

1 Part 1

1. Write table definitions from the schema, including any foreign key constraints that you think should be included. You may choose any reasonable data types for attributes, but then be consistent when you write queries in the remaining parts of this problem.

```
CREATE TABLE Room (  
    floor CHAR(50),  
    line CHAR(1),  
    basePrice DECIMAL(10, 2),  
    configCode VARCHAR(50),  
    PRIMARY KEY (floor, line)  
);  
  
CREATE TABLE Guest (  
    gID VARCHAR(20),  
    fname VARCHAR(50),  
    lname VARCHAR(50),  
    phone VARCHAR(20),  
    memStatus VARCHAR(20),  
    PRIMARY KEY (gID)  
);  
  
CREATE TABLE Night (  
    month VARCHAR(20),  
    day INT,  
    weekend BOOLEAN,  
    holiday BOOLEAN,  
    -- both of the above might affect availability  
    PRIMARY KEY (month, day)  
);  
  
CREATE TABLE Reserve (  
    month VARCHAR(20),  
    day INT,  
    floor CHAR(50),  
    line CHAR(1),  
    gID VARCHAR(20),  
    pricePaid DECIMAL(10, 2),  
    notes TEXT,  
    PRIMARY KEY (month, day, floor, line, gID),  
    FOREIGN KEY (floor, line) REFERENCES Room(floor, line),  
    FOREIGN KEY (month, day) REFERENCES Night(month, day),  
    FOREIGN KEY (gID) REFERENCES Guest(gID)  
);
```

```
CREATE TABLE Adjoin (  
    floor CHAR(50),  
    line1 CHAR(1),  
    line2 CHAR(1),  
    PRIMARY KEY (floor, line1, line2),  
    FOREIGN KEY (floor, line1) REFERENCES Room(floor, line),  
    FOREIGN KEY (floor, line2) REFERENCES Room(floor, line)  
);
```

2. Write an SQL query to find the name (fname, lname) and phone number of each guest who reserved a room in March with a price higher than the basePrice.

```
SELECT  
    fname,  
    lname,  
    phone  
FROM  
    Guest  
NATURAL JOIN  
    Reserve  
NATURAL JOIN  
    Room  
WHERE  
    Reserve.month = 'March'  
    AND Reserve.pricePaid > Room.basePrice;
```

3. Write a Relational Algebra query to find the name and phone number of each guest who reserved a room in March with a price higher than the basePrice.

$$\pi_{\text{fname, lname, phone}} (\sigma_{\text{month}='March' \wedge \text{pricePaid} > \text{basePrice}} (\text{Guest} \bowtie \text{Reserve} \bowtie \text{Room}))$$

4. Write a Domain Relational Calculus query to find the name and phone number of each guest who reserved a room in March with a price higher than the basePrice.

$\{ \langle \text{fname}, \text{lname}, \text{phone} \rangle \mid \exists \text{gID}, \text{fname}, \text{lname}, \text{phone} (\langle \text{gID}, \text{fname}, \text{lname}, \text{phone} \rangle \in \text{Guest} \\ \wedge \exists \text{floor}, \text{line}, \text{month}, \text{day}, \text{gID}, \text{pricePaid} \\ (\langle \text{month}, \text{day}, \text{floor}, \text{line}, \text{gID}, \\ \text{pricePaid} \rangle \in \text{Reserve} \\ \wedge \text{month} = \text{'March'} \\ \wedge \exists \text{floor}, \text{line}, \text{basePrice} (\langle \text{floor}, \text{line}, \text{basePrice} \rangle \in \text{Room} \\ \wedge \text{pricePaid} > \text{basePrice})) \}$

5. Write an SQL query to find the gID and phone number of each guest who booked a pair of adjoining rooms for March 22 and March 23.

