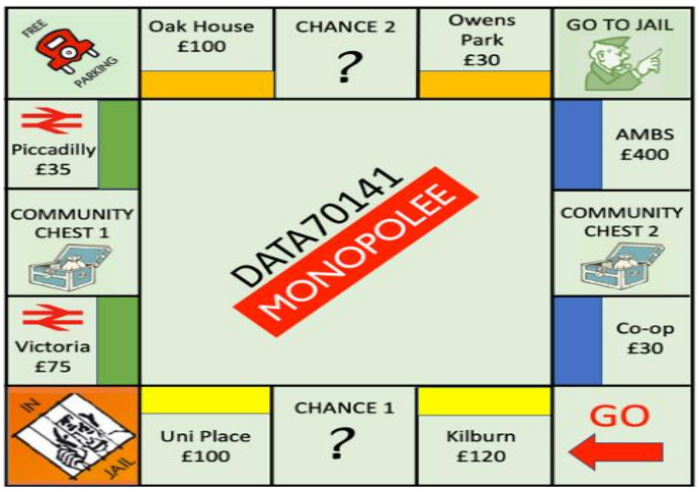
DATA70141: Understanding Databases

ASSIGNMENT 1:

Monopoly Championship 2024 (Using SQLite)

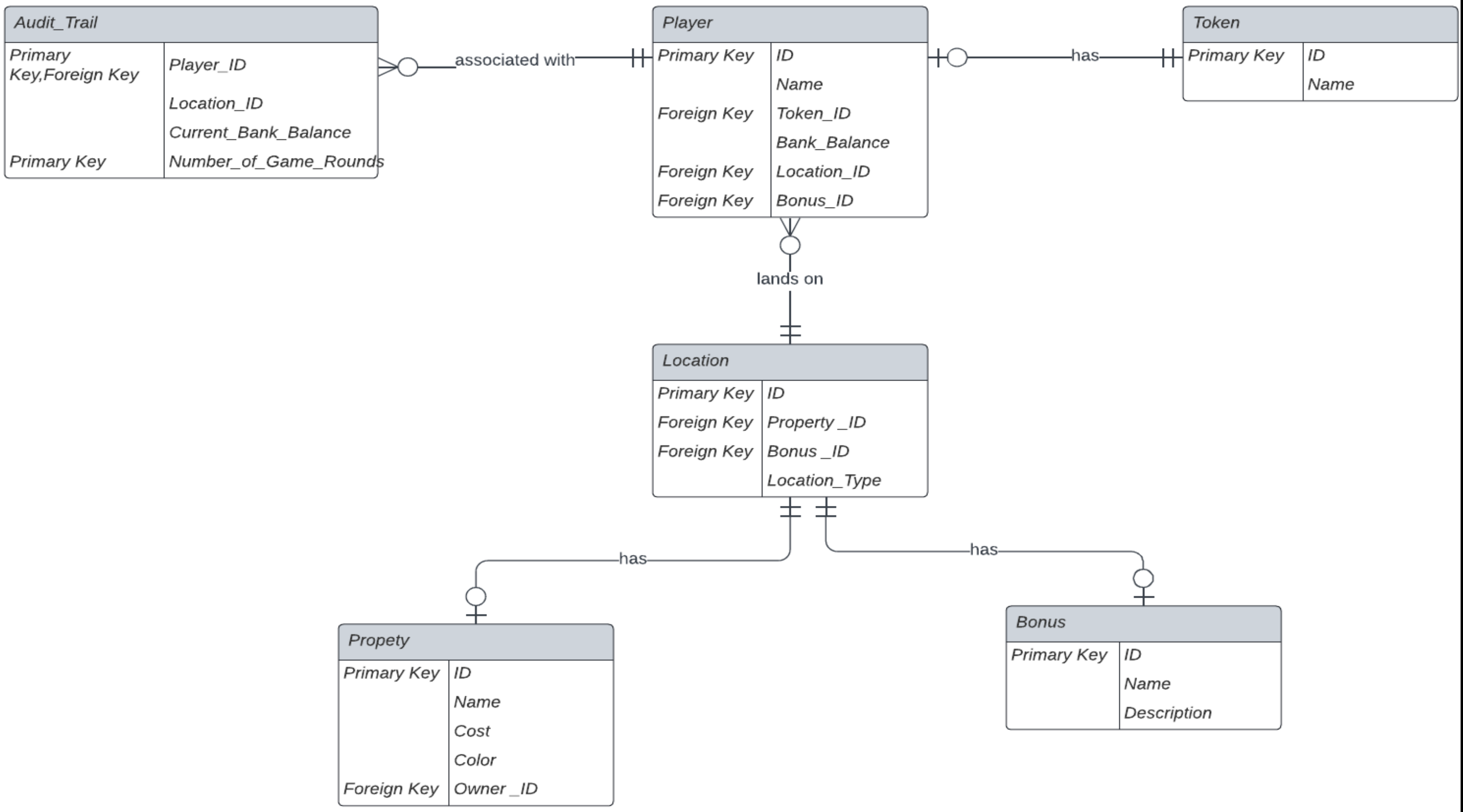
NAME: KASHISH KHARYAL

University ID: 11356488



CONTENTS

ER Diagram in Crow's Foot Notation



ENTITIES:

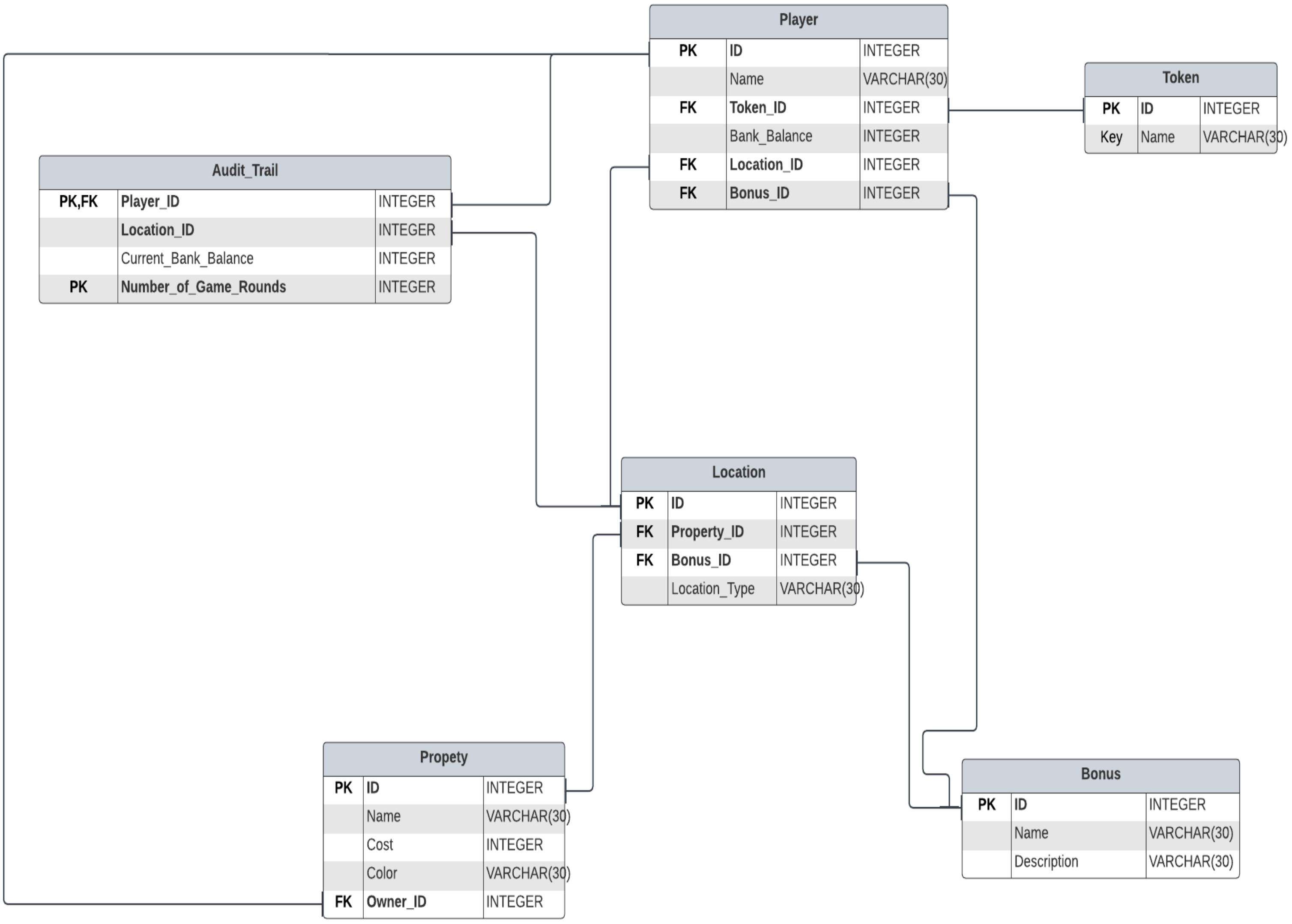
- The **strong Entities** and their constraints are as follows:
  1. Token Entity:
     - This entity represents the different tokens that the players can choose from.
  2. Property Entity:
     - This entity represents the properties on the game board that players can purchase.
  3. Bonus Entity:
     - Represents the various bonuses that can be found on the board.
  4. Location Entity:
     - Represents locations on the game board. Location can be either a property or a bonus.
  5. Player Entity:
     - Represents the players in the game, their attributes, chosen token, bank balances, current location, and active bonuses.
- **Weak Entities** and their constraints (Tables which will not exist without a parent table)
  1. Audit_Trail
     - This entity will record the audit trail of gameplay after every turn, including player movements, bank balances and game rounds.

Relations:

1. **Player – Token Relation**
   - Each player must have a token (one-and-only-one relationship)
   - A token may or may not be chosen by the player (zero-or-one relationship).

2. **Player – Location Relation**
    - Each player must be on any one location on the board at any point of time (one-and-only-one relationship)
    - A location may have one or more players or may not have any player.(Zero-or-many relationship)
3. **Player – Audit_Trail**
    - Each player will have 0 audit_trail entries at the start of the game and will have more number od entries based on the rounds played. (zero-or-many relationship)
    - Audit_trail table will have each gamplay turn associated to a specific player in a single gameround.(one-and-only-one relationship)
4. **Location-Property Relation**
    - Each location may or may not be a property (zero-or-one relationship)
    - Each property is a unique location. (one-and-only-one relationship)
5. **Location- Bonus relationship**
    - Each location may or may not be a Bonus (zero-or-one relationship)
    - Each bonus is a unique location. (one-and-only-one relationship)
6. **Bonus- Player**
    - Each bonus location may have multiple players or no players (zero-or-many)
    - A player may be on a single bonus or may not be on a bonus(zero-or-one)

Relational Database Schema Diagram



Entity Relationship Diagram: DESIGN FEATURES

ENTITIES and CONSTRAINTS

- The **strong Entities** and their constraints are as follows:

6. Token Entity:
    - This entity represents the different tokens that the players can choose from.
    - Primary key: 'ID'
    - UNIQUE and NOT NULL Constraint: 'Name'
7. Property Entity:
    - This entity represents the properties on the game board that players can purchase.
    - Primary Key: 'ID'
    - UNIQUE ad NOT NULL Constraint: 'Name'
    - Foreign Key: 'Owner_ID' is referencing the player table's 'ID'. This will track property ownership.
8. Bonus Entity:
    - Represents the various bonuses that can be found on the board.
    - Primary Key: 'ID'
    - UNIQUE ad NOT NULL Constraint: 'Name'
9. Location Entity:
    - Represents locations on the game board. Location can be either a property or a bonus.
    - Primary Key: 'ID'
    - NOT NULL Constraint: 'Location_Type'
    - Foreign Key:
        - 'Bonus_ID' is referencing the Bonus table's 'ID.
        - 'Property_ID' is referencing the Property table's 'ID'.
10. Player Entity:
    - Represents the players in the game, their attributes, chosen token, bank balances, current location, and active bonuses.
    - Primary Key: 'ID'
    - UNIQUE and NOT NULL Constraint: 'Name'
    - Foreign Key:
        - 'Token_ID' referencing the Token table
        - 'Location_ID' is refercing the Location table
        - 'Bonus_ID' is referencing the Bonus table

- **Weak Entities** and their constraints (Tables which will not exist without a parent table)

2. Audit_Trail
    - This entity will record the audit trail of gameplay after every turn, including player movements, bank balances and game rounds.
    - Candidate Key: 'Player_ID', 'Number_of_Game_Rounds'.
    - NOT NULL Constraint: Player_ID', 'Number_of_Game_Rounds', 'Current_Bank_Balance','Location_ID'.
    - Foreign Key:
        - 'Player_ID' is referencing the Player table's 'ID'.
        - Location_ID is referencing the Location table's 'ID'.

