Department of Computer Engineering

Bilkent University

CS 353 Term Project

Group 5 - Social Betting Platform

Design Report

İdil YILMAZ - 21703556
Doğukan Ertunga KURNAZ - 21702331
Turgut Alp EDİS - 21702587
Utku GÖKÇEN - 21703746

Instructor: Prof. Dr. Özgür Ulusoy
Teaching Assistant(s): Mustafa Can Çavdar

Apr 06, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Database System course CS353.

Table of contents

Table of contents 2

Revised E/R Model 3
- **Changes in Entities** 3
- **Changes in Relations** 4

Table Schemas 7
- **User** 7
- **Normal User** 8
- **Normal User Friend** 9
- **Normal User Follows** 9
- **Editor** 10
- **Editor Request** 10
- **Slip Creator** 11
- **Admin** 11
- **Bet** 12
- **Bet Slip** 13
- **Comments** 13
- **Item Coupon** 14
- **Matches** 14
- **Results** 15
- **Volleyball Results** 16
- **Basketball Results** 16
- **Football Results** 17
- **LOL Results** 17
- **Esports Team** 18
- **Team** 19
- **Sports** 19
- **Contest** 20
- **Competitors** 20
- **Match Comment** 21
- **Bet Slip Comment** 21
- **Competitor Contest** 22
- **Banned Users** 23
- **Placed On** 23
- **Banned Editors** 24
- **Plays** 25
- **Like Comment** 25
- **Share Bet Slip** 26
- **Edit Bet** 27
- **Buys** 27

UI Design & SQL Statements 28

3.1 Sign Up Page **28**
3.2 Login Page **30**
3.3 Home Page **31**
3.4 Social Page **35**
3.5 Raffle Page **36**

Project Web Page **38**

References **38**

1. Revised E/R Model

After we received the comments for the E/R Model, we changed the E/R model. The changes can be splitted into two parts which are entity and relation.

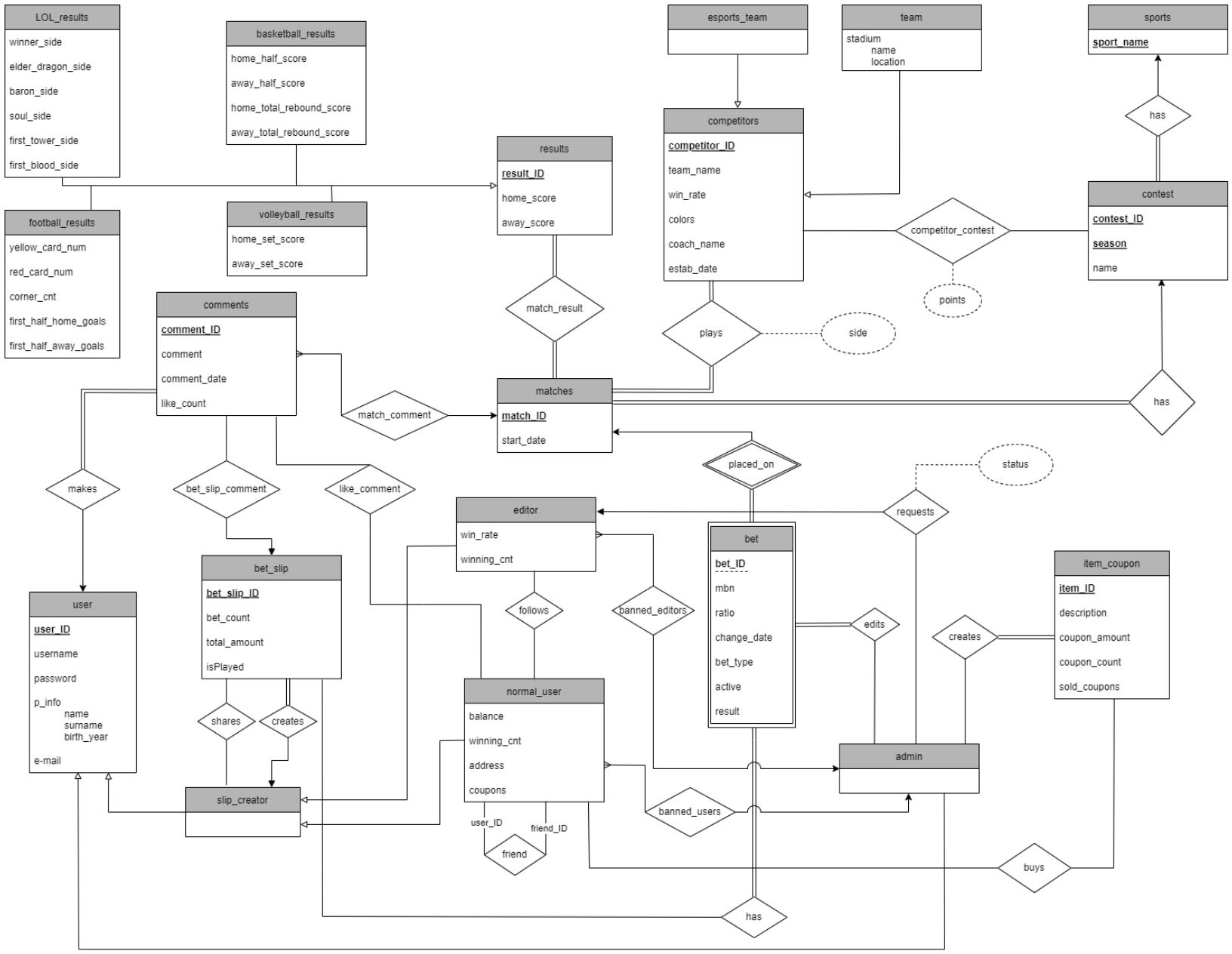
1.1. Changes in Entities

- team_name, colors, estab_date and coach_name attributes are taken from team and esports_team entities and placed into the competitors entity since these attributes are common for both esports_team and team entities.
- start_date is deleted from the primary keys of matches since it is no longer needed.
- bet entity is transformed to the weak entity.
- The attributes active and result are added to the bet entity to show the current status of the bet and result of the bet respectively.
- match_amount attribute in the bet_slip entity is changed to bet_count.
- total_amount attribute is added to bet_slip entity to hold the amount which is deposited by the user.
- slip_creator entity is created to generalize the creator of the bet slips.
- item_coupon is added to the diagram to show the extra functionality of our project which is that the user buys coupons for the item and if the coupon hits, he or she earns the item which his or her coupon indicates.

1.2. Changes in Relations

- The relation between contest and sports is changed to total participation for the contest entity since every contest belongs to a sport.
- The relation between contest and matches is changed to total participation for the matches entity because every match is played within a contest.
- The relation between competitors and matches is changed to total participation for both entities since every match is played with competitors and every competitor has a match.
- placed_bets relation is removed because it is no longer needed.
- All total participations for the bet_slip is removed since bet_slip can be belong to a user or an editor.
- The relations between slip_creator and bet_slip are shares and creates which indicate that slip_creator can share many bet slips and all bet slips are created by slip_creator.
- like_comment relation between comments and normal_user is added to show that the user can like the comments.
- The relation between comments and user is changed to total participation for the comments entity since every comment belongs to a user.