-- this query works only with the assumption that the adjoining rooms
↪ are inserted in the table in alphanumerical order

```
SELECT DISTINCT G.gID, G.phone
FROM Reserve R1
JOIN Adjoin A ON R1.floor = A.floor AND R1.line = A.line1
JOIN Reserve R2 ON R1.gID = R2.gID AND R2.floor = A.floor AND R2.line
↪ = A.line2
JOIN Guest G ON R1.gID = G.gID
WHERE R1.month = 'March' AND R1.day = 22
AND R2.month = 'March' AND R2.day = 23;
```

6. Write a Relational Algebra query to find the gID and phone number of each guest who booked a pair of adjoining rooms for March 22 and March 23.

$$\pi_{\text{gID}, \text{phone}} \left(\left(\left(\sigma_{\text{month} = \text{'March'} \wedge \text{day} = 22}(\text{Reserve}) \bowtie_{\substack{\text{Reserve.floor} = \text{Adjoin.floor} \\ \text{Reserve.line} = \text{Adjoin.line1}}} \text{Adjoin} \right) \right. \right. \\ \left. \left. \bowtie_{\text{Reserve.gID} = \text{Guest.gID}} \text{Guest} \right. \right. \\ \left. \left. \bowtie \left(\sigma_{\text{month} = \text{'March'} \wedge \text{day} = 23}(\text{Reserve}) \bowtie_{\substack{\text{Reserve.floor} = \text{Adjoin.floor} \\ \text{Reserve.line} = \text{Adjoin.line2}}} \text{Adjoin} \right) \right) \right)$$

7. Write a Domain Relational Calculus query to find the gID and phone number of each guest who booked a pair of adjoining rooms for the nights of March 22 and March 23.

$$\begin{aligned} \{ \langle \text{gID}, \text{phone} \rangle \mid & \exists \text{fname}, \text{lname}, \text{phone}, \text{gID} (\langle \text{gID}, \text{fname}, \text{lname}, \text{phone} \rangle \in \text{Guest} \\ & \wedge \exists \text{floor}, \text{line}, \text{month}, \text{day}, \text{pricePaid} \\ & (\langle \text{month}, \text{day}, \text{floor}, \text{line}, \text{gID}, \text{pricePaid} \rangle \in \text{Reserve} \\ & \wedge \text{month} = \text{'March'} \wedge \text{day} = 22 \\ & \wedge \exists \text{line1}, \text{line2} (\langle \text{floor}, \text{line1}, \text{line2} \rangle \in \text{Adjoin} \\ & \wedge \text{line} = \text{line1} \\ & \wedge \exists \text{month}, \text{day}, \text{pricePaid} \\ & (\langle \text{month}, \text{day}, \text{floor}, \text{line2}, \text{gID}, \text{pricePaid} \rangle \in \text{Reserve} \\ & \wedge \text{month} = \text{'March'} \wedge \text{day} = 23 \wedge \text{line} = \text{line2}))) \} \end{aligned}$$

8. Write an SQL query to find the name, gID, and phone number of the guest who paid the highest total price for all of their bookings in February. (If there are ties, list all of the guests who are tied for the highest.)

```
CREATE VIEW FebruaryTotals AS
SELECT gID, SUM(pricePaid) AS TotalPrice
FROM Reserve
WHERE month = 'February'
GROUP BY gID;

SELECT fname, lname, Guest.gID, phone, TotalPrice
FROM Guest
JOIN FebruaryTotals ON Guest.gID = FebruaryTotals.gID
WHERE TotalPrice = (
    SELECT MAX(TotalPrice) FROM FebruaryTotals
);
```

9. Write an SQL query to find the gIDs, names, and phone numbers of guests who've booked rooms with every configCode during April.

```
SELECT gID, fname, lname, phone
FROM Guest
NATURAL JOIN Reserve
NATURAL JOIN Room
WHERE month = 'April'
GROUP BY gID, fname, lname, phone
```

```
HAVING COUNT(DISTINCT configCode) = (  
    SELECT COUNT(DISTINCT configCode) FROM Room  
);
```