Database Normalisation Level:

- **1st Normal Form:** Each row in a table must be unique and each column consists atomic values.
    1. **The following tables are normalised based on 1NF:**
        1. Token (Primary key : ID)
        2. Player (Primary key : ID)
        3. Location (Primary key : ID)
        4. Property (Primary key : ID)
        5. Bonus (Primary key : ID)
        6. Audit_Trail (Primary key : ID)
    2. **LOCATION TABLE(1NF):**
        - Location table is a generated as a result of normalization.
        - The Location table has a reference of both property and bonus which will be used in finding the **Player's current location**

- I have not applied anymore normalisation as the current normalisation level is well suited for my database design. In future we can normalise it further to 2NF as well.

GAME RULES

- R1 If a player lands on a property without an owner, they must buy it.
- R2 If player P lands on a property owned by player Q, then P pays Q a rent equal to the cost of the property. If Q owns all the properties of a particular colour, P pays double rent.
- R3 If a player is in jail, they must roll a 6 to get out. They immediately roll again.
- R4 If a player lands on or passes GO they receive £200.
- R5 If a player rolls a 6, they move 6 squares; whatever location they land on has no effect. They then get another roll immediately.
- R6 If a player lands on "Go to Jail", they move to Jail, without passing GO.
- R7 If a player lands on a Chance or Community Chest location, the action described by the bonus happens.

ASSUMPTIONS

1. **Dice Roll:**
   We are considering the dice roll value to be given manually.
2. **Player's Location after dice roll**:
   Consider we already know the new location of the player after it moves from its current location.
3. We know the player name and ID associated with the player.
4. **Virtual Bank:**
   Consider the players get paid bonus from a virtual bank with unlimited funds.
5. If a player is in jail , they have to roll a 6 to get out . But, **they will not move 6 steps**. Roll again immediately.

SQL queries for table creation

```
-- Create the Token table
CREATE TABLE Token (
	ID INTEGER PRIMARY KEY,
	Name VARCHAR(30) NOT NULL UNIQUE
);

-- Create the Property table
CREATE TABLE Property (
	ID INTEGER PRIMARY KEY,
	Name VARCHAR(30) NOT NULL UNIQUE,
	Cost INTEGER NOT NULL,
	Color VARCHAR(30) NOT NULL,
	Owner_ID INTEGER,
	FOREIGN KEY (Owner_ID) REFERENCES Player(ID)
);

-- Create the Bonus table
CREATE TABLE Bonus (
	ID INTEGER PRIMARY KEY,
	Name VARCHAR(30) NOT NULL UNIQUE,
	Description VARCHAR(30) NOT NULL
);

-- Create the Location table
CREATE TABLE Location (
	ID INTEGER PRIMARY KEY AUTOINCREMENT,
	Location_type VARCHAR(30) NOT NULL,
	Bonus_ID INTEGER,
	Property_ID INTEGER,
	FOREIGN KEY (Bonus_ID) REFERENCES Bonus(ID),
	FOREIGN KEY (Property_ID) REFERENCES Property(ID)
);

-- Create the Player table
CREATE TABLE Player (
	ID INTEGER PRIMARY KEY,
	Name VARCHAR(30) NOT NULL UNIQUE,
	Token_ID INTEGER,
	Bank_Balance INTEGER,
	Location_ID INTEGER,
	Bonus_ID INTEGER,
	FOREIGN KEY (Token_ID) REFERENCES Token(ID),
	FOREIGN KEY (Location_ID) REFERENCES Location(ID),
	FOREIGN KEY (Bonus_ID) REFERENCES Bonus(ID)
);

-- Create the Audit_Trail table
CREATE TABLE Audit_Trail (
	Player_ID INTEGER NOT NULL,
	Location_ID INTEGER NOT NULL,
	Current_Bank_Balance INTEGER NOT NULL,
	Number_of_Game_Rounds INTEGER NOT NULL,
	FOREIGN KEY (Player_ID) REFERENCES Player(ID),
	PRIMARY KEY (Player_ID, Number_of_Game_Rounds)
);
```

Insert Statements for initial setup.

```
-- Inserting data into the Token table
INSERT INTO Token VALUES
(1,'Dog'),
(2,'Car'),
(3,'Battleship'),
(4,'Top Hat'),
(5,'Thimble'),
(6,'Boot');

-- Inserting data into the Property table
INSERT INTO Property (ID, Name, Cost, Color, Owner_ID) VALUES
(1, 'Oak House', 100, 'Orange', NULL),
(2, 'Owens Park', 30, 'Orange', NULL),
(3, 'AMBS', 400, 'Blue', NULL),
(4, 'Co-Op', 30, 'Blue', NULL),
(5, 'Kilburn', 120, 'Yellow', NULL),
(6, 'Uni Place', 100, 'Yellow', NULL),
(7, 'Victoria', 75, 'Green', NULL),
(8, 'Piccadilly', 35, 'Green', NULL);

-- Inserting data into the Bonus table
INSERT INTO Bonus (ID, Name, Description) VALUES
(1, 'Chance 1', 'Pay each of the other players £50'),
(2, 'Chance 2', 'Move forward 3 spaces'),
(3, 'Community Chest 1', 'For winning a Beauty Contest, you win £100'),
(4, 'Community Chest 2', 'Your library books are overdue. Play a fine of £30'),
(5, 'Free Parking', 'No action'),
(6, 'Go to Jail', 'Go to Jail, do not pass GO, do not collect £200'),
(7, 'GO', 'Collect £200');

-- Insert data into the Location table for Properties
INSERT INTO Location (Location_Type, Bonus_ID, Property_ID)
SELECT 'Property', NULL, ID
FROM Property;
-- Insert data into the Location table for Bonuses
INSERT INTO Location (Location_Type, Bonus_ID, Property_ID)
SELECT 'Bonus', ID, NULL
FROM Bonus;

-- Inserting data into the Player table
INSERT INTO Player (ID,Name,Token_ID,Bank_Balance,Location_ID,Bonus_ID) VALUES
(1,'Mary', 3, 190, 13, 5),
(2,'Bill', 1, 500, 5, NULL),
(3,'Jane', 2, 150, 1, NULL),
(4,'Norman', 5, 250, 3, NULL);

--adding the Ownwer_ID column of Player table using UPDATE,
--This is done beacuse the Player and Property Table have circular references and SQLITE does not support that

UPDATE Property
SET Owner_ID = 4
WHERE ID = 1;

UPDATE Property
SET Owner_ID = 4
```

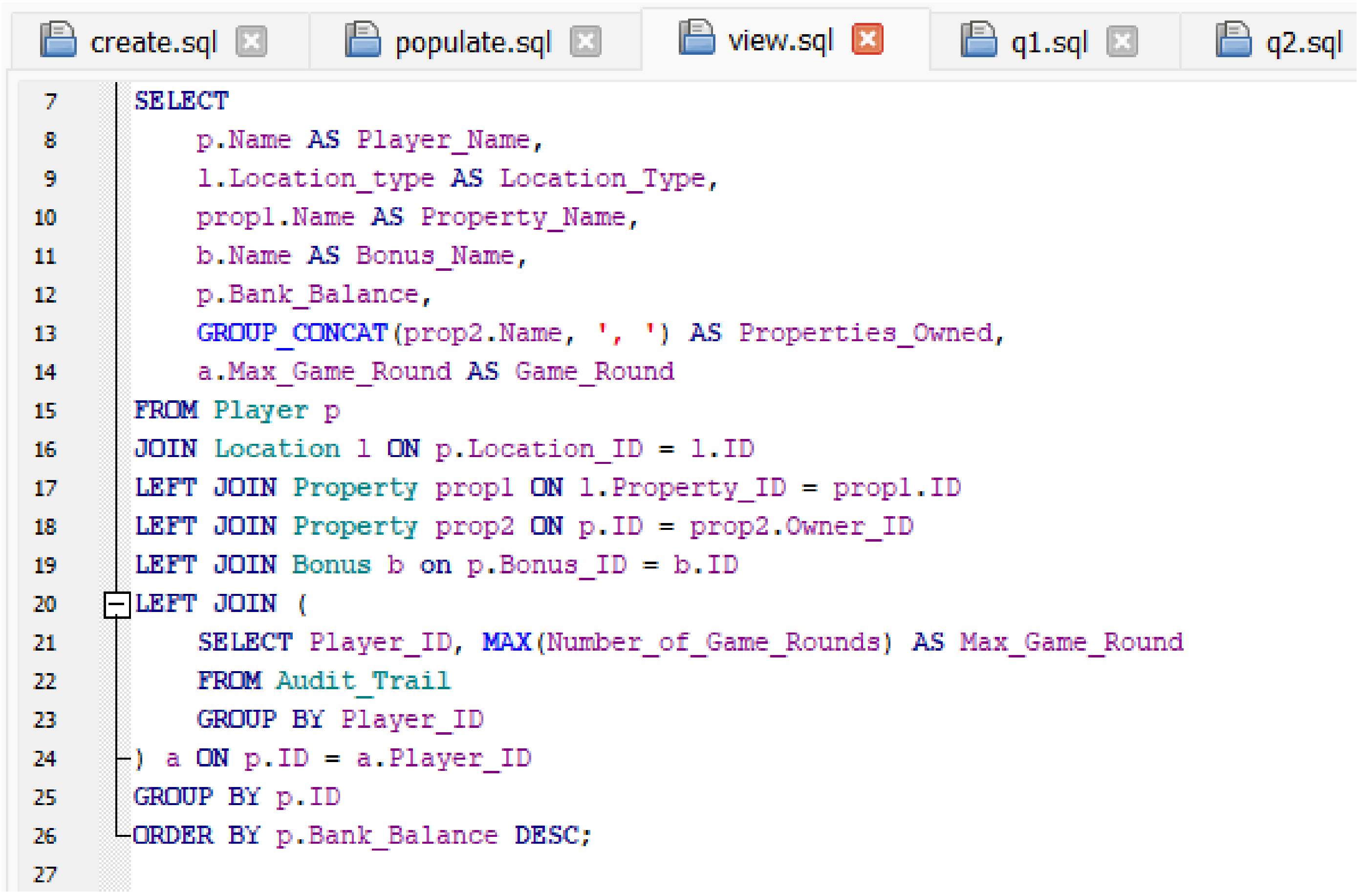
```
WHERE ID = 2;

UPDATE Property
SET Owner_ID = 3
WHERE ID = 4;

UPDATE Property
SET Owner_ID = 1
WHERE ID = 6;

UPDATE Property
SET Owner_ID = 2
WHERE ID = 7;
```

Please Note:

6. **The ‘UPDATE’ statements used in the above code is used for setting the ‘Owner_ID’ values in the property table.**
7. **This update is necessary because we have a CIRCULAR REFERENCE between player and property.**
8. **This is done to avoid referncial integrity constraints error in SQLITE**

