2024 Monopoly World Championships in Havana

MANCHESTER 1824
The University of Manchester

DATA 70141 - Understanding Databases
Assignment-1

Abhishek Kumar
11358614

CONTENTS

OBJECTIVE

Model the gameplay of a simplified version of Monopoly using a relational database and SQL queries.

Database and the queries must be compatible with SQLite.

Tools used:

1. **SQLite**
2. **DB Browser**

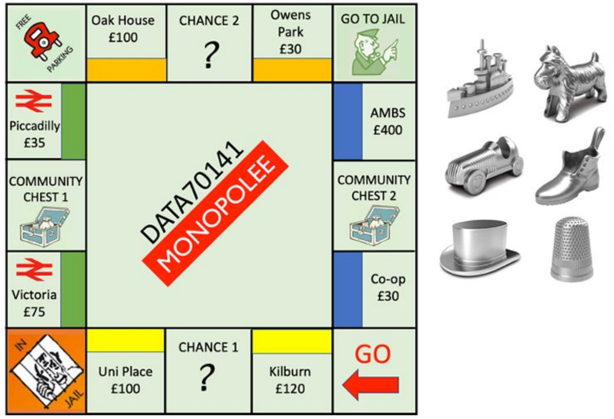
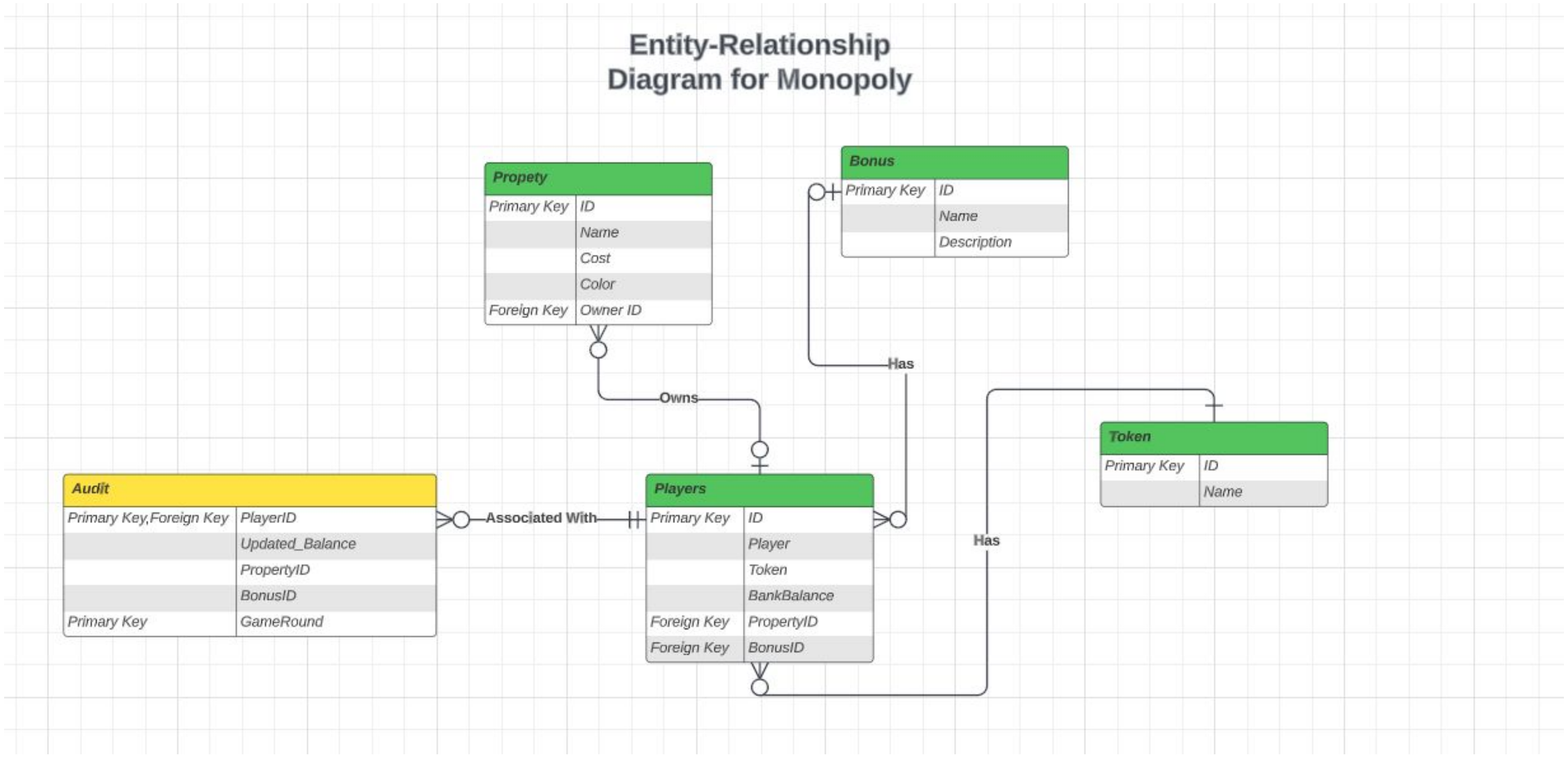


Figure 1: The Monopolee Board

ER Diagram (Crow's Foot Notation)



Propety

Primary Key	ID
	Name
	Cost
	Color
Foreign Key	Owner ID

Bonus

Primary Key	ID
	Name
	Description

Token

Primary Key	ID
	Name

Audit

Primary Key,Foreign Key	PlayerID
	Updated_Balance
	PropertyID
	BonusID
Primary Key	GameRound

Players

Primary Key	ID
	Player
	Token
	BankBalance
Foreign Key	PropertyID
Foreign Key	BonusID

ER Diagram Design Choices

MANCHESTER 1824
The University of Manchester

There are strong and weak entities identified for the ER model along with their relationships.

Strong entities:

1. **Token** - Identifies the token used by each player. Every player needs to have a distinct token.
   Relationship: zero-or-many relationship with **player**.

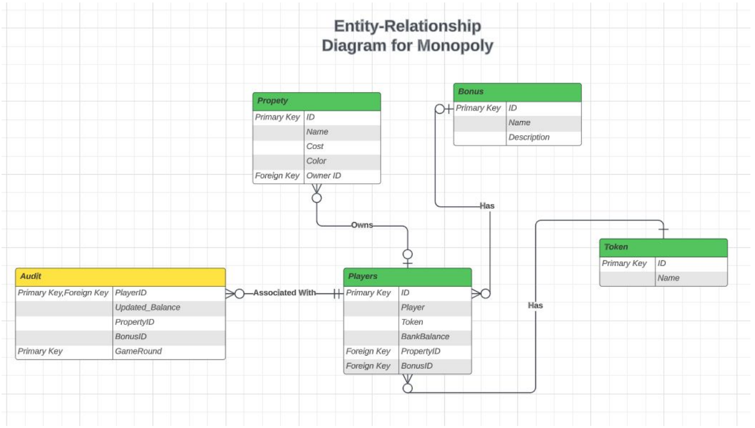
2. **Property** - Every property can have one owner at most.
   Relationship: **zero-or-one** relationship with **player**.

3. **Bonus** - Entity represents special locations on the board excluding properties. Every player can use the same bonus.
   Relationship: **zero-or-many** relationship with **Player**.

