post-id),
          foreign key (TID) references User
);
```

2.3 Post_Like

Relational Model: Post_Like(<u>post-id, TID</u>)

Functional Dependencies: NONE

Candidate Keys: {(post-id, TID)}

Normal Form: BCNF

Table Definition:

```
create table Post_Like(
          post-id          char(8),
          TID               char(11) primary key,
          primary key(post-id, TID)
          foreign key (TID) references User,
          foreign key (post-id) references Post
);
```

2.4 Post_Share

Relational Model: Post_Share(<u>post-id, creator-TID, sharer-TID</u>)

Functional Dependencies: NONE

Candidate Keys: {(post-id, creator-TID, sharer-TID)}

Normal Form: BCNF

Table Definition:

```
create table Post_Share(
        post-id         char(8),
        creator-TID     char(11),
        sharer-TID      char(11),
        primary key(post-id, creator-TID, sharer-TID),
        foreign key (creator-TID) references User (TID),
        foreign key (sharer-TID) references User (TID),
        foreign key (post-id) references Post
);
```

2.5 Coupon_User

Relational Model: Coupon_User(<u>coupon-id</u>, date, deposit, time, TID)

Functional Dependencies: coupon-id -> date, time, deposit

Candidate Keys: {(coupon-id)}

Normal Form: BCNF

Table Definition:

```
create table coupon(
        coupon-id           char(8),
        deposit             int not null,
        time                varchar(5) not null,
        primary key(coupon-id),
        foreign key (TID) references User
);
```

2.6 Post_Coupon

Relational Model: Post_Coupon(post-id, coupon-id)

Functional Dependencies: NONE

Candidate Keys: {(post-id, coupon-id)}

Normal Form: BCNF

Table Definition:

```
create table post-coupon(
        post-id         char(8),
        coupon-id       char(8),
        primary key(post-id, coupon-id),
        foreign key (post-id) references Post,
        foreign key (coupon-id) references Coupon_User
);
```

2.7 Match

Relational Model: Match(match-id, min-bet, league, date, time, stadium)

Functional Dependencies: match-id -> min-bet, league, date, time, stadium

Candidate Keys: {(match-id)}

Normal Form: BCNF

Table Definition:

```
create table Match (
        match-id        char(8),
        min-bet         char(8) not null,
        league          varchar(30) not null,
        date            varchar(10) not null,
        time            varchar(5) not null,
        stadium         varchar(30) not null,
        primary key(match-id)
);
```

2.8 Coupon_Match

Relational Model: Coupon_Match(match-id, coupon-id)

Functional Dependencies: NONE

Candidate Keys: {(match-id, coupon-id)}

Normal Form: BCNF

Table Definition:

```
create table Coupon_Match (
        match-id        char(8),
        coupon-id       char(8),
        primary key(match-id, coupon-id),
        foreign key (match-id) references Match,
        foreign key (coupon-id) references Coupon_User
);
```

2.9 Team

Relational Model: Team(team-id, team-name, branch)

Functional Dependencies: team-id -> team-name, branch

Candidate Keys: {(team-id)}

Normal Form: BCNF

Table Definition:

```
create table Team (
        team-id         char(8),
        team-name       varchar(50) not null,
        branch          varchar(10) not null,
        primary key(team-id)
);
```

2.10 Player

Relational Model: Player_Team(player-id, team-id, name)

Functional Dependencies: player-id, team-id -> name

Candidate Keys: {(player-id, team-id)}

Normal Form: BCNF

Table Definition:

```
create table Player (
        player-id       char(8),
        team-id         varchar(10),
        name            varchar(10) not null,
        primary key(player-id, team-id),
        foreign key (team-id) references Team
);
```

2.11 Team_Match

Relational Model: Team_Match(match-id, team-id1, team-id2, score1, score2)

Functional Dependencies: match-id, team-id1, team-id2 -> score1, score2

Candidate Keys: {(match-id, team-id1, team-id2)}

Normal Form: BCNF

Table Definition:

```
create table Team (
        team-id1        char(8),
        team-id2        char(8),
        match-id        char(8),
        branch          varchar(10) not null,
        score1          varchar(3),
        score2          varchar(3),
        primary key(match-id,team-id1,team-id2),
        foreign key (match-id) references Match,
        foreign key (team-id1) references Team (team-id),
        foreign key (team-id2) references Team (team-id)
);
```

2.12 Football Match

Relational Model: F_Match(match-id, referee)

Functional Dependencies: match-id -> referee

Candidate Keys: {(match-id)}

Normal Form: BCNF

Table Definition:

```
create table F_Match (
        match-id        char(8),
        referee         varchar(10) not null,
        primary key(match-id),
        foreign key (match-id) references Match
);
```

2.13 Basketball Match

Relational Model: B_Match(match-id, narrator)

Functional Dependencies: match-id -> narrator

Candidate Keys: {(match-id)}

Normal Form: BCNF

Table Definition:

```
create table B_Match (
        match-id        char(8),
        narrator        varchar(30) not null,
        primary key(match-id),
        foreign key (match-id) references Match
);
```

2.14 Odd

Relational Model: Odd(match-id, type, odd-rate)

Functional Dependencies: match-id, type -> odd-rate

Candidate Keys: {(match-id, type)}

Normal Form: BCNF

Table Definition:

```
create table Odd (
        match-id        char(8),
        type            varchar(8),
        odd-rate        varchar(7),
        primary key(match-id,type),
        foreign key (match-id) references Match
);
```

2.15 Coupon_Odd

Relational Model: Coupon_Odd(coupon-id, match-id, type)

Functional Dependencies: NONE

Candidate Keys: {(coupon-id, match-id)}

Normal Form: BCNF

Table Definition:

```
create table B_Match (
        coupon-id       char(8),
        match-id        char(8),
        type            varchar(10) not null,
        primary key(coupon-id,match-id),
        foreign key (match-id) references Match,
        foreign key (coupon-id) references Coupon_User
);
```

2.16 Comment_Like

Relational Model: Comment_Like(TID, comment-id, content, date, time)

Functional Dependencies: TID, comment-id -> content, date, time

Candidate Keys: {(TID, comment-id)}

Normal Form: BCNF

Table Definition:

```
create table B_Match (
        TID             char(8),
        comment-id      char(8),
        content         varchar(255) not null,
        date            varchar(10) not null,
        time            varchar(5) not null,
        primary key(TID, comment-id),
        foreign key (TID) references User
);
```

2.17 Comment_Write

Relational Model: Comment_Write(TID, comment-id, post-id)

Functional Dependencies: NONE

Candidate Keys: {(TID, comment-id, post-id)}

Normal Form: BCNF

Table Definition:

```
create table B_Match (
        TID             char(8),
        comment-id      char(8),
        post-id         char(8),
        primary key(TID,comment-id,post-id),
        foreign key (TID) references User ,
        foreign key (comment-id) references Comment_Like,
        foreign key (post-id) references Post,
);
```

2.18 Follow

Relational Model: Follow(<u>follower-TID, followed-TID</u>)

Functional Dependencies: NONE

Candidate Keys: {(follower-TID, followed-TID)}

Normal Form: BCNF

Table Definition:

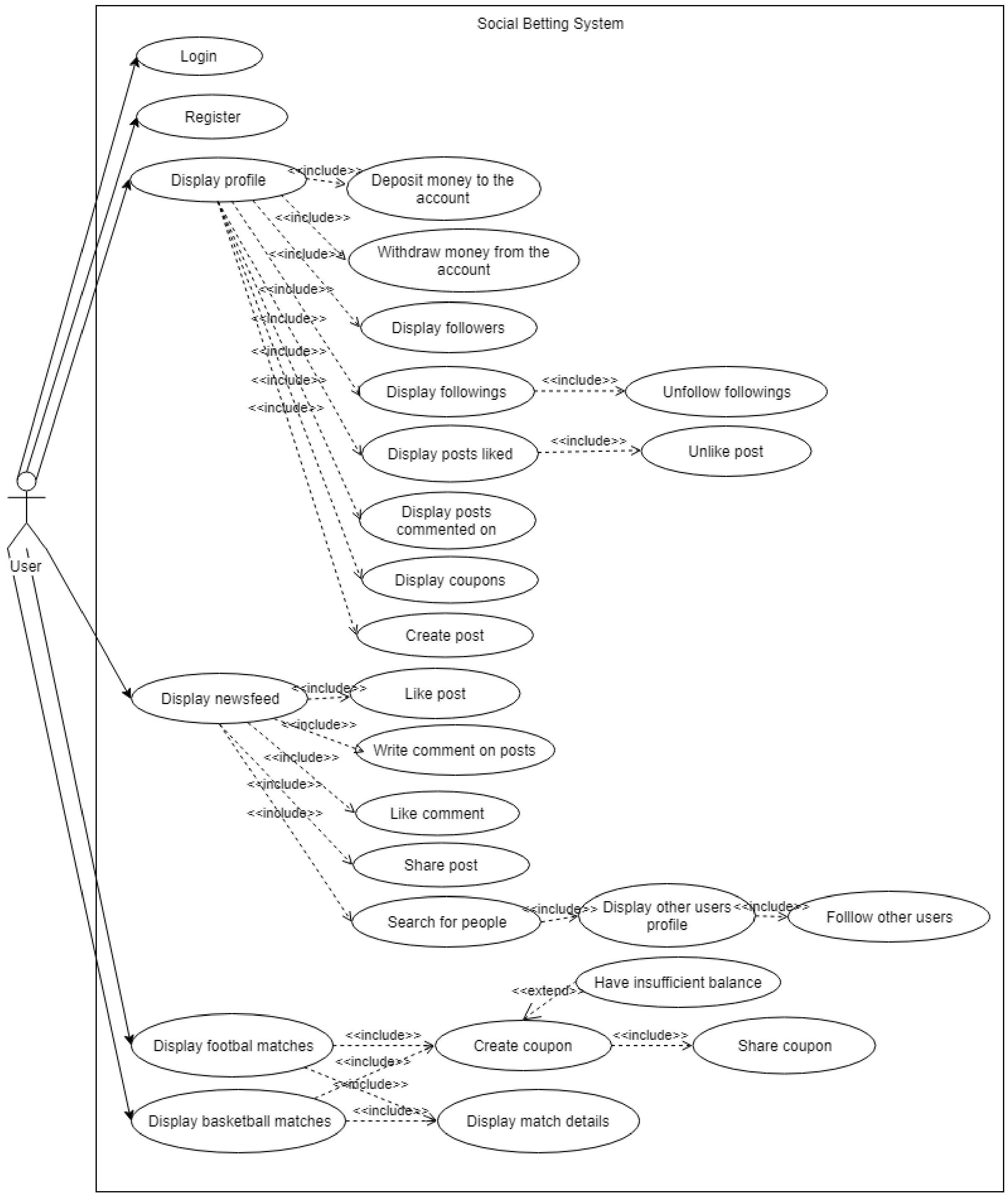
```
create table B_Match (
        follower-TID    char(8),
        followed-TID    char(8),
        primary key(follower-TID,followed-TID),
        foreign key (follower-TID) references User (TID),
        foreign key (followed-TID) references User (TID)
);
```

3. Functional Dependencies and Normalization of Tables

The Relation Schemas part of our design report contains all the functional dependencies and normal forms. Since the relations are all in Boyce-Codd Normal Form (BCNF), there is no need for any decomposition nor normalization.

4. Functional Components

4.1 Use Case and Scenarios



4.1.1 Register

Users can register the system(BetterBettor) with the Turkish identification number, name, surname, e-mail, password, birth-date and phone-number.

4.1.2 Login

Users can login to the system with corresponding e-mail and password.

4.1.3 Display Football Matches

Users can display football matches.

4.1.4 Display Basketball Matches

Users can display basketball matches.

4.1.5 Display Profile

Users can display their profile.

4.1.6 Display Newsfeed

Users can display their newsfeed.

4.1.7 Search for People

Users are able to search other users.

4.1.8 Deposit Money to the Account

Users are able to deposit money to make bets to their accounts.

4.1.9 Withdraw Money from the Account

Users are able to withdraw money from their account.

4.1.10 Display Followers

Users can display their followers.

4.1.11 Display Followings

Users can display people who are followed by them.

4.1.12 Display Posts Commented On

Users are able to display posts that they commented on.

4.1.13 Display Posts Liked

Users are able to display posts that they liked.

4.1.14 Display Coupons

Users are able to display other their coupons.

4.1.15 Display Other Users' Profile

Users are able to display other users' profile.

4.1.16 Like Comment

Users are able to like comments on posts.

4.1.17 Follow Other Users

Users can follow other users and see their posts on their newsfeed.

4.1.18 Unfollow Followings

Users are able to unfollow the people who is followed before.

4.1.19 Share Post

Users are able to create posts both by providing an explanation or not providing one. In order to create a post, users must either share a coupon that they created or share another's post.

4.1.20 Write Comment on Post

Users are able to write comments on posts.

4.1.21 Like Post

Users are able to like posts if they already haven't.

4.1.22 Unlike Post

Users are able to unlike any post they liked before.

4.1.23 Have Insufficient Balance

When users tries to create a coupon but they do not have enough balance for the stake they provided, they will not be able to create that coupon.

4.2 Algorithms

4.2.1 Post-User Related Algorithms

Firstly, posts have to be created by users. Each post are created by one user, that's why we have. One user are able to create many posts. That's why we have 1-M cardinality. A post instance contain post-id, date, time, and type which is an explanation of the post. All posts do belong to users. Therefore, we have a total participation towards post entity.

Created posts can be liked by users. Some posts may have no likes. Therefore, we do not have a total participation towards post. Also, many user can like the same post, which is why we have M-M cardinality between post and user entities in Post_Like relation.

Created posts can be shared by users. Some posts may have not been shared at all. Therefore, we do not have a total participation towards post. Also, many user can share the same post, which is why we have M-M cardinality between post and user entities in Post_Like relation.

4.2.2 Comment-User Related Algorithms

All the comments have to be made by users. Therefore, we have a total participation towards comment entity. Each comment has one creator. Comments will be made on posts. Therefore, we have a ternary relationship.

Comments made can be liked by users. Some comments may have not been liked at all. Therefore, we do not have a total participation towards comment entity. Also, many user can like the same post, which is why we have M-M cardinality between user and comment entities in Comment_Like relation.

4.2.3 Coupon Related Algorithm

Posts may contain coupons which is made by users. Also, a post may not contain a coupon instead simply a text can be posted without a coupon. Each coupon is specified with its
User can make many coupons and each coupon can be related at most one user. Not every coupon does not have to belong any user since we plan to make recommended coupons which are not made by users they are made by admins.

Each coupon has to have an odd type to be played however not every odd should participate a coupon. For instance coupons may have ft1(full-time1) but may not have 1_1 (half time 1, full time 1).

4.2.4 Match Related Algorithms

Users can add matches with the odds of these matches to their coupons. Each coupon should have matches however not every match should be included to the coupons. Matches are identified uniquely with match-id.

Without a match an odd cannot be exist. Each match has different types of odds. When the user selects a match to add to his/her coupon, all types of odds are available but not every odd of the match should be selected.

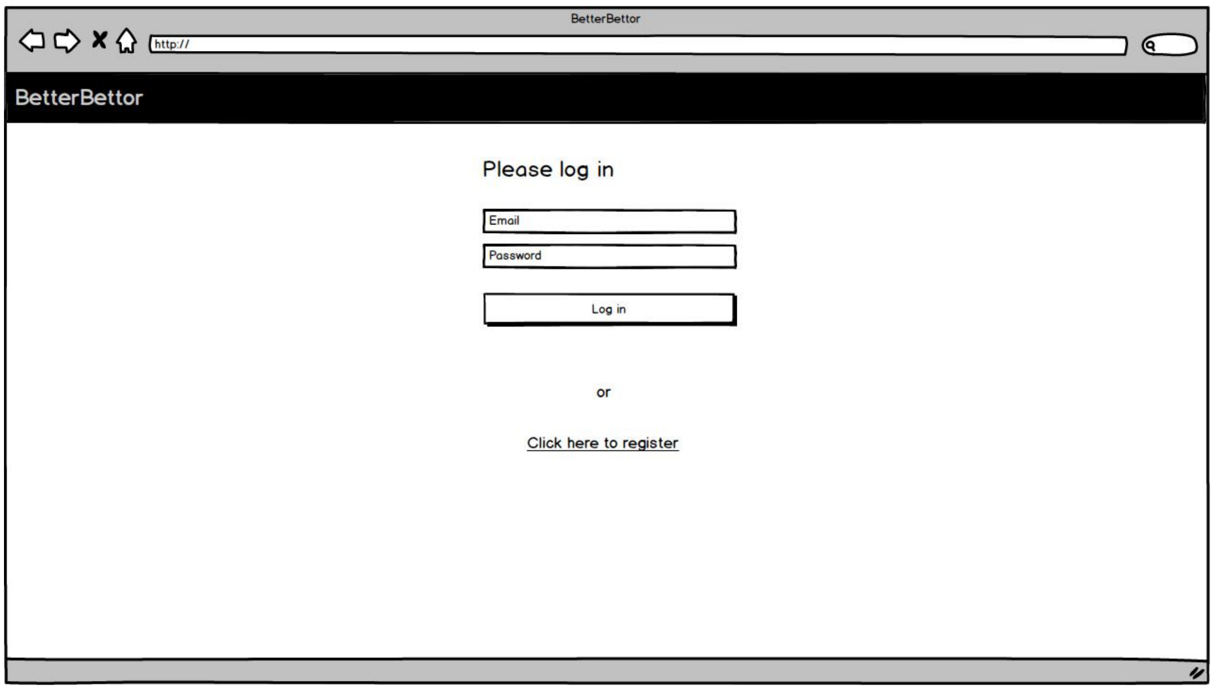
When the match is on two teams are required but some teams may not have a match.
Teams are specified with their unique id.

4.2.5 Team-Player Related Algorithms

Each Team have at least one player nonetheless not every player has to be in a team.
Players are identified with their unique id.

5. User Interface Design and Corresponding SQL Statements

5.1 Log in



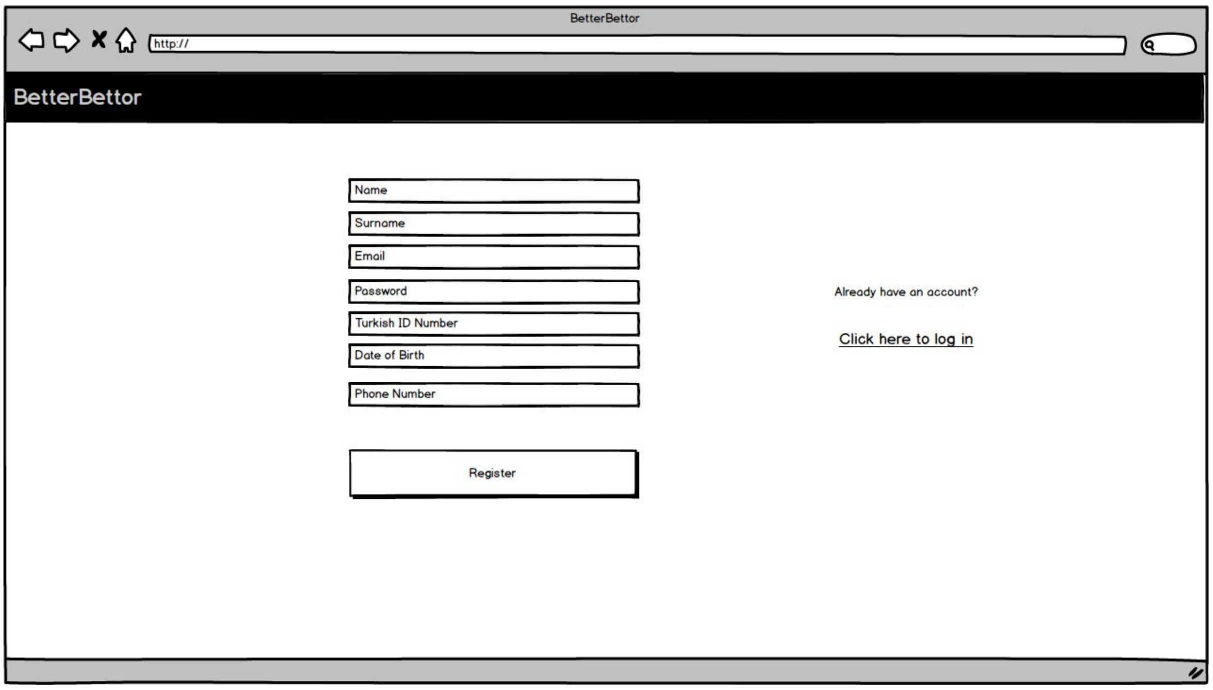
Input: @email, @password

Process: When the user enters the email and password, system searches to find a tuple from User table with given email and password. If such an user exists, the system let the user to log in to the system. If email and password do not match with any of user, system warns and wants the user to re-enter the information.

SQL Statement:

```
select case when exists(
        select *
        from User U
        where U.email = @email and U.password = password
) then cast (1 as bit)
else cast(0 as bit)
```

5.2 User Registration



Input: @name @surname @email @password @TID @birth-date @phone-number

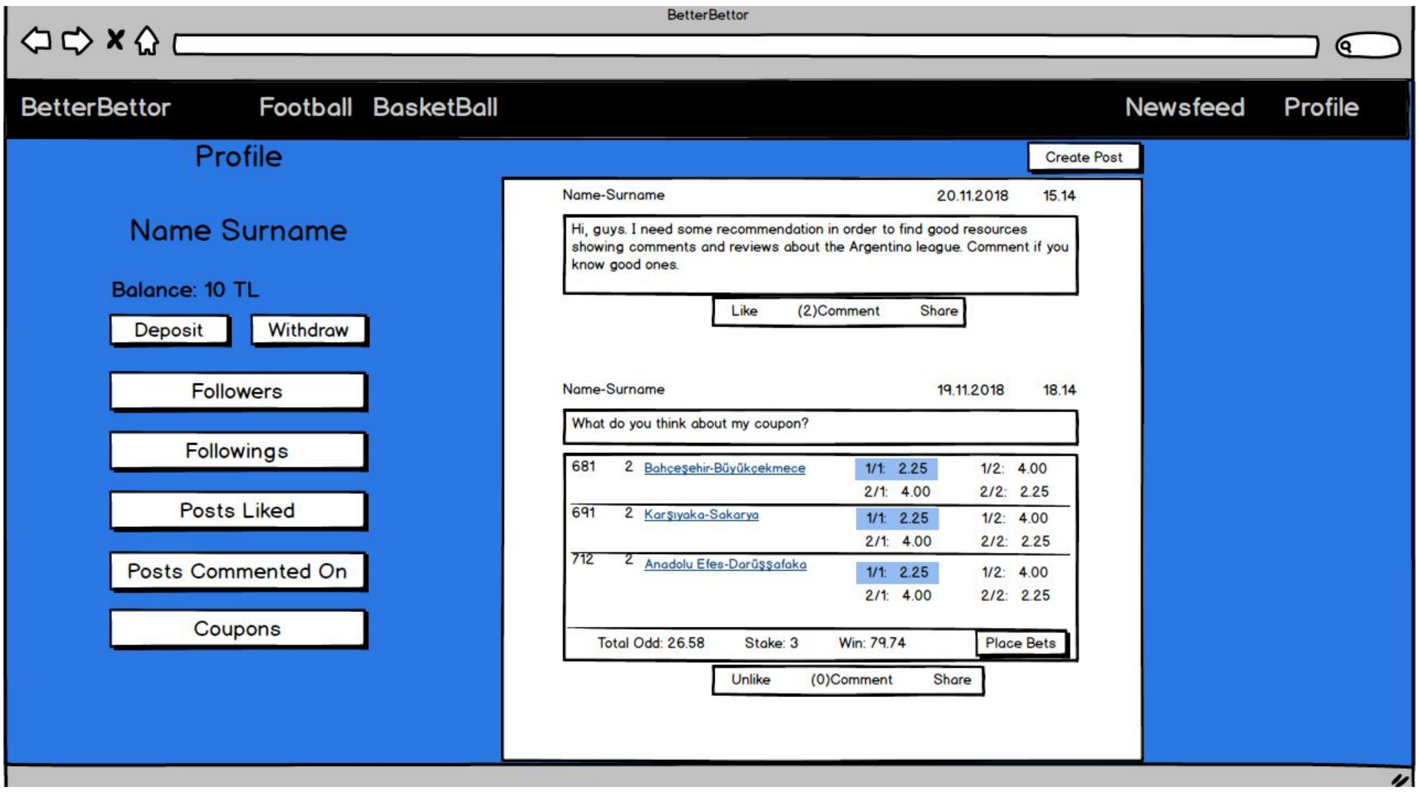
Process: Customer can register the system by entering the his/her name, surname, email, password Turkish identification number, date of birth, phone number.

SQL Statement:

Check if user exists
select * from User where TID = @TID or email = @email

Create new User
insert into User(email, password, name, surname, phone-number, birth-date, TID)
values (@email, @password, @name, @surname, @phone-number, @birth-date, @TID);

5.3 Profile



Name-Surname 20.11.2018 15.14

Hi, guys. I need some recommendation in order to find good resources showing comments and reviews about the Argentina league. Comment if you know good ones.

Name-Surname 19.11.2018 18.14

What do you think about my coupon?

681	2	Bahçeşehir-Büyükçekmece	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
691	2	Karşıyaka-Sakarya	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
712	2	Anadolu Efes-Darüşşafaka	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25

Total Odd: 26.58 Stake: 3 Win: 79.74

Input: @currentTID

Process: A user with a unique TID has a profile page. The name, surname, balance, posts that the user shared in the past will be available in the page. We will use a couple tables to retrieve these information.

SQL Statements:

1. Name, surname and the balance of the user

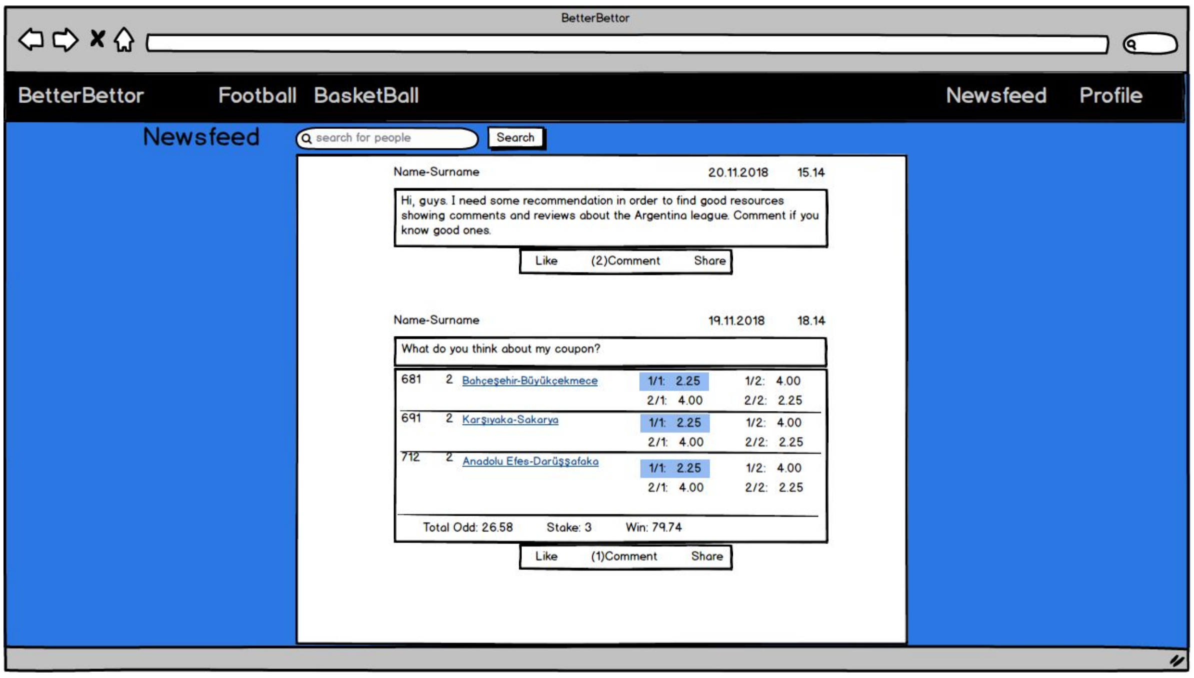
```
select name, surname, balance
from User U
where U.TID = @currentTID
```

2.

```
with postOfUser(post-id, date, timePost, description) as (
        select date, time as timePost, description
        from Post_Create natural joins User U
        where U.TID = @currentTID
)
```