Leaderboard gameView

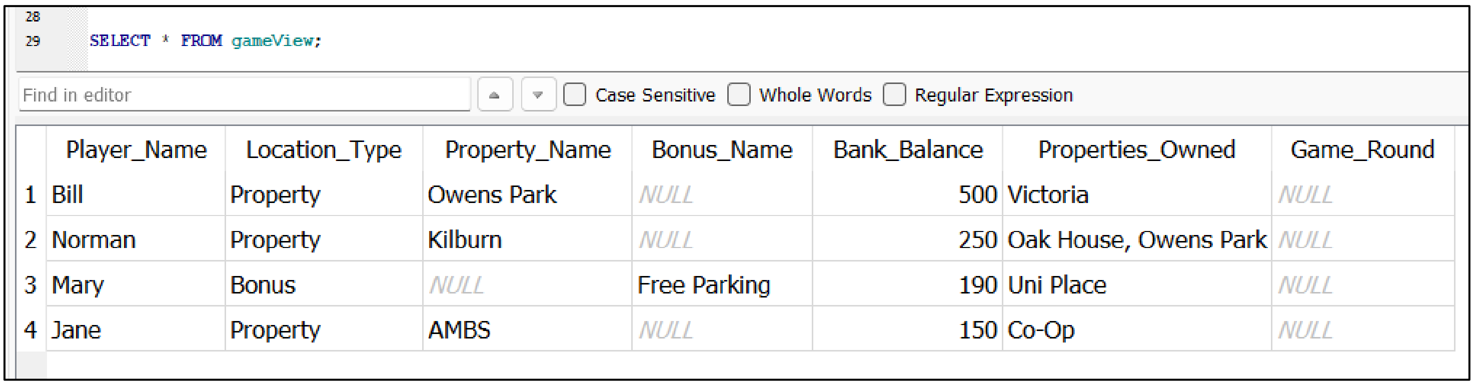
view.sql



```sql
SELECT
    p.Name AS Player_Name,
    l.Location_type AS Location_Type,
    prop1.Name AS Property_Name,
    b.Name AS Bonus_Name,
    p.Bank_Balance,
    GROUP_CONCAT(prop2.Name, ', ') AS Properties_Owned,
    a.Max_Game_Round AS Game_Round
FROM Player p
JOIN Location l ON p.Location_ID = l.ID
LEFT JOIN Property prop1 ON l.Property_ID = prop1.ID
LEFT JOIN Property prop2 ON p.ID = prop2.Owner_ID
LEFT JOIN Bonus b on p.Bonus_ID = b.ID
LEFT JOIN (
    SELECT Player_ID, MAX(Number_of_Game_Rounds) AS Max_Game_Round
    FROM Audit_Trail
    GROUP BY Player_ID
) a ON p.ID = a.Player_ID
GROUP BY p.ID
ORDER BY p.Bank_Balance DESC;
```

Initial State of the Leaderboard:

Select * gameView:



```sql
SELECT * FROM gameView;
```

	Player_Name	Location_Type	Property_Name	Bonus_Name	Bank_Balance	Properties_Owned	Game_Round
1	Bill	Property	Owens Park	NULL	500	Victoria	NULL
2	Norman	Property	Kilburn	NULL	250	Oak House, Owens Park	NULL
3	Mary	Bonus	NULL	Free Parking	190	Uni Place	NULL
4	Jane	Property	AMBS	NULL	150	Co-Op	NULL

All tables Initial State:

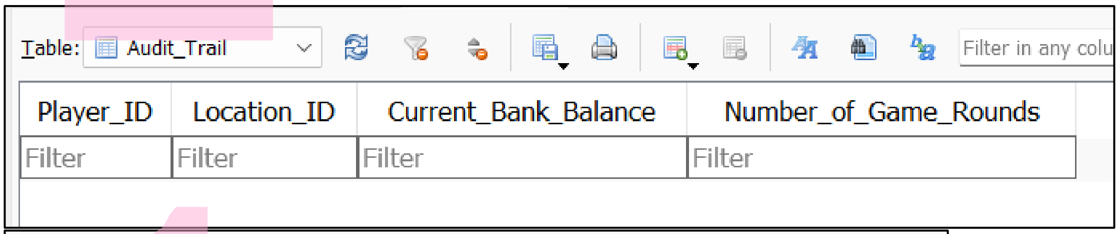


Table: Audit_Trail

Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
Filter	Filter	Filter	Filter

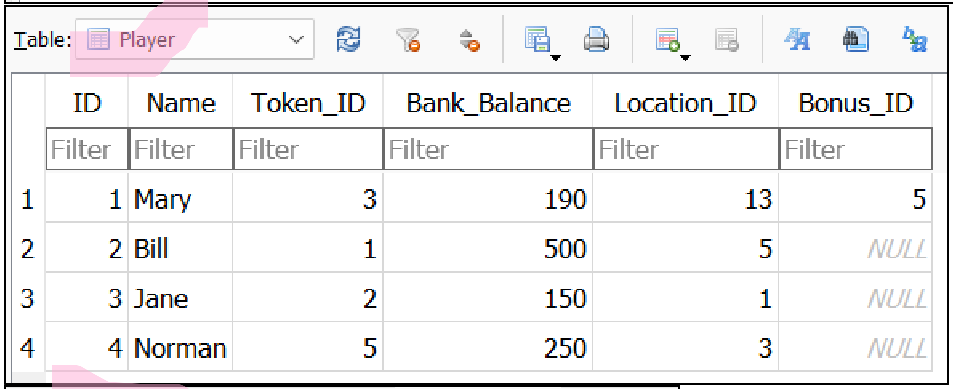


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Mary	3	190	13	5
2	2	Bill	1	500	5	NULL
3	3	Jane	2	150	1	NULL
4	4	Norman	5	250	3	NULL

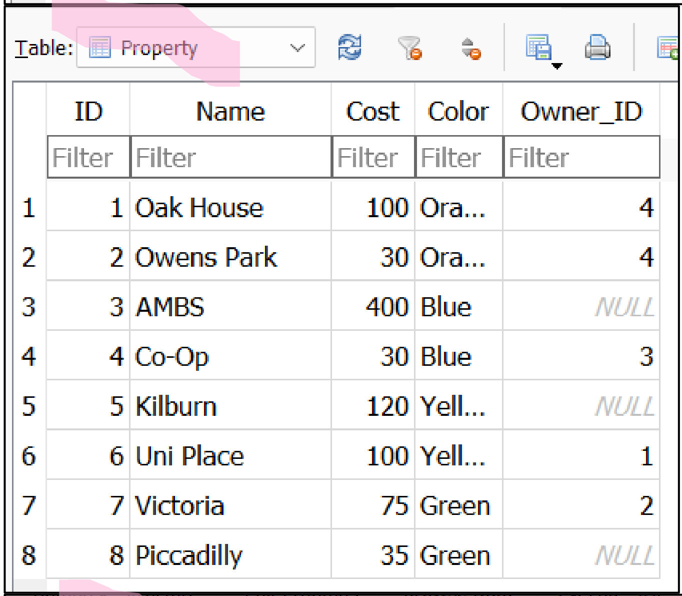


Table: Property

	ID	Name	Cost	Color	Owner_ID
	Filter	Filter	Filter	Filter	Filter
1	1	Oak House	100	Ora...	4
2	2	Owens Park	30	Ora...	4
3	3	AMBS	400	Blue	NULL
4	4	Co-Op	30	Blue	3
5	5	Kilburn	120	Yell...	NULL
6	6	Uni Place	100	Yell...	1
7	7	Victoria	75	Green	2
8	8	Piccadilly	35	Green	NULL

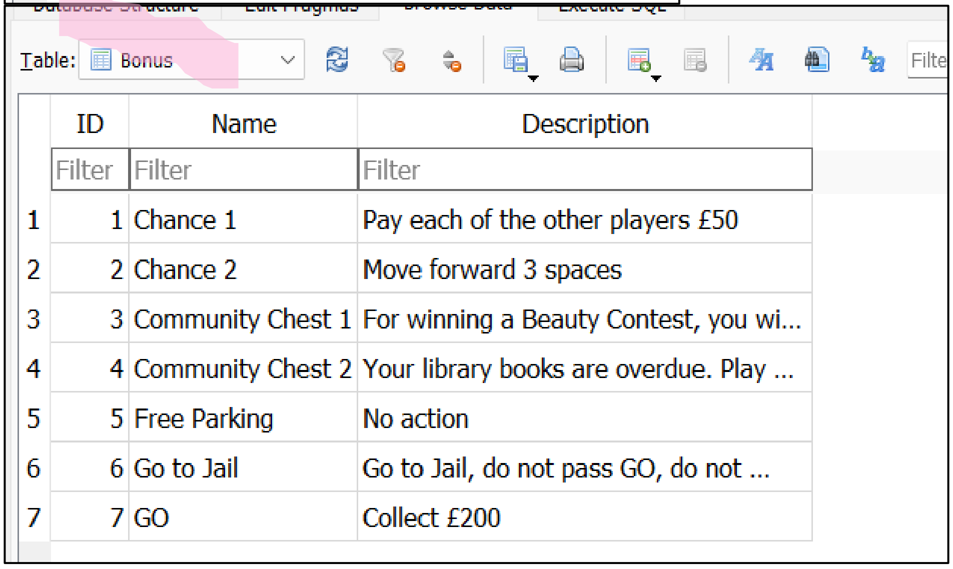


Table: Bonus

	ID	Name	Description
	Filter	Filter	Filter
1	1	Chance 1	Pay each of the other players £50
2	2	Chance 2	Move forward 3 spaces
3	3	Community Chest 1	For winning a Beauty Contest, you wi...
4	4	Community Chest 2	Your library books are overdue. Play ...
5	5	Free Parking	No action
6	6	Go to Jail	Go to Jail, do not pass GO, do not ...
7	7	GO	Collect £200

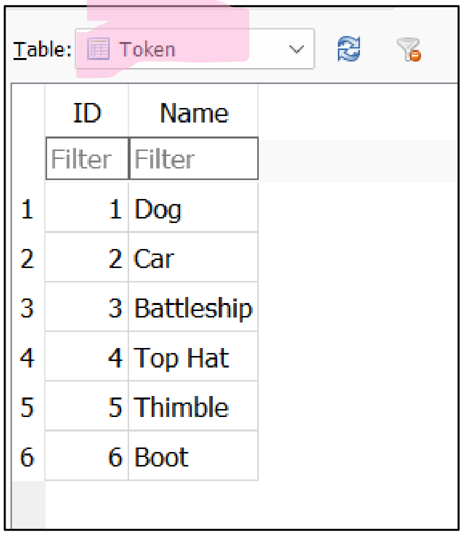


Table: Token

	ID	Name
	Filter	Filter
1	1	Dog
2	2	Car
3	3	Battleship
4	4	Top Hat
5	5	Thimble
6	6	Boot

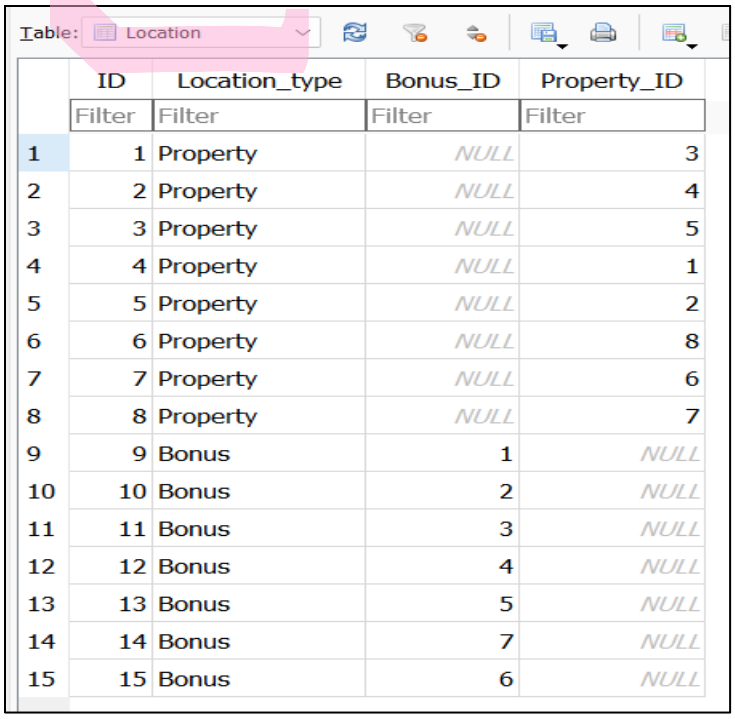


Table: Location

	ID	Location_type	Bonus_ID	Property_ID
	Filter	Filter	Filter	Filter
1	1	Property	NULL	3
2	2	Property	NULL	4
3	3	Property	NULL	5
4	4	Property	NULL	1
5	5	Property	NULL	2
6	6	Property	NULL	8
7	7	Property	NULL	6
8	8	Property	NULL	7
9	9	Bonus	1	NULL
10	10	Bonus	2	NULL
11	11	Bonus	3	NULL
12	12	Bonus	4	NULL
13	13	Bonus	5	NULL
14	14	Bonus	7	NULL
15	15	Bonus	6	NULL