- The relation between results and matches is changed to total participation for both entities since every match has a result and every result corresponds to a match.
- request relation is added to indicate that being an editor requires admin permission.
- Normal users and editors are slip creators.
- Slip creators are users.
- Slip creators can create and share bet slips.
- The relation between item_coupon and normal_user is defined as buys relation which shows that many normal users can buy many coupons.
- The relation between item_coupon and admin is defined as creates relation which shows that admin creates all coupons.


LOL_results
winner_side
elder_dragon_side
baron_side
soul_side
first_tower_side
first_blood_side
basketball_results
home_half_score
away_half_score
home_total_rebound_score
away_total_rebound_score
esports_team
team
stadium
name
location
sports
sport_name
has
competitors
competitor_ID
team_name
win_rate
colors
coach_name
estab_date
results
result_ID
home_score
away_score
contest
contest_ID
season
name
competitor_contest
points
football_results
yellow_card_num
red_card_num
corner_cnt
first_half_home_goals
first_half_away_goals
volleyball_results
home_set_score
away_set_score
match_result
plays
side
comments
comment_ID
comment
comment_date
like_count
matches
match_ID
start_date
match_comment
has
placed_on
status
makes
bet_slip_comment
like_comment
editor
win_rate
winning_cnt
requests
bet
bet_ID
mbn
ratio
change_date
bet_type
active
result
bet_slip
bet_slip_ID
bet_count
total_amount
isPlayed
item_coupon
item_ID
description
coupon_amount
coupon_count
sold_coupons
user
user_ID
username
password
p_info
name
surname
birth_year
e-mail
follows
banned_editors
edits
creates
shares
creates
normal_user
balance
winning_cnt
address
coupons
admin
slip_creator
banned_users
user_ID
friend_ID
friend
buys
has

2. Table Schemas

2.1. User

Relational Model: user(<u>user_ID</u>, username, password, name, surname, birth_year, e-mail)

Functional Dependencies:
user_ID -> username, password, name, surname, birth_year, e-mail
username -> user_ID
e-mail -> user_ID

Candidate Keys: {(user_ID), (username), (e-mail)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE user(
    user_ID INT,
    username VARCHAR(16) NOT NULL UNIQUE,
    password VARCHAR(16) NOT NULL,
    name VARCHAR(20) NOT NULL,
    surname VARCHAR(20) NOT NULL,
    birth_year INT NOT NULL,
    e-mail VARCHAR(255) NOT NULL UNIQUE,
    PRIMARY KEY(user_ID)
    );
```

2.2. Normal User

Relational Model: normal_user(<u>n_user_ID</u>, balance, winning_cnt, address, coupons)
n_user_ID: FK to slip_creator(creator_ID)

Functional Dependencies:
n_user_ID -> balance, winning_cnt, coupons, address

Candidate Keys: {(n_user_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE normal_user(

```
n_user_ID INT,
balance INT,
winning_cnt INT,
address VARCHAR(MAX),
coupons INT,
PRIMARY KEY (n_user_ID),
FOREIGN KEY (n_user_ID) REFERENCES
slip_creator(creator_ID)
ON DELETE CASCADE
);
```

2.3. Normal User Friend

Relational Model: normal_user_friend(<u>user_ID</u>, <u>friend_ID</u>)
user_ID: FK to normal_user(n_user_ID)
friend_ID: FK to normal_user(n_user_ID)

Functional Dependencies:
user_ID, friend_ID -> user_ID, friend_ID

Candidate Keys: {(user_ID, friend_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE normal_user_friend(

```
user_ID INT,
friend_ID INT,
PRIMARY KEY (user_ID, friend_ID),
FOREIGN KEY (user_ID) REFERENCES
normal_user(n_user_ID) ON DELETE CASCADE,
FOREIGN KEY (friend_ID) REFERENCES
normal_user(n_user_ID) ON DELETE CASCADE
);
```

2.4. Normal User Follows

Relational Model: normal_user_follows(<u>editor_ID, user_ID</u>)
editor_ID: FK to editor(editor_ID)
user_ID: FK to normal_user(n_user_ID)

Functional Dependencies:
editor_ID, user_ID -> editor_ID, user_ID

Candidate Keys: {(editor_ID, user_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE normal_user_follows(
editor_ID INT,
user_ID INT,
PRIMARY KEY (editor_ID, user_ID),
FOREIGN KEY (editor_ID) REFERENCES
editor(editor_ID) ON DELETE CASCADE,
FOREIGN KEY (user_ID) REFERENCES
normal_user(n_user_ID) ON DELETE CASCADE
);

2.5. Editor

Relational Model: editor(<u>editor_ID</u>, win_rate, winning_cnt)
editor_ID: FK to slip_creator(creator_ID)

Functional Dependencies:
editor_ID -> win_rate, winning_cnt

Candidate Keys: {(editor_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE editor(
editor_ID INT,
win_rate INT,
winning_cnt INT,
PRIMARY KEY (editor_ID),
FOREIGN KEY (editor_ID) REFERENCES
slip_creator(creator_ID) ON DELETE CASCADE
);

2.6. Editor Request

Relational Model: editor_request(editor_ID, admin_ID, status)
editor_ID: FK to editor(editor_ID)
admin_ID: FK to admin(admin_ID)

Functional Dependencies:
editor_ID -> admin_ID

Candidate Keys: {(editor_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE editor_request(
    editor_ID INT,
    admin_ID INT,
    status VARCHAR(10),
    PRIMARY KEY (editor_ID),
    FOREIGN KEY (admin_ID) REFERENCES
    admin(admin_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (editor_ID) REFERENCES
    editor(editor_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CHECK( status IN ( 'APPROVED', 'PENDING' ) )
    );
```

2.7. Slip Creator

Relational Model: slip_creator(creator_ID)
creator_ID: FK to user(user_ID)

Functional Dependencies:
creator_ID -> creator_ID

Candidate Keys: {(creator_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE slip_creator(
creator_ID INT,
PRIMARY KEY (creator_ID),
FOREIGN KEY (creator_ID) REFERENCES
user(user_ID)
ON DELETE CASCADE ON UPDATE CASCADE
);

2.8. Admin

Relational Model: admin(<u>admin_ID</u>)
admin_ID: FK to user(user_ID)

Functional Dependencies:
admin_ID -> admin_ID

Candidate Keys: {(admin_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE admin(
admin_ID INT,
PRIMARY KEY (admin_ID),
FOREIGN KEY (admin_ID) REFERENCES
user(user_ID) ON DELETE CASCADE
);

2.9. Bet

Relational Model: bet(<u>bet_ID, match_ID</u>, mbn, ratio, change_date, bet_type, active, result)
match_ID: FK to matches(match_ID)

Functional Dependencies:
bet_ID, match_ID -> mbn, ratio, change_date, bet_type, active, result

Candidate Keys: {(bet_ID, match_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE bet(

```
bet_ID INT,
match_ID INT,
mbn INT,
ratio FLOAT (2,5),
change_date TIMESTAMP,
bet_type VARCHAR(30),
active BOOLEAN,
result VARCHAR(10),
PRIMARY KEY (bet_ID, match_ID),
FOREIGN KEY (match_ID) REFERENCES
matches(match_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
CHECK result IN ('WON', 'RESULT', 'PENDING')
);
```