10. Write an SQL query to find rooms with configuration code 'DeluxeKing' that are available (not reserved by anyone) on March 22 and March 23 with base price less than \$150 each night.

```
SELECT floor, line  
FROM Room  
WHERE basePrice < 150  
AND configCode = 'DeluxeKing'  
AND NOT EXISTS (  
    SELECT 1  
    FROM Reserve  
    WHERE Reserve.floor = Room.floor  
    AND Reserve.line = Room.line  
    AND month = 'March'  
    AND day IN (22, 23)  
);
```

11. Using your netID or your NYU N-number as your gID, Write any SQL statements that would be needed for you to reserve room 5C for the nights of March 22 and March 23 at 90% of the basePrice. (You may make up a phone number and other personally identifying info, other than the gID and your name.)

```
INSERT INTO Guest (gID, fname, lname, phone, memStatus) VALUES (  
    ↪ jfg388', 'Junior', 'Garcia', '123-456-7890', 'Gold');  
-- reservation for March 22 at 90% of the basePrice for Junior Garcia  
↪ (jfg388)  
INSERT INTO Reserve (month, day, floor, line, gID, pricePaid, notes)  
VALUES ('March', 22, '5', 'C', 'jfg388', (SELECT 0.9 * basePrice FROM  
    ↪ Room WHERE floor = '5' AND line = 'C'), 'reserved at 90%  
    ↪ basePrice');  
  
-- reservation for March 23 at 90% of the basePrice for Junior Garcia  
↪ (jfg388)  
INSERT INTO Reserve (month, day, floor, line, gID, pricePaid, notes)  
VALUES ('March', 23, '5', 'C', 'jfg388', (SELECT 0.9 * basePrice FROM  
    ↪ Room WHERE floor = '5' AND line = 'C'), 'reserved at 90%  
    ↪ basePrice');
```

12. Briefly discuss the pros and cons of adding a table "Available1(month, day, floor, line, status)" that indicates whether or not each room is available on each night. Which

operations that you expect to be done frequently would be easier and which would be harder. Explain.

This table, as the name implies, allows users to quickly check which rooms are available on which nights without needing to join the Reserve, Room, and Night tables, for example. This might be necessary in a setting where this query occurs frequently, and such information needs to be provided to the user rapidly. Additionally, if the managers of the hotel booking system want to capture trends on room availability over time, this table might be all they need to create offers for different time periods.

The drawback of this approach is data redundancy and inconsistency. Maintaining this table is a projection of data already available in the existing tables of the application, namely Reserve. An insert into the Reserve table implies a modification of the Availability table, which might incur space costs, especially as the application grows.

2 Part 2

1. Indicate whether each of these statements is True or False, according to the ER diagram. Please write out the whole word “True” or “False” to avoid handwriting ambiguity issues.
 - (a) A client can have reservations for two flights on the same *f_date*. **True**
 - (b) A client can have two reservations with different seats for the same *flt_num*, *airline*, and *f_date*. **False**
 - (c) Every client has a reservation on at least one flight. **True**
 - (d) Every client has a reservation on at most one flight. **False**
 - (e) Every flight has at least one client with a reservation on it. **False**
 - (f) Every flight has at most one client with a reservation on it. **False**
2. Derive a relation schema from the E-R diagram, using the technique we studied. You may show your answer as a schema diagram or by writing SQL table definitions.

```
CREATE TABLE Client (  
    cid INT PRIMARY KEY,  
    fname VARCHAR(255),  
    lname VARCHAR(255),
```

```
        email VARCHAR(255)
    );

CREATE TABLE Flight (
    airline VARCHAR(255),
    flt_num VARCHAR(255),
    f_date DATE,
    orig VARCHAR(255),
    dest VARCHAR(255),
    PRIMARY KEY (airline, flt_num, f_date)
);

CREATE TABLE Reserve (
    cid INT,
    airline VARCHAR(255),
    flt_num VARCHAR(255),
    f_date DATE,
    seat VARCHAR(255),
    PRIMARY KEY (cid, airline, flt_num, f_date),
    FOREIGN KEY (cid) REFERENCES Client(cid),
    FOREIGN KEY (airline, flt_num, f_date) REFERENCES Flight(airline,
        ↪ flt_num, f_date)
);
```

