CS 353 Database Systems

BombarBet a Social Betting Platform
Project Functionality Document

Batuhan Tosyalı 21702055
Berk Güler 21402268
Enver Yiğitler 21702285
Ufuk Bombar 21703486

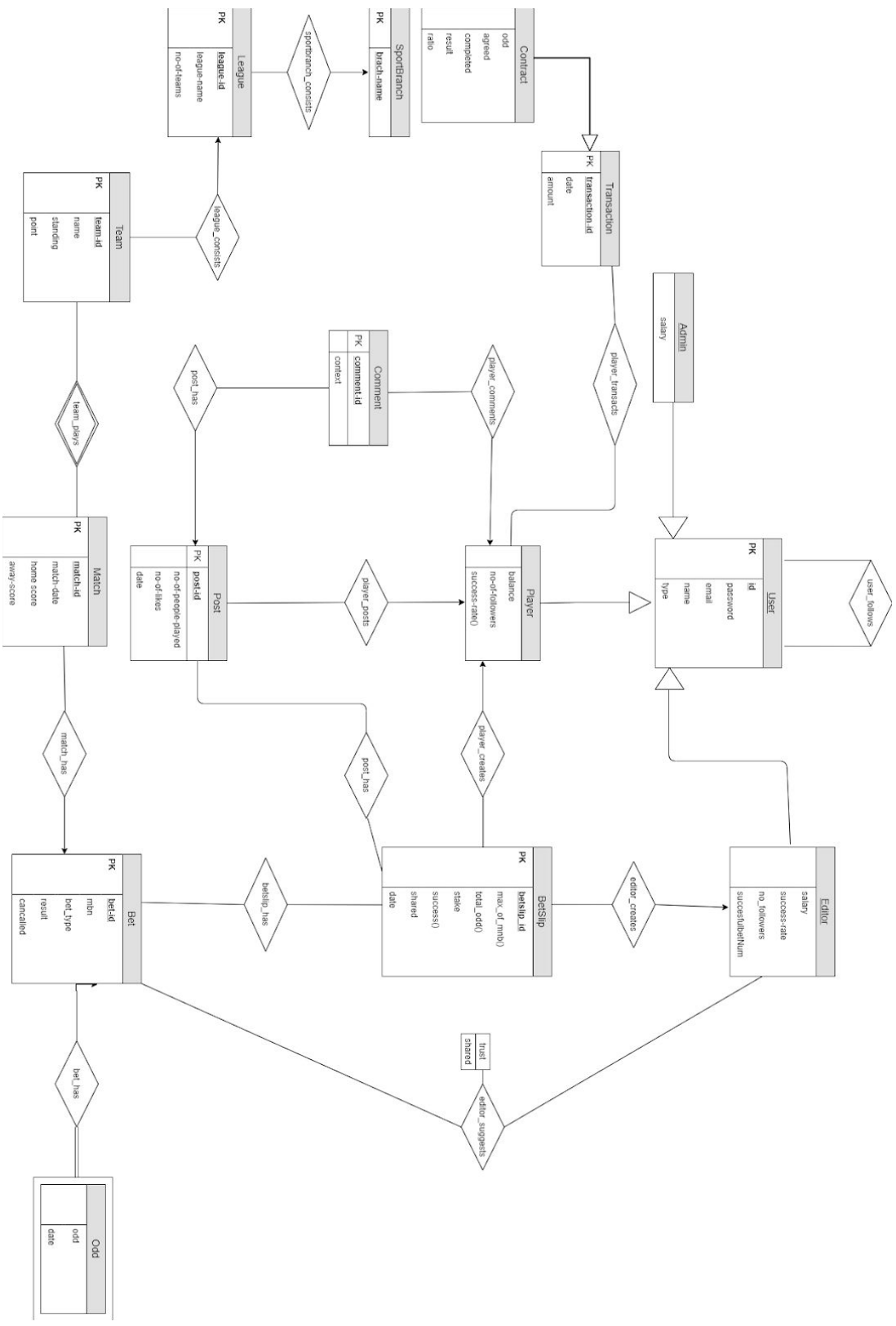
Table of Contents

1. Revised ER Diagram

We have the following changes done in our ER diagram.

- Table **User** has been splitted up to the tables named **Editor , Player** and **Admin**. Editors are well-known people among their communities and they're paid a salary by the system to share suggestions on bets and bet slips. **Admin** is a special kind of user that can ban players and add or remove matches and bets.
  Added salary to the **editor** and **admin.**
- The tables **BasketballBets, HandballBets, FootballBets,** and **TennisBets** are removed. Now the table **"bets"** is responsible for all of the matches and all of the sports. Which is making the table more flexible, now if we were to add a new type of bet like corner we do not need to add another column, the types of bets are holded in bet_type and the sport type is stored in sport_type.
- We added a new functionality, money transactions between users, all of the transactions are held in table **transaction**, the new attributes for that table is the date of transaction and the amount of money in the transaction, and lastly id of the transaction, which is the primary key of the transaction.
- We also added a new functionality which is called contract, where players lend money to another player who has a high success rate, in order to make money for both. The table for this is called simply **contract** and its only attribute is odd.
- Match is no longer a weak entity.
- Entity **Odds** has been added to the table as a **weak entity**. Adding it made it easier to change odds from the position of admins.