QUERY 1

Jane rolls a 3: **She lands on Bonus location 'GO', therefore the player collects 200.**

Initialize Trigger to update the Audit_Trail table after every game turn in **ROUND 1**

```
DROP TRIGGER IF EXISTS UpdateAuditTrail;
CREATE TRIGGER UpdateAuditTrail
AFTER UPDATE OF Location_ID on Player
WHEN NEW.Location_ID != OLD.Location_ID OR NEW.Bank_Balance != OLD.Bank_Balance
BEGIN
    INSERT INTO Audit_Trail (Player_ID, Location_ID, Current_Bank_Balance,Number_of_Game_Rounds)
    VALUES (NEW.ID, NEW.Location_ID, NEW.Bank_Balance,1);
END;
```

Initialize Trigger Fires after player lands on Bonus -'GO' to update 200 bonus

```
DROP TRIGGER IF EXISTS UpdateBankBalanceBEFOREGO;
CREATE TRIGGER UpdateBankBalanceBEFOREGO
BEFORE UPDATE of Bonus_ID on Player
WHEN NEW.Bonus_ID = (SELECT ID FROM Bonus WHERE Name = 'GO')
BEGIN
    UPDATE Player
    SET Bank_Balance = Bank_Balance + 200
    WHERE ID = NEW.ID;
END;
```

- **Trigger:UpdateBankBalanceBEFOREGO** will update the player's balance based on 'GO' .
- Jane's location gets updated to 'GO',

```
UPDATE Player
SET
    Location_ID = (SELECT l.ID FROM Location l INNER JOIN Bonus b on l.Bonus_ID = b.ID WHERE b.Name = 'GO' ),
    Bonus_ID = (SELECT ID FROM Bonus WHERE Name='GO')
WHERE ID= 3;
```

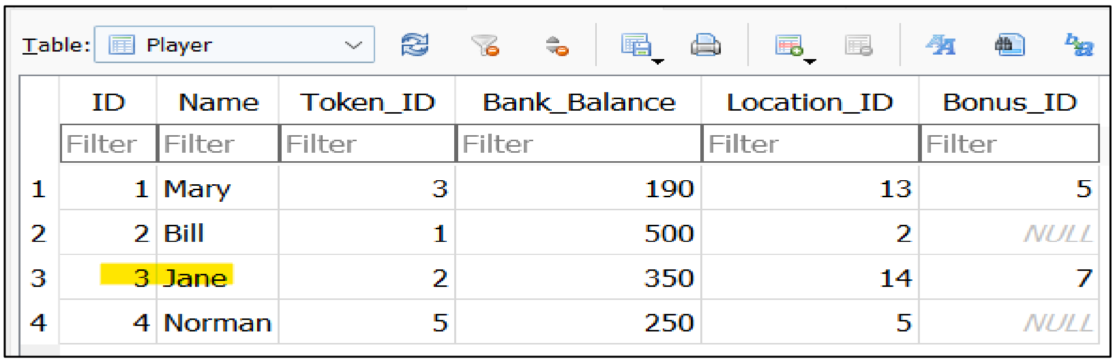


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Mary	3	190	13	5
2	2	Bill	1	500	2	NULL
3	3	Jane	2	350	14	7
4	4	Norman	5	250	5	NULL

Figure 1- Player Table after Jane rolls a 3.

Right after the update of Location_ID on the player, The **Trigger: UpdateAuditTrailTable** will fire off to add the current player's move.

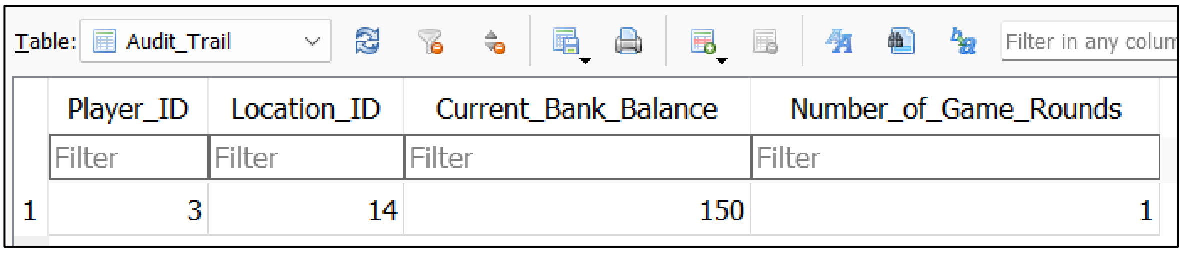


Table: Audit_Trail

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
	Filter	Filter	Filter	Filter
1	3	14	150	1

Figure 2- Audit_Trail Table after Jane roll

QUERY 2

Norman rolls a 1: **He lands on Bonus location 'Chance 1', therefore the player pays £50 to every other player.**

- Initialize **Trigger: UpdateBankBalanceBEFOREChance1,** this will update the bank balance of the player based on the Chance 1 description.

```
DROP TRIGGER IF EXISTS UpdateBankBalanceBEFOREChance1;

CREATE TRIGGER UpdateBankBalanceAFterChance1
BEFORE UPDATE of Bonus_ID on Player
WHEN NEW.Bonus_ID = (SELECT ID FROM Bonus WHERE Name = 'Chance 1')
BEGIN
    --Subtract from player 4
    UPDATE Player
    SET Bank_Balance = Bank_Balance - (50*((SELECT count(*) FROM Player) - 1))
    WHERE ID = NEW.ID;

    --Add to all other players
    UPDATE Player
    SET Bank_Balance = Bank_Balance + 50
    WHERE ID != NEW.ID;
END;
```

- Right before updating the Player table, Trigger: **UpdateBankBalanceBEFOREChance1** will fire and update the bank balance according to Chance 1 for Norman player.
- Now, We will update Norman's Location ID to Chance 1.

```
UPDATE Player
SET
    Location_ID = (SELECT l.ID FROM Location l INNER JOIN Bonus b on l.Bonus_ID = b.ID WHERE b.Name = 'Chance 1' ),
    Bonus_ID = (SELECT ID FROM Bonus WHERE Name='Chance 1')
WHERE
ID = 4;
```

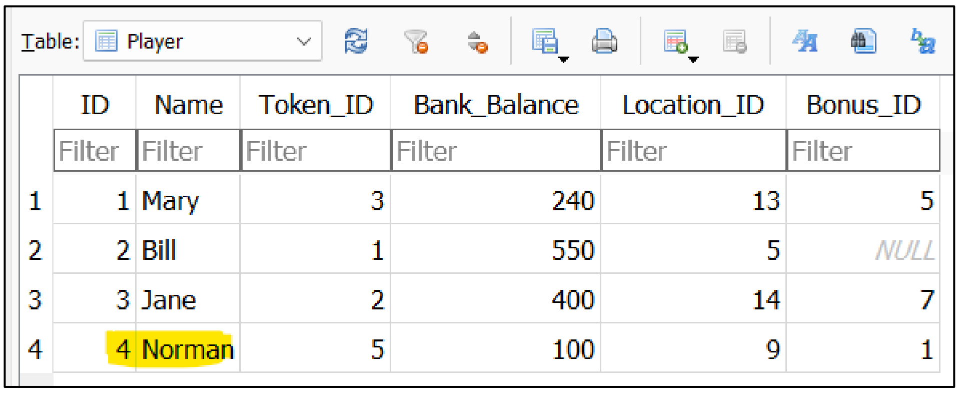


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Mary	3	240	13	5
2	2	Bill	1	550	5	NULL
3	3	Jane	2	400	14	7
4	4	Norman	5	100	9	1

Figure 3 Player table after Norman rolls a 1

- After update on Player table, **Trigger: UpdateAuditTrailTable will fire off.**

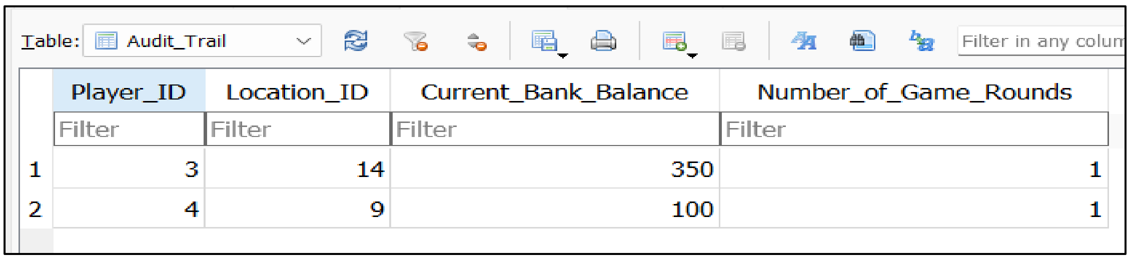


Table: Audit_Trail

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
	Filter	Filter	Filter	Filter
1	3	14	350	1
2	4	9	100	1

QUERY 3

Mary rolls a 4: **She lands on Bonus location 'Go To Jail', therefore the player directly goes to jail and will not come out until a 6 is rolled.**

Updating Mary's location ID and bonus ID

```
UPDATE Player
SET Location_ID = (SELECT l.ID FROM Location l INNER JOIN Bonus b on l.Bonus_ID = b.ID WHERE b.Name = "Go to Jail" ),
    Bonus_ID = (SELECT b.ID FROM Bonus b WHERE b.Name= "Go to Jail")
WHERE ID = 1;
```

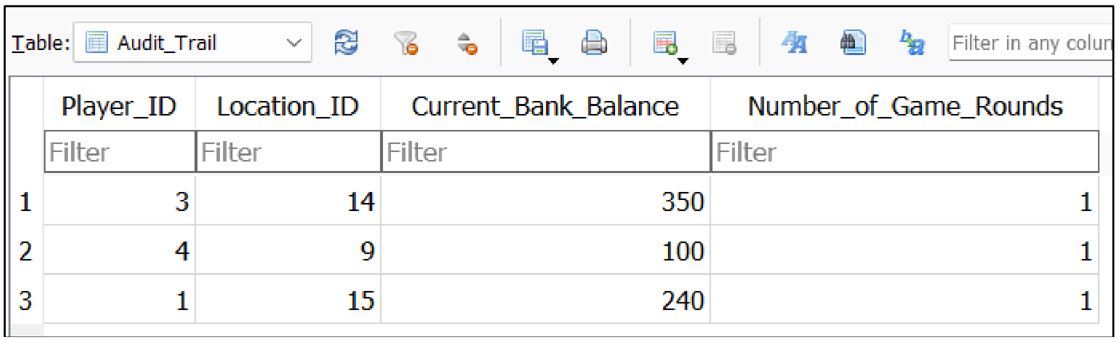


Table: Audit_Trail

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
1	3	14	350	1
2	4	9	100	1
3	1	15	240	1

Figure 4 Player table after Mary rolls a 4

➢ After update on Player table, **Trigger: UpdateAuditTrailTable** will fire off.