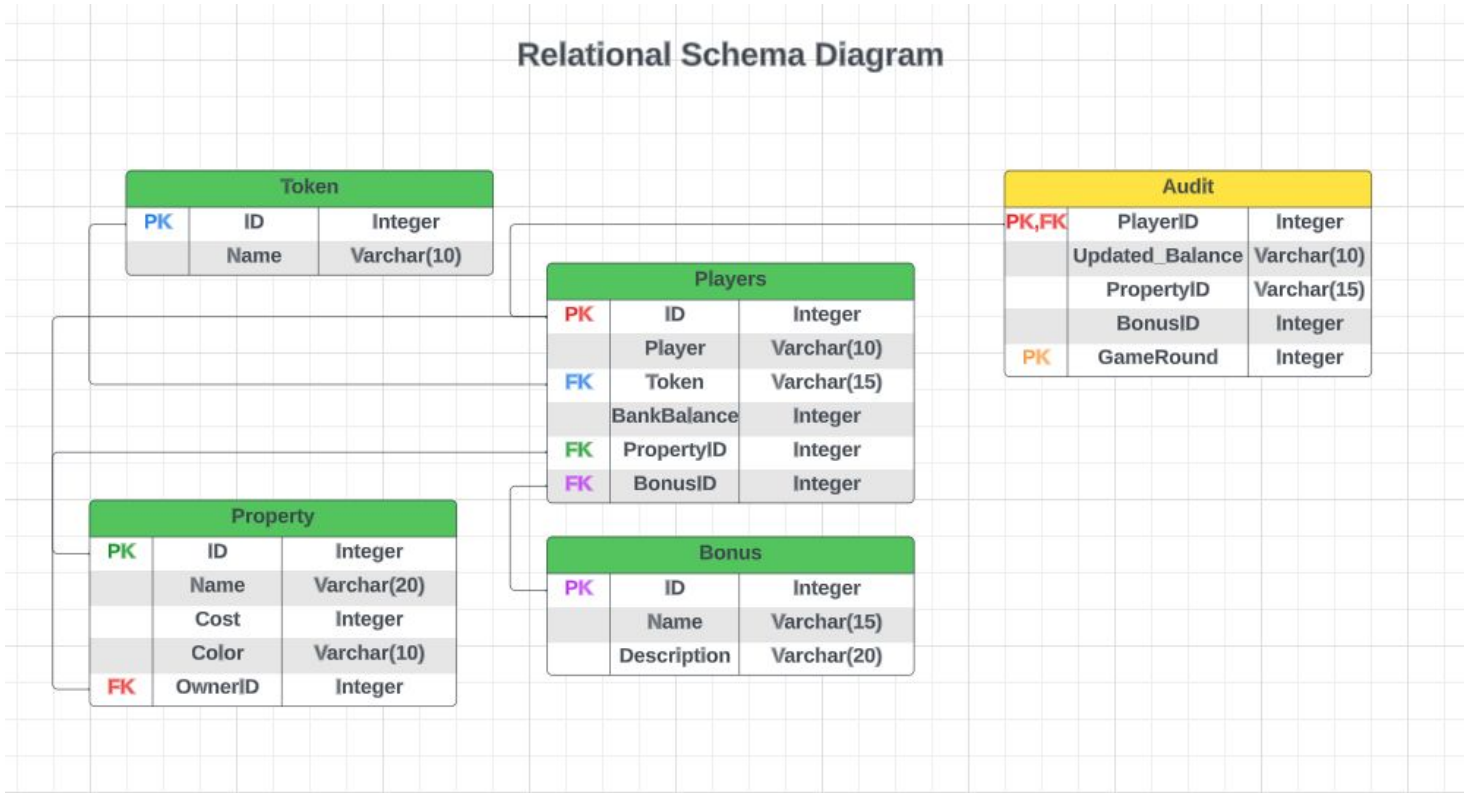
4. **Player** - This entity represents the players participating in the game. Every player must have one token. Also, every player can own zero or multiple properties. For bonus, every player can have at most one bonus at any given time in the game.
   Relationship: **one-and-only-one** relationship with **token**.
   zero-or-many relationship with **property**.
   zero-or-one relationship with **bonus**.
   zero-or-many relationship with **audit**.

Weak entities:

1. **Audit** - This entity records the state of every player throughout the game. Every record from audit cannot have multiple players since it represents the state of a player at any given time in the game.
   Relationship: **one-and-only-one** relationship with **player**.



Relational Schema



Relational Schema Diagram

Token

PK	ID	Integer
	Name	Varchar(10)

Players

PK	ID	Integer
	Player	Varchar(10)
FK	Token	Varchar(15)
	BankBalance	Integer
FK	PropertyID	Integer
FK	BonusID	Integer

Audit

PK,FK	PlayerID	Integer
	Updated_Balance	Varchar(10)
	PropertyID	Varchar(15)
	BonusID	Integer
PK	GameRound	Integer

Property

PK	ID	Integer
	Name	Varchar(20)
	Cost	Integer
	Color	Varchar(10)
FK	OwnerID	Integer

Bonus

PK	ID	Integer
	Name	Varchar(15)
	Description	Varchar(20)

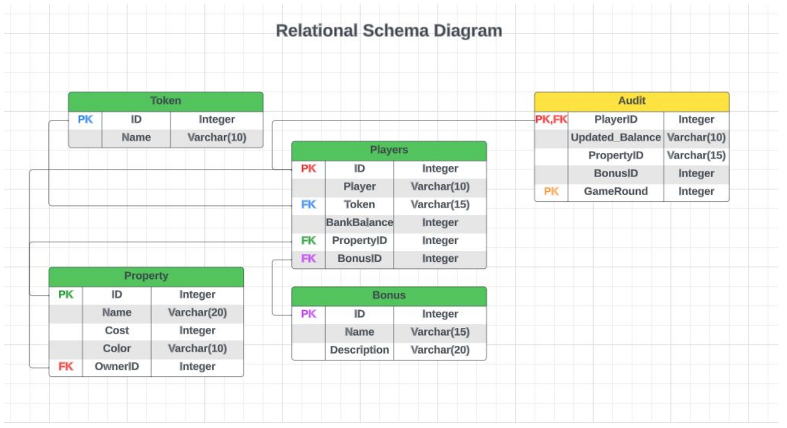
Relational Schema-Entity and Constraints

1. <u>**Token Table**</u> -
   Attributes - ID,Name
   Primary Key - ID
   UNIQUE and **NOT NULL** constraints - Name

2. <u>**Property Table**</u> -
   Attributes - ID,Name,Cost,Color,OwnerID
   Primary Key - ID
   Foreign Key - **OwnerID** References **Player(ID)**
   Unique Constraints - Name
   Not Null Constraints - ID,Name

3. <u>**Bonus Table**</u> -
   Attributes - ID,Name,Description
   Primary Key - ID
   Unique Constraint - Name
   Not Null Constraint - ID,Name

4. <u>**Player Table**</u> -
   Attributes -
   ID,Player,Token,BankBalance,PropertyID,BonusID
   Primary Key - ID
   Foreign Key - **BonusID** References **Bonus(ID)**
   PropertyID References **Property(ID)**
   Token References **Token(ID)**
   Unique Constraints - Player,Token
   Not Null Constraint - ID,Player,Token



Relational Schema Diagram

Token		
PK	ID	Integer
	Name	Varchar(10)

Players		
PK	ID	Integer
	Player	Varchar(10)
FK	Token	Varchar(15)
	BankBalance	Integer
FK	PropertyID	Integer
FK	BonusID	Integer

Audit		
PK,FK	PlayerID	Integer
	Updated_Balance	Varchar(10)
	PropertyID	Varchar(15)
	BonusID	Integer
PK	GameRound	Integer

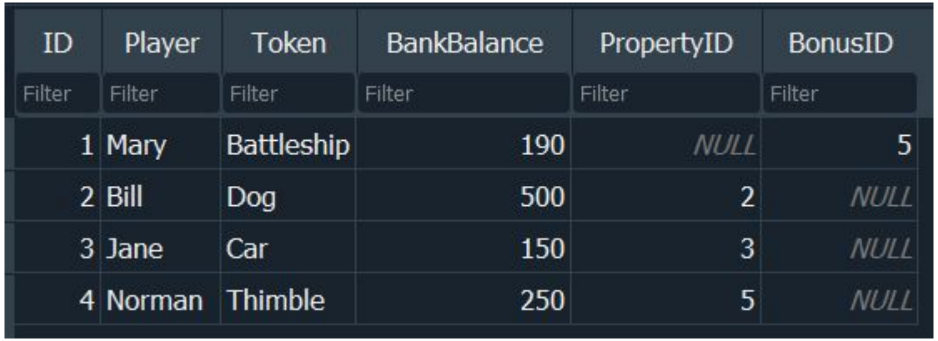
Property		
PK	ID	Integer
	Name	Varchar(20)
	Cost	Integer
	Color	Varchar(10)
FK	OwnerID	Integer