The revised ER diagram can be found in the following :


League
PK
league-id
league-name
no-of-teams
sportsranch_consists
SportBranch
PK
brach-name
Contract
odd
agreed
completed
result
ratio
Transaction
PK
transaction-id
date
amount
Admin
salary
Team
PK
team-id
name
standing
point
league_consists
player_transacts
Comment
PK
comment-id
context
post_has
player_comments
team_plays
Match
PK
match-id
match-date
home score
away-score
Post
PK
post-id
no-of-people-played
no-of-likes
date
player_posts
Player
balance
no-of-followers
success-rate()
User
PK
id
password
email
name
type
user_follows
match_has
post_has
player_creates
Bet
PK
bet-id
mbn
bet_type
result
cancelled
betslip_has
BetSlip
PK
betslip_id
max_of_mnb()
total_odd()
stake
success()
shared
date
editor_creates
Editor
salary
success-rate
no_followers
succesfulbetNum
trust
shared
editor_suggests
bet_has
Odd
odd
date

2. Relational Schemas, Functional Dependencies and Normalizations

2.1. User

Relational Model:

User(user_id, type, password, email, name)

Functional Dependencies:

email -> id, type, password, name

Candidate Keys:

{(id), (email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE User(
    id          VARCHAR(30) PRIMARY KEY,
    type        VARCHAR(10) NOT NULL,
    password    VARCHAR(255) NOT NULL,
    email       VARCHAR(255) NOT NULL,
    name        VARCHAR(255) NOT NULL);
```

2.2. Admin

Relational Model:

Admin(id, salary)

FK: id references user

Functional Dependencies:

-

Candidate Keys:

{(id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Admin(
    id          VARCHAR(30) PRIMARY KEY,
    salary      NUMERIC(6,2) NOT NULL,
    FOREIGN KEY(id) REFERENCES User(id));
```

2.3. Player

Relational Model:

Player(id, balance , no_followers)

Functional Dependencies:

-

Candidate Keys:

{(id , email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Player (
        id                  VARCHAR(30) PRIMARY KEY,
        balance             NUMERIC(10,2) NOT NULL,
        no_followers        INT NOT NULL
        FOREIGN KEY(id) REFERENCES User(id));
```

2.4. Editor

Relational Model:

Editor(id, salary, success_rate , successful_bets)

FK: id references user

Functional Dependencies:

-

Candidate Keys:

{(id , email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Editor(
        id                  VARCHAR(30) PRIMARY KEY,
        salary              NUMERIC(6, 2) NOT NULL,
        success_rate        NUMERIC(0, 2) NOT NULL,
        successfull_bets    INT NOT NULL,
        no-of-followers     INT NOT NULL
        FOREIGN KEY (id) REFERENCES User(id));
```

2.5. BetSlip

Relational Model:

BetSlip(betslip_id, creator_user_id, stake, shared)

FK: creator_user_id references the user who creates the bet

Functional Dependencies:

-

Candidate Keys:

{(betslip_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Betslip (
        betslip_id          INT PRIMARY KEY,
        creator_user_id     VARCHAR(30) NOT NULL,
        stake               NUMERIC(10,2) NOT NULL,
        shared              INT NOT NULL,

        FOREIGN KEY(creator_user_id) REFERENCES User(id));
```

2.6. Bet

Relational Model:

Bet(<u>bet_id</u>, match_id, sport_type, odd, mbn, bet_type, cancelled)
FK: match_id

Functional Dependencies:

match_id -> sport_type

Candidate Keys:

{(bet_id), (match_id)}

Normal Form:

2NF

Table Definition:

```
CREATE TABLE Bet (
        bet_id          INT PRIMARY KEY,
        match_id        INT NOT NULL,
        mbn             INT NOT NULL,
        bet_type        VARCHAR(30),
        cancelled       INT NOT NULL,
        FOREIGN KEY(match_id) REFERENCES Match(match_id));
```

2.7. Odd

Relational Model:

Odd(odd, date, bet_id)
FK: bet_id refers to the Bet

Functional Dependencies:

-

Candidate Keys:

{ (date, bet_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Editor_Suggests(
        odd         NUMERIC(3, 2) NOT NULL,
        date        DATETIME NOT NULL,
        bet_id      INT NOT NULL,
        FOREIGN KEY(bet_id) REFERENCES Bet(bet_id));
```

2.8. Match

Relational Model:

Match(match_id, home_team_id, away_team_id, match_date, home_score, away_score)

FK: home_team_id references home team

FK: away_team_id references away team

Functional Dependencies:

-

Candidate Keys:

{(match_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Match (
        match_id        INT PRIMARY KEY,
        home_team_id    INT NOT NULL,
        away_team_id    INT NOT NULL,
        match_date      DATETIME NOT NULL,
        home_score      INT NOT NULL,
        away_score      INT NOT NULL,
        FOREIGN KEY(home_team_id) REFERENCES Team(team_id),
        FOREIGN KEY(away_team_id) REFERENCES Team(team_id));
```

2.9. Team

Relational Model:

Team(team_id, league_id, name, standing, point)

FK: league_id references League

Functional Dependencies:

-

Candidate Keys:

{(team_id)}

Normal Form:

BCNF

Table Definition:

CREATE TABLE Team (

team_id **INT PRIMARY KEY**,
league_id **INT NOT NULL**,
name **VARCHAR**(30) **NOT NULL**,
standing **INT NOT NULL**,
point **INT NOT NULL**,
FOREIGN KEY(league_id) **REFERENCES** League(league_id));

2.10. League

Relational Model:

League(<u>league_id</u>, branch_name, league_name, no_of_teams)
FK: branch_name references SportsBranch

Functional Dependencies:

-

Candidate Keys:

{(league_id , league_name)}

Normal Form:

BCNF

Table Definition:

CREATE TABLE League (

league_id **INT PRIMARY KEY NOT NULL**,
branch_name **VARCHAR**(30) **NOT NULL**,
league_name **VARCHAR**(30) **NOT NULL**,
no_of_teams **INT NOT NULL**,
FOREIGN KEY(branch_name) **REFERENCES** SportBranch(branch_name));

2.11. SportBranch

Relational Model:

SportBranch(<u>branch_name</u>)

Functional Dependencies:

-

Candidate Keys:

-

Normal Form:

BCNF

Table Definition:

CREATE TABLE SportBranch (

branch_name **VARCHAR**(30) **PRIMARY KEY**);

2.12. Post

Relational Model:

Post(<u>post_id</u>, betslip_id, user_id, no_of_people_played, no_of_likes, date)

Functional Dependencies:

-

Candidate Keys:

{(post_id)}

Normal Form:

BCNF

Table Definition:

CREATE TABLE Post (
post_id **INT PRIMARY KEY**,
no_of_people_played **INT NOT NULL**,
betslip_id **INT**,
user_id **VARCHAR**(30),
no_of_likes **INT**,
date **DATE**,
FOREIGN KEY(betslip_id) **REFERENCES** Betslip(betslip_id)
FOREIGN KEY(user_id) **REFERENCES** User(user_id));

2.13. Comment

Relational Model:

Comment(<u>comment_id</u>, post_id, user_id, context)
FK: user_id references Player
FK: post_id references Post

Functional Dependencies:

-

Candidate Keys:

{(comment_id)}

Normal Form:

BCNF

Table Definition:

CREATE TABLE Comment(
comment_id **INT PRIMARY KEY**,
context **VARCHAR**(2048));

2.14. Transaction

Relational Model:

Transaction(transaction_id, type, date, amount, reciever_player_id, sender_player_id)

FK: reciever_player_id references receiver player

FK: giver_player_id references sender player

Functional Dependencies:

reciever_player_id, sender_player_id -> transaction_id, type, date, amount

Candidate Keys:

{(transaction_id), (reciever_player_id, sender_player_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Transaction (
    transaction_id      INT PRIMARY KEY AUTO_INCREMENT,
    type                VARCHAR(10) NOT NULL,
    date                DATETIME NOT NULL,
    amount              NUMERIC(10,2) NOT NULL,
    reciever_player_id  VARCHAR(30) NOT NULL,
    sender_player_id    VARCHAR(30) NOT NULL,
    FOREIGN KEY(reciever_player_id) REFERENCES User(id),
    FOREIGN KEY(sender_player_id) REFERENCES User(id));
```

2.15. Contract

Relational Model:

Contract(transaction_id, odd, agreed,completed,result,ratio)

FK: transaction-id references Contract

Functional Dependencies:

-

Candidate Keys:

{(id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Contract(
    transaction_id      VARCHAR(30) PRIMARY KEY,
    odd                 NUMERIC(3,2) NOT NULL,
    agreed              BOOL NOT NULL,
    completed           BOOL NOT NULL
    result              BOOL NOT NULL
    ratio               BOOL NOT NULL
    FOREIGN KEY(transaction_id) REFERENCES
Transaction(transaction_id));
```

2.16. UserFollows

Relational Model:

UserFollows(follower_id, followee_id)
FK: follower_id refers to the follower user
FK: followee_id refers to the followee user

Functional Dependencies:

-

Candidate Keys:

-

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE UserFollows (
        follower_id         VARCHAR(30),
        followee_id         VARCHAR(30),
        FOREIGN KEY(follower_id) REFERENCES User(id),
        FOREIGN KEY(followee_id) REFERENCES User(id),
        PRIMARY KEY (follower_id, followee_id));
```

2.17. EditorSuggests

Relational Model:

EditorSuggests(user_id, bet-id,trust,share)
FK: bet_id refers to the Bet
FK: user_id refers to the User and therefore Editor

Functional Dependencies:

-

Candidate Keys:

-

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Editor_Suggests(
        bet_id          INT,
        user_id         INT,
        trust           INT,
        share           VARCHAR(30)
        FOREIGN KEY(bet_id) REFERENCES Bet(bet_id),
        FOREIGN KEY(user_id) REFERENCES User(user_id),
        PRIMARY KEY(bet_id, user_id));
```

2.18. BetSlipHas

Relational Model:

BetSlipHas(<u>betslip_id</u>, <u>bet_id</u>)
FK: betslip_id refers to the BetSlip
FK: bet_id refers to the Bet

Functional Dependencies:

-

Candidate Keys:

-

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE BetSlipHas(
    betslip_id      INT NOT NULL,
    bet_id          INT NOT NULL,
    FOREIGN KEY(betslip_id) REFERENCES Betslip(betslip_id),
    FOREIGN KEY(bet_id) REFERENCES Bet(bet_id),
    PRIMARY KEY (betslip_id, bet_id));
```

3. Functionalities

3.1. Additional Functionalities

3.1.1. Transaction and Contract

Our extra functionality is adding the contract system, which a user can send a contract to another user in order to pay the money of the user which the contract sended. In return the user who sent the requests gets a percentage of the bet if the bet succeeded.