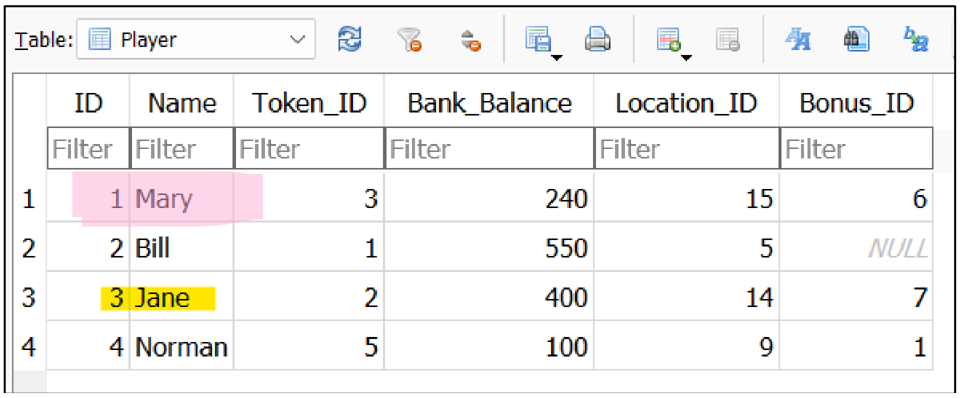


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
1	1	Mary	3	240	15	6
2	2	Bill	1	550	5	NULL
3	3	Jane	2	400	14	7
4	4	Norman	5	100	9	1

QUERY 4

Bill rolls a 2: **He lands on Property location 'AMBS' which is unowned, therefore the player will pay 400 to buy the property.**

CONDITION 1: UPDATE Bank_Balance of **Bill** by removing the rent amount (if AMBS is owned by someone and also check if the owner of AMBS owns all properties of that color or not, this will check double rent case as well

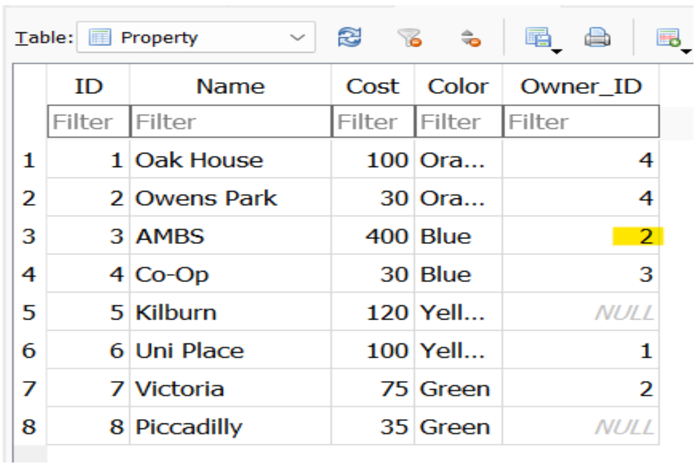
```
UPDATE Player -- Current player
SET
    Bank_Balance =
    CASE
        WHEN (Select count(*) from Property
              WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='AMBS')
              AND Color=( SELECT Color FROM Property WHERE name='AMBS'))
              =
             (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='AMBS'))
        THEN Bank_Balance - (SELECT Cost*2 FROM Property WHERE Name='AMBS')--Doubles rent
        ELSE Bank_Balance - (SELECT Cost FROM Property WHERE Name='AMBS')  -- This will also cover the case if
        owner_ID is null so it will take only rent
    END
WHERE ID = 2;
```

CONDITION 2: Update the **owner's** bank balance after receiving the rent (If an owner is existing for the property 'AMBS')

```
UPDATE Player – Owner updation
SET
    Bank_Balance =
    CASE
        WHEN (Select count(*) from Property
              WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='AMBS')
              AND Color=( SELECT Color FROM Property WHERE name='AMBS'))
              =
             (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='AMBS'))
        THEN Bank_Balance + (SELECT Cost*2 FROM Property WHERE Name='AMBS')--Doubles rent
        --WHEN Owner_ID is NULL
        --THEN Bank_Balance
        ELSE Bank_Balance + (SELECT Cost FROM Property WHERE Name='AMBS')
    END
WHERE ID = (SELECT Owner_ID FROM Property WHERE Name='AMBS');
```

CONDITION 3: If the property AMBS is unowned then make bill the owner of AMBS. → This is satisfied in as AMBS is unowned.

```
UPDATE Property
SET
    Owner_ID = 2
WHERE Name='AMBS' AND Owner_ID is NULL;
```



	ID	Name	Cost	Color	Owner_ID
	Filter	Filter	Filter	Filter	Filter
1	1	Oak House	100	Ora...	4
2	2	Owens Park	30	Ora...	4
3	3	AMBS	400	Blue	2
4	4	Co-Op	30	Blue	3
5	5	Kilburn	120	Yell...	NULL
6	6	Uni Place	100	Yell...	1
7	7	Victoria	75	Green	2
8	8	Piccadilly	35	Green	NULL

Figure 5Bill is now the ownwer of AMBS

- Update bills Location_ID and Bonus_ID to AMBS

```
UPDATE Player
SET
        Location_ID = (SELECT l.ID FROM Location l INNER JOIN Property p on l.Property_ID = p.ID WHERE p.Name = "AMBS" ),
        Bonus_ID = NULL
WHERE ID = 2;
```

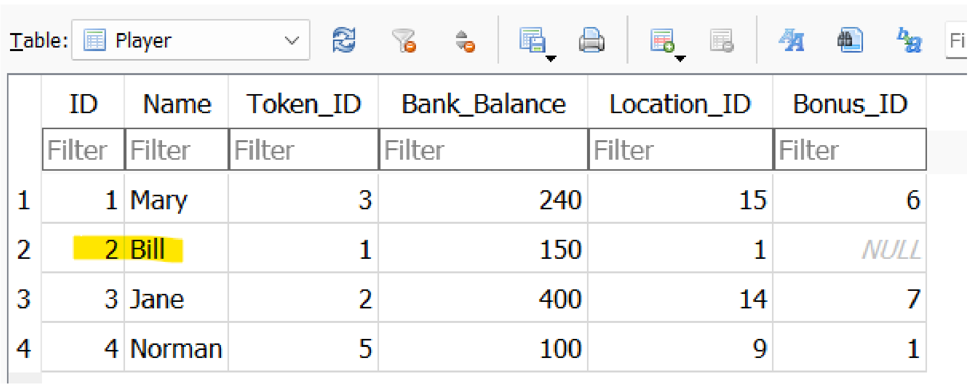


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
1	1	Mary	3	240	15	6
2	2	Bill	1	150	1	NULL
3	3	Jane	2	400	14	7
4	4	Norman	5	100	9	1

Figure 6 Player table after bill rolls a 2

- After update on Player table, **Trigger: UpdateAuditTrailTable** will fire off.

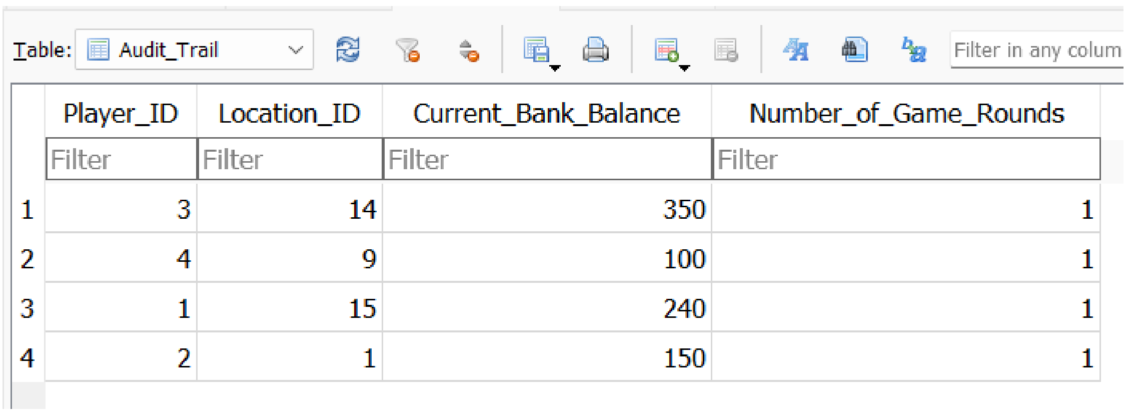
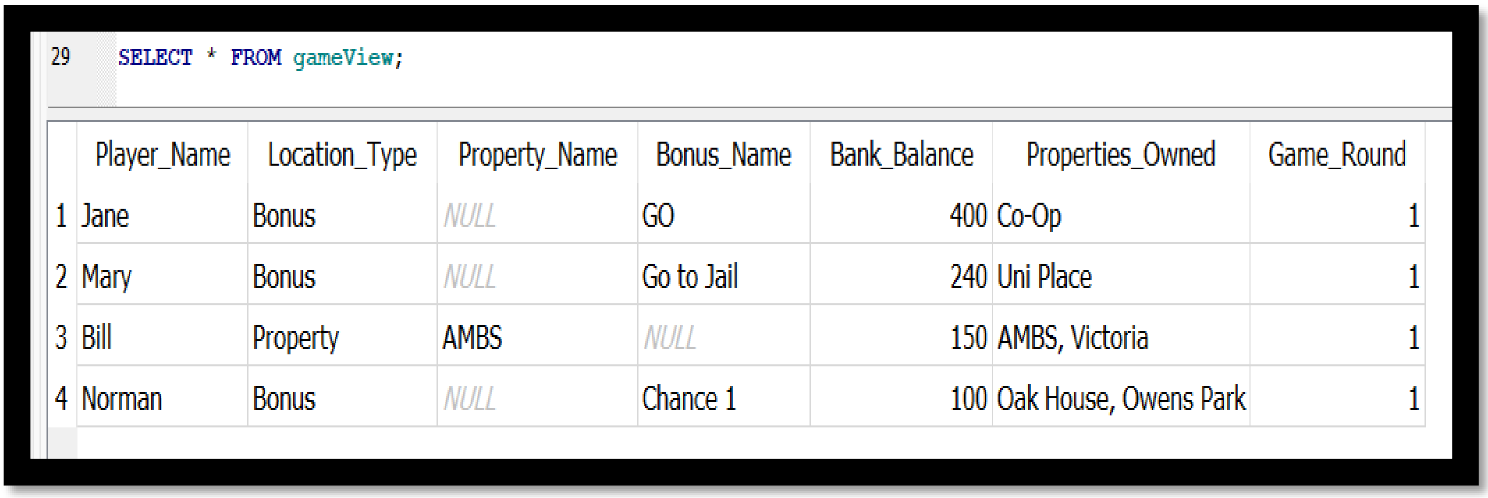


Table: Audit_Trail

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
1	3	14	350	1
2	4	9	100	1
3	1	15	240	1
4	2	1	150	1

Leaderboard View after Gameplay ROUND 1 (All 4 players have played)

```
SELECT * FROM gameView;
```



```
SELECT * FROM gameView;
```

	Player_Name	Location_Type	Property_Name	Bonus_Name	Bank_Balance	Properties_Owned	Game_Round
1	Jane	Bonus	NULL	GO	400	Co-Op	1
2	Mary	Bonus	NULL	Go to Jail	240	Uni Place	1
3	Bill	Property	AMBS	NULL	150	AMBS, Victoria	1
4	Norman	Bonus	NULL	Chance 1	100	Oak House, Owens Park	1

QUERY 5

Jane rolls a 5: **She lands on Property location 'Victoria'(Green) which is owned by Bill, therefore Player will pay 75 rent to Owner.**

(Double rent conditions failed as owner bill does not OWN all properties of green color).