2.10. Bet Slip

Relational Model: bet_slip(<u>bet_slip_ID</u>, creator_ID, bet_count, total_amount, isPlayed)
creator_ID: FK to slip_creator(creator_ID)

Functional Dependencies:
bet_slip_ID -> creator_ID, bet_count, total_amount, isPlayed

Candidate Keys: {(bet_slip_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE bet_slip(

```
bet_slip_ID INT,
bet_count INT,
total_amount INT,
isPlayed BOOL,
PRIMARY KEY (bet_slip_ID),
);
```

2.11. Comments

Relational Model: comments(comment_ID, user_ID, comment, comment_date, like_count)
user_ID: FK to user(user_ID)

Functional Dependencies:
comment_ID -> user_ID, comment, comment_date, like_count

Candidate Keys: {(comment_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE comment(
    comment_ID INT,
    user_ID INT,
    comment VARCHAR(MAX),
    comment_date TIMESTAMP,
    like_count INT,
    PRIMARY KEY (comment_ID),
    FOREIGN KEY(user_ID) REFERENCES
    user(user_ID) ON DELETE CASCADE ON UPDATE
    CASCADE
    );
```

2.12. Item Coupon

Relational Model: item_coupon(item_ID, description, coupon_amount, coupon_count, sold_coupons)

Functional Dependencies:
item_ID -> description, coupon_amount, coupon_count, sold_coupons

Candidate Keys: {(item_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE item_coupon(
item_ID INT,
description VARCHAR(MAX),
coupon_amount INT,
coupon_count INT,
sold_coupons INT,
PRIMARY KEY (item_ID),
);

2.13. Matches

Relational Model: matches(<u>match_ID</u>, start_date, contest_ID, season, sport_name)
contest_ID, sport_name, season: FK to contest(contest_ID, sport_name, season)
sport_name: FK to sports(sport_name)

Functional Dependencies:
match_ID -> start_date, contest_ID, season, sport_name

Candidate Keys: {(match_ID)}

Normal Form: 3NF

Table Definition: CREATE TABLE matches(
match_ID INT,
start_date TIMESTAMP,
contest_ID INT,
season VARCHAR(20),
sport_name VARCHAR(15),
PRIMARY KEY(match_ID),
FOREIGN KEY(contest_ID, season, sport_name)
REFERENCES contest(contest_ID, season,
sport_name) ON DELETE CASCADE ON UPDATE
CASCADE,
FOREIGN KEY(sport_name) REFERENCES
sports(sport_name) ON DELETE CASCADE ON
UPDATE CASCADE
);

2.14. Results

Relational Model: results(<u>result_ID</u>, <u>match_ID</u>, home_score, away_score)
match_ID: FK to matches(match_ID)

Functional Dependencies:
result_ID, match_ID -> home_score, away_score

Candidate Keys: {(result_ID, match_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE results(
    result_ID INT,
    home_score INT,
    away_score INT,
    PRIMARY KEY(result_ID),
    FOREIGN KEY(match_ID) REFERENCES
    matches(match_ID) ON DELETE CASCADE ON
    UPDATE CASCADE
    );
```

2.15. Volleyball Results

Relational Model: volleyball_results(<u>v_result_ID</u>, home_set_score, away_set_score**)**
v_result_ID: FK to results(result_ID)

Functional Dependencies:
v_result_ID -> home_set_score, away_set_score

Candidate Keys: {(v_result_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE volleyball_results(

```
v_result_ID INT,
home_set_score INT,
away_set_score INT,
PRIMARY KEY(v_result_ID),
FOREIGN KEY (v_result_ID) REFERENCES
results(result_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

2.16. Basketball Results

Relational Model: basketball_results(b_result_ID, home_half_score, away_half_score, home_total_rebound_score, away_total_rebound_score**)**
b_result_ID: FK to results(result_ID)

Functional Dependencies:
b_result_ID -> home_half_score, away_half_score, home_total_rebound_score, away_total_rebound_score

Candidate Keys: {(b_result_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE basketball_results(

```
b_result_ID INT,
home_half_score INT,
away_half_score INT,
home_total_rebound_score INT,
away_total_rebound_score INT,
PRIMARY KEY(b_result_ID),
FOREIGN KEY (b_result_ID) REFERENCES
results(result_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

2.17. Football Results

Relational Model: football_results(f_result_ID, yellow_card_num, red_card_num, corner_cnt, first_half_home_goals, first_half_away_goals**)**
f_result_ID: FK to results(result_ID)

Functional Dependencies:
f_result_ID -> yellow_card_num, red_card_num, corner_cnt, first_half_home_goals, first_half_away_goals

Candidate Keys: {(f_result_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE football_results(
    f_result_ID INT,
    yellow_card_num INT,
    red_card_num INT,
    corner_cnt INT,
    first_half_home_goals INT,
    first_half_away_goals INT,
    PRIMARY KEY(f_result_ID),
    FOREIGN KEY (f_result_ID) REFERENCES
    results(result_ID) ON DELETE CASCADE ON
    UPDATE CASCADE
    );
```

2.18. LOL Results

Relational Model: LOL_results(l_result_ID, winner_side, elder_dragon_side, baron_side, soul_side, first_tower_side, first_blood_side**)**
l_result_ID: FK to results(result_ID)

Functional Dependencies:
l_result_ID -> winner_side, elder_dragon_side, baron_side, soul_side, first_tower_side, first_blood_side

Candidate Keys: {(l_result_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE lol_results(

```
l_result_ID INT,
winner_side VARCHAR(4),
elder_dragon_side VARCHAR(4),
baron_side VARCHAR(4),
soul_side VARCHAR(4),
first_tower_side VARCHAR(4),
first_blood_side VARCHAR(4),
PRIMARY KEY(l_result_ID),
FOREIGN KEY (l_result_ID) REFERENCES
results(result_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
CHECK(winner_side IN ('HOME', 'AWAY')),
CHECK(elder_dragon_side IN ('HOME', 'AWAY')),
CHECK(baron_side IN ('HOME', 'AWAY')),
CHECK(soul_side IN ('HOME', 'AWAY')),
CHECK(first_tower_side IN ('HOME', 'AWAY')),
CHECK(first_blood_side IN ('HOME', 'AWAY'))
);
```

2.19. Esports Team

Relational Model: esports_team(<u>competitor_ID</u>**)**
competitor_ID: FK to competitors(competitor_ID)

Functional Dependencies:
competitor_ID -> competitor_ID

Candidate Keys: {(competitor_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE esports_team(

```
competitor_ID INT,
PRIMARY KEY(competitor_ID),
FOREIGN KEY (competitor_ID) REFERENCES
competitors(competitor_ID) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

2.20. Team

Relational Model: team(<u>competitor_ID,</u> name, location)