The following query gets the transactions of a user. Assume our id is usrid

```
@INPUTS = @userid
SELECT id, amount, ratio
FROM Contract NATURAL JOIN Transaction
WHERE completed = 1 AND successful = 1 AND agreed = 1 AND id = @userid;
```

3.2. Common Functionalities

3.2.1. Signing up

Assume you are signing up as a normal user or an editor.
@INPUTS = @user_name , @password_name, @email_name , @user_id

If the user is a player:
INSERT INTO User
VALUES(user_id, “Player”, @user_name, @user_password, @email_name);

INSERT INTO Player(id, balance , no_followers)
VALUES(@user_name, 1.00, 0);

If user is an editor:
INSERT INTO User(user_id, type, password, email, name)
VALUES(@user_name, “Editor”, @password_name , @email_name, @user_name);

INSERT INTO Editor(id, salary, success_rate , successful_bets)
VALUES(@user_name, 0, 0, 0);

If you are an admin:
INSERT INTO User(user_id, type, password, email, name)
VALUES(@user_name, “Admin”, @password_name , @email_name, @user_name);

INSERT INTO Admin(id, salary)
VALUES(@user_name, 0);

3.2.2. Logging in

Now assume you are logging in to the system
@INPUTS = @username , @password

If you are a player:
SELECT *
FROM Player
WHERE EXISTS(
    SELECT *
    FROM User

```
        WHERE User.name = @username AND User.password = @password);
```

If user is an editor:

```
SELECT *
FROM Editor
WHERE EXISTS(
        SELECT *
        FROM User
        WHERE User.name = @username AND User.password = @password);
```

If you are an admin:

```
SELECT *
FROM Admin
WHERE EXISTS(
        SELECT *
        FROM User
        WHERE User.name = @username AND User.password = @password);
```

3.3. Topic Specific Functionalities

3.3.1. User Makes a Bet

a) We create a view for easing our job.

```
CREATE VIEW AllMatches AS
        SELECT *
        FROM Team NATURAL JOIN Bet NATURAL JOIN Match NATURAL
        JOIN League NATURAL JOIN SportBranch
```

b)

We will do the specific filtering queries and then intersect those queries.Let's consider a few examples.Assume we filter matches where mbn is 1 and matches are football matches

```
SELECT *
FROM AllMatches
WHERE MBN = 2 and and sport-type = Football
```

However, if we filter different things within the same column we need to use union for these.For example assume we are looking for sports that are either football or basketball

```
SELECT *
FROM AllMatches
```

WHERE sport-type = Football **or** sport-type = Basketball

c) User clicks on a bet and the bet is inserted to their betslip.

Inputs **=** @betslip_id, @bet_id

INSERT INTO Betslip_Has
VALUES(@betslip_id, @bet_id**)**

d) First we find the maximum of the mbn's that players have inserted into their betslip. Using that value we'll compare it with the number of bets in the betslip, and if the number of bets in the betslip is less than the bet with maximum MBN then the betslip will be unplayable.

Max mbn query:
SELECT max(mnb)
FROM BetslipHas **NATURAL JOIN** Bet
WHERE betslip_id = input_betslip_id);

Number of bets in bet slip query:
SELECT count(bet_id)
FROM BetslipHas
WHERE betslip_id = user_betslip_id

ii)

- Inputs= @bet_id

Here, assume we start the query via knowing our betslip_id, which is our input for this query named as input_bet_id since we want to check whether the owner of the bet slip has the money. We first use natural join between player and betslip in order to get the correct id pairs. Therefore we use Exist and check the owner id's of the betslip, find the desired result, and check if the balance is bigger than stake. If the query is not an empty table we know that the player has the money to fund his/her bet

SELECT *
FROM Player **NATURAL JOIN** BetSlip
WHERE EXISTS (Player.id = @bet_id and Player.balance >= BetSlip.stake)

e) Assume we start via knowing the betslip_id and post_id. First we need find the number of likes of the post to increment.

Inputs= @bestlip_id , @post_id

```
UPDATE Post
SET no_of_likes = no_of_likes + 1
WHERE Post.id = @post_id
```

Assume from the start we know an editor_id to work with, also we know match_id and bet_type as well. Since user_id and bet_id both are a primary key, it automatically handles when the editor tries to create more than one bets in the match.

INPUTS = @editor_id , @match_id , @bet_type

```
INSERT INTO Editory_Suggests
VALUES(@user_id, @bet_id, @trust);
```

3.3.2. Editor Suggests

To share their bet suggestions, editors will update their shared attribute of bet suggestion to 1. Then Users will see only the best suggestion of the editors they follow.

Inputs: @editor_id, @betslip_id

```
UPDATE Betslip
SET shared = 1
WHERE editor_id = @editor_id AND betslip_id = @betslip_id
```

Users will see the only bet suggestions of the editors they follow.

All the bet suggestions of editors that user follows:
Inputs: @user_id

```
SELECT betslip_id
FROM Editors
NATURAL JOIN Betslip
WHERE  user_id IN
        (SELECT followee
        FROM User AS  U NATURAL JOIN UserFollows
        WHERE U.user_id = @user_id));
```

When a user wants to play the same betslip, another new betslip will be inserted with the same bets into the Betslip relation.

Put all bets of the editor’s bet slip into the player's betslip.