Bonus		
PK	ID	Integer
	Name	Varchar(15)
	Description	Varchar(20)

5. <u>**Audit Table**</u> -
   Attributes -
   PlayerID,Updated_Balance,PropertyID,BounusID,GameRound
   Candidate Keys - PlayerID, GameRound
   Foreign Key - **PlayerID** References **Players(ID)**
   Not Null Constraints- PlayerID,GameRound

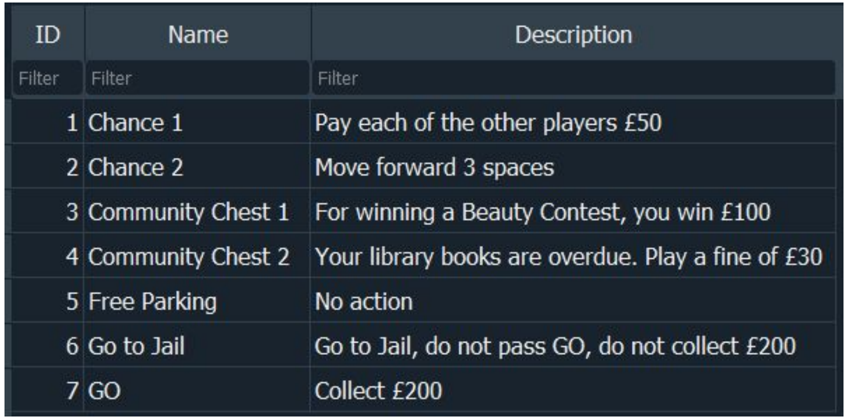
Implementation— Initial State of the Game

Figure 1 : Players table



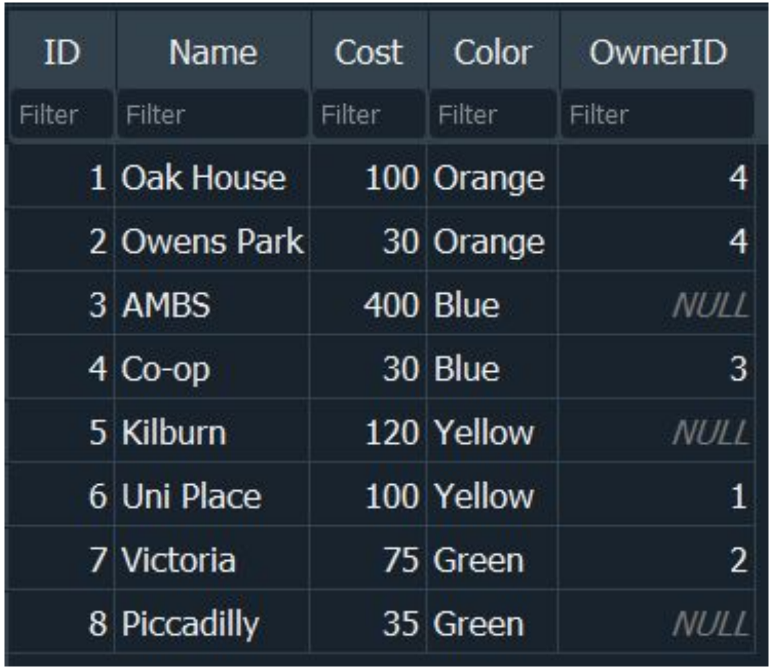
ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	190	NULL	5
2	Bill	Dog	500	2	NULL
3	Jane	Car	150	3	NULL
4	Norman	Thimble	250	5	NULL

Figure 2 : Bonus table



ID	Name	Description
Filter	Filter	Filter
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

Figure 3 : Property table



ID	Name	Cost	Color	OwnerID
Filter	Filter	Filter	Filter	Filter
1	Oak House	100	Orange	4
2	Owens Park	30	Orange	4
3	AMBS	400	Blue	NULL
4	Co-op	30	Blue	3
5	Kilburn	120	Yellow	NULL
6	Uni Place	100	Yellow	1
7	Victoria	75	Green	2
8	Piccadilly	35	Green	NULL

Implementation— Rules of the game

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.

Implementation— Gameplay Rounds

Gameplay Round 1:

G1 Jane rolls a 3

G2 Norman rolls a 1

G3 Mary rolls a 4

G4 Bill rolls a 2

Gameplay Round 2:

G5 Jane rolls a 5

G6 Norman rolls a 4

G7 Mary rolls a 6, and then a 5

G8 Bill rolls a 6, and then a 3

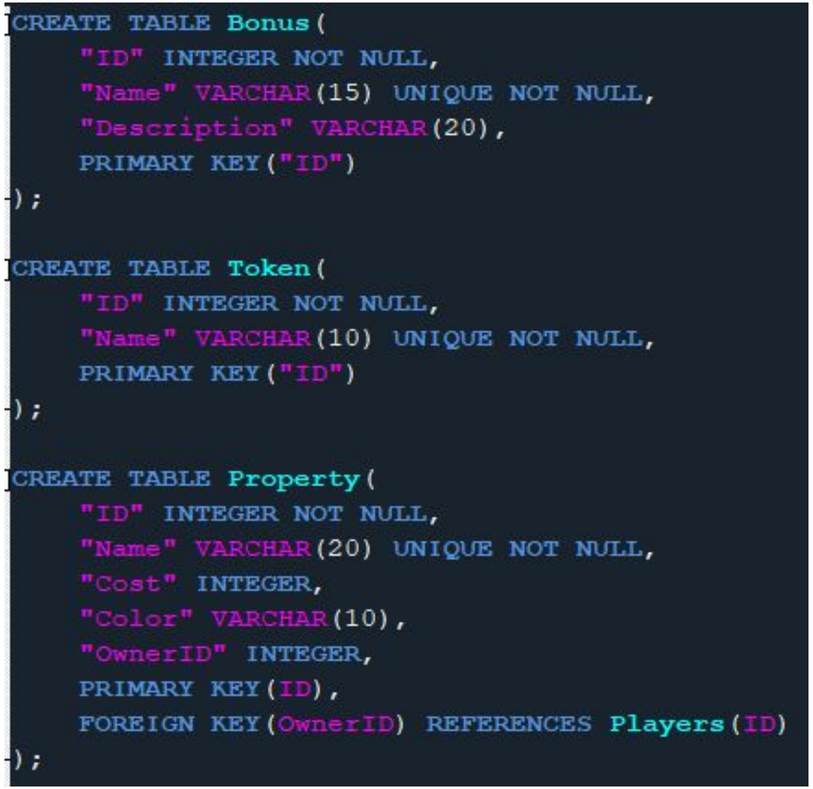
Assumptions

Following assumptions were made before playing the game:

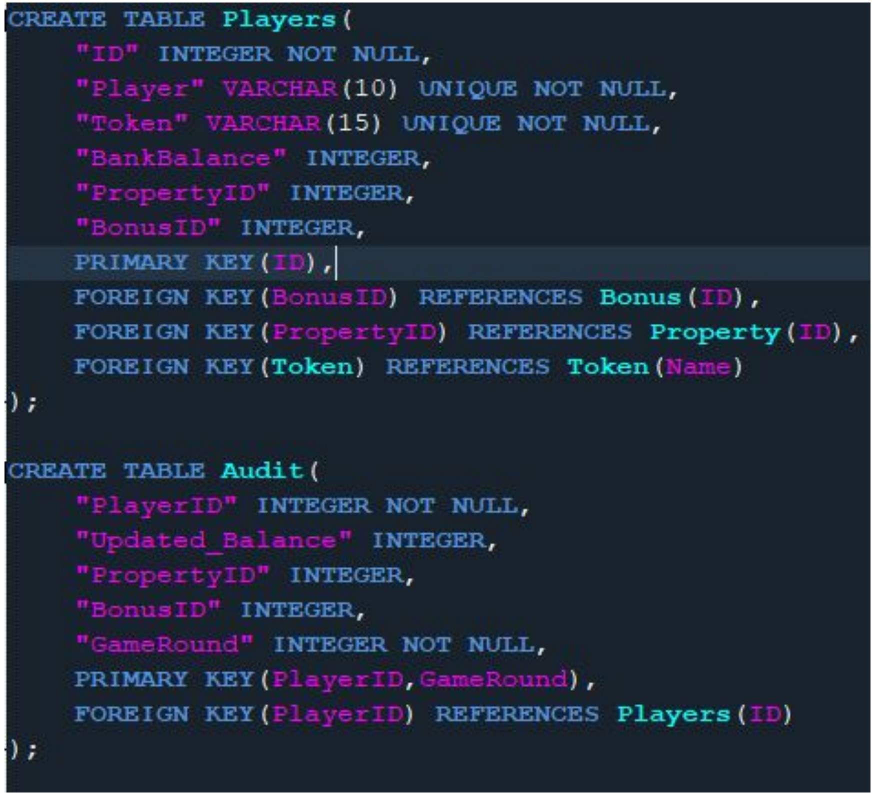
1. We are considering a virtual bank which is giving a credit of £200 whenever a player passed ‘GO’.
2. The dice roll is not automated. We already know the position after the roll where the player is landing considering the monopoly board given in the question.
3. In the initial state, Jane (P3) starts from AMBS and she hasn’t bought the unowned property.
4. 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.

create.sql

File contains queries required to successfully create the tables, with the correct constraints.



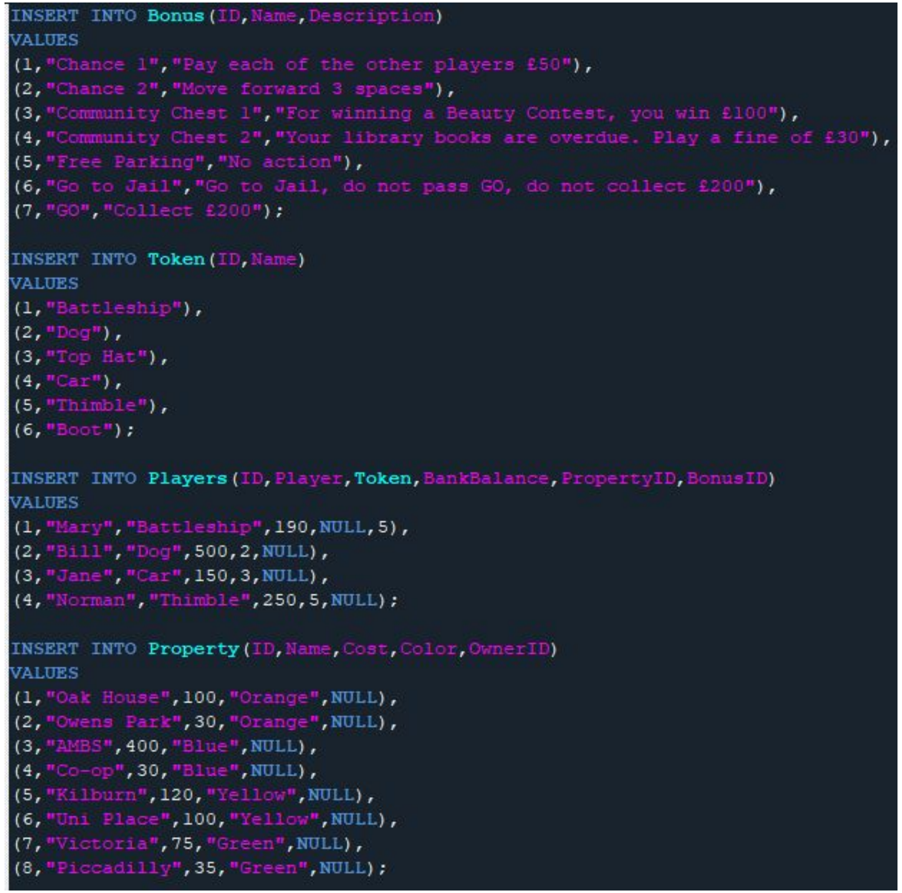
```
CREATE TABLE Bonus(
    "ID" INTEGER NOT NULL,
    "Name" VARCHAR(15) UNIQUE NOT NULL,
    "Description" VARCHAR(20),
    PRIMARY KEY("ID")
);

CREATE TABLE Token(
    "ID" INTEGER NOT NULL,
    "Name" VARCHAR(10) UNIQUE NOT NULL,
    PRIMARY KEY("ID")
);

CREATE TABLE Property(
    "ID" INTEGER NOT NULL,
    "Name" VARCHAR(20) UNIQUE NOT NULL,
    "Cost" INTEGER,
    "Color" VARCHAR(10),
    "OwnerID" INTEGER,
    PRIMARY KEY(ID),
    FOREIGN KEY(OwnerID) REFERENCES Players(ID)
);
```



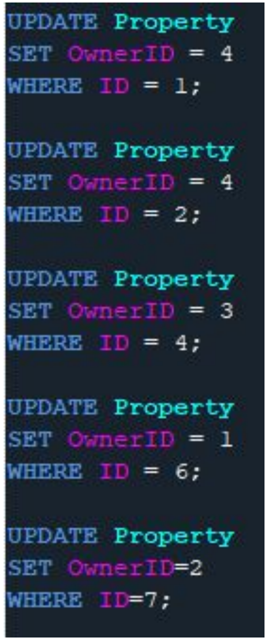
```
CREATE TABLE Players(
    "ID" INTEGER NOT NULL,
    "Player" VARCHAR(10) UNIQUE NOT NULL,
    "Token" VARCHAR(15) UNIQUE NOT NULL,
    "BankBalance" INTEGER,
    "PropertyID" INTEGER,
    "BonusID" INTEGER,
    PRIMARY KEY(ID),
    FOREIGN KEY(BonusID) REFERENCES Bonus(ID),
    FOREIGN KEY(PropertyID) REFERENCES Property(ID),
    FOREIGN KEY(Token) REFERENCES Token(Name)
);

CREATE TABLE Audit(
    "PlayerID" INTEGER NOT NULL,
    "Updated_Balance" INTEGER,
    "PropertyID" INTEGER,
    "BonusID" INTEGER,
    "GameRound" INTEGER NOT NULL,
    PRIMARY KEY(PlayerID,GameRound),
    FOREIGN KEY(PlayerID) REFERENCES Players(ID)
);
```

populate.sql

File contains queries required to successfully populate the database to match the initial state.



```
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 INTO Token(ID,Name)
VALUES
(1,"Battleship"),
(2,"Dog"),
(3,"Top Hat"),
(4,"Car"),
(5,"Thimble"),
(6,"Boot");

INSERT INTO Players(ID,Player,Token,BankBalance,PropertyID,BonusID)
VALUES
(1,"Mary","Battleship",190,NULL,5),
(2,"Bill","Dog",500,2,NULL),
(3,"Jane","Car",150,3,NULL),
(4,"Norman","Thimble",250,5,NULL);

INSERT INTO Property(ID,Name,Cost,Color,OwnerID)
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);
```