Functional Dependencies:
competitor_ID -> name, location

Candidate Keys: {(competitor_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE team(
    competitor_ID INT,
    name VARCHAR(50) NOT NULL,
    location VARCHAR(200) NOT NULL,
    PRIMARY KEY(competitor_ID),
    FOREIGN KEY (competitor_ID) REFERENCES
    competitors(competitor_ID) ON DELETE CASCADE
    ON UPDATE CASCADE
    );
```

2.21. Sports

Relational Model: sports(<u>sport_name</u>)

Functional Dependencies:
sport_name -> sport_name

Candidate Keys: {(sport_name)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE sports(
    sport_name VARCHAR(15),
    PRIMARY KEY(sport_name)
    );
```

2.22. Contest

Relational Model: contest(contest_ID, sport_name, season, name)
sport_name: FK to sports(sports_name)

Functional Dependencies:
contest_ID, sport_name, season -> name

Candidate Keys: {(contest_ID, sport_name, season)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE contest(
    contest_ID INT,
    sport_name VARCHAR(15),
    name VARCHAR(30),
    season VARCHAR(20),
    PRIMARY KEY (contest_ID, sport_name, season),
    FOREIGN KEY (sport_name) REFERENCES
    sports(sport_name) ON DELETE CASCADE ON
    UPDATE CASCADE,
    CHECK(sport_name IN ('VOLLEYBALL,
    'FOOTBALL', 'BASKETBALL, 'LOL'))
    );
```

2.23. Competitors

Relational Model: competitors(competitor_ID, team_name, win_rate, colors, estab_date, coach_name)

Functional Dependencies:
competitor_ID -> team_name, win_rate, colors, estab_date, coach_name

Candidate Keys: {(competitor_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE competitors(
competitor_ID INT,
team_name VARCHAR(50),
win_rate FLOAT,
colors VARCHAR(30),
coach_name VARCHAR(MAX),
estab_date INT,
PRIMARY KEY(competitor_ID)
);

2.24. Match Comment

Relational Model: match_comment(<u>match_ID, comment_ID</u>)
match_ID : FK to matches(match_ID)
comment_ID: FK to comments(comment_ID)

Functional Dependencies:
match_ID, comment_ID -> match_ID, comment_ID

Candidate Keys: {(match_ID, comment_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE match_comment(
match_ID INT,
comment_ID INT,
PRIMARY KEY(match_ID, comment_ID),
FOREIGN KEY(match_ID) REFERENCES
matches(match_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY(comment_ID) REFERENCES
comments(comment_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);

2.25. Bet Slip Comment

Relational Model:bet_slip_comment(bet_slip_ID, comment_ID)
bet_slip_ID: FK to bet_slip(bet_slip_ID)
comment_ID: FK to comments(comment_ID)

Functional Dependencies:
comment_ID, bet_slip_ID -> comment_ID, bet_slip_ID

Candidate Keys: {(comment_ID, bet_slip_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE bet_slip_comment(
comment_ID INT,
bet_slip_ID INT,
PRIMARY KEY(comment_ID, bet_slip_ID),
FOREIGN KEY(comment_ID) REFERENCES
comments(comment_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY(bet_slip_ID) REFERENCES
bet_slip(bet_slip_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

2.26. Competitor Contest

Relational Model: competitor_contest(competitor_ID, contest_ID, season, points)
competitor_ID: FK to competitors(competitor_ID)
contest_ID, season: FK to contest(contest_ID, season)

Functional Dependencies:
competitor_ID, contest_ID, season -> points

Candidate Keys: {(competitor_ID, contest_ID, season)}

Normal Form: BCNF

Table Definition: CREATE TABLE competitor_contest(
competitor_ID INT,
contest_ID INT,
season VARCHAR(20),
points INT,
PRIMARY KEY(competitor_ID, contest_ID, season),
FOREIGN KEY (competitor_ID) REFERENCES
competitors(competitor_ID) ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY(contest_ID, season) REFERENCES
contest(contest_ID, season) ON DELETE CASCADE
ON UPDATE CASCADE
);

2.27. Banned Users

Relational Model: banned_users(n_user_ID, admin_ID)
admin_ID: FK to admin(admin_ID)
n_user_ID: FK to normal_user(n_user_ID)

Functional Dependencies:
admin_ID, n_user_ID -> admin_ID, n_user_ID

Candidate Keys: {(admin_ID, n_user_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE banned_users(
admin_ID INT,
n_user_ID INT,
PRIMARY KEY(admin_ID, n_user_ID),
FOREIGN KEY(admin_ID) REFERENCES
admin(admin_ID) ON DELETE CASCADE,
FOREIGN KEY(n_user_ID) REFERENCES
normal_user(n_user_ID) ON DELETE CASCADE
);

2.28. Placed On

Relational Model: placed_on(bet_slip_ID, match_ID, bet_ID)
bet_slip_ID: FK to bet_slip(bet_slip_ID)
bet_ID, match_ID: FK to bet(bet_ID, match_ID)
match_ID: FK to matches(match_ID)

Functional Dependencies:
bet_slip_ID, match_ID -> bet_ID

Candidate Keys: {(bet_slip_ID, match_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE placed_on(
bet_slip_ID INT,
bet_ID INT,
match_ID INT,
PRIMARY KEY(bet_slip_ID, match_ID),
FOREIGN KEY(bet_slip_ID) REFERENCES
bet_slip(bet_slip_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY(bet_ID, match_ID) REFERENCES
bet(bet_ID, match_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY(match_ID) REFERENCES
matches(match_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

2.29. Banned Editors

Relational Model: banned_editors(admin_ID, editor_ID)
admin_ID: FK to admin(admin_ID)
editor_ID season: FK to editor(editor_ID)

Functional Dependencies:
admin_ID, editor_ID -> admin_ID, editor_ID

Candidate Keys: {(admin_ID, editor_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE banned_editors(
admin_ID INT,
editor_ID INT,
PRIMARY KEY(admin_ID, editor_ID),
FOREIGN KEY(admin_ID) REFERENCES
admin(admin_ID) ON DELETE CASCADE,
FOREIGN KEY(editor_ID) REFERENCES
editor(editor_ID) ON DELETE CASCADE
);

2.30. Plays

Relational Model: plays(<u>match_ID, competitor_ID</u>, side)
match_ID: FK to matches(match_ID)
competitor_ID: FK to competitors(competitor_ID)

Functional Dependencies:
match_ID, competitor_ID -> side

Candidate Keys: {(match_ID, competitor_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE plays(
match_ID INT,
competitor_ID INT,
side VARCHAR(4),
PRIMARY KEY(match_ID, competitor_ID),
FOREIGN KEY(match_ID) REFERENCES
matches(match_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY (competitor_ID) REFERENCES
competitors(competitor_ID) ON DELETE CASCADE
ON UPDATE CASCADE,
CHECK (side IN (‘HOME’, ‘AWAY’))
);

2.31. Like Comment

Relational Model: like_comment(comment_ID, n_user_ID)
comment_ID: FK to comments(comment_ID)
n_user_ID: FK to normal_user(n_user_ID)

Functional Dependencies:
comment_ID, n_user_ID -> comment_ID, n_user_ID

Candidate Keys: {(comment_ID, n_user_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE like_comment(
    comment_ID INT,
    n_user_ID INT,
    PRIMARY KEY(comment_ID, n_user_ID),
    FOREIGN KEY(n_user_ID) REFERENCES
    normal_user(n_user_ID) ON DELETE CASCADE ON
    UPDATE CASCADE,
    FOREIGN KEY (comment_ID) REFERENCES
    comments(comment_ID) ON DELETE CASCADE ON
    UPDATE CASCADE
    );
```