Inputs: @new_betslip_id, @editors_betslip_id

```
INSERT INTO BetslipHas
SELECT betslip_id ,bet_id
FROM Betslip, Bet
WHERE betslip_id = @new_betslip_id AND bet_id IN
        (SELECT bet_id
        FROM BetslipHas
        WHERE betslip_id = @editors_betslip_id);
```

a)

i) First do the query that the performances will be shared to everyone

```
SELECT Editor.successfull_bets
FROM Editor NATURAL JOIN Player;
```

Therefore with this query, we can get every editor's succesful_bet number for each player.

ii) Assume we want to get the players who follow the specific editor and id is given as @editorid, we are getting a table in the end that gets the ids of the followers and the corresponding successful bets that he can see.
INPUTS = @editorid

```
SELECT follower_id , successful_bets
FROM Editor, UserFollows
WHERE Editor.user_id = @editorid AND followee_id = @editorid;
```

With that query we will get the id of the followers who are following the specified editor and get his specified betnum.

b) Assume a user want to like a post (either a comment or a betslip) the id of the post is postid
Inputs= @postid

```
UPDATE Post
SET no-of-likes = no-of-likes + 1
WHERE post_id = @postid
```

And now assume a user wants to comment on a post (either a comment or a betslip) and the id of the post is @postid, also username is @userid and context is @context
Inputs = @postid , @userid , @context

INSERT INTO Comment
VALUES (@postid , @userid , @context)

And lastly let's write a query to find the editors who are followed by the desired player. Assume our users id is @userid
Inputs = @userid
SELECT followee
FROM UserFollows
WHERE UserFollows.follower = @userid

3.3.3. Admin Changes Odd of a Bet or Removes a Bet

a) For this specific scenario assume our filter is sport-type = Basketball and bet_type = MS1

SELECT *
FROM Bet
WHERE Bet.sport_type = Basketball **AND** Bet.bet_type = MS1

Now assume our filter is sport-type = Tennis or sport_type = Football

SELECT *
FROM Bet
WHERE Bet.sport_type = Tennis **OR** Bet.sport_type = Football

b) To change the odds of a specific bet we insert a new odd to Odd table with its insertion date.

Inputs: @bet_id, @new_odd, @date
INSERT INTO Odd
VALUES(@bet_id, @new_odd, @date)

c) To cancel a specific bet we just set the cancelled attribute of the bet.

Input: @bet_id
UPDATE Bet
SET cancelled = 1
WHERE bet_id = @bet_id

d) To find bet odds of bet slips we use dates of bet slips and dates of odds. We find odd values that are just before the date bet slip is created.

Inputs: @betslip_id

```
WITH  odds_before_betslip_date(bet_id, max_date, type) AS
        SELECT bet_id, max(Odd.date) AS max_date
        FROM Betslip NATURAL JOIN BetslipHas NATURAL JOIN Odd
        WHERE betslip_id=@betslip_id AND Betslip.date > Odd.date)
        GROUP BY bet_id)

SELECT bet_id, odd
FROM odds_before_betslip_date NATURAL JOIN Odd
WHERE max_date = Odd.date;
```

4. User Interface Design

Given below the mockup diagram.

Admin Settings Page

Username

Open Profile | Delete User | List Transactions of User

User Deleted...

Match Name	Match Date	MS1	MSX	MS2	2.5A	2.5U	HND1	HNDX	HND2
GS-FB	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
BJK-TS	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
JM-KL	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64

Delete Match

Update Odds

Admin Settings Page (continued)

Match Name	Match Date	MS1	MSX	MS2	2.5A	2.5U	HND1	HNDX	HND2

Add Match

Bet Page for Editor

Editor Profile to All

Player Profile to Others

Ufuk Bombar @ubombar

Followers 0
Following 0

Amount | Send Money

Followers

Following

Contracts

Contracts Pending

My Posts

Create a Contract

User Profile to Himself

İlber Şengör @isengor

Followers 4
Following 1

Balance : 1345 TL

Followers

@ubombar
@wondrous
@envrygtlr
@adanalibatuhan

Following

@alibaba

Contracts

Contracts Pending

My Posts

Bet Played on 1.1.2020
Gain: 123 TL

Match Name	Match Date	Type	MNB	Odd
BJK-TS	2.11.2020	MS1	12	3.14
CS-EEE	2.12.2020	MSX	12	1.59

@alibaba very good ilber keep it up!
@isengor yes

Play Bet

Played Bets

Bet Played on 1.1.2020

Match Name	Match Date	Type	MNB	Odd
BJK-TS	2.11.2020	MS1	12	3.14
CS-EEE	2.12.2020	MSX	12	1.59

Bet Played on 31.12.2019

Match Name	Match Date	Type	MNB	Odd
BJK-TS	2.11.2020	MS1	12	3.14
CS-EEE	2.12.2020	MSX	12	1.59

Bet Page for Player

Home Page

Match Name	Match Date	Type	MNB	Odd
BJK-TS	Today	MS1	12	3.14
CS-EEE	2.12.2020	MSX	12	1.59

Amount

Play Bet

İlber Şengör @isengor

Followers 4
Following 1

Contracts
@alex

Balance : 1345 TL

Contract

Open Profile

Home Page logged in

Football | Basketball | Volleyball | Handball

İlber Şengör @isengor

Followers 4
Following 1

Football Matches

Match Name	**Match Date**	**MS1**	**MSX**	**MS2**	**2.5A**	**2.5U**	**HND1**	**HNDX**	**HND2**
GS-FB	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
BJK-TS	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
JM-KL	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
ALS-ML	Tomorrow	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
CS-EEE	4.25.2020	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
IE-COMD	4.26.2020	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64