➢ Initialize **Trigger: UpdateAuditTrail** to update the Audit_Trail table after every game turn in **ROUND 2**.

```
DROP TRIGGER IF EXISTS UpdateAuditTrail; --Drop the already existing Round 1 Trigger for Audit_Trail Table

CREATE TRIGGER UpdateAuditTrail
AFTER UPDATE OF Location_ID on Player
WHEN NEW.Location_ID != OLD.Location_ID OR NEW.Bank_Balance != OLD.Bank_Balance
BEGIN
        INSERT INTO Audit_Trail (Player_ID, Location_ID, Current_Bank_Balance, Number_of_Game_Rounds)
        VALUES (NEW.ID, NEW.Location_ID, NEW.Bank_Balance,2);
END;
```

→ This CASE is satisfied in as Victoria is Owned property and owner is Bill (Player_ID=2)

CONDITION 1: UPDATE Bank_Balance of **Jane** by removing the rent amount (if Victoria is owned by someone and also check if the owner of Victoria owns all properties of that color or not, this will check the double rent case as well

```
UPDATE Player -- Current player
SET
        Bank_Balance =
        CASE
                WHEN (Select count(*) from Property
                        WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='Victoria')
                        AND Color=( SELECT Color FROM Property WHERE name='Victoria'))
                        =
                       (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='Victoria'))
                THEN Bank_Balance - (SELECT Cost*2 FROM Property WHERE Name='Victoria') --Doubles rent
                ELSE Bank_Balance - (SELECT Cost FROM Property WHERE Name='Victoria') -- This will cover the case if
                owner_ID is NULL, then the current player will buy the property.
        END
WHERE ID = 3;
```

CONDITION 2: Update the **owner's** bank balance after receiving the rent (If an owner is existing for the property 'Victoria')

```
UPDATE Player – Owner updation
SET
        Bank_Balance =
        CASE
                WHEN (Select count(*) from Property
                        WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='Victoria')
                        AND Color=( SELECT Color FROM Property WHERE name='Victoria'))
                        =
                       (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='Victoria'))
                THEN Bank_Balance + (SELECT Cost*2 FROM Property WHERE Name='Victoria')--Doubles rent

                ELSE Bank_Balance + (SELECT Cost FROM Property WHERE Name='Victoria')
        END
WHERE ID = (SELECT Owner_ID FROM Property WHERE Name='Victoria'); -- only considers not null owner_ID
```

CONDITION 3: If the property Victoria is unowned then make Jane the owner of Victoria.

```
UPDATE Property
SET
        Owner_ID = 3
WHERE Name='Victoria' AND Owner_ID is NULL;
```

- Update Jane's Location_ID and Bonus_ID to Victoria.

```
UPDATE Player
SET
        Location_ID = (SELECT l.ID FROM Location l INNER JOIN Property p on l.Property_ID = p.ID WHERE p.Name = 'Victoria' ),
        Bonus_ID = NULL
WHERE ID = 3;
```

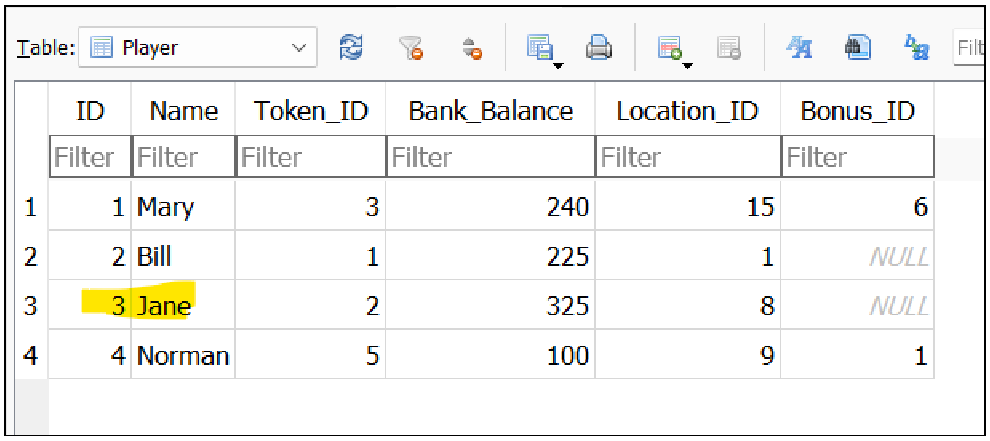


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Mary	3	240	15	6
2	2	Bill	1	225	1	NULL
3	3	Jane	2	325	8	NULL
4	4	Norman	5	100	9	1

Figure 7Player table after Jane rolls a 5

- After update on Player table, **Trigger: UpdateAuditTrailTable** will fire off.

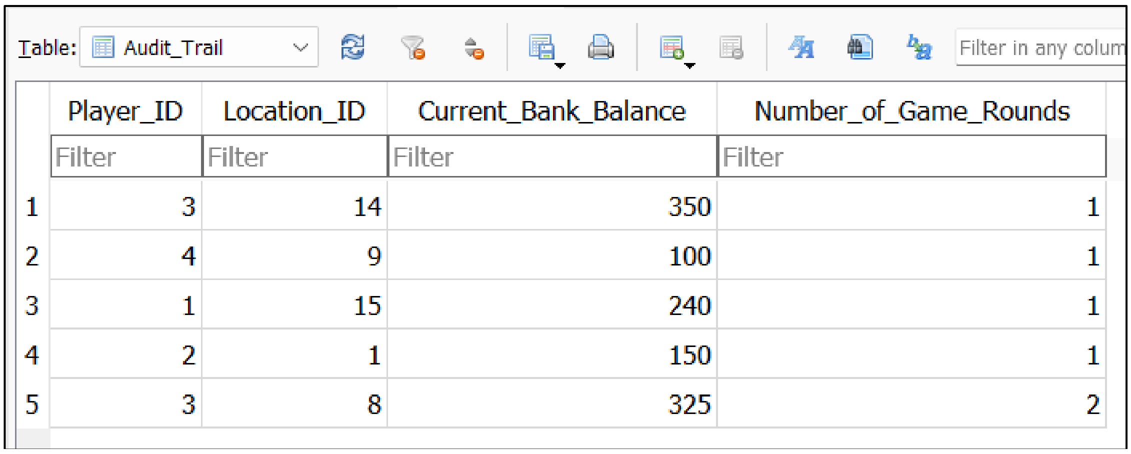


Table: Audit_Trail

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
	Filter	Filter	Filter	Filter
1	3	14	350	1
2	4	9	100	1
3	1	15	240	1
4	2	1	150	1
5	3	8	325	2

QUERY 6

Norman rolls a 4: **He lands on Bonus location 'Community Chest 1', therefore the player wins 100 for winning beauty contest.**

Initialise Trigger: UpdateBankBalanceBEFORECommunityChest1for community chest bonus update.

```
DROP TRIGGER IF EXISTS UpdateBankBalanceBEFORECommunityChest1;
CREATE TRIGGER UpdateBankBalanceBEFORECommunityChest1
BEFORE UPDATE of Bonus_ID on Player
WHEN NEW.Bonus_ID = (SELECT ID FROM Bonus WHERE Name = 'Community Chest 1')
BEGIN
        --Player wins 100 in beauty contest
        UPDATE Player
        SET Bank_Balance = Bank_Balance + 100
        WHERE ID = NEW.ID;
END;
```

Update Norman's (Player_ID= 4) Location ID and Bonus_ID to Community Chest 1

```
UPDATE Player
SET
        Location_ID = (SELECT l.ID FROM Location l INNER JOIN Bonus b on l.Bonus_ID = b.ID WHERE b.Name = 'Community
        Chest 1' ),
        Bonus_ID = (SELECT ID FROM Bonus WHERE Name='Community Chest 1')
WHERE
ID = 4;
```

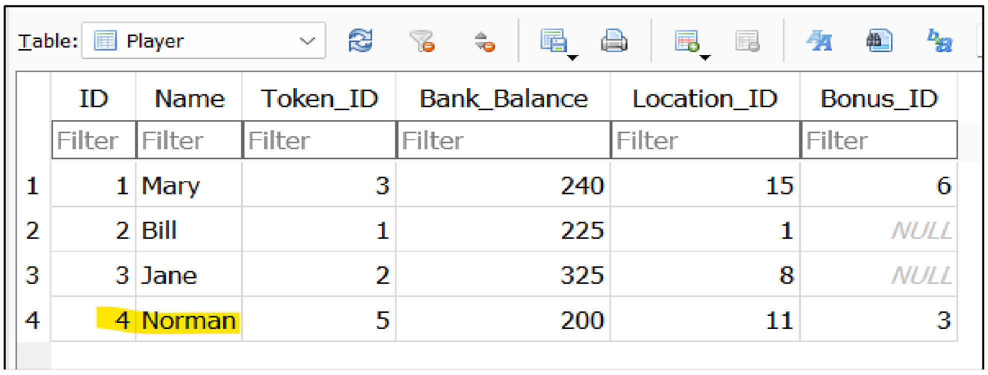


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Mary	3	240	15	6
2	2	Bill	1	225	1	NULL
3	3	Jane	2	325	8	NULL
4	4	Norman	5	200	11	3

Figure 8 Player table after Norman rolls a 4

➢ After update on Player table, **Trigger: UpdateAuditTrailTable** will fire off.

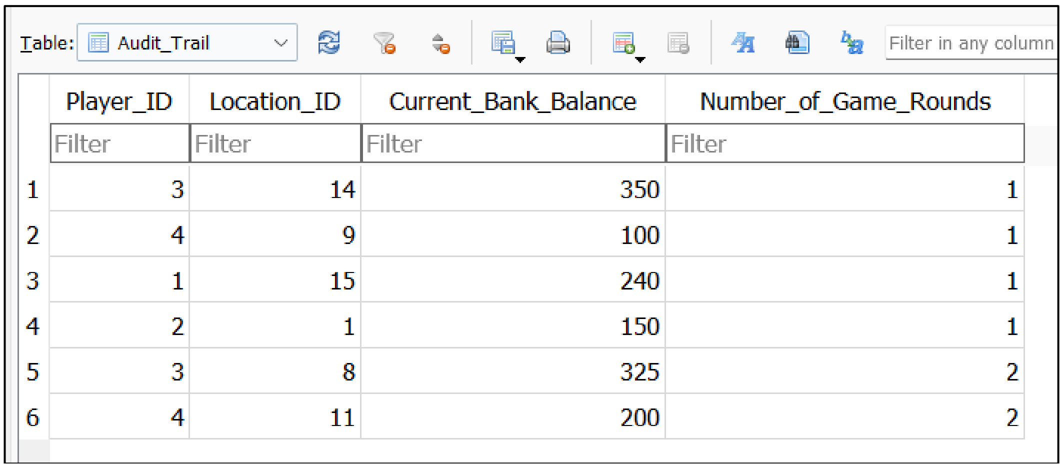


Table: Audit_Trail

Filter in any column

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
	Filter	Filter	Filter	Filter
1	3	14	350	1
2	4	9	100	1
3	1	15	240	1
4	2	1	150	1
5	3	8	325	2
6	4	11	200	2

Mary rolls a 6, and then a 5: **She lands on property location 'Oak House' (orange color) which is owned by Norman, therefore, Player Mary will pay double rent to Owner: Norman as Norman owns all properties of orange color which implies that the player will pay double rent to owner. Player Mary pays 100*2=200 to Norman.**

-- Mary moves 11 steps (no changes on roll 6 as she gets out of Jail)

CONDITION 1: UPDATE Bank_Balance of **Mary** by removing the rent amount (if Oak House is owned by someone and check if the owner of Oak House owns all properties of that color or not, this will check the double rent case as well)

```
UPDATE Player -- Current player
SET
        Bank_Balance =
        CASE
                WHEN (Select count(*) from Property
                        WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='Oak House')
                        AND Color=( SELECT Color FROM Property WHERE name='Oak House'))
                        =
                       (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='Oak House'))
                THEN Bank_Balance - (SELECT Cost*2 FROM Property WHERE Name='Oak House') --Doubles rent
                ELSE Bank_Balance - (SELECT Cost FROM Property WHERE Name='Oak House') -- This will cover the case if owner_ID is
                null so it will take only rent
        END
WHERE ID = 1;
```

CONDITION 2: Update the **owner's** bank balance after receiving the rent, If an owner exists for the property 'Oak House')

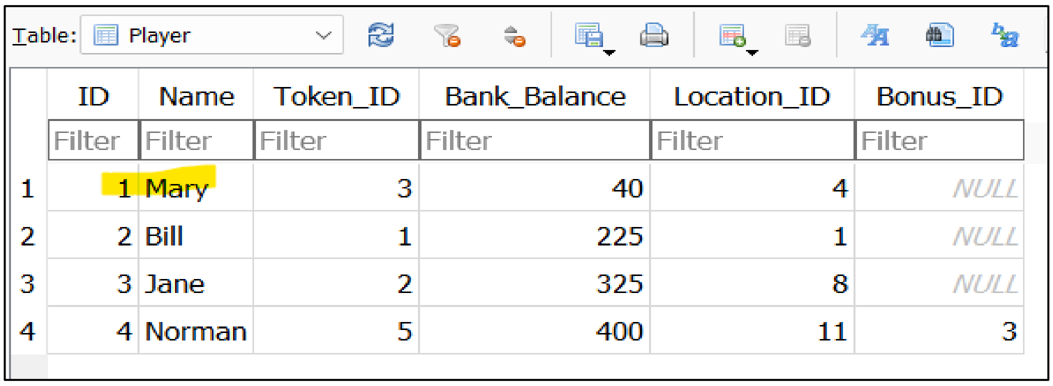
```
UPDATE Player -- Owner
SET
        Bank_Balance =
        CASE
                WHEN (Select count(*) from Property
                        WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='Oak House')
                        AND Color=( SELECT Color FROM Property WHERE name='Oak House'))
                        =
                       (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='Oak House'))
                THEN Bank_Balance + (SELECT Cost*2 FROM Property WHERE Name='Oak House')--Doubles rent

                ELSE Bank_Balance + (SELECT Cost FROM Property WHERE Name='Oak House')
        END
WHERE ID = (SELECT Owner_ID FROM Property WHERE Name='Oak House'); -- only considers not null ownwer ID
```