2.32. Share Bet Slip

Relational Model: shared_slip(bet_slip_ID, sharer_ID)
bet_slip_ID: FK to bet_slip(bet_slip_ID)
sharer_ID: FK to slip_creator(creator_ID)

Functional Dependencies:
bet_slip_ID, sharer_ID -> bet_slip_ID, sharer_ID

Candidate Keys: {(bet_slip_ID, sharer_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE shared_slip(
bet_slip_ID INT,
sharer_ID INTi
PRIMARY KEY(bet_slip_ID, sharer_ID),
FOREIGN KEY(bet_slip_ID) REFERENCES
bet_slip(bet_slip_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY (sharer_ID) REFERENCES
slip_creator(creator_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);

2.33. Edit Bet

Relational Model: edits(admin_ID, bet_ID, match_ID)
admin_ID: FK to admin(admin_ID)
bet_ID, match_ID: FK to bet(bet_ID, match_ID)

Functional Dependencies:
admin_ID, bet_ID, match_ID-> admin_ID, bet_ID, match_ID

Candidate Keys: {(admin_ID, bet_ID, match_ID)}

Normal Form: BCNF

Table Definition: CREATE TABLE edits(
admin_ID INT,
bet_ID INT,
match_ID INT,
PRIMARY KEY(admin_ID, bet_ID, match_ID),
FOREIGN KEY (admin_ID) REFERENCES
admin(admin_ID) ON DELETE CASCADE ON
UPDATE CASCADE,
FOREIGN KEY(bet_ID, match_ID) REFERENCES
bet(bet_ID, match_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);

2.34. Buys

Relational Model: buys(<u>item_ID, n_user_ID</u>)
item_ID: FK to item_coupon(shop_item_id, item_type)
n_user_ID: FK to normal_user(n_user_ID)

Functional Dependencies:
item_ID, n_user_ID -> item_ID, n_user_ID

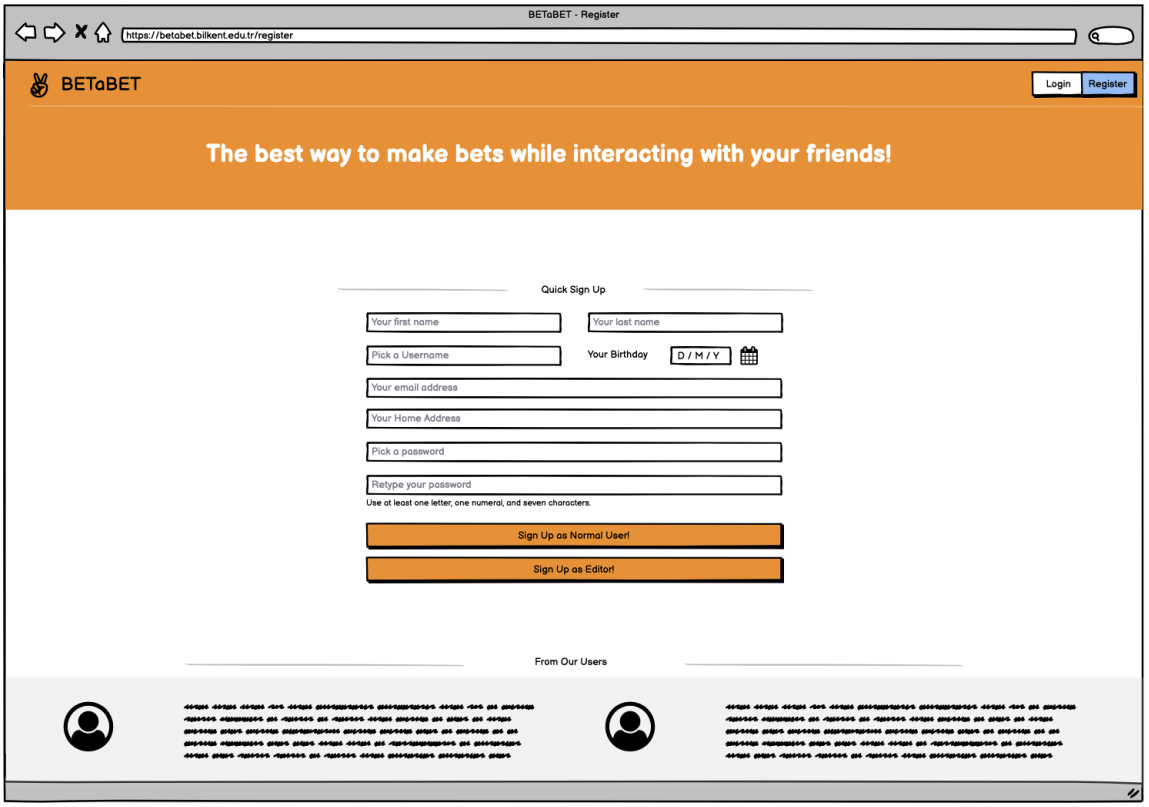
Candidate Keys: {(item_ID, n_user_ID)}

Normal Form: BCNF

Table Definition:
```
CREATE TABLE buys(
    item_ID INT,
    n_user_ID INT,
    PRIMARY KEY(item_ID, n_user_ID), FOREIGN
    KEY(item_ID) REFERENCES item_coupon(item_ID)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(n_user_ID) REFERENCES
    norma_user(n_user_ID) ON DELETE CASCADE ON
    UPDATE CASCADE
    );
```

3. UI Design & SQL Statements

3.1 Sign Up Page



Inputs: @name, @surname, @username, @birth_year, @e-mail, @address, @password

Process: The user will be asked to specify his/her first name, last name, a unique username, birthday, email, address, and password which contains at least one letter, one numeral, and seven characters. Then, the user can sign up as a regular user or as an editor. All editor registers must be approved by the admin. Admin can see the pending editor approvals on the admin dashboard.