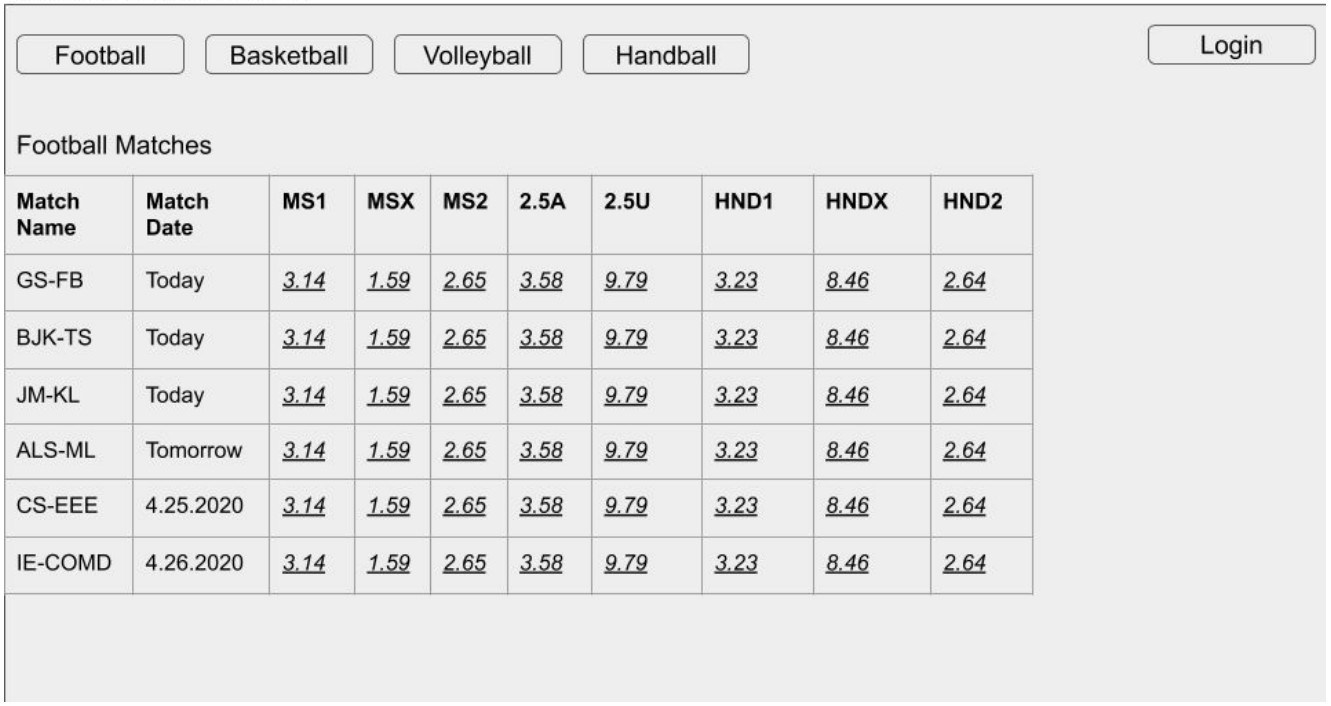
Contracts
@alex

Balance : 1345 TL

Contract

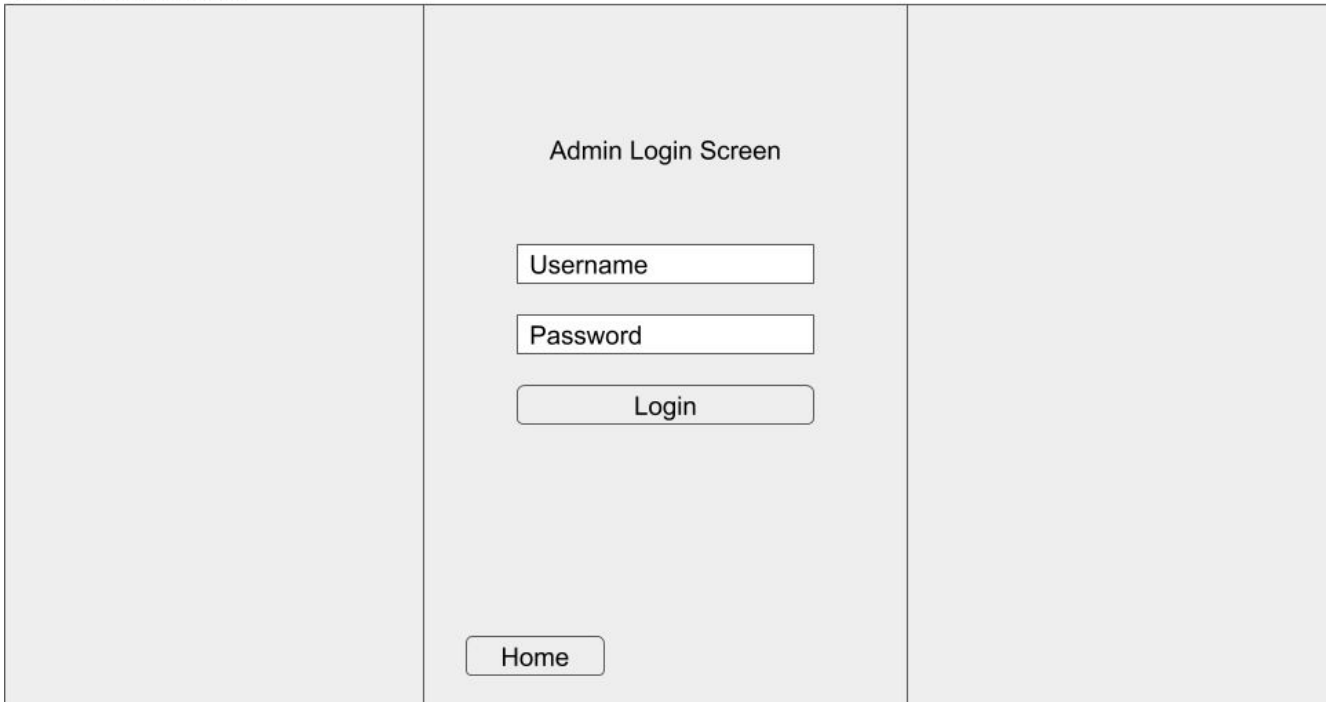
Open Profile

Home Page not logged in

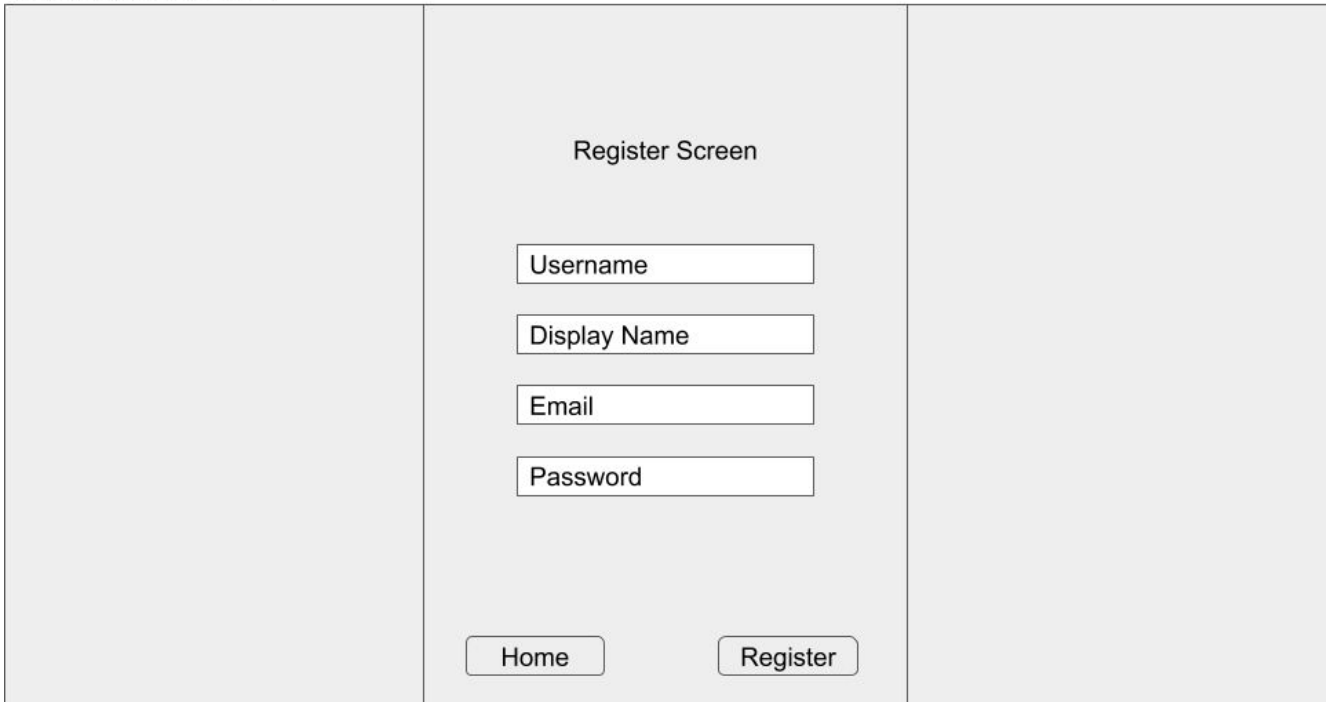


Match Name	Match Date	MS1	MSX	MS2	2.5A	2.5U	HND1	HNDX	HND2
GS-FB	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
BJK-TS	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
JM-KL	Today	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
ALS-ML	Tomorrow	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
CS-EEE	4.25.2020	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64
IE-COMD	4.26.2020	3.14	1.59	2.65	3.58	9.79	3.23	8.46	2.64