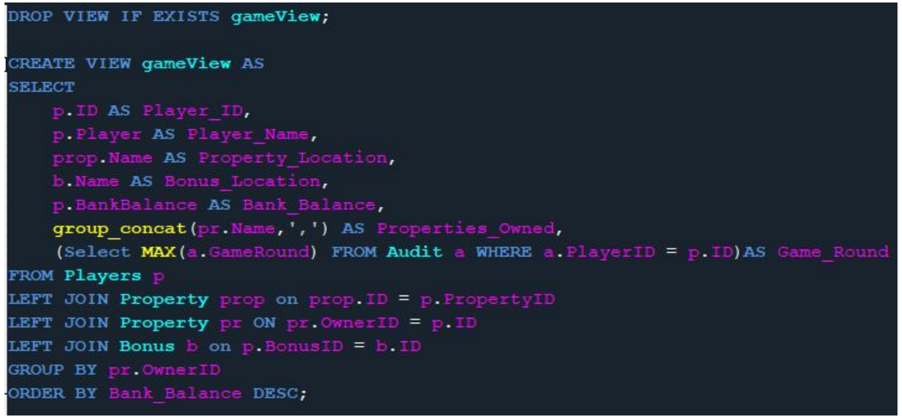


```
UPDATE Property
SET OwnerID = 4
WHERE ID = 1;

UPDATE Property
SET OwnerID = 4
WHERE ID = 2;

UPDATE Property
SET OwnerID = 3
WHERE ID = 4;

UPDATE Property
SET OwnerID = 1
WHERE ID = 6;

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

view.sql

File contains an SQL View that displays a leaderboard of the gameplay.

Query

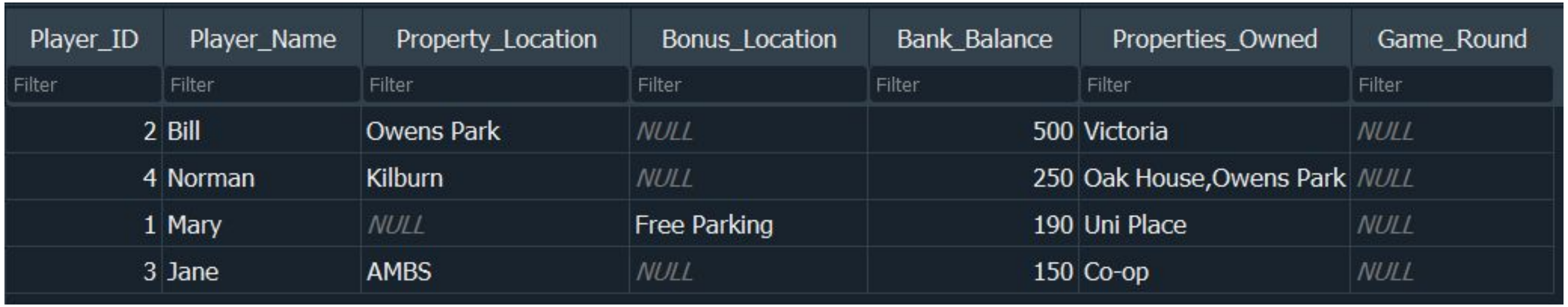


```
DROP VIEW IF EXISTS gameView;

CREATE VIEW gameView AS
SELECT
    p.ID AS Player_ID,
    p.Player AS Player_Name,
    prop.Name AS Property_Location,
    b.Name AS Bonus_Location,
    p.BankBalance AS Bank_Balance,
    group_concat(pr.Name,',') AS Properties_Owned,
    (Select MAX(a.GameRound) FROM Audit a WHERE a.PlayerID = p.ID)AS Game_Round
FROM Players p
LEFT JOIN Property prop on prop.ID = p.PropertyID
LEFT JOIN Property pr ON pr.OwnerID = p.ID
LEFT JOIN Bonus b on p.BonusID = b.ID
GROUP BY pr.OwnerID
ORDER BY Bank_Balance DESC;
```

Key Aspects:

1. **DDL/DML command**
2. **View -** gameView
3. **Aggregate function (group_concat) -** to display all owned properties
4. **JOIN, LEFT JOIN -** to display the consolidated view from multiple tables.
5. **GROUP BY**
6. **ORDER BY Clause**
7. **Aliases**
8. **Subquery**

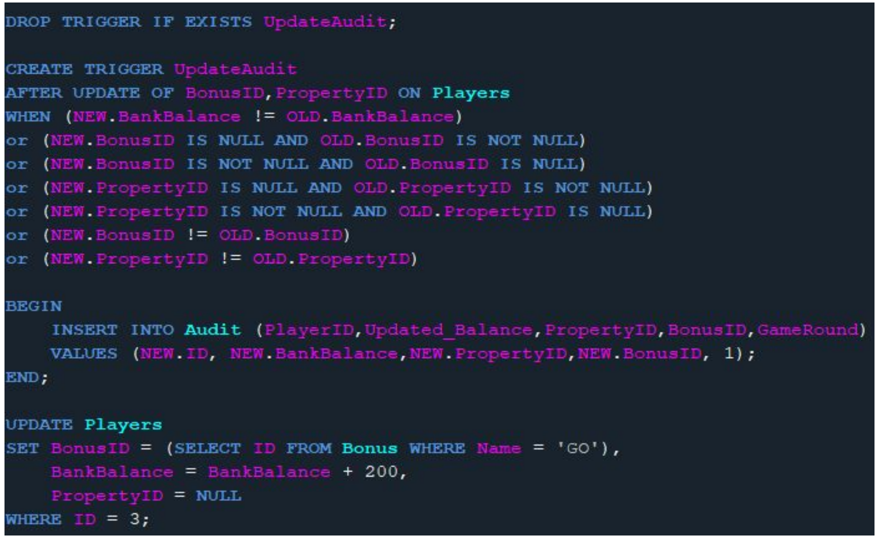
gameView before starting the game



Player_ID	Player_Name	Property_Location	Bonus_Location	Bank_Balance	Properties_Owned	Game_Round
Filter	Filter	Filter	Filter	Filter	Filter	Filter
2	Bill	Owens Park	NULL	500	Victoria	NULL
4	Norman	Kilburn	NULL	250	Oak House,Owens Park	NULL
1	Mary	NULL	Free Parking	190	Uni Place	NULL
3	Jane	AMBS	NULL	150	Co-op	NULL

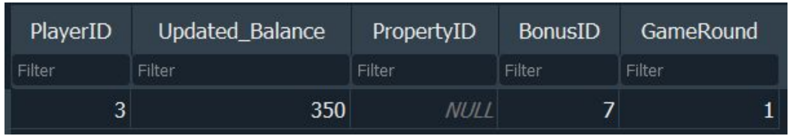
q1.sql Simulation of G1 - Jane rolls a 3

Query



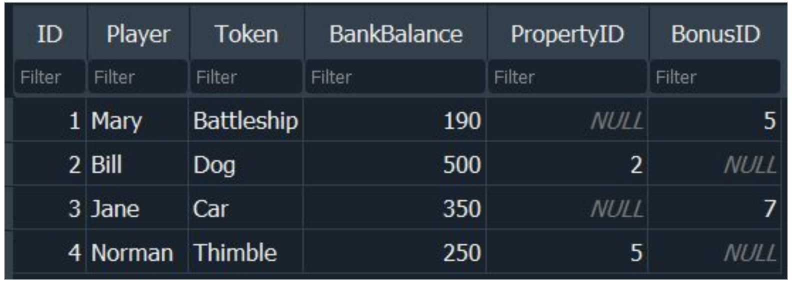
```
DROP TRIGGER IF EXISTS UpdateAudit;

CREATE TRIGGER UpdateAudit
AFTER UPDATE OF BonusID,PropertyID ON Players
WHEN (NEW.BankBalance != OLD.BankBalance)
or (NEW.BonusID IS NULL AND OLD.BonusID IS NOT NULL)
or (NEW.BonusID IS NOT NULL AND OLD.BonusID IS NULL)
or (NEW.PropertyID IS NULL AND OLD.PropertyID IS NOT NULL)
or (NEW.PropertyID IS NOT NULL AND OLD.PropertyID IS NULL)
or (NEW.BonusID != OLD.BonusID)
or (NEW.PropertyID != OLD.PropertyID)

BEGIN
    INSERT INTO Audit (PlayerID,Updated_Balance,PropertyID,BonusID,GameRound)
    VALUES (NEW.ID, NEW.BankBalance,NEW.PropertyID,NEW.BonusID, 1);
END;