3. Modification to include frequent fiers programs and bonus points: Figure 1.
4. SQL Table definitions after modification to include frequent fiers programs and bonus points.

```
CREATE TABLE Airline (
    airline_name VARCHAR(255) PRIMARY KEY,
    ff_program VARCHAR(255)
);

CREATE TABLE Client (
    cid INT PRIMARY KEY,
    fname VARCHAR(255),
    lname VARCHAR(255)L,
    email VARCHAR(255)
);

CREATE TABLE Flight (
    airline_name VARCHAR(255),
```

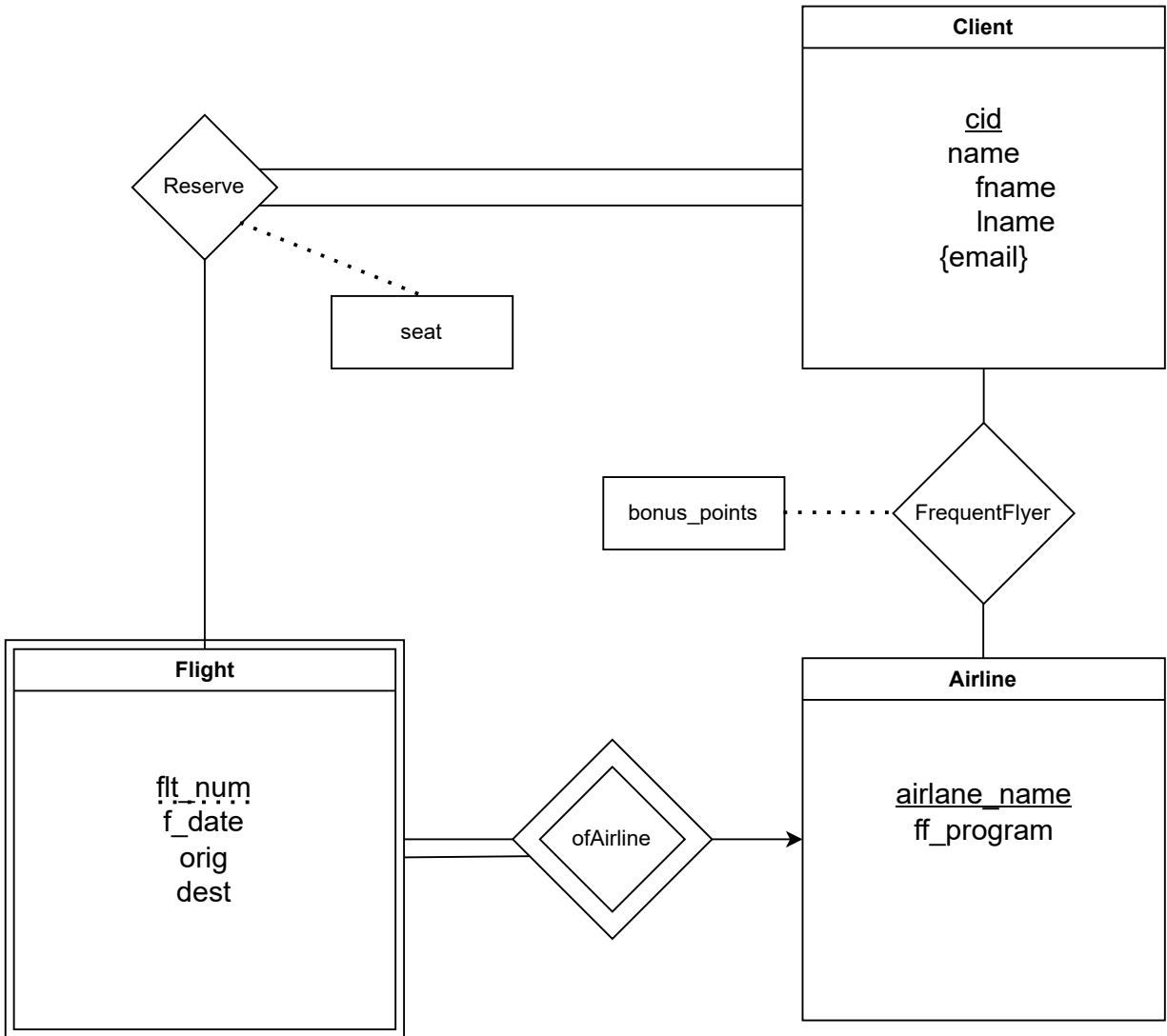


Figure 1: Modified Entity-Relationship Diagram of Airplane Application


```
    flt_num VARCHAR(255),
    f_date DATE,
    orig VARCHAR(255),
    dest VARCHAR(255),
    PRIMARY KEY (airline_name, flt_num, f_date),
    FOREIGN KEY (airline_name) REFERENCES Airline(airline_name)
);

CREATE TABLE FrequentFlyer (
    cid INT,
    airline_name VARCHAR(255),
    bonus_points INT,
    PRIMARY KEY (cid, airline_name),
    FOREIGN KEY (cid) REFERENCES Client(cid),
    FOREIGN KEY (airline_name) REFERENCES Airline(airline_name)
);

CREATE TABLE Reserve (
    cid INT,
    airline_name VARCHAR(255),
    flt_num VARCHAR(255),
    f_date DATE,
    seat VARCHAR(255),
    PRIMARY KEY (cid, airline_name, flt_num, f_date),
    FOREIGN KEY (cid) REFERENCES Client(cid),
    FOREIGN KEY (airline_name, flt_num, f_date) REFERENCES Flight(
        ↪ airline_name, flt_num, f_date)
);
```

3 Part 3: My own Premier League Betting System

1. Textual description

I am a huge Premier League (soccer) fan, so this system will allow users to track bets on different soccer games' scores. The system tracks various teams in the Premier League, each game played during the season, and bets made by users on these games.

There is a **Team** entity set that represents the football teams participating in the Premier League, with the following attributes: *team_name*, and *location*.

There is a **Game** entity set involving two teams (*home_team*, *away_team*) and occurring on a specific *game_date*. Two teams cannot play against each other on the same

game_date. Other attributes include *score_home*, *score_away*, and *stadium*.

There is also a **User** entity, representing users who bet on games. Attributes include *user_id*, *first_name*, *last_name* and *email*.

Additionally, a **Bet** entity represents individual bets made by users on the games, including attributes like *bet_amount*, and *predicted_score_home*, and *predicted_score_away*. For simplicity, we assume a user can only place one bet on a specific game (ie. user A can only place one bet for the game between Arsenal and ManCity) and give us their predicted score through the *predicted_score_home* and *predicted_score_away* values. However, they can still place multiples bets on different games (User A can place bets on 10 games of the season, as long as they are all different games).

A Bet can have an associated BetEnhancer which allows the system managers to increase the profits of the bet for any reason. The reason is the discriminator of this weak entity sets and it includes additional attributes like a multiplier and an additional amount.