```
select post-id, date, timePost, description, coupon-id, deposit, time, match-id, type,
odd-rate, teamid-1, team-id2
from (select coupon-id, deposit, time, match-id, type, odd-rate, teamid-1, team-id2
          from (select match-id, type, odd-rate, teamid-1, teamid-2
                    from (select match-id, min-bet, teamid-1, teamid-2
                              from (select match-id, teamid-1, teamid-2
                                        from Team_Match ) natural joins Match)
                              natural joins Odd)
          natural join Coupon join Coupon using (coupon-id) )
) natural join postOfUser
```

5.4 Newsfeed



Name-Surname 20.11.2018 15.14

Hi, guys. I need some recommendation in order to find good resources showing comments and reviews about the Argentina league. Comment if you know good ones.

Like (2)Comment Share

Name-Surname 19.11.2018 18.14

What do you think about my coupon?

681	2	Bahçeşehir-Büyükçekmece	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
691	2	Karşıyaka-Sakarya	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
712	2	Anadolu Efes-Darüşşafaka	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25

Total Odd: 26.58 Stake: 3 Win: 79.74

Like (1)Comment Share

Input: @currentTID

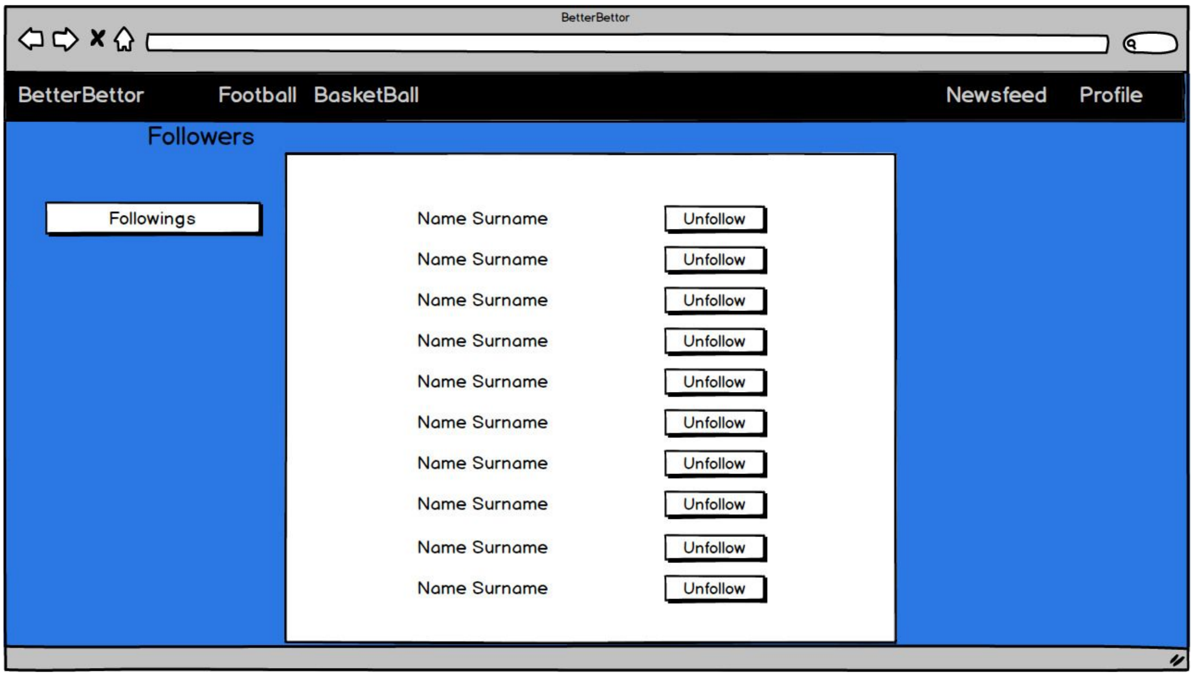
Process: The user will be able to see posts of people that user follows in the newsfeed. The user will be able to like, comment on and share any post he/she wants. The user will be able to search people in the search bar by writing the name of the user. Firstly, we need to find the people the user follows, then show the post of all of these people. We will use the SQL code in Following part.

SQL Statement:

```
with followings(TID) as (
        select followed-TID as TID
        from Follow F
        where F.follower-TID = @currentTID
)
with postOfUser(post-id, date, timePost, description) as (
        select date, time as timePost, description
        from Post_Create natural joins followings
)

select post-id, date, timePost, description, coupon-id, deposit, time, match-id, type,
odd-rate, teamid-1, team-id2
from (select coupon-id, deposit, time, match-id, type, odd-rate, teamid-1, team-id2
        from (select match-id, type, odd-rate, teamid-1, teamid-2
                    from (select match-id, min-bet, teamid-1, teamid-2
                              from (select match-id, teamid-1, teamid-2
                                        from Team_Match ) natural joins Match)
                              natural joins Odd)
          natural join Coupon join Coupon using (coupon-id) )
) natural join postOfUser
```

5.5 Followers



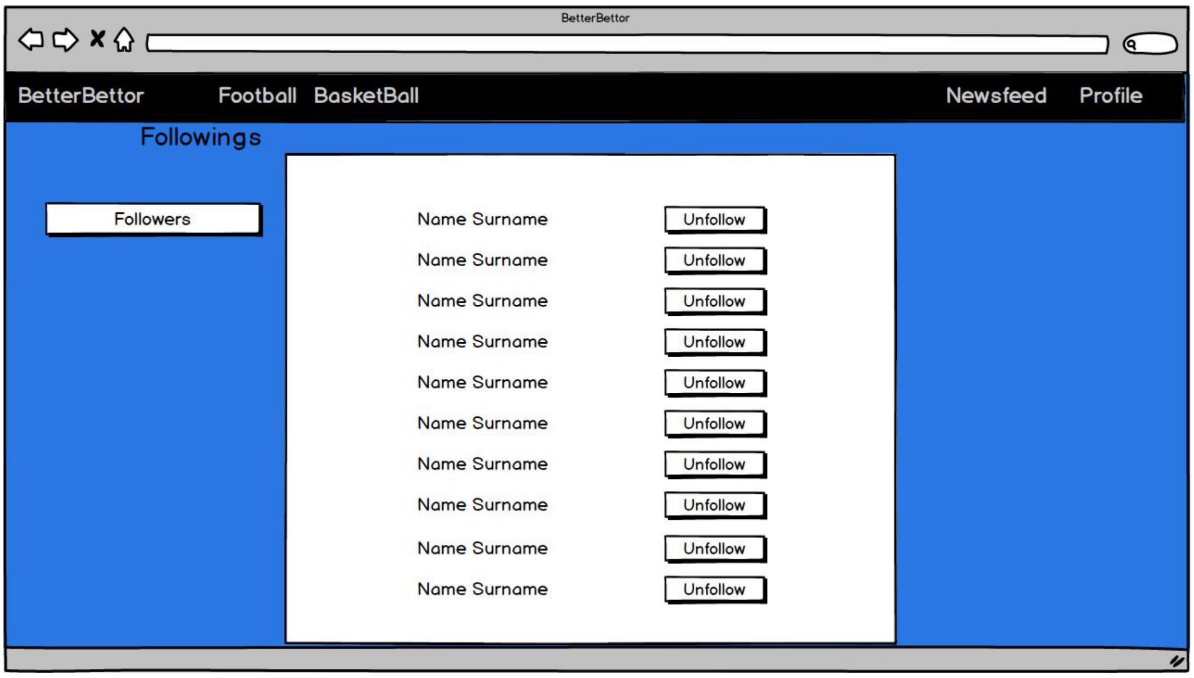
Input: @currentTID

Process: The list of followers of the user will be listed in this page.

SQL Statement:

```
select name, surname
from ( select follower-TID as TID
        from Follow F
        where F.followed-TID = @currentTID
        ) natural joins User
```

5.6 Followings



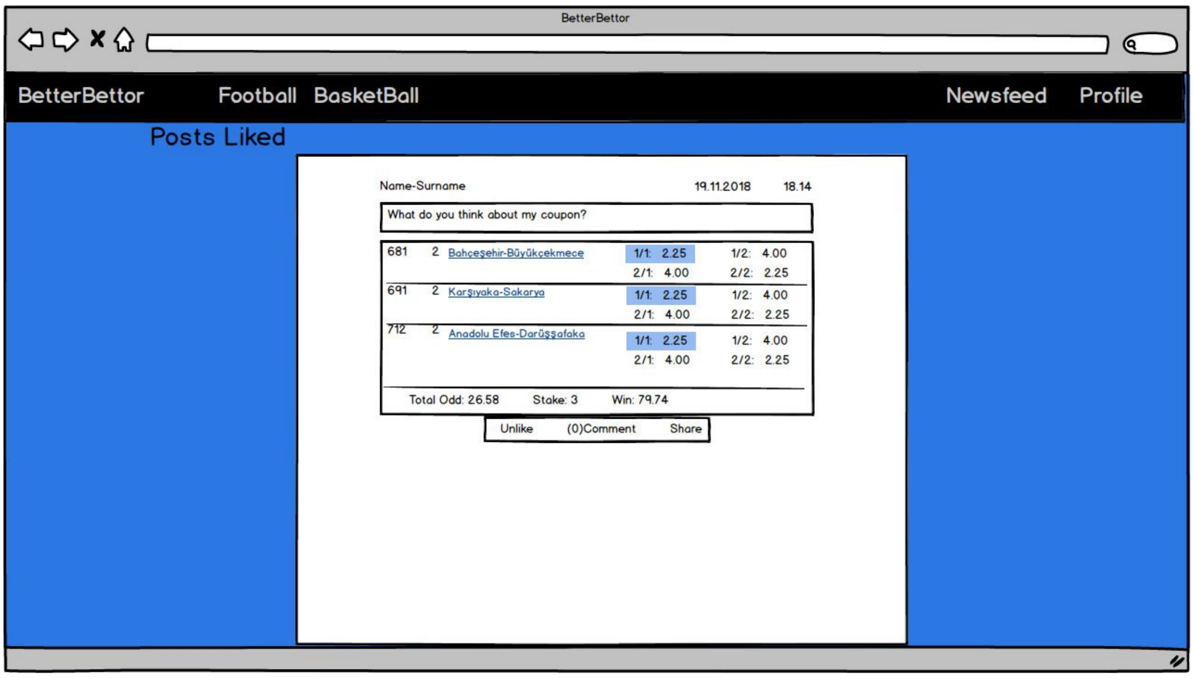
Input: @currentTID

Process: The list of followings of the user will be listed in this page.

SQL Statement:

```
select name, surname
from ( select followed-TID as TID
        from Follow F
        where F.follower-TID = @currentTID
        ) natural joins User
```

5.7 Post Liked



Name-Surname 19.11.2018 18.14

What do you think about my coupon?

681	2	Bahçeşehir-Büyükçekmece	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
691	2	Karşıyaka-Sakarya	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
712	2	Anadolu Efes-Darüşşafaka	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25

Total Odd: 26.58 Stake: 3 Win: 79.74

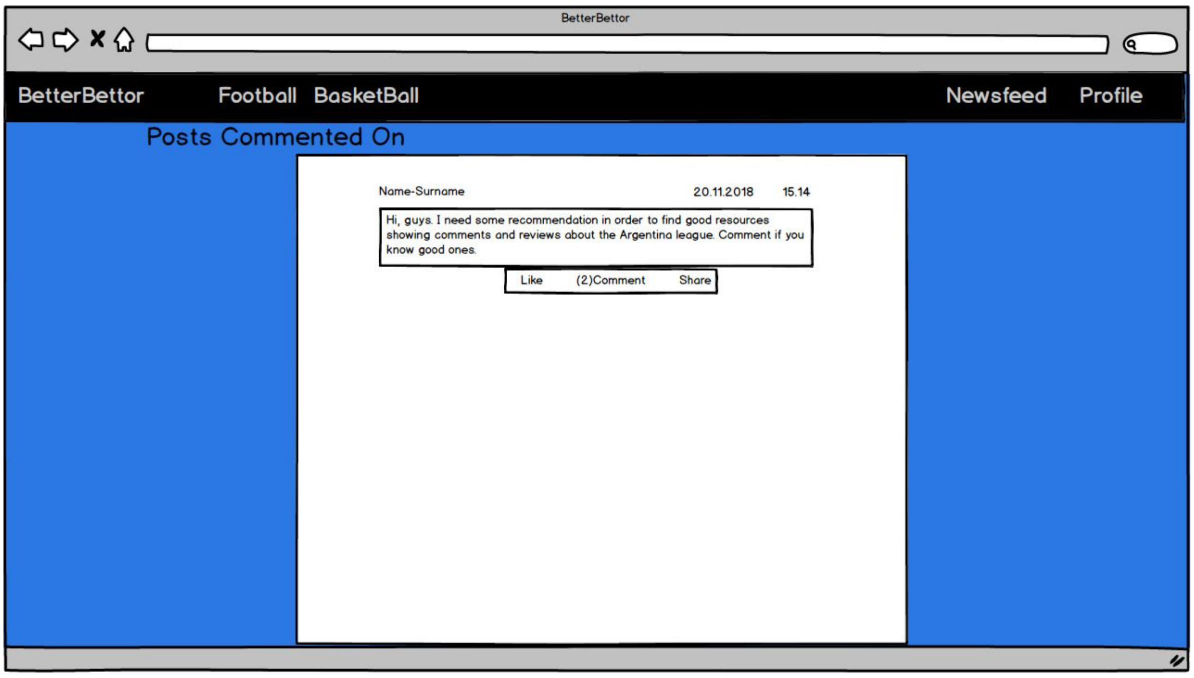
Input: @currentTID

Process: After login user may go through his/her profile. In his/her profile, user is able to see which posts he/she liked before.

SQL Statement:

```
select post-id
from Post_Like
where Post_Like.TID = @currentTID;
```

5.8 Post Commented On



Name-Surname 20.11.2018 15.14

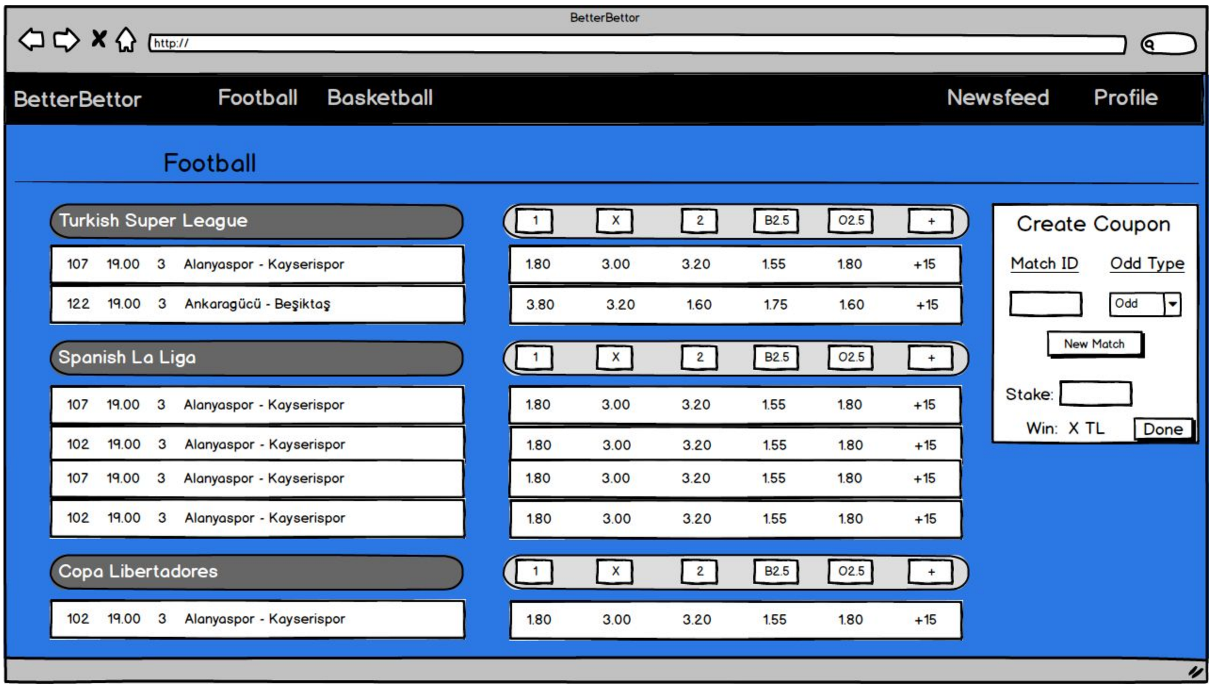
Hi, guys. I need some recommendation in order to find good resources showing comments and reviews about the Argentina league. Comment if you know good ones.

Input: @currentTID

Process: After login user may go through his/her profile. In his/her profile, user is able to see which posts he/she commented before.

SQL Statement:
select comment-id, post-id
from Comment_Write
where Comment_Write.TID = @currentTID;

5.9 Football



Turkish Super League				1	X	2	B2.5	O2.5	+
107	19.00	3	Alanyaspor - Kayserispor	1.80	3.00	3.20	1.55	1.80	+15
122	19.00	3	Ankaragücü - Beşiktaş	3.80	3.20	1.60	1.75	1.60	+15
Spanish La Liga				1	X	2	B2.5	O2.5	+
107	19.00	3	Alanyaspor - Kayserispor	1.80	3.00	3.20	1.55	1.80	+15
102	19.00	3	Alanyaspor - Kayserispor	1.80	3.00	3.20	1.55	1.80	+15
107	19.00	3	Alanyaspor - Kayserispor	1.80	3.00	3.20	1.55	1.80	+15
102	19.00	3	Alanyaspor - Kayserispor	1.80	3.00	3.20	1.55	1.80	+15
Copa Libertadores				1	X	2	B2.5	O2.5	+
102	19.00	3	Alanyaspor - Kayserispor	1.80	3.00	3.20	1.55	1.80	+15

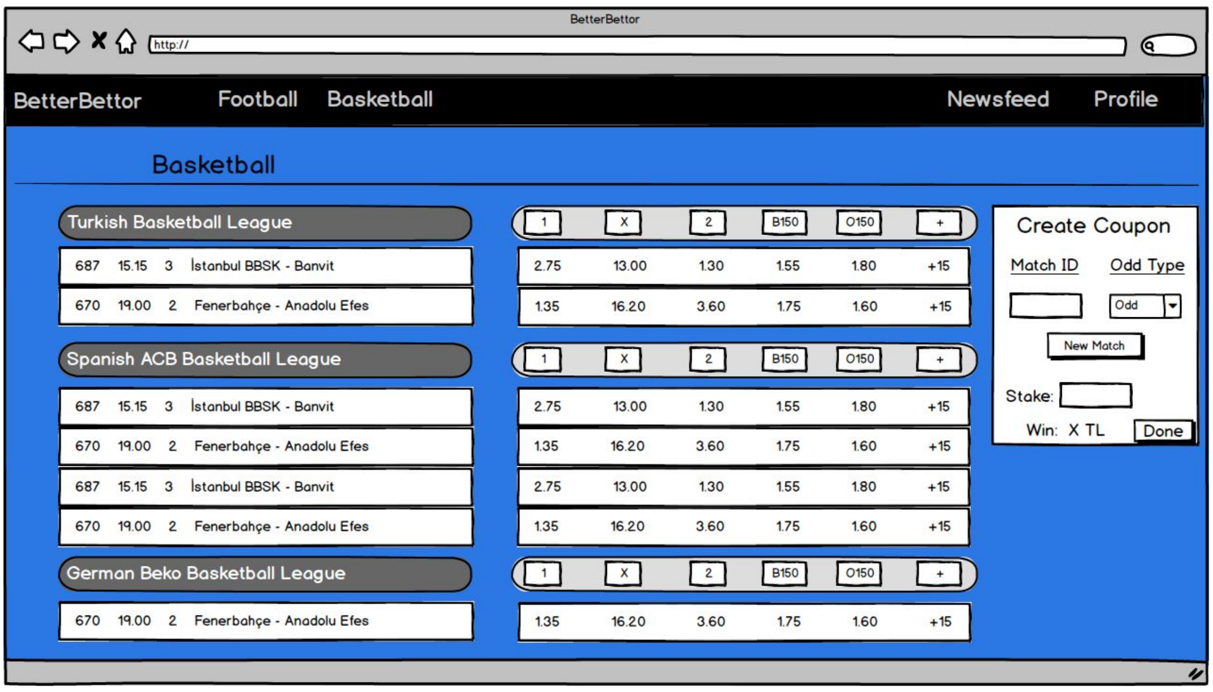
Input: @currentTID

Process: When the user click Football from the main bar of BetterBettor, he/she are able to see Football matches which can he/she bet on. Football matches are grouped by their league and are listed based on match-id, time, minimum bet number, corresponding teams and odds respectively. 'football matches' is a view stated in 6.1.4. User will be able to add matches to their incomplete coupon by match-ids and selected odds. The coupon will not be added to database until it is completed and placed.

SQL Statements:

```
select match-id, min-bet, league, time, team-name1, team-name2, type, odd-rate
from football matches  join Team T1, T2
where team-id1 = T1.team-id and team-id2 = T2.team-id
```

5.10 Basketball



Turkish Basketball League				1	X	2	B150	O150	+
687	15.15	3	İstanbul BBSK - Banvit	2.75	13.00	1.30	1.55	1.80	+15
670	19.00	2	Fenerbahçe - Anadolu Efes	1.35	16.20	3.60	1.75	1.60	+15

Spanish ACB Basketball League				1	X	2	B150	O150	+
687	15.15	3	İstanbul BBSK - Banvit	2.75	13.00	1.30	1.55	1.80	+15
670	19.00	2	Fenerbahçe - Anadolu Efes	1.35	16.20	3.60	1.75	1.60	+15
687	15.15	3	İstanbul BBSK - Banvit	2.75	13.00	1.30	1.55	1.80	+15
670	19.00	2	Fenerbahçe - Anadolu Efes	1.35	16.20	3.60	1.75	1.60	+15

German Beko Basketball League				1	X	2	B150	O150	+
670	19.00	2	Fenerbahçe - Anadolu Efes	1.35	16.20	3.60	1.75	1.60	+15

Create Coupon

Match ID: ____ Odd Type: Odd

New Match

Stake: ____

Win: X TL

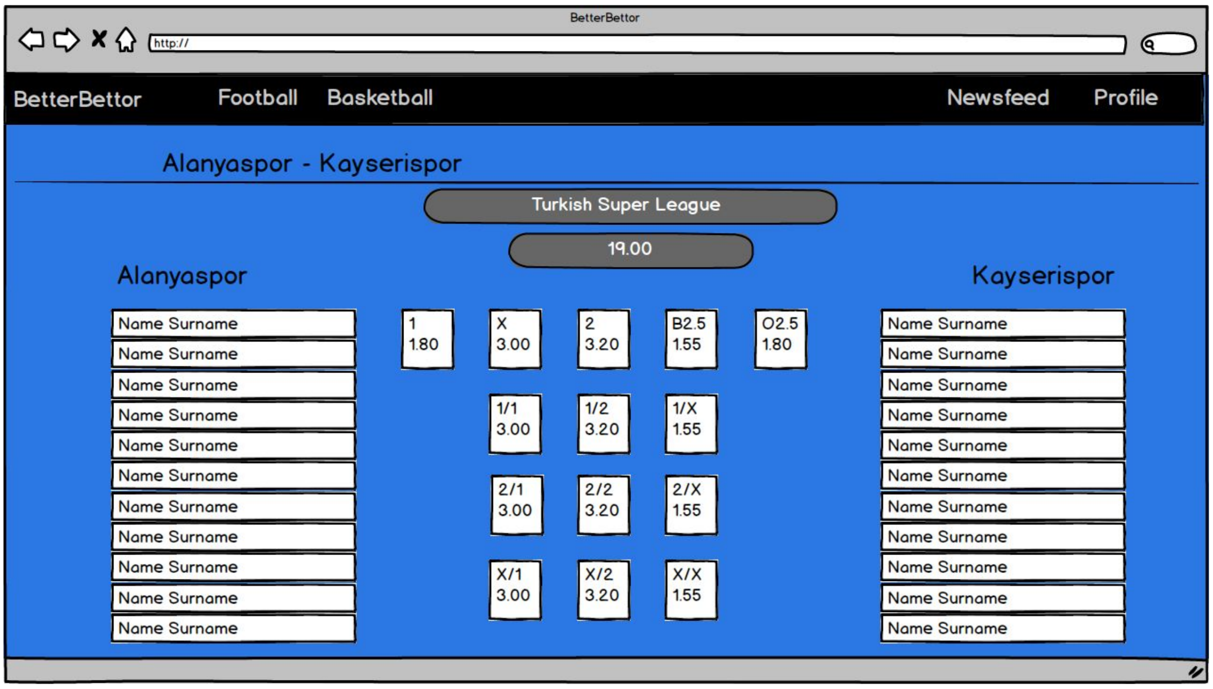
Done

Input:@currentTID

Process: When the user click Basketball from the main bar of BetterBettor, he/she are able to see Basketball matches which can he/she bet on. Basketball matches are grouped by their league and are listed based on match-id, time, minimum bet number, corresponding teams and odds respectively. ‘basketball matches’ is a view stated in 6.1.4.User will be able to add matches to their incomplete coupon by match-ids and selected odds. The coupon will not be added to database until it is completed and placed.

SQL Statement:
select match-id, min-bet, league, time, team-name1, team-name2, type, odd-rate
from basketball matches join Team T1, T2
where team-id1 = T1.team-id and team-id2 = T2.team-id

5.11 Match Details



Input: @currentTID, @match-id

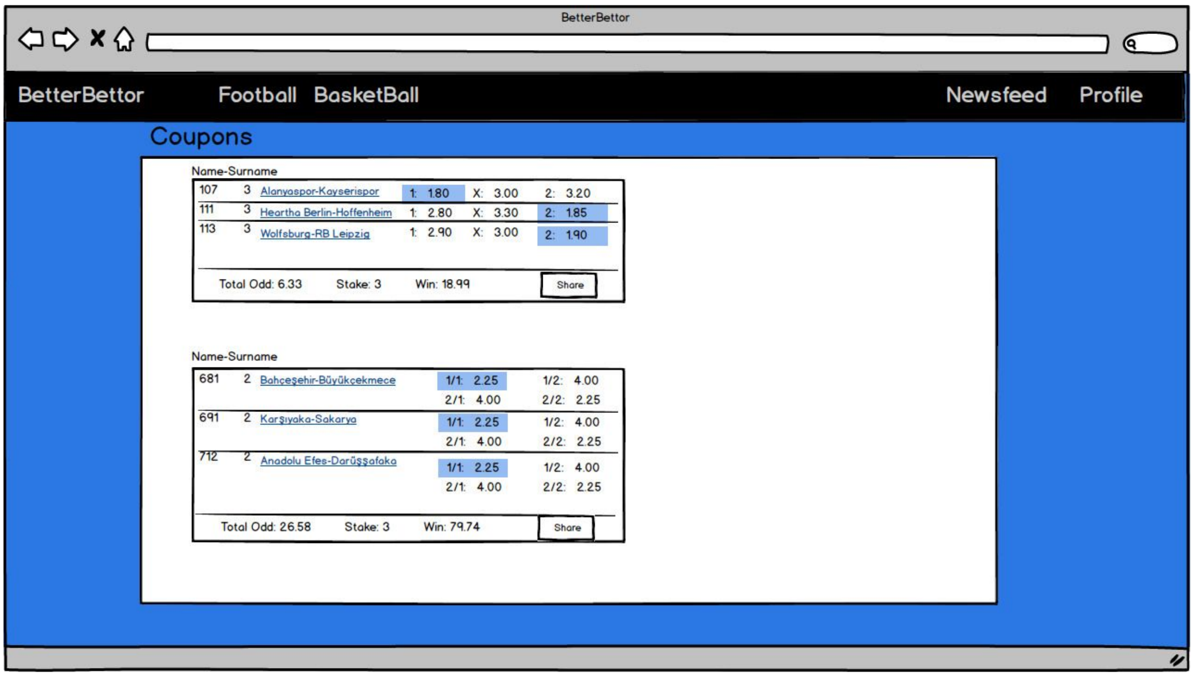
Process: When the user clicks on a match from Football or Basketball menus, they will see the details of the match containing players of the teams and all of the odds extra to the ones that were shown in previous menu.

SQL Statements:

select name
from Match natural join Team_Match natural join Team natural join Player_Team natural join player
where match-id = @match-id

select *
from Match natural join contains natural join odd
where match-id = @match-id

5.12 Coupons



Name-Surname

107	3	Alanyaspor-Kayserispor	1: 1.80	X: 3.00	2: 3.20
111	3	Heartha Berlin-Hoffenheim	1: 2.80	X: 3.30	2: 1.85
113	3	Wolfsburg-RB Leipzig	1: 2.90	X: 3.00	2: 1.90

Total Odd: 6.33 Stake: 3 Win: 18.99 Share

Name-Surname

681	2	Bahçeşehir-Büyükçekmece	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
691	2	Karşıyaka-Sakarya	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
712	2	Anadolu Efes-Darüşşafaka	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25

Total Odd: 26.58 Stake: 3 Win: 79.74 Share

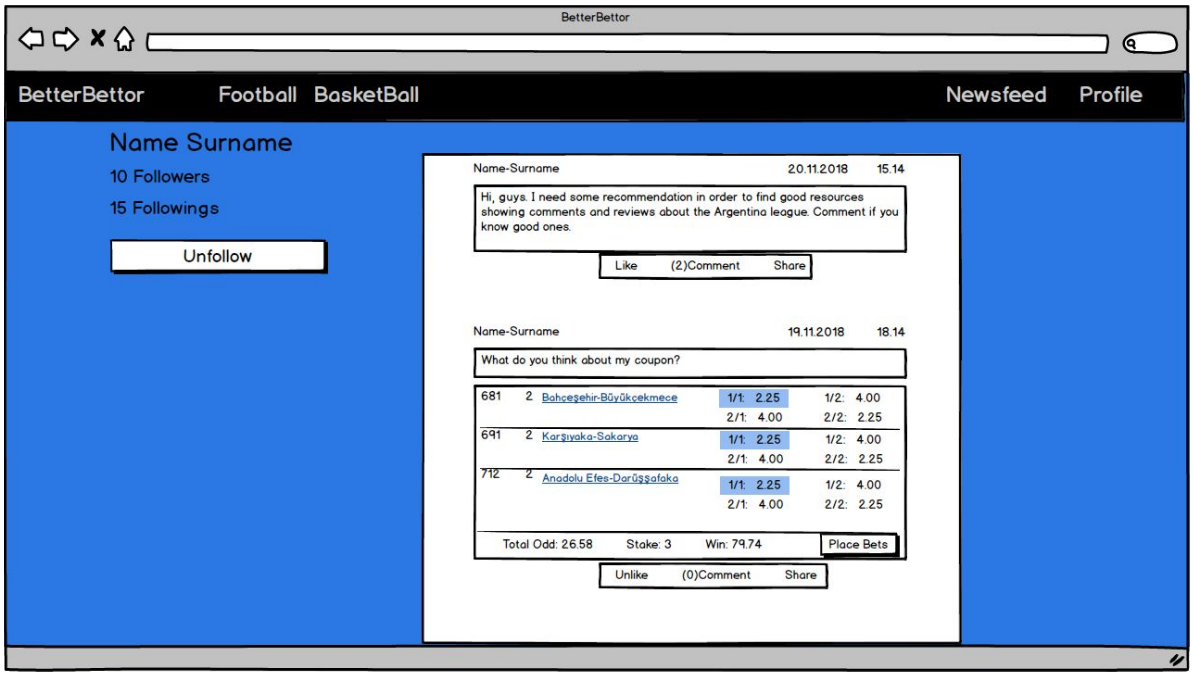
Input: @currentTID

Process: A user will be able to see their coupons by clicking "Coupons" button in their profiles. Coupons will be displayed according to the time they were created.

SQL Statement:

```
select *
from    (select coupon-id
        from Coupon_User) natural join Coupon_Match
group by coupon-id
having TID = @currentTID
```

5.13 Other User’s Profile



Name Surname
10 Followers
15 Followings

Name-Surname 20.11.2018 15.14

Hi, guys. I need some recommendation in order to find good resources showing comments and reviews about the Argentina league. Comment if you know good ones.

Name-Surname 19.11.2018 18.14

What do you think about my coupon?

681	2	Bahçeşehir-Büyükçekmece	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
691	2	Karşıyaka-Sakarya	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25
712	2	Anadolu Efes-Darüşşafaka	1/1: 2.25	1/2: 4.00
			2/1: 4.00	2/2: 2.25

Total Odd: 26.58 Stake: 3 Win: 79.74

Input: @currentTID @profile-owner-TID

Process: When a user clicks another user’s name, they will be directed to their profile which they will be able to see their follower and following user numbers and their posts..

SQL Statement:

Name Surname:

```
select name, surname
        from User U
        where U.TID = @profile-owner-TID
```

Number of Followings:

```
select count(TID)
        from ( select followed-TID as TID
                from Follow F
                where F.follower-TID = @profile-owner-TID
                ) natural joins User
```

Number of Followers:

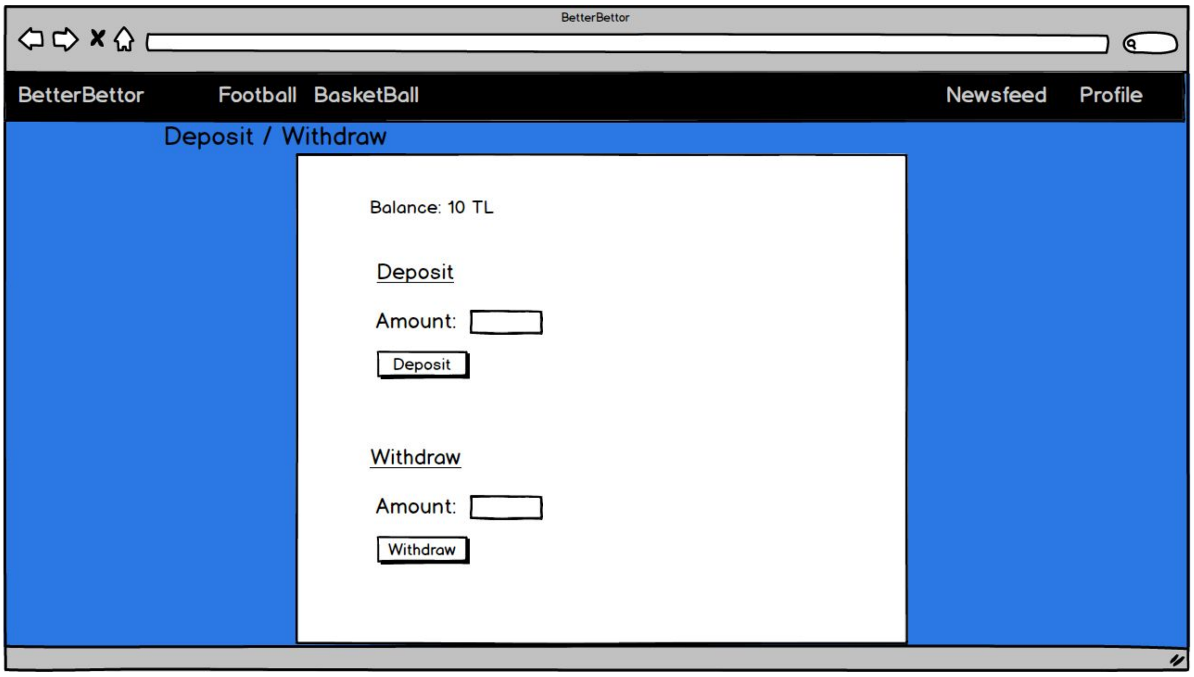
```
select count(TID)
        from ( select follower-TID as TID
                from Follow F
                where F.followed-TID = @profile-owner-TID
                ) natural joins User
```

Posts:

```
with postOfUser(post-id, date, timePost, description) as (
        select date, time as timePost, description
        from Post_Create natural joins User U
        where U.TID = @profile-owner-TID
)


select post-id, date, timePost, description, coupon-id, deposit, time, match-id, type, odd-rate,
teamid-1, team-id2
        from (select coupon-id, deposit, time, match-id, type, odd-rate, teamid-1, team-id2
                from (select match-id, type, odd-rate, teamid-1, teamid-2
                        from (select match-id, min-bet, teamid-1, teamid-2
                                from (select match-id, teamid-1, teamid-2
                                        from Team_Match ) natural joins Match)
                                natural joins Odd)
                natural join Coupon join Coupon using (coupon-id) )
        ) natural join postOfUser
```

5.14 Deposit / Withdraw



Input: @currentTID, @amount

Process: When the user clicked “Deposit” or “Withdraw” in their profile, they will be directed to this page. When they input amount and click Deposit or Withdraw buttons, their balance will chance accordingly.

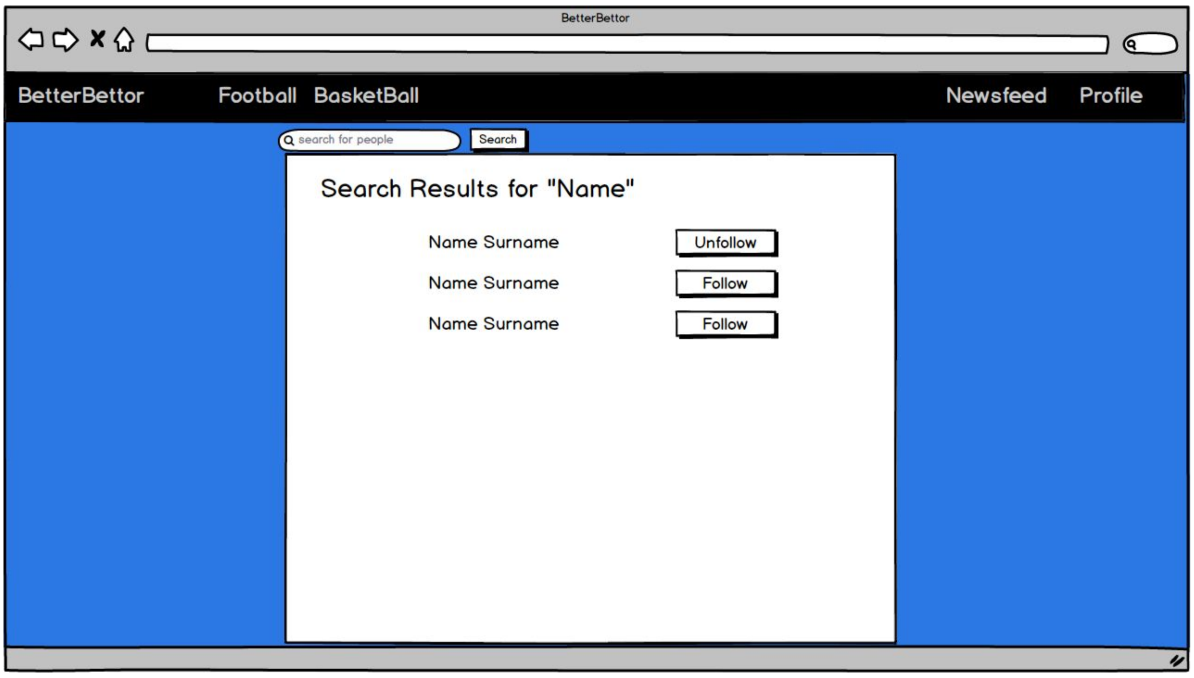
SQL Statement:

```
select balance from User where TID = @currentTID

update User
set balance = balance + @amount
where TID = @currentTID

update User
set balance = balance - @amount
where TID = @currentTID
```

5.15 Search Results



Search Results for "Name"

Name Surname
Name Surname
Name Surname

Input: @input @currentTID

Process: A user will be able to search for other users by typing their name or surname. Because input may contain several words representing full name, it will be parsed into separate words before SQL operation.

SQL Statement:

```
select name, surname
        from User U
        where U.name = @input or U.surname = @input
```

6. Advanced Database Components

6.1 Views

6.1.1 Newsfeed

```
create view [newsfeed] as

        with followings(TID) as (
                select followed-TID as TID
                from Follow F
                where F.follower-TID = @currentTID
        )
        with postOfUser(post-id, date, timePost, description) as (
                select date, time as timePost, description
                from Post_Create natural joins followings
        )

        select post-id, date, timePost, description, coupon-id, deposit, time, match-id, type,
        odd-rate, teamid-1, team-id2
        from (select coupon-id, deposit, time, match-id, type, odd-rate, teamid-1, team-id2
                from (select match-id, type, odd-rate, teamid-1, teamid-2
                        from (select match-id, min-bet, teamid-1, teamid-2
                                from (select match-id, teamid-1, teamid-2
                                        from Team_Match ) natural joins Match)
                                natural joins Odd)
                natural join Coupon join Coupon using (coupon-id) )
        ) natural join postOfUser
```

6.1.2 Follower of A User

```
create view [followerOfAUser] as
select name, surname
        from ( select follower-TID as TID
                from Follow F
                where F.followed-TID = @currentTID
                ) natural joins User
```

6.1.3 Match Tuples

This views are used in 5.9 and 5.10

```
create view [football matches] as
select match-id, min-bet, league, time, team-id1, team-id2, type, odd-rate
from    (select match-id, min-bet, league, time, team-id1, team-id2
        from    (select match-id, min-bet, league, time
                from    ((select match-id
                        from F_Match) natural join Match
                        ) natural join Team_Match
                group by league
                ) natural join Odd
        )
```

```
create view [basketball matches] as
select match-id, min-bet, league, time, team-id1, team-id2, type, odd-rate
from    (select match-id, min-bet, league, time, team-id1, team-id2
        from    (select match-id, min-bet, league, time
                from    ((select match-id
                        from B_Match) natural join Match
                        ) natural join Team_Match
                group by league
                ) natural join Odd
        )
```

6.2 Reports

6.2.1 Number of coupons of each user

Bet system is interested in which user is most dedicated for the system and they can try to surprise him/her with extra bonus etc. They will be able to see the number of coupons of each user.

<u>-SQL query of this report:</u>

```
select coupon-id, count(*)
from Coupon_User
group by TID
```

6.2.2 Deposit from each User

Since we know that the system is interested in which user makes more coupons, deposit from each user may also be beneficial. System can give more appealing odds for that specific users as they spent more money and the system encourage users to play much.

<u>-SQL query of this report:</u>

```
select deposit, count(*)
from Coupon_User
group by TID
```

6.2.3 Total Odd Rate for each Coupon

We need the multiplication of the odds of the matches in each coupon to compute the total-odd to specify the winning rate of the users.

<u>-SQL query of this report:</u>