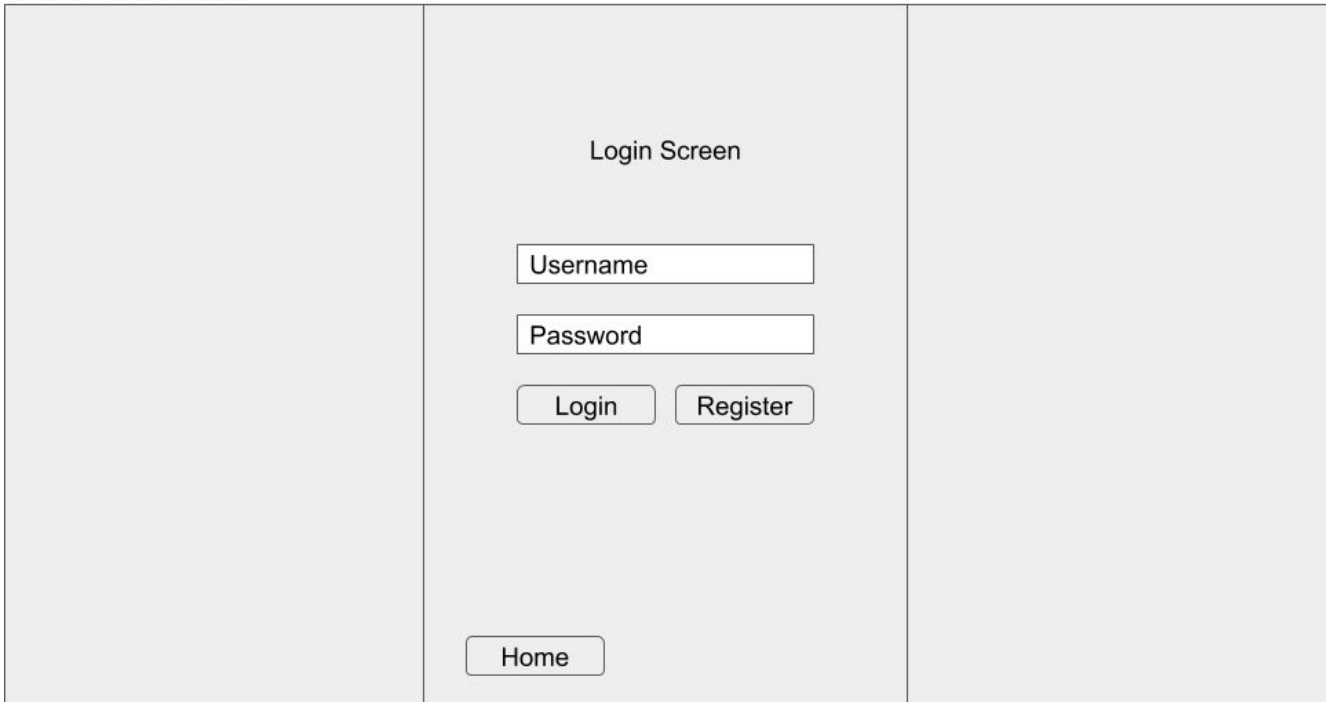
Admin Login Screen



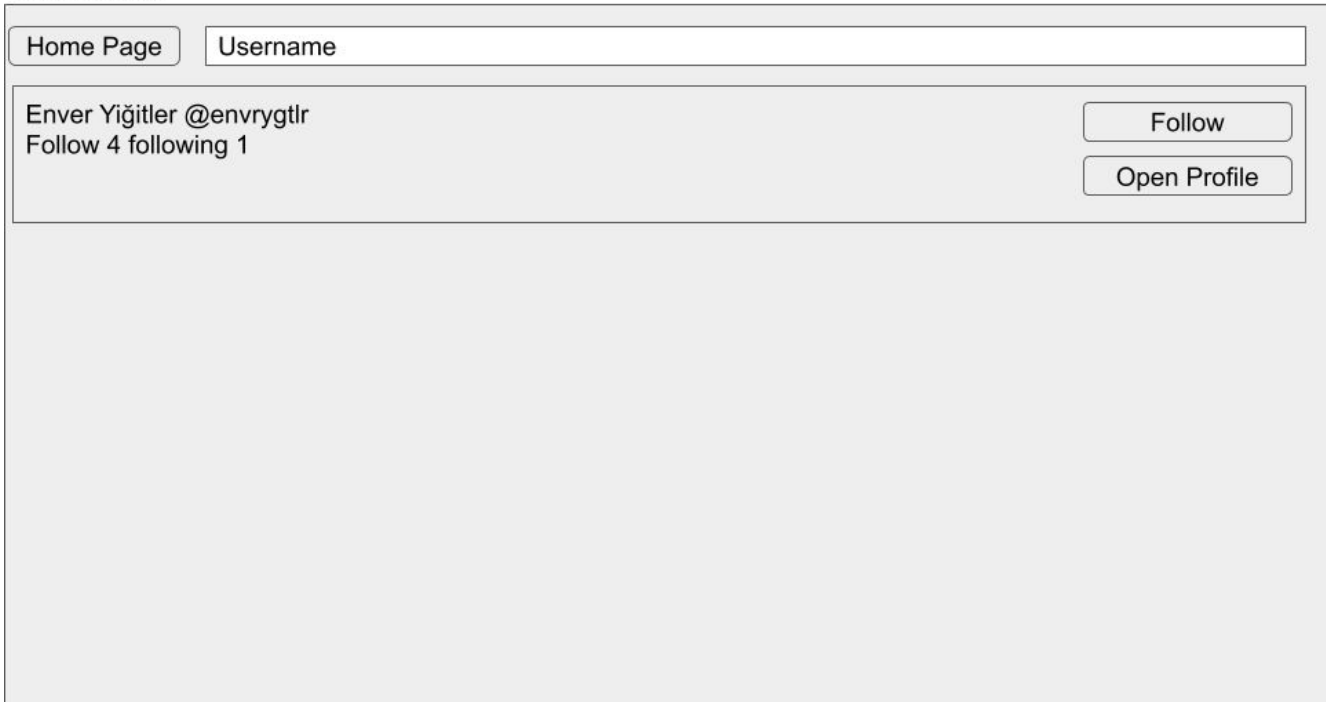
Player Register Screen



Player Login Screen



Search Page



5. Github Link

https://aybukeertekin.github.io/Bilkent_CS353_Database_Project/