2. ER diagram: Figure 2

3. Relational Schema

```
Team(team_name, location)
    PRIMARY KEY (team_name)
```

```
Game(game_date, home_team, away_team, score_home, score_away, stadium)
    PRIMARY KEY (game_date, home_team, away_team)
    FOREIGN KEY (home_team) REFERENCES Team(team_name)
    FOREIGN KEY (away_team) REFERENCES Team(team_name)
```

```
User(user_id, first_name, last_name, email)
    PRIMARY KEY (user_id)
```

```
Bet(bet_id, user_id, game_date, home_team, away_team, bet_amount,
    predicted_score_home, predicted_score_away)
    PRIMARY KEY (bet_id)
    FOREIGN KEY (user_id) REFERENCES User(user_id)
    FOREIGN KEY (game_date, home_team, away_team)
        REFERENCES Game(game_date, home_team, away_team)
```

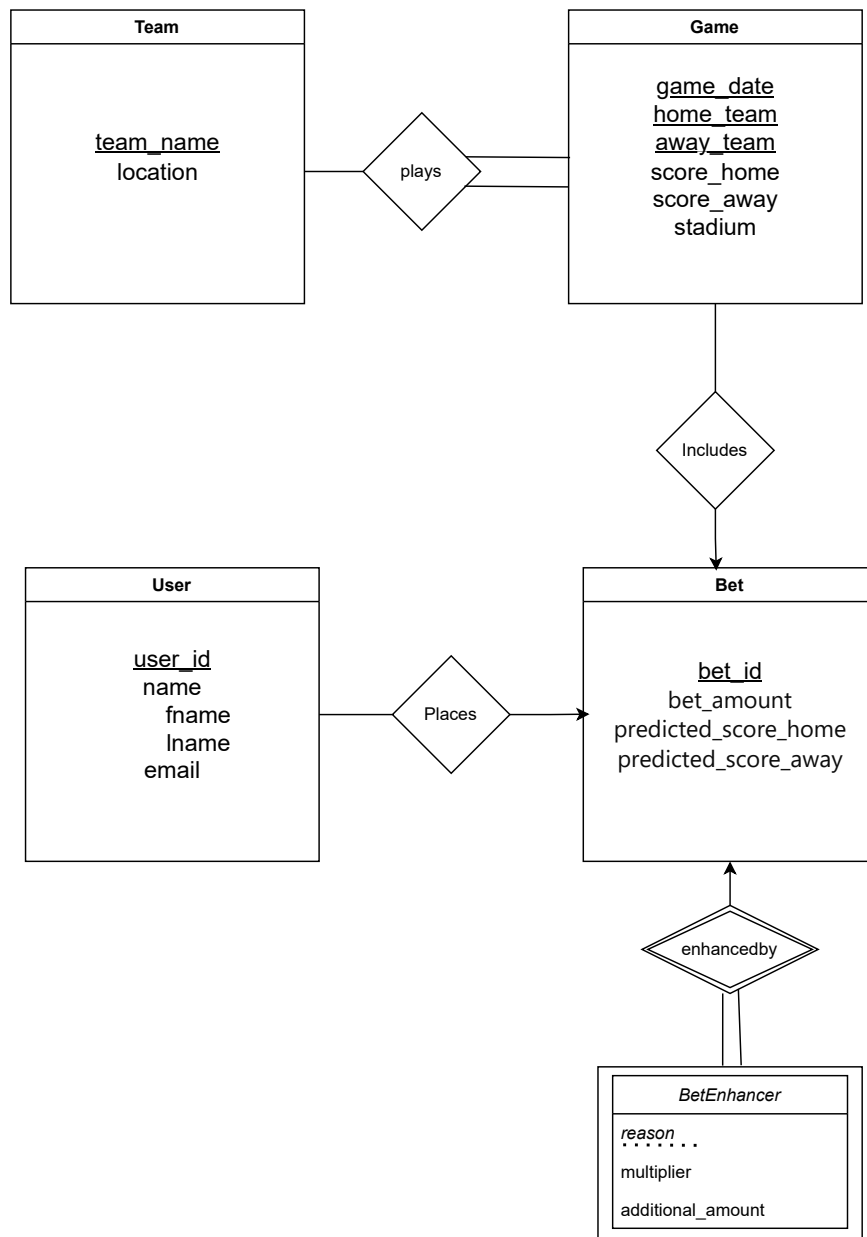


Figure 2: Premier League Betting System

```
BetEnhancer(enhancer_id, bet_id, reason, multiplier, bonus_amount)
    PRIMARY KEY (enhancer_id)
    FOREIGN KEY (bet_id) REFERENCES Bet(bet_id)
```