UPDATE Players
SET BonusID = (SELECT ID FROM Bonus WHERE Name = 'GO'),
    BankBalance = BankBalance + 200,
    PropertyID = NULL
WHERE ID = 3;
```

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
Filter	Filter	Filter	Filter	Filter
3	350	NULL	7	1

Players table



ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	190	NULL	5
2	Bill	Dog	500	2	NULL
3	Jane	Car	350	NULL	7
4	Norman	Thimble	250	5	NULL

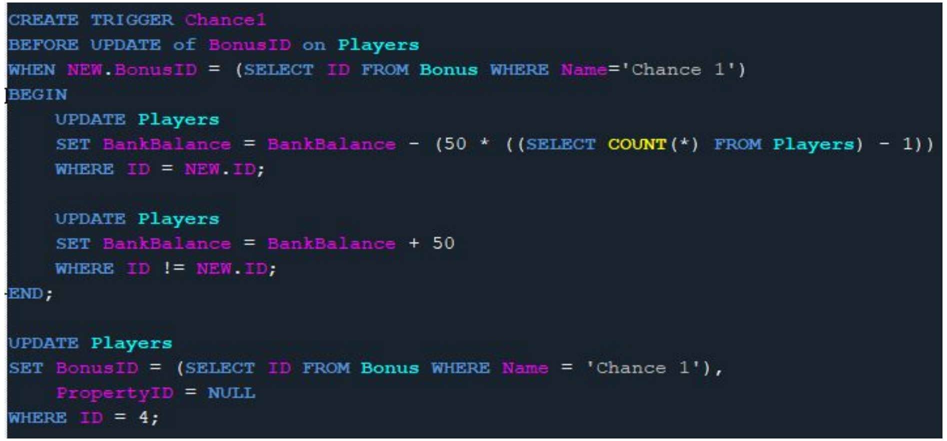
Action performed:
Initial location - AMBS
Update location - GO
Rule - Add £200 to Bank Balance.

Key Aspects:

1. **Trigger - UpdateAudit** after update in Player table
2. **Nested Query** to update BonusID
3. **DML** commands

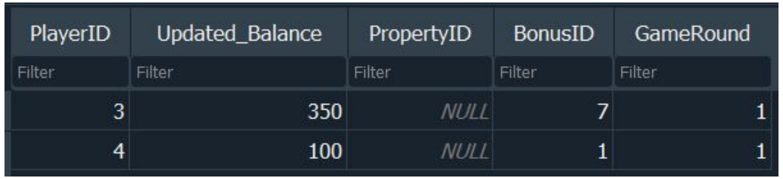
q2.sql Simulation of G2 - Norman rolls a 1

Query



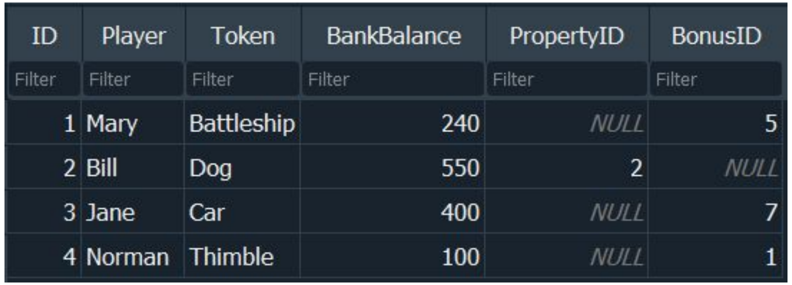
```
CREATE TRIGGER Chance1
BEFORE UPDATE of BonusID on Players
WHEN NEW.BonusID = (SELECT ID FROM Bonus WHERE Name='Chance 1')
BEGIN
    UPDATE Players
    SET BankBalance = BankBalance - (50 * ((SELECT COUNT(*) FROM Players) - 1))
    WHERE ID = NEW.ID;

    UPDATE Players
    SET BankBalance = BankBalance + 50
    WHERE ID != NEW.ID;
END;

UPDATE Players
SET BonusID = (SELECT ID FROM Bonus WHERE Name = 'Chance 1'),
    PropertyID = NULL
WHERE ID = 4;
```

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
Filter	Filter	Filter	Filter	Filter
3	350	NULL	7	1
4	100	NULL	1	1

Players table



ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	240	NULL	5
2	Bill	Dog	550	2	NULL
3	Jane	Car	400	NULL	7
4	Norman	Thimble	100	NULL	1

Action performed:
Initial location - Kilburn
Update location - Chance1
Rule - Pay each of the other player £50.

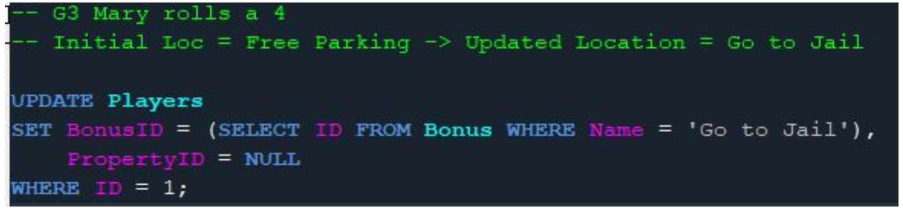
Key Aspects:

1. **Trigger - UpdateAudit** is already initialized from q1.sql

2. **Trigger - Chance1** before update of BonusID

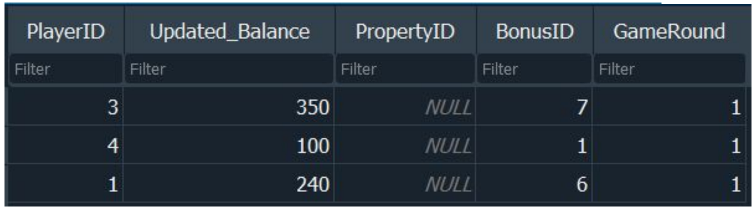
q3.sql Simulation of G3 - Mary rolls a 4

Query



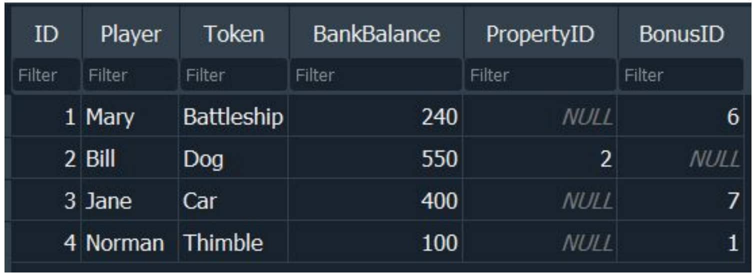
```sql
-- G3 Mary rolls a 4
-- Initial Loc = Free Parking -> Updated Location = Go to Jail

UPDATE Players
SET BonusID = (SELECT ID FROM Bonus WHERE Name = 'Go to Jail'),
    PropertyID = NULL
WHERE ID = 1;
```

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
3	350	NULL	7	1
4	100	NULL	1	1
1	240	NULL	6	1

Players table



ID	Player	Token	BankBalance	PropertyID	BonusID
1	Mary	Battleship	240	NULL	6
2	Bill	Dog	550	2	NULL
3	Jane	Car	400	NULL	7
4	Norman	Thimble	100	NULL	1

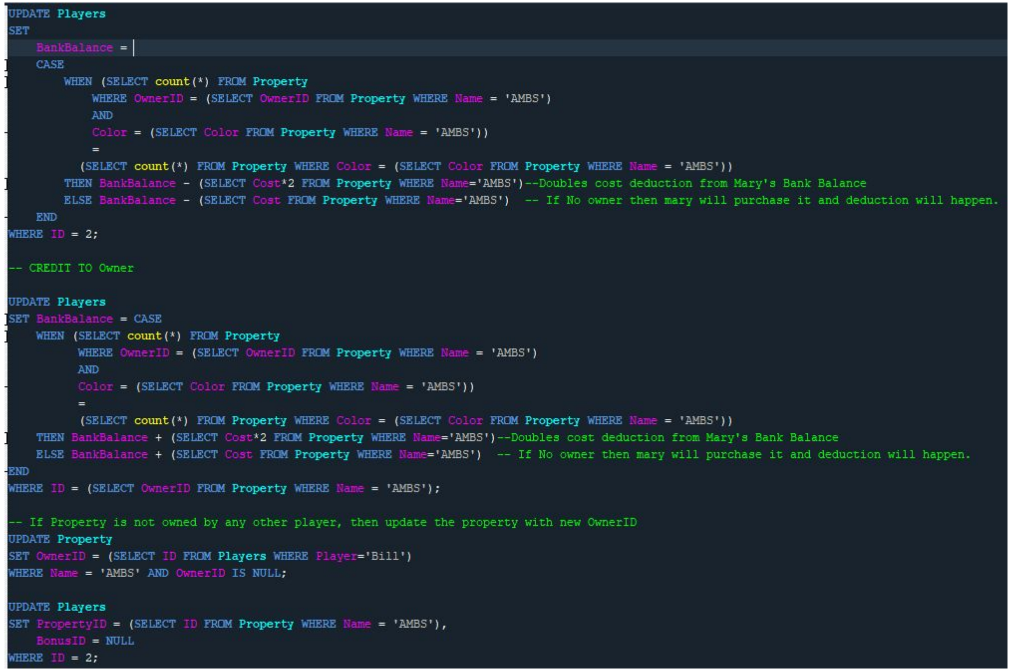
Action performed:
Initial location - Free Parking
Update location - Go to Jail
Rule - move to Jail, without passing GO

Key Aspects:

1. **Trigger - UpdateAudit** is already initialized from q1.sql

q4.sql Simulation of G4 - Bill rolls a 2

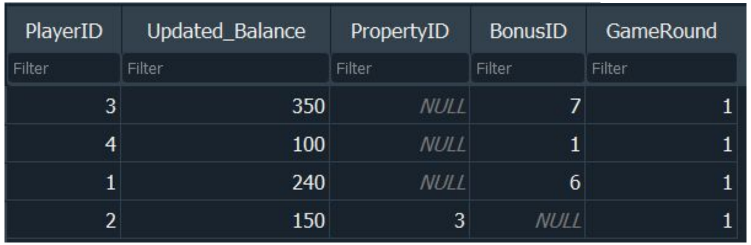
Query