SQL Statements:

For Normal User:

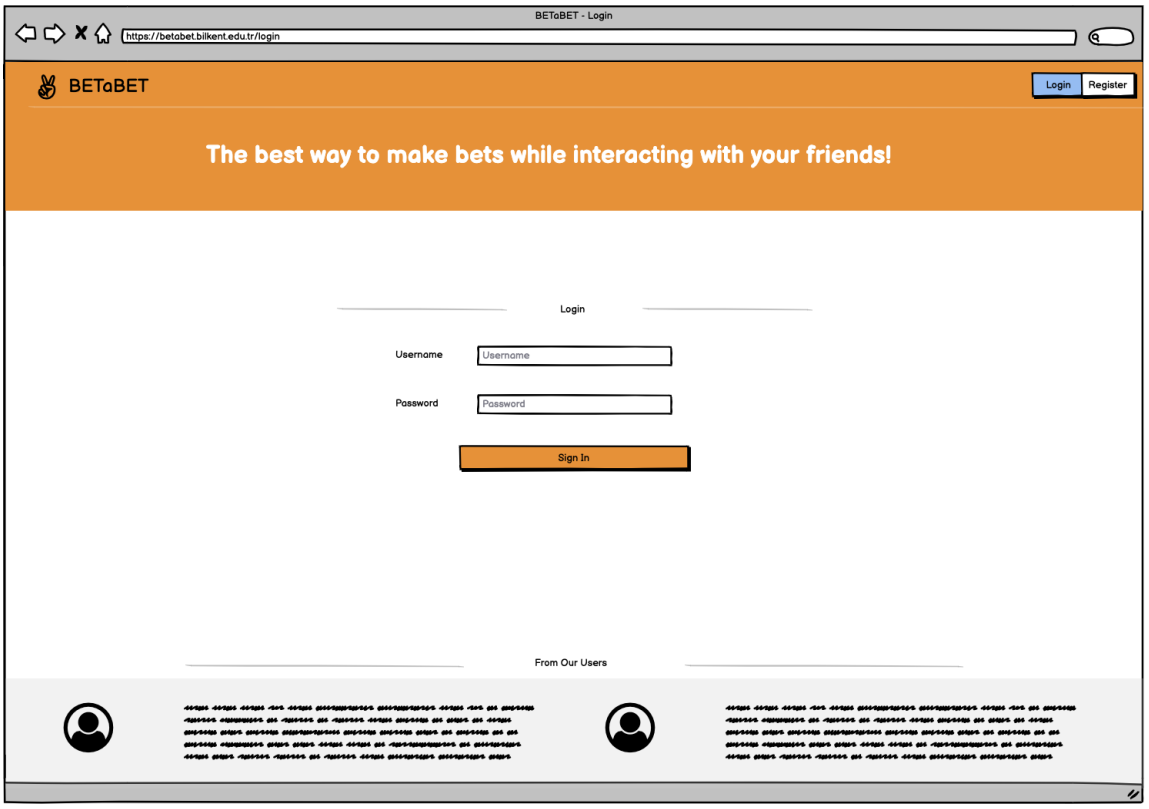
INSERT INTO user(username, name, surname, birth_year, e-mail, password) VALUES (@username, @name, @surname, @birth_year, @e-mail, @password);

INSERT INTO normal_user(balance, winning_cnt, address, coupons)
VALUES(0, 0, @address, 0);

For Editor:
INSERT INTO user(username, name, surname, birth_year, e-mail, password) VALUES (@username, @name, @surname, @birth_year, @e-mail, @password);

INSERT INTO editor_request(editor_ID, admin_ID, status) VALUES (SELECT user_ID FROM user WHERE username = @username, NULL, 'PENDING');

3.2 Login Page



Inputs: @username, @password

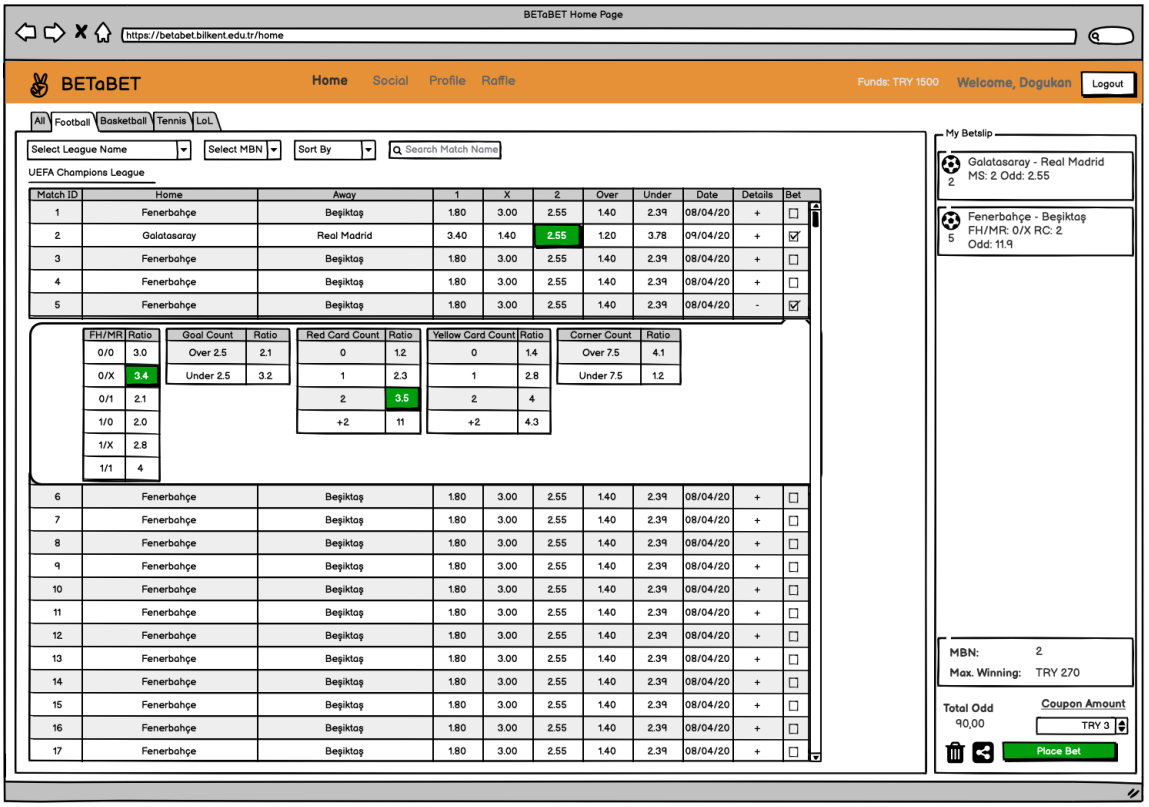
Process: Users will be asked to enter his/her username and password. Users can login to the application by clicking the "Sign In" button. System checks if the user is banned or not.

SQL Statements:

Login:
WITH current_user AS (SELECT user_ID From user WHERE username = @username)

SELECT username, user_ID FROM user WHERE username = @username AND password = @password AND NOT EXISTS (SELECT n_user_ID FROM banned_users WHERE n_user_ID = current_user) AND NOT EXISTS (SELECT editor_ID FROM banned_editors WHERE editor_ID = current_user) AND NOT EXISTS (SELECT editor_ID from editor_request WHERE editor_ID = current_user AND status = 'PENDING')

3.3 Home Page



Inputs: @username**,** @sport_name, @contest_name, @mbn, @sort_type, @search_match, @bet_ID, @match_ID, @total_amount, @bet_type, @user_ID

Process: On the homepage of the BETaBET, users can select a sport name to bet on and after their selection, they can also select league name(s) and minimum bet number. Users have an option to sort matches or they can directly search match names via the search box. The matches will be listed in order to their match IDs and users can bet directly by clicking the ratios next to each match in the minimized view. Also, users have the ability to expand the details of the match in order to see a detailed view of other bet types. In the expanded window users can also select their side by clicking the ratios. Those selections will be added automatically to the betslip area which is on the right side of the page. After users have finished with betting selections, they can see the final results of their betslip's information in the bottom right panel. This panel contains the MBN, Maximum Winning, Total Odd, and Coupon Amount options. Users can specify their amount to bet on by entering or by using the up and down arrows inside that area. If users' selections are correct in terms of MBN and available funds, the place bet button will be available to click on it. The other options are deletion of this betslip and a share button. Users can share their betslip on the social part of the BETaBET website and interact with others. If a user chooses an amount above their available funds or lower than their MBN, an appropriate message will be shown in order to inform the user.