4. SQL Create Table Statements

```
CREATE TABLE Team (
    team_name VARCHAR(255),
    location VARCHAR(255),
    PRIMARY KEY (team_name)
);
```

```
CREATE TABLE Game (
    game_date DATE,
    home_team VARCHAR(255),
    away_team VARCHAR(255),
    score_home INT,
    score_away INT,
    stadium VARCHAR(255),
    PRIMARY KEY (game_date, home_team, away_team),
    FOREIGN KEY (home_team) REFERENCES Team(team_name),
    FOREIGN KEY (away_team) REFERENCES Team(team_name)
);
```

```
CREATE TABLE User (
    user_id INT PRIMARY KEY,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    email VARCHAR(255)
);
```

```
CREATE TABLE Bet (
    bet_id INT,
    user_id INT,
    game_date DATE,
    home_team VARCHAR(255),
    away_team VARCHAR(255),
    bet_amount DECIMAL(10, 2),
    predicted_score_home INT,
    predicted_score_away INT,
    PRIMARY KEY (bet_id),
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (game_date, home_team, away_team) REFERENCES Game(
        ↪ game_date, home_team, away_team)
);
```

```
);  
  
CREATE TABLE BetEnhancer (  
    enhancer_id INT,  
    bet_id INT,  
    reason VARCHAR(255),  
    multiplier DECIMAL(3,2),  
    bonus_amount DECIMAL(10, 2),  
    PRIMARY KEY (enhancer_id),  
    FOREIGN KEY (bet_id) REFERENCES Bet(bet_id)  
);
```

5. Questions with SQL Queries answers

- (a) What is the total amount bet by each user across all games?

```
SELECT User.user_id, User.first_name, User.last_name, SUM(Bet.  
    ↳ bet_amount) AS total_bet_amount  
FROM User  
NATURAL JOIN Bet  
GROUP BY User.user_id, User.first_name, User.last_name;
```

- (b) Which game had the highest total bets placed on it, and what is the total amount?

```
SELECT game_date, home_team, away_team, SUM(bet_amount) AS  
    ↳ total_bet_amount  
FROM Game  
NATURAL JOIN Bet  
GROUP BY game_date, home_team, away_team  
ORDER BY total_bet_amount DESC  
LIMIT 1;
```

- (c) Who are the users that have a total bet amount greater than \$1000?


```
CREATE VIEW UserTotalBets AS  
SELECT user_id, SUM(bet_amount) AS total_bet_amount  
FROM Bet  
GROUP BY user_id;  
  
SELECT User.user_id, User.first_name, User.last_name,  
    ↳ total_bet_amount  
FROM User  
JOIN UserTotalBets ON User.user_id = UserTotalBets.user_id  
WHERE total_bet_amount > 1000;
```

CS 6083: Principles of Database Systems
Midterm Exam
Professor Phyllis Frankl

Name: Junior Francisco Garcia
NYU ID: jfg388
Date: 2024-03-17

4 Part 4: Academic Integrity Statement

I certify that my solutions to this exam are my own work. I did not consult with other people (with the possible exception of seeking clarification from Prof Frankl) and did not use any generative AI systems to help me with the exam. I understand that if the Professor discovers that I have violated this agreement, I will get a zero on the exam and will be referred to the CS Department and/or Tandon's disciplinary officers for possible additional sanctions.

Signature: 
Date: March 16, 2024