```sql
UPDATE Players
SET
    BankBalance = 
    CASE
        WHEN (SELECT count(*) FROM Property
            WHERE OwnerID = (SELECT OwnerID 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 BankBalance - (SELECT Cost*2 FROM Property WHERE Name='AMBS')--Doubles cost deduction from Mary's Bank Balance
        ELSE BankBalance - (SELECT Cost FROM Property WHERE Name='AMBS')  -- If No owner then mary will purchase it and deduction will happen.
    END
WHERE ID = 2;

-- CREDIT TO Owner

UPDATE Players
SET BankBalance = CASE
    WHEN (SELECT count(*) FROM Property
          WHERE OwnerID = (SELECT OwnerID 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 BankBalance + (SELECT Cost*2 FROM Property WHERE Name='AMBS')--Doubles cost deduction from Mary's Bank Balance
    ELSE BankBalance + (SELECT Cost FROM Property WHERE Name='AMBS')  -- If No owner then mary will purchase it and deduction will happen.
END
WHERE ID = (SELECT OwnerID FROM Property WHERE Name = 'AMBS');

-- If Property is not owned by any other player, then update the property with new OwnerID
UPDATE Property
SET OwnerID = (SELECT ID FROM Players WHERE Player='Bill')
WHERE Name = 'AMBS' AND OwnerID IS NULL;

UPDATE Players
SET PropertyID = (SELECT ID FROM Property WHERE Name = 'AMBS'),
    BonusID = NULL
WHERE ID = 2;
```

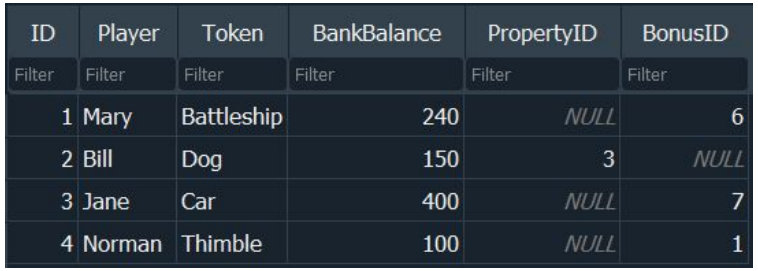
Action performed:
Initial location - Owens Park
Update location - AMBS
Rule - player lands on a property without an owner, they must buy it. Deduct £400 from balance

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
Filter	Filter	Filter	Filter	Filter
3	350	NULL	7	1
4	100	NULL	1	1
1	240	NULL	6	1
2	150	3	NULL	1

Players table

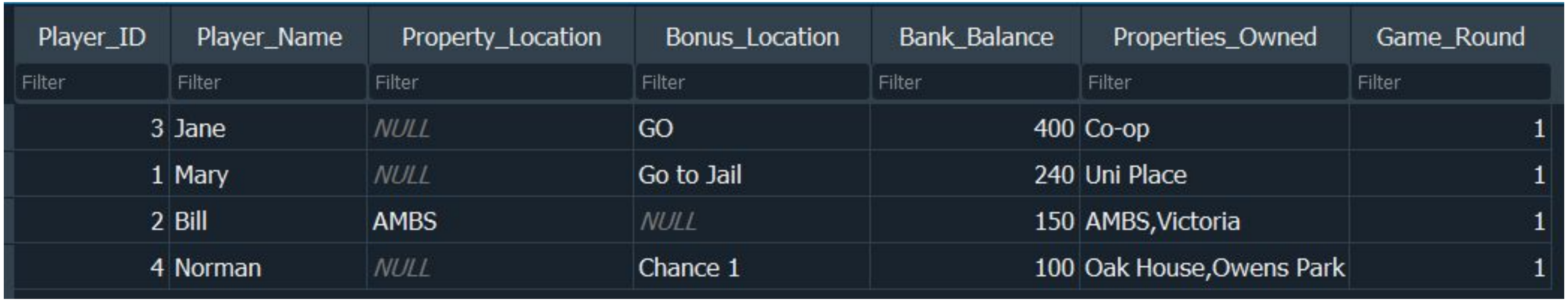


ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	240	NULL	6
2	Bill	Dog	150	3	NULL
3	Jane	Car	400	NULL	7
4	Norman	Thimble	100	NULL	1

Key Aspects:
1. **Trigger - UpdateAudit** is already initialized from q1.sql
2. **CASE** Expressions - implement conditional logic
3. **Subqueries**
4. **SET** operators(=,NULL)

State after Gameplay round 1

Figure 4 : Leaderboard table (ORDER BY Balance DESC)



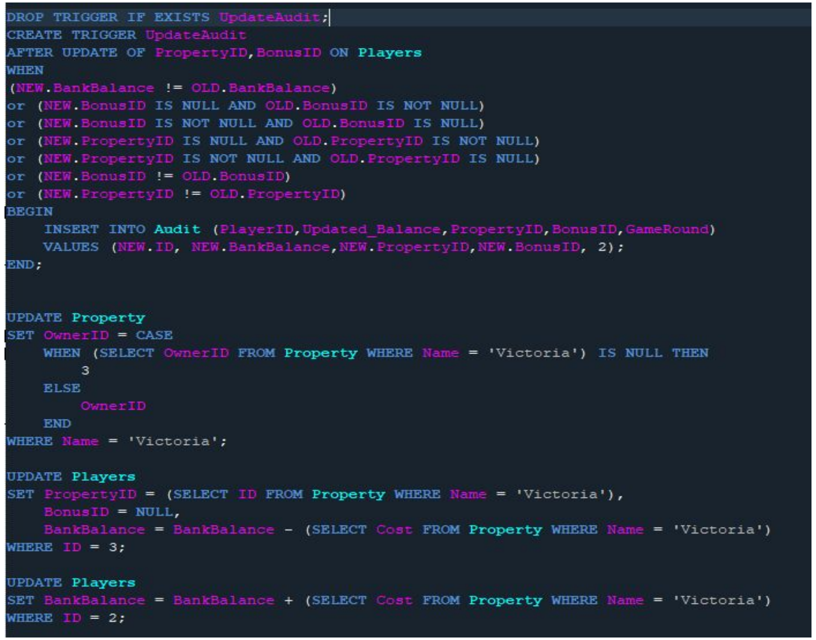
Player_ID	Player_Name	Property_Location	Bonus_Location	Bank_Balance	Properties_Owned	Game_Round
Filter	Filter	Filter	Filter	Filter	Filter	Filter
3	Jane	*NULL*	GO	400	Co-op	1
1	Mary	*NULL*	Go to Jail	240	Uni Place	1
2	Bill	AMBS	*NULL*	150	AMBS,Victoria	1
4	Norman	*NULL*	Chance 1	100	Oak House,Owens Park	1

Key Aspects:

1. This table is generated from **view.sql** which shows the **balance, current location of each player (property or bonus) and properties owned** with round value as 1.

q5.sql Simulation of G5 - Jane rolls a 5

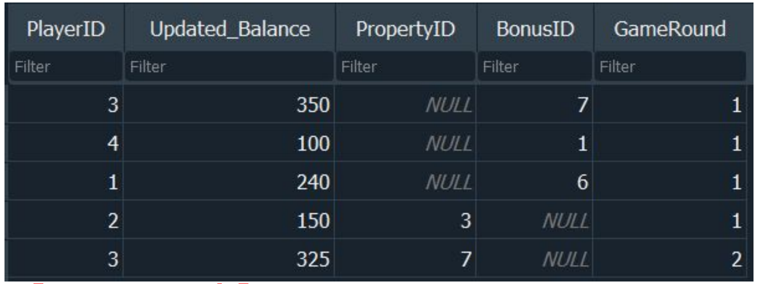
Query