SQL Statements:

Filtering bets and matches with keywords and selections:

```
WITH s_filter AS ( SELECT match_ID from matches WHERE sport_name =
@sport_name),

b_filter AS ( SELECT match_ID FROM bet WHERE active = “true” AND mbn <=
@mbn),

c_filter AS ( SELECT match_ID FROM matches NATURAL JOIN contest WHERE
contest_name IN @contest_name),

esport_filter AS ( SELECT match_ID FROM plays NATURAL JOIN esports_team
WHERE team_name LIKE @search_text),

team_filter AS ( SELECT match_ID FROM plays NATURAL JOIN team WHERE
team_name LIKE @search_text),

final_filter AS ( SELECT match_ID FROM s_filter INTERSECT b_filter
INTERSECT c_filter INTERSECT esport_filter INTERSECT team_filter)

SELECT * FROM final_filter
```

Show all possible bets of a specified sport with the filter:

```
WITH data AS ( SELECT * FROM final_filter NATURAL JOIN matches),

all_competitors AS ( SELECT name, id FROM ( SELECT competitor_ID as id, name
AS name FROM esports_team ) AS tmp UNION ( SELECT competitor_ID AS id,
team_name AS name FROM team) ),

curr_competitors AS ( SELECT competitor_ID as id, side, match_ID FROM plays
NATURAL JOIN final_filter),

all_side AS ( SELECT name, side, match_ID FROM all_competitors NATURAL
JOIN curr_competitors),

b_data AS ( SELECT * FROM bet NATURAL JOIN final_filter ),

o_ratios AS ( SELECT match_ID, MAX(change_date) AS change_date FROM (
SELECT * FROM bet NATURAL JOIN final_filter WHERE active = ‘false’ ) AS
inactives GROUP BY match_ID )
```

SELECT * FROM data NATURAL JOIN b_data NATURAL JOIN all_side NATURAL JOIN o_ratios

Creation of Initial Bet Slip:

INSERT INTO bet_slip(creator_ID, bet_count, total_amount, isPlayed) VALUES (SELECT user_ID FROM user WHERE username = @username, 0, 0, FALSE)

Selection and Addition of Bet to the Slip:

INSERT INTO placed_on(bet_slip_ID, match_ID, bet_ID) VALUES (SELECT bet_slip_ID FROM bet_slip WHERE isPlayed = FALSE AND creator_ID = (SELECT user_ID FROM user WHERE username = @username)), @bet_ID, @match_ID)

System check if minimum bet number is satisfied:

WITH user_bet_slip AS (SELECT bet_slip_ID FROM bet_slip WHERE creator_ID = (SELECT user_ID FROM users WHERE username = @username) AND isPlayed = FALSE),

current_bets as (SELECT * FROM user_bet_slip NATURAL JOIN bet),

current_cnt_bet AS (SELECT Count(bet_slip_ID) AS bet_count FROM current_bets),

max_mbn_cnt AS (SELECT Max(mbn) AS max_mbn FROM current_bets),

SELECT CASE
WHEN current_cnt_bet.bet_count < max_mbn_cnt.max_mbn THEN “MBN condition is not satisfied!”
WHEN current_cnt_bet.bet_count >= max_mbn_cnt.max_mbn THEN “MBN condition is not satisfied!”
    END AS response
FROM current_cnt_bet, max_mbn_cnt

System check if user balance is enough to place the bet:

SELECT CASE
    WHEN user.balance < 2 THEN “Insufficient funds.”
    WHEN user.balance > 2 THEN “Sufficient funds.”
    END AS response
FROM users WHERE user_ID = (SELECT user_ID FROM user WHERE username = @username)

User places money on a bet slip:

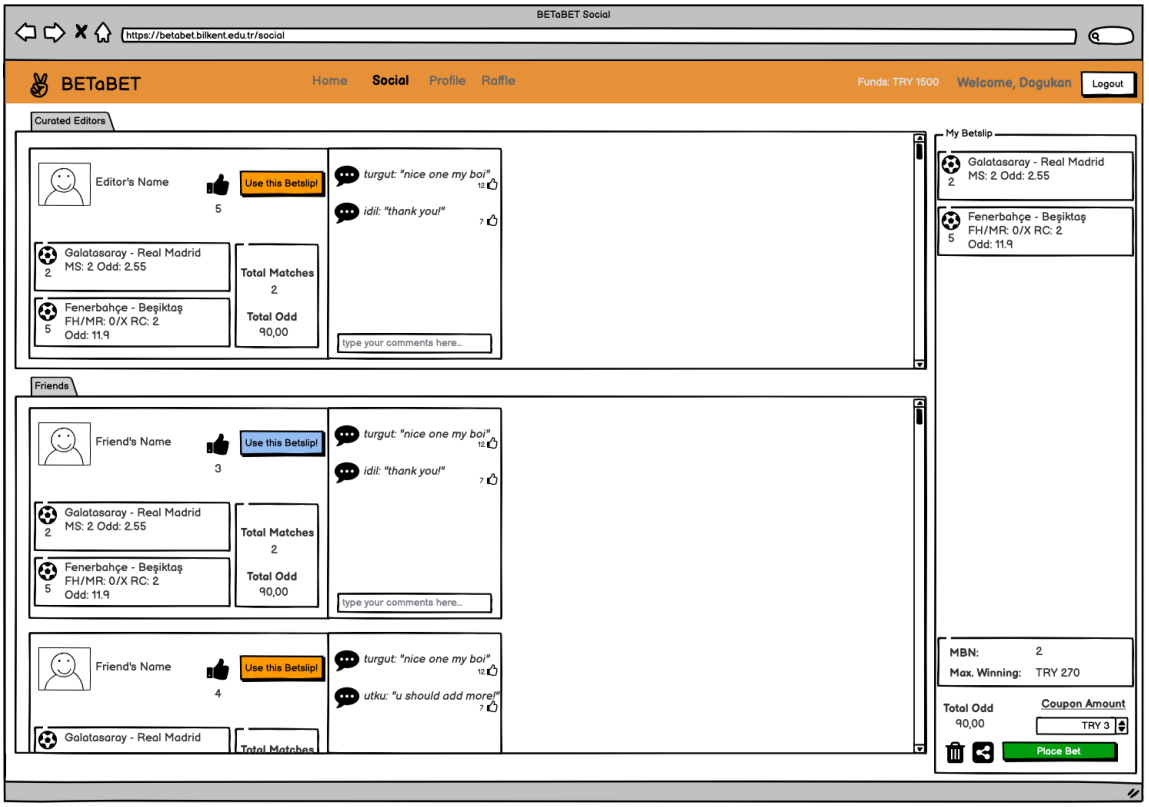
```
UPDATE bet_slip
SET total_amount = @total_amount
WHERE creator_ID = (SELECT user_ID FROM user WHERE username =
@username)
```

User places a bet:

```
UPDATE bet_slip
SET isPlayed = TRUE
WHERE creator_ID = (SELECT user_ID FROM user WHERE username =
@username)

UPDATE user
SET balance = balance - @total_amount
WHERE user_ID = (SELECT user_ID FROM user WHERE username = @username)
```

3.4 Social Page



Inputs: @username, @user_ID, @friend_ID, @bet_slip_ID, @comment_ID, @comment, @like_count, @comment_date, @selected_bet_slip_ID, @selected_comment_ID

Process: When users click on the social tab from the navbar on the website, they can see the feeds from their friends and curated editors. In this feed, users can make comments and like the shared bet slips and like the comments made by other users on the betslips. They can also quickly add the shared betslip to their own betslip by clicking the button named "Use this Betslip!".