```
select odd-rate, multiply(odd_rate) as (total-odd)
from (   select *
         from (   select coupon-id
                  from Coupon_User
                  where TID = @currentTID) natural join Coupon_Match) natural join Odd
where type = @selectedType
```

6.3 Triggers

- When the coupon is deleted by the user the post related to this coupon, if exist, it will be deleted automatically.
- When the post is deleted all the likes and comments of corresponding post are deleted.
- When the Coupon wins balance of the User is incremented.
- When the Coupon is created balance of the User is decremented.

6.4 Constraints

- Only certain parts of the service can be used (such as seeing matches with their odds but not be able to bet) if user is not logged in.
- Users cannot bet unless they are logged in.
- Users cannot write, like or comment any post if they are not logged in.
- Users cannot make coupon if they have not enough balance.
- Passwords cannot be shorter than 8 characters and cannot include special characters and space.
- A coupon must fulfill the minimum bet number constraints of the matches. A user cannot make a coupon with lower than
- Any online payment that isn’t completed (due to insufficient funds, internal problems…) will cancel the coupon made
- When the user unfollow another user corresponding likes and comments are deleted.

6.5 Stored Procedures

- A benefit of stored procedures is that you can centralize data access logic into a single place that is then easy for DBA's to optimize. And because they compiled just once at the beginning, it will make our database more rapid.

- Stored procedures also have a security benefit in that you can grant execute rights to a stored procedure but the user will not need to have read/write permissions on the underlying tables. Because we are going to assign on our model with stored procedure.

- Stored Procedure is a set pre-compiled SQL statement that used to perform a special task. So we are going to create a complicated stored procedure for a task and then use it for our database for several times without writing all those complicated queries over and over again. For example accessing all odd types for every different matches.

- Retrieving and sending information, in other words information transmission will be quite easy with stored procedure.

7. Implementation Plan

For our system functionalities and user interface in our hypertext dictionary system we will use HTML, CSS and JavaScript for front-end implementation. Back-end implementation of the project will be done via NodeJS. Database implementation will be done via MySQL.

8. Website

https://github.com/ybalcanci/BetterBettor/

3.1 User

Relational Model:

User(<u>TID</u>, name, surname, email, password, birth-date, phone-number, balance)

Functional Dependencies:

TID -> name, surname, email, password, birth-date, phone-number, balance

Candidate Keys: {(TID), (email)}

Normal Form: BCNF

Table Definition:

```
create table User(
        tid bigint primary key,
        name varchar(255) not null,
        surname varchar(255) not null,
        email varchar(30) not null,
        password varchar(30) not null,
        birth_date varchar(10) not null,
        phone_number varchar(14) not null,
        balance int not null
);
```

3.2 Post_Create

Relational Model: Post_Create(<u>post-id</u>, date, time, description, TID)

Functional Dependencies: post-id -> date, time, description, TID

Candidate Keys: {(post-id)}

Normal Form: BCNF

Table Definition:

```
create table Post(
        post_id int primary key,
        match_date varchar(10) not null,
		match_time varchar(5) not null,
		description varchar(255) not null,
		tid bigint not null,
		foreign key (tid) references User(tid)
	);
```

3.3 Post_Like

Relational Model: Post_Like(<u>post-id, TID</u>)

Functional Dependencies: NONE

Candidate Keys: {(post-id, TID)}

Normal Form: BCNF

Table Definition:

```
	create table Post_Like(
		post_id int primary key,
		tid bigint,
		foreign key (tid) references User(tid),
		foreign key (post_id) references Post(post_id)
	);
```

3.4 Post_Share

Relational Model: Post_Share(<u>post-id, sharer-TID, creator-TID</u>)

Functional Dependencies: NONE

Candidate Keys: {(post-id, creator-TID, sharer-TID)}

Normal Form: BCNF

Table Definition:

```
create table Post_Share(
        post_id int,
        creator_tid bigint,
        sharer_tid bigint,
        primary key(post_id, sharer_tid),
        foreign key (creator_tid) references User (tid),
        foreign key (sharer_tid) references User (tid),
        foreign key (post_id) references Post(post_id)
);
```

3.5 Coupon_User

Relational Model: Coupon_User(<u>coupon-id</u>, creator_tid, deposit, coupon_time, date)

Functional Dependencies: coupon-id -> date, time, deposit

Candidate Keys: {(coupon-id)}

Normal Form: BCNF

Table Definition:

```
create table Coupon(
        coupon_id int primary key,
        creator_tid bigint,
        deposit int not null,
        date varchar(10),
        coupon_time varchar(5) not null,
        foreign key (creator_tid) references User(tid)
)
```

3.6 Post_Coupon

Relational Model: Post_Coupon(post-id, coupon-id)

Functional Dependencies: NONE

Candidate Keys: {(post-id, coupon-id)}

Normal Form: BCNF

Table Definition:

```
create table Post_Coupon(
        post_id int,
        coupon_id int,
        primary key(post_id, coupon_id),
        foreign key (post_id) references Post(post_id),
        foreign key (coupon_id) references Coupon(coupon_id)
);
```

3.7 Match

Relational Model: gmatch(match-id, min-bet, league, date, time, stadium)

Functional Dependencies: match-id -> min-bet, league, date, time, stadium

Candidate Keys: {(match-id)}

Normal Form: BCNF

Table Definition:

```
create table gmatch(
        match_id int primary key,
        min_bet int not null,
        league varchar(30) not null,
        date varchar(10) not null,
        time varchar(5) not null,
        stadium varchar(30) not null
);
```

3.8 Coupon_Match

Relational Model: Coupon_Match(match-id, coupon-id)

Functional Dependencies: NONE

Candidate Keys: {(match-id, coupon-id)}

Normal Form: BCNF

Table Definition:

```
create table Coupon_Match (
        match_id int,
        coupon_id int,
        primary key(match_id, coupon_id),
        foreign key (match_id) references gmatch(match_id),
        foreign key (coupon_id) references Coupon(coupon_id)
);
```

3.9 Team

Relational Model: Team(team-id, team-name, branch)

Functional Dependencies: team-id -> team-name, branch

Candidate Keys: {(team-id)}

Normal Form: BCNF

Table Definition:

```
create table Team (
        team_id int,
        team_name varchar(50) not null,
        branch varchar(10) not null,
        primary key(team_id)
);
```

3.10 Player

Relational Model: Player_Team(player-id, team-id, name)

Functional Dependencies: player-id, team-id -> name

Candidate Keys: {(player-id, team-id)}

Normal Form: BCNF

Table Definition:

```
create table Player (
        kit_number int,
        team_id int,
        name varchar(10) not null,
        primary key(kit_number, team_id),
        foreign key (team_id) references Team(team_id)
);
```

3.11 Team_Match

Relational Model: Team_Match(match-id, team-id1, team-id2, score1, score2)

Functional Dependencies: match-id, team-id1, team-id2 -> score1, score2

Candidate Keys: {(match-id, team-id1, team-id2)}

Normal Form: BCNF

Table Definition:

```
create table Team_Match (
        team_id1 int,
        team_id2 int,
        match_id int primary key,
        branch varchar(10) not null,
        score1 int,
        score2 int,
        foreign key (match_id) references gmatch(match_id),
        foreign key (team_id1) references Team (team_id),
        foreign key (team_id2) references Team (team_id)
);
```

3.12 Football Match

Relational Model: F_Match(match-id, referee)

Functional Dependencies: match-id -> referee

Candidate Keys: {(match-id)}

Normal Form: BCNF

Table Definition:

```
create table F_Match (
        match_id int,
        referee varchar(10) not null,
        primary key(match_id),
        foreign key (match_id) references gmatch(match_id)
);
```

3.13 Basketball Match

Relational Model: B_Match(match-id, narrator)

Functional Dependencies: match-id -> narrator

Candidate Keys: {(match-id)}

Normal Form: BCNF

Table Definition:

```
create table B_Match (
        match_id int,
        narrator varchar(30) not null,
        primary key(match_id),
        foreign key (match_id) references gmatch(match_id)
);
```

3.14 Odd

Relational Model: Odd(match_id, type, odd_rate)

Functional Dependencies: match_id, type -> odd_rate

Candidate Keys: {(match-id, type)}

Normal Form: BCNF

Table Definition:

```
        create table Odd (
                match_id int,
                type varchar(8),
                odd_rate float,
                primary key(match_id,type),
                foreign key (match_id) references gmatch(match_id)
        );
```

3.15 Coupon_Odd

Relational Model: Coupon_Odd(coupon-id, match-id, type)

Functional Dependencies: NONE

Candidate Keys: {(coupon-id, match-id)}

Normal Form: BCNF

Table Definition:

```

                coupon_id int,
                match_id int,
                type varchar(10) not null,
                primary key (coupon_id,match_id),
                foreign key (match_id, type) references Odd(match_id, type),
                foreign key (coupon_id) references Coupon(coupon_id)
        );
```

3.16 Comment_Like

Relational Model: Comment_Like(comment-id, TID, date, time)

Functional Dependencies: TID, comment-id -> content, date, time

Candidate Keys: {(TID, comment-id)}

Normal Form: BCNF

Table Definition:

```
        create table Comment_Like (
                comment_id int primary key,
                tid bigint ,
                cl_date varchar(10) not null,
                cl_time varchar(5) not null,
                foreign key (tid) references User(tid),
                foreign key (comment_id) references Comment_Write(comment_id)
        );
```

3.17 Comment_Write

Relational Model: Comment_Write(<u>comment-id, post-id</u>, TID)

Functional Dependencies: NONE

Candidate Keys: {(TID, comment-id, post-id)}

Normal Form: BCNF

Table Definition:

```
        create table Comment_Write (
                tid bigint,
                comment_id int,
                post_id int,
                content varchar(255) not null,
                primary key(comment_id, post_id),
                foreign key (tid) references User(tid) ,
                foreign key (post_id) references Post(post_id)
        );
```

3.18 Follow

Relational Model: Follow(<u>follower-TID, followed-TID</u>)

Functional Dependencies: NONE

Candidate Keys: {(follower-TID, followed-TID)}

Normal Form: BCNF

Table Definition:

```
        create table Follow (
        follower_tid bigint,
        followed_tid bigint,
        primary key (follower_tid, followed_tid),
        foreign key (follower_tid) references User(tid),
        foreign key (followed_tid) references User(tid)
);
```