CONDITION 3: If the property Oak House is unowned then make Mary the owner of Victoria.

```
    UPDATE Property
    SET
            Owner_ID = 1
    WHERE Name='Oak House' AND Owner_ID is NULL;
```

- Update Jane's Location_ID and Bonus_ID to Oak House

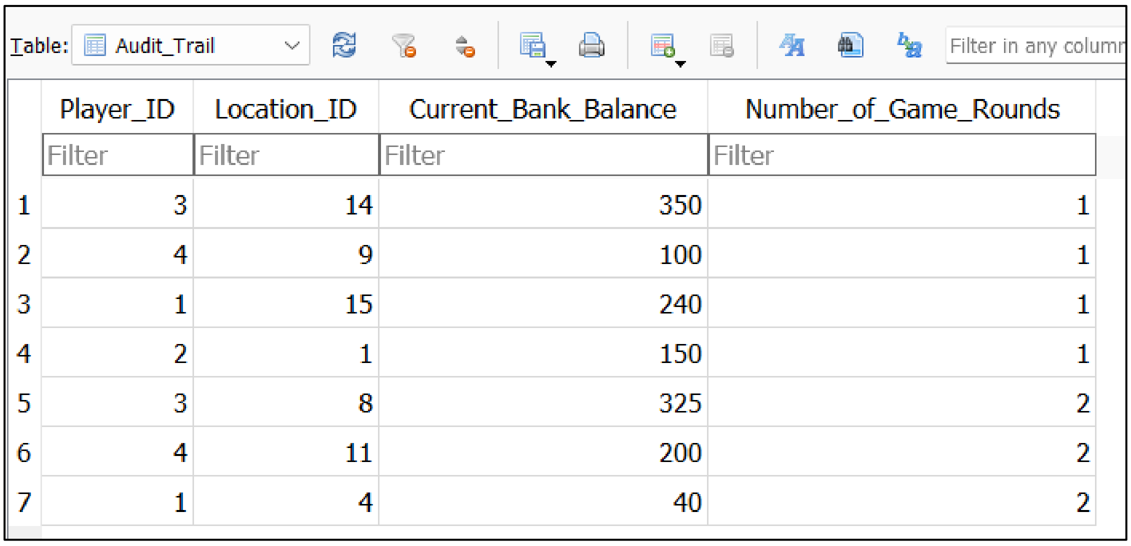
```
    UPDATE Player
    SET
          Location_ID = (SELECT l.ID FROM Location l INNER JOIN Property p on l.Property_ID = p.ID WHERE p.Name = 'Oak House' ),
          Bonus_ID = NULL
    WHERE ID = 1;
```



	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
1	1	Mary	3	40	4	NULL
2	2	Bill	1	225	1	NULL
3	3	Jane	2	325	8	NULL
4	4	Norman	5	400	11	3

Figure 9 Player table after Mary rolls a 6, and then a 5

➢ After update on Player table, **Trigger: UpdateAuditTrailTable** will fire off.



	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
1	3	14	350	1
2	4	9	100	1
3	1	15	240	1
4	2	1	150	1
5	3	8	325	2
6	4	11	200	2
7	1	4	40	2

QUERY 8

Bill rolls a 6, and then a 3: **He lands on Bonus location 'Community Chest 1', therefore the player wins 100 for winning beauty contest. Also, he will get 200 as he passes 'GO'.**

- Use existing **Trigger: UpdateBankBalanceBEFORECommunityChest1** for community chest 1 bonus update(+100)

Update Bill's bonus ID to Community Chest 1 -- **This will fire the trigger: UpdateBankBalanceBEFORECommunityChest1**
Adding 200 to the bank balance since the player passes GO

```
    UPDATE Player
    SET
        Location_ID = (SELECT l.ID FROM Location l INNER JOIN Bonus b on l.Bonus_ID = b.ID WHERE b.Name = 'Community Chest 1' ),
        Bonus_ID = (SELECT ID FROM Bonus WHERE Name='Community Chest 1'),
        Bank_Balance = Bank_Balance + 200
     WHERE ID = 2;
```

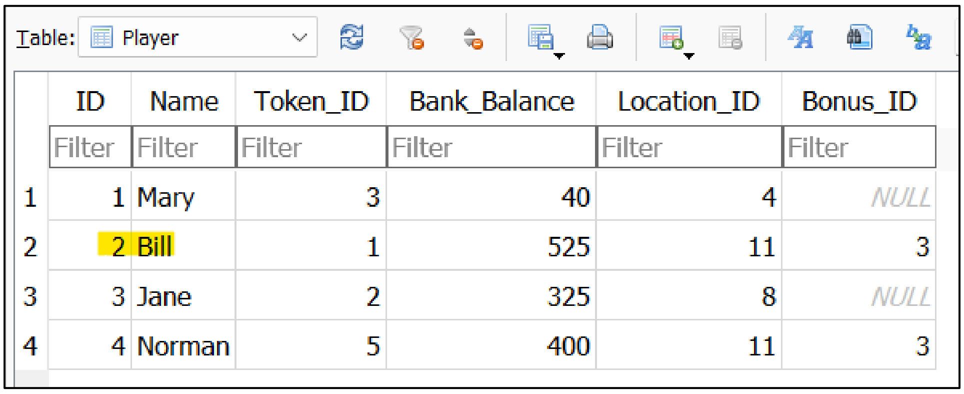


Table: Player

	ID	Name	Token_ID	Bank_Balance	Location_ID	Bonus_ID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Mary	3	40	4	NULL
2	2	Bill	1	525	11	3
3	3	Jane	2	325	8	NULL
4	4	Norman	5	400	11	3

Figure 10 Player table after Bill rolls a 6, and then a 3

- After update on Player table, **Trigger: UpdateAuditTrailTable** will fire off.

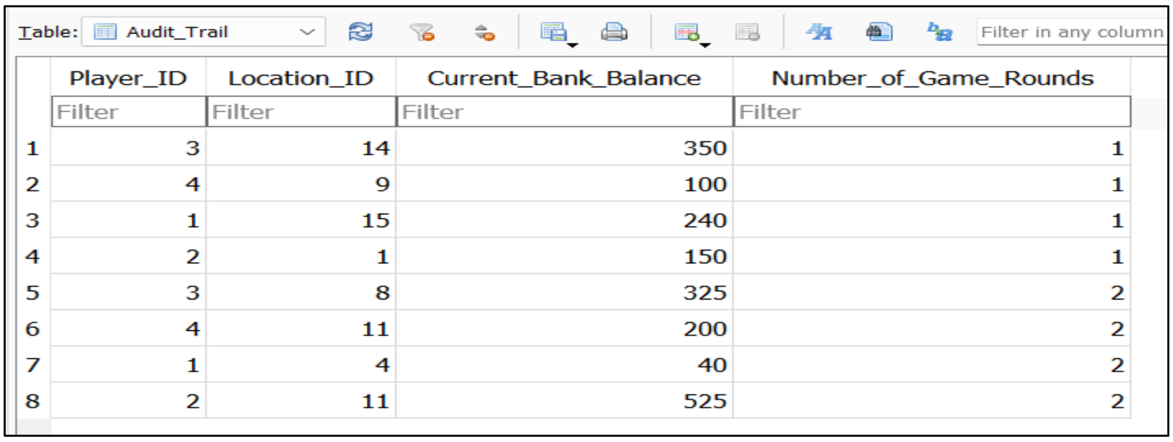


Table: Audit_Trail

Filter in any column

	Player_ID	Location_ID	Current_Bank_Balance	Number_of_Game_Rounds
	Filter	Filter	Filter	Filter
1	3	14	350	1
2	4	9	100	1
3	1	15	240	1
4	2	1	150	1
5	3	8	325	2
6	4	11	200	2
7	1	4	40	2
8	2	11	525	2

➢ Leaderboard View after Gameplay ROUND 2 (All 4 players have played)

SELECT * FROM gameView;

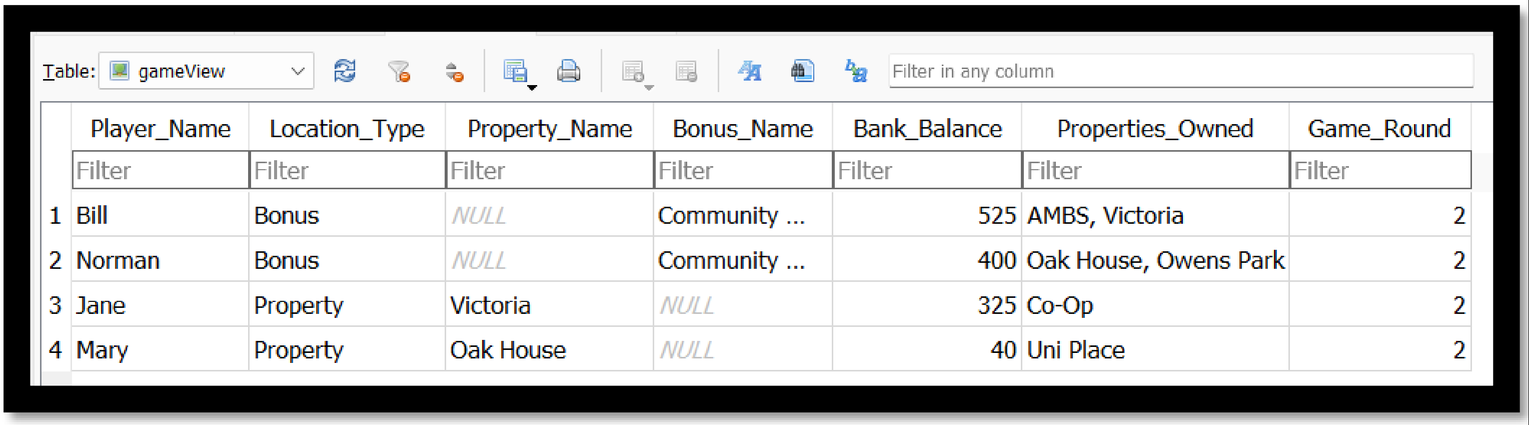


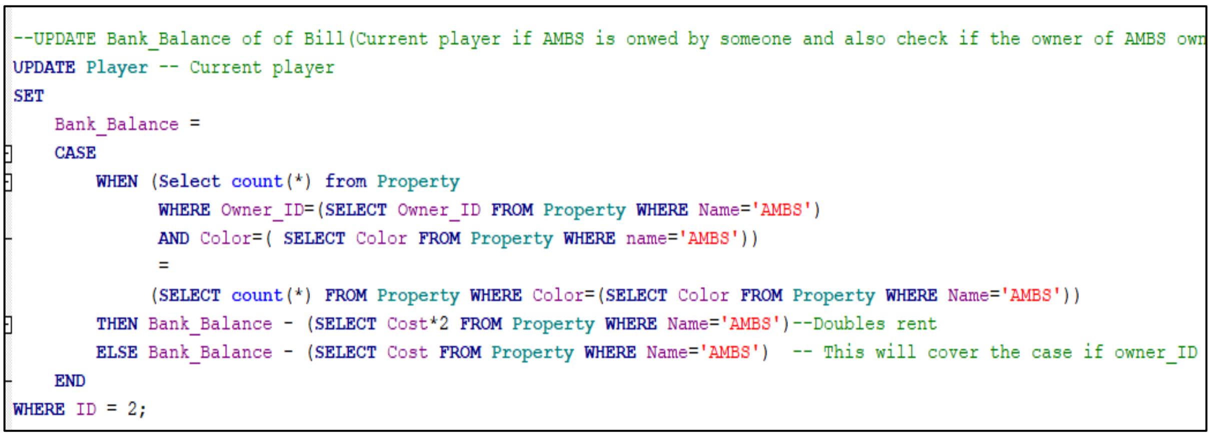
Table: gameView

Filter in any column

	Player_Name	Location_Type	Property_Name	Bonus_Name	Bank_Balance	Properties_Owned	Game_Round
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Bill	Bonus	NULL	Community ...	525	AMBS, Victoria	2
2	Norman	Bonus	NULL	Community ...	400	Oak House, Owens Park	2
3	Jane	Property	Victoria	NULL	325	Co-Op	2
4	Mary	Property	Oak House	NULL	40	Uni Place	2