```sql
DROP TRIGGER IF EXISTS UpdateAudit;
CREATE TRIGGER UpdateAudit
AFTER UPDATE OF PropertyID,BonusID ON Players
WHEN
(NEW.BankBalance != OLD.BankBalance)
or (NEW.BonusID IS NULL AND OLD.BonusID IS NOT NULL)
or (NEW.BonusID IS NOT NULL AND OLD.BonusID IS NULL)
or (NEW.PropertyID IS NULL AND OLD.PropertyID IS NOT NULL)
or (NEW.PropertyID IS NOT NULL AND OLD.PropertyID IS NULL)
or (NEW.BonusID != OLD.BonusID)
or (NEW.PropertyID != OLD.PropertyID)
BEGIN
    INSERT INTO Audit (PlayerID,Updated_Balance,PropertyID,BonusID,GameRound)
    VALUES (NEW.ID, NEW.BankBalance,NEW.PropertyID,NEW.BonusID, 2);
END;


UPDATE Property
SET OwnerID = CASE
    WHEN (SELECT OwnerID FROM Property WHERE Name = 'Victoria') IS NULL THEN
        3
    ELSE
        OwnerID
    END
WHERE Name = 'Victoria';

UPDATE Players
SET PropertyID = (SELECT ID FROM Property WHERE Name = 'Victoria'),
    BonusID = NULL,
    BankBalance = BankBalance - (SELECT Cost FROM Property WHERE Name = 'Victoria')
WHERE ID = 3;

UPDATE Players
SET BankBalance = BankBalance + (SELECT Cost FROM Property WHERE Name = 'Victoria')
WHERE ID = 2;
```

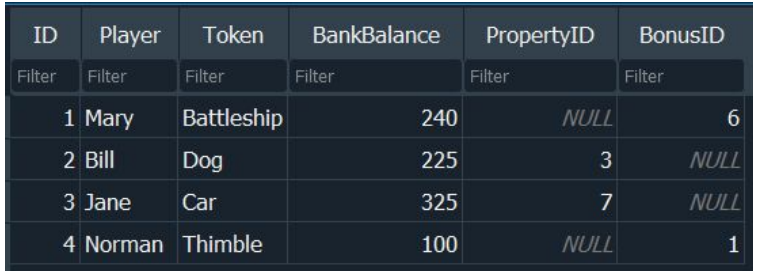
Action performed:
Initial location - GO
Update location - Victoria
Rule - Victoria is already owned by P2. Give £75 rent to player 2.

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
Filter	Filter	Filter	Filter	Filter
3	350	NULL	7	1
4	100	NULL	1	1
1	240	NULL	6	1
2	150	3	NULL	1
3	325	7	NULL	2

Players table



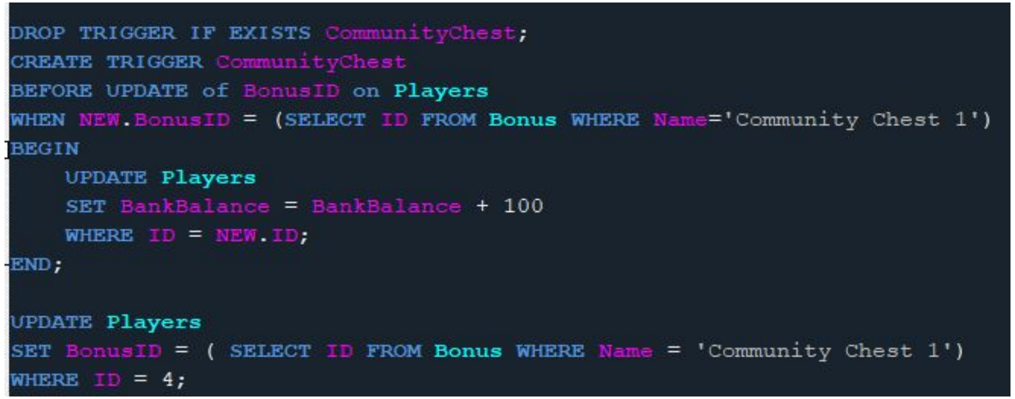
ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	240	NULL	6
2	Bill	Dog	225	3	NULL
3	Jane	Car	325	7	NULL
4	Norman	Thimble	100	NULL	1

Key Aspects:

1. **Trigger - UpdateAudit** is dropped and executed again with round 2 value.

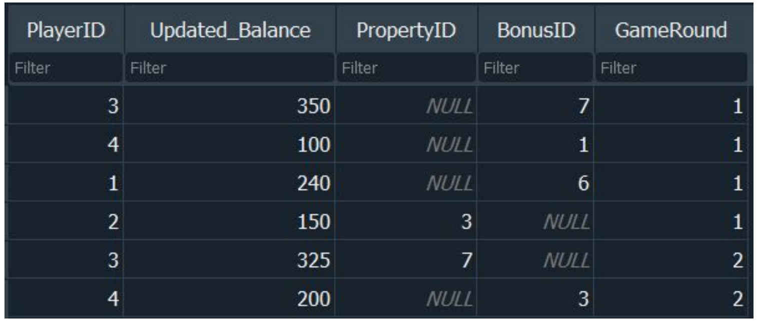
q6.sql Simulation of G6 - Norman rolls a 4

Query



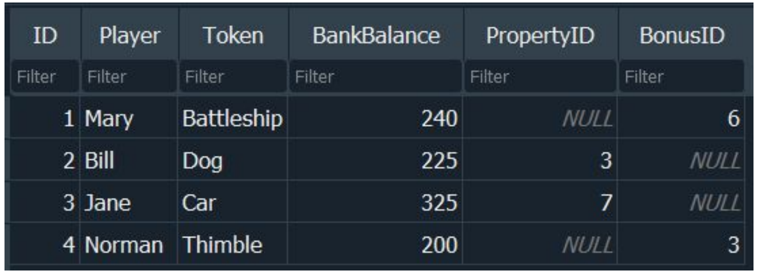
```
DROP TRIGGER IF EXISTS CommunityChest;
CREATE TRIGGER CommunityChest
BEFORE UPDATE of BonusID on Players
WHEN NEW.BonusID = (SELECT ID FROM Bonus WHERE Name='Community Chest 1')
BEGIN
    UPDATE Players
    SET BankBalance = BankBalance + 100
    WHERE ID = NEW.ID;
END;

UPDATE Players
SET BonusID = ( SELECT ID FROM Bonus WHERE Name = 'Community Chest 1')
WHERE ID = 4;
```

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
3	350	NULL	7	1
4	100	NULL	1	1
1	240	NULL	6	1
2	150	3	NULL	1
3	325	7	NULL	2
4	200	NULL	3	2

Players table



ID	Player	Token	BankBalance	PropertyID	BonusID
1	Mary	Battleship	240	NULL	6
2	Bill	Dog	225	3	NULL
3	Jane	Car	325	7	NULL
4	