SQL Statements:

Display bet slips shared by other users:
WITH friend_ID_set AS (SELECT friend_ID AS user_ID FROM normal_user_friend WHERE user_ID = @user_id),

friend_info AS (SELECT username, user_ID as sharer_ID FROM friend_ID_set NATURAL JOIN user),

friend_slip_ID AS (SELECT * FROM (bet_slip NATURAL JOIN (SELECT user_ID AS sharer_ID FROM friend_ID_set) AS sharing_user)),

friend_slip_bet AS (SELECT * FROM (placed_on NATURAL JOIN friend_slip_ID)),

friend_bet_slip_data AS (SELECT * FROM friend_slip_bet NATURAL JOIN bet),

match_data AS (SELECT * FROM friend_bet_slip_data NATURAL JOIN plays),

all_competitors AS (SELECT name, id FROM (SELECT competitor_ID as id, name AS name FROM esports_team) AS tmp UNION (SELECT competitor_ID AS id, team_name AS name FROM team)),

SELECT * FROM match_data NATURAL JOIN all_competitors NATURAL JOIN (SELECT sharer_ID AS user_ID, username FROM friend_info),

User comments on a betslip:
INSERT INTO comment (comment, user_ID, comment_date)
VALUES (@comment, @user_ID, @comment_date)

DECLARE @comment_id INT
SET @comment_id = SCOPE_IDENTITY()

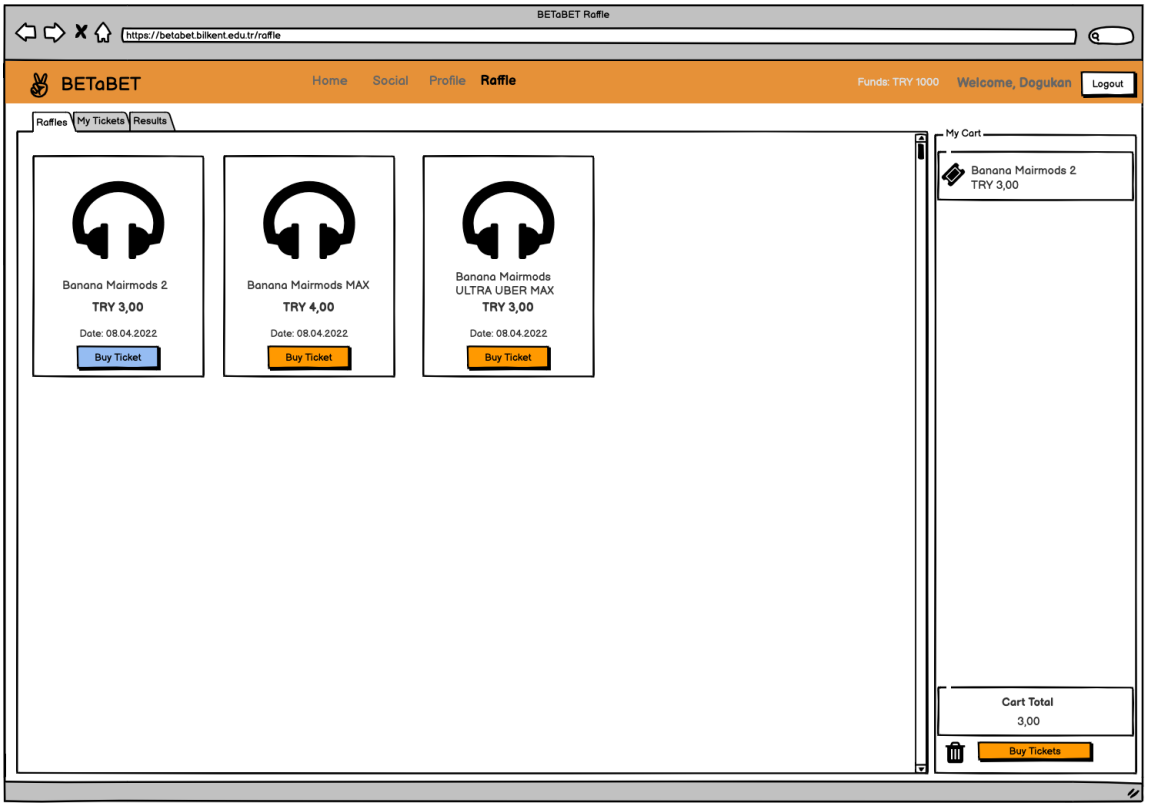
INSERT INTO bet_slip_comment(comment_ID, bet_slip_id)
VALUES (@comment_id, @selected_bet_slip_id)

User deletes a comment:

DELETE FROM bet_slip_comment WHERE (comment_ID = @comment_ID AND user_ID = @user_ID)
DELETE FROM comment WHERE comment_ID = @comment_ID

User likes a comment:
INSERT INTO like_comment (user_ID, comment_ID) VALUES (@user_ID, @selected_comment_ID)

3.5 Raffle Page



Inputs: @username, @item_ID, @coupon_amount, @description

Process: When users click on the raffle tab from the navbar on the website, they can see the open raffles created by the admin. Users can buy raffle tickets from this tab by clicking the "Buy Ticket" button. The selected tickets will be added to the user's cart in the right panel. Users can add multiple raffle tickets to their cart and buy all of them by using the credits they deposited to the website. At the top left of the table, there are other options available for the users. Such as, users can see the tickets that they bought previously. Another option is that users can also keep track of the raffle results from the "Results" tab. If the user won the raffle the product which they bought the ticket for, will be sent to their home address by the BETaBET team.

SQL Statements:

List items:
SELECT DISTINCT description, coupon_amount FROM item_coupon WHERE item_ID = @item_ID AND description = @description

User buys an item:
INSERT INTO buys (item_ID, user_ID) VALUES (@item_id, @user_id)

4. Project Web Page

https://turgut-edis.github.io/SocialBettingProject/

5. References

[1] “EGW: E Gamers World” https://tr.egamersworld.com/bets

[2] “Nesine.com” https://www.nesine.com/

[3] “Misli.com” https://www.misli.com/

[4] “diagrams.net” https://app.diagrams.net/