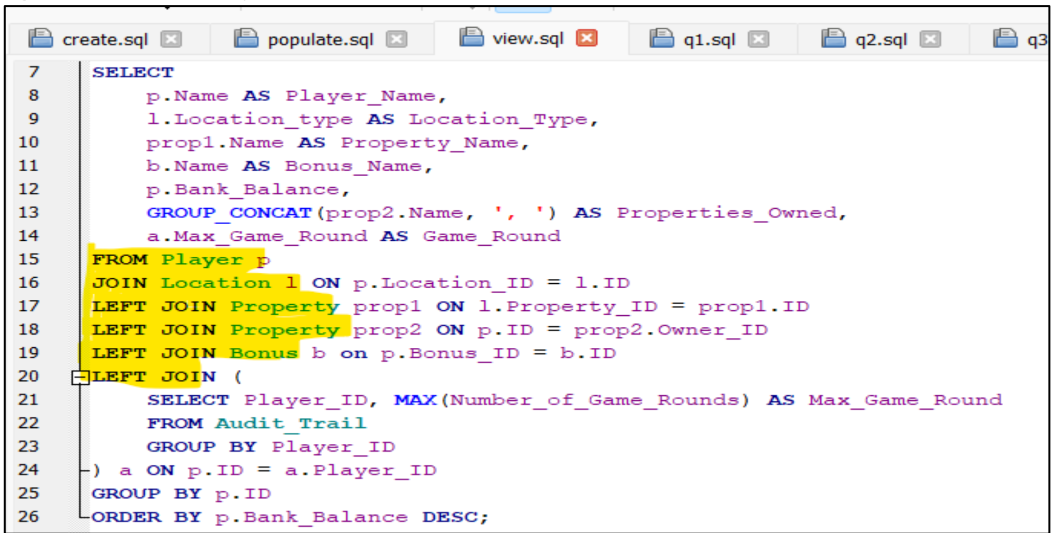
Key Features/Concepts used.

- **Data Definition Language (DDL) Commands:**
  - CREATE: Used to create the monopoly database tables and views
  - Constraints such as 'PRIMARY KEY', 'FOREIGN KEY' and 'UNIQUE' are applied to enforce data integrity.
- **Data Manipulation Language (DML) Commands:**
  - 'INSERT INTO': Inserting data into all 6 monopoly tables
  - 'UPDATE': Update the player attributes, audit_trail table after every turn
- **CASE – WHEN**
  - Applied conditional statements using CASE within UPDATE statements.
  - For eg) – In Query 5 : We determine the amount of rent and bank balance updates based on weather the owner of a property owns all properties of the same color.
- **Aggregate Functions –**
  - 'GROUP_CONCAT' – concatenate the names of properties owned by each player. This creates comma-seperated list.
  - 'MAX' – find the max of a column
  - Used in **view.sql**.
- **Nested Queries**
  - Subqueries or subselects are used to retrieve a specific data between tables
  - For eg) Used in Query 4 to extract the same color properties owned by a specific owner.



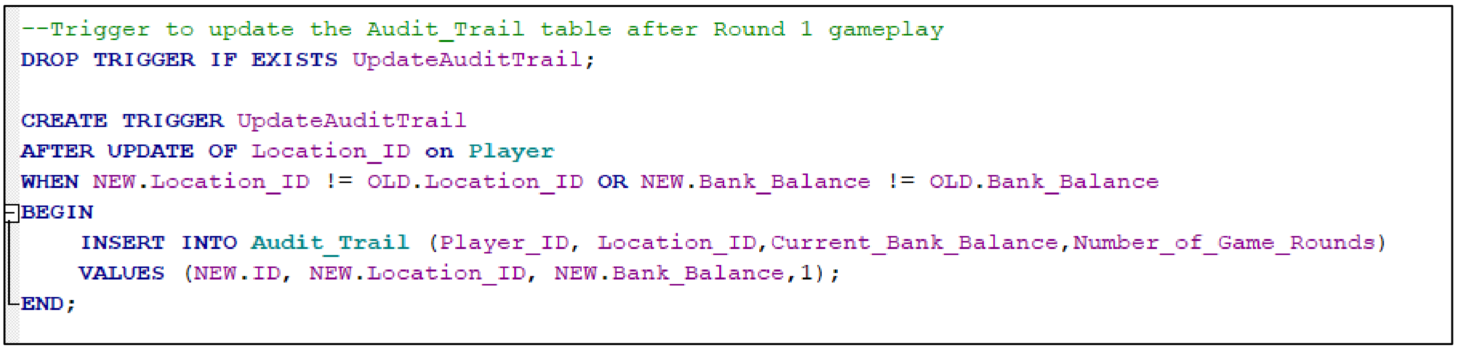
```sql
--UPDATE Bank_Balance of of Bill(Current player if AMBS is onwed by someone and also check if the owner of AMBS own
UPDATE Player -- Current player
SET
    Bank_Balance =
    CASE
        WHEN (Select count(*) from Property
              WHERE Owner_ID=(SELECT Owner_ID FROM Property WHERE Name='AMBS')
              AND Color=( SELECT Color FROM Property WHERE name='AMBS'))
              =
             (SELECT count(*) FROM Property WHERE Color=(SELECT Color FROM Property WHERE Name='AMBS'))
        THEN Bank_Balance - (SELECT Cost*2 FROM Property WHERE Name='AMBS')--Doubles rent
        ELSE Bank_Balance - (SELECT Cost FROM Property WHERE Name='AMBS')  -- This will cover the case if owner_ID
    END
WHERE ID = 2;
```

- **Joins**
  - Used single joins and multiple joins to improve the query efficiency and extract the reslts
  - Eg) used in view.sql



```sql
SELECT
    p.Name AS Player_Name,
    l.Location_type AS Location_Type,
    prop1.Name AS Property_Name,
    b.Name AS Bonus_Name,
    p.Bank_Balance,
    GROUP_CONCAT(prop2.Name, ', ') AS Properties_Owned,
    a.Max_Game_Round AS Game_Round
FROM Player p
JOIN Location l ON p.Location_ID = l.ID
LEFT JOIN Property prop1 ON l.Property_ID = prop1.ID
LEFT JOIN Property prop2 ON p.ID = prop2.Owner_ID
LEFT JOIN Bonus b on p.Bonus_ID = b.ID
LEFT JOIN (
    SELECT Player_ID, MAX(Number_of_Game_Rounds) AS Max_Game_Round
    FROM Audit_Trail
    GROUP BY Player_ID
) a ON p.ID = a.Player_ID
GROUP BY p.ID
ORDER BY p.Bank_Balance DESC;
```

- **Triggers**
  - Triggers are defined to automate some action in response to a specific event.
  - 4 Triggers are used:
    1. UpdateAuditTrail
    2. UpdateBankBalanceBEFOREGO
    3. UpdateBankBalanceBEFOREChance1
    4. UpdateBankBalanceBEFORECCommunityChest1



```sql
--Trigger to update the Audit_Trail table after Round 1 gameplay
DROP TRIGGER IF EXISTS UpdateAuditTrail;

CREATE TRIGGER UpdateAuditTrail
AFTER UPDATE OF Location_ID on Player
WHEN NEW.Location_ID != OLD.Location_ID OR NEW.Bank_Balance != OLD.Bank_Balance
BEGIN
    INSERT INTO Audit_Trail (Player_ID, Location_ID,Current_Bank_Balance,Number_of_Game_Rounds)
    VALUES (NEW.ID, NEW.Location_ID, NEW.Bank_Balance,1);
END;
```

- **Views:**
  - 'CREATE VIEW' statement is used to define 'gameView' view to create a virtual table for leaderboard representation.
- **Column Alias** – tables are used by an alias name and columns are accessed by the alias.
  - 'p.Name' is aliased as Player_Name in view.sql
- **Grouping and ordering :**
  - 'GROUP BY' – In gameView, It groups the results of the player's ID, to ensure that the player's information is displayed only once in VIEW.
  - 'ORDER BY' – In gameView, The bank_balance of the players are orderd in descending order to show the leaderboard.

Future Enhancements

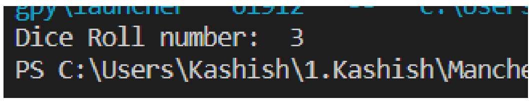
The current game using SQLITE was made using as many automations as possible. There are a few suggestions which can make this game even more dynamic and automated by integrating PYTHON.

Please check some of my ideas:

1. **Automating DICE ROLL:**
   - We can establish a front-end page to play this game , where the die roll can be automated using the following logic:

```
➢ import random
➢
➢ # This function will use the random module in python and generate a random
  # number between 1,2,3,4,5,6
➢ def  diceRoll():
➢     return random.randint(1,6)
➢ #display the diceRoll result
➢ diceRoll = diceRoll()
➢ print("Dice Roll number: ", diceRoll)
```

   - **Result:**



```
Dice Roll number:  3
PS C:\Users\Kashish\1.Kashish\Manche
```

2. **Storing the players in a dictionary containing their current position**

```
➢ # Player Name and its starting position
➢ players = {
➢     "Mary":0,
➢     "Bill":0,
➢     "Jane":0,
➢     "Norman":0
➢ }
```

3.Storing the Board Game in a List

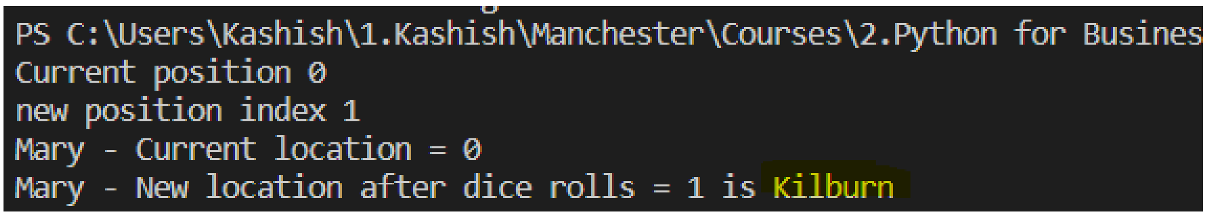
```
➢ # Monopoly board List in clockwise order
➢ board = [
➢ "GO", "Kilburn", "Chance 1", "Uni Place", "In Jail", "Victoria",
➢ "Community Chest 1", "Piccadilly", "Free Parking", "Oak House",
➢ "Chance 2", "Owens Park", "Go To Jail", "AMBS", "Community Chest 2",
➢ "Co-op"
➢ ]
```

4. Getting the Player location after Dice Roll:

```
#Simulating a player's turn
def simulate_player_turn(name):
    current_pos = players.get(name)
    print("Current position",current_pos)
    # calculating New position Index
    new_pos_index = (current_pos +diceRoll)%len(board)
```

```python
    print("new position index",new_pos_index)

    # calculating New Position Value
    new_pos = board[new_pos_index]
    players.update({name: new_pos_index})

    #Updating player's position (using new_pos)
    print("%s - Current location = %s"%(name,current_pos))
    print("%s - New location after dice rolls = %s is %s"%(name,diceRoll,new_pos))

simulate_player_turn("Mary")
```

Output showing the new player location:



```
PS C:\Users\Kashish\1.Kashish\Manchester\Courses\2.Python for Busines
Current position 0
new position index 1
Mary - Current location = 0
Mary - New location after dice rolls = 1 is Kilburn
```

CONCLUSION

The Monopoly championship taking place in Havana was completed using SQLITE in DB Browser.

The database design was created and explained clearly with ER diagram using Crow's Foot Notation and deriving the relational database schema with accurate constraints for the entities. The redundancy was handled with normalisation. The gameplay was simulated using SQLITE queries for all 8 gameplay turns and the view of the leaderboard was also clearly defined to extract the current state of the game at any point. Key concepts were used while simulating the gameplay and many rules were automated using Triggers, which in turn reduced the code redundancy.

Some future enhancements are also suggested which will lead to complete automation of the gameplay using a graphical user interface by integrating the frontend (Using python) and backend (using SQLITE).

NAME: KASHISH KHARYAL

University ID: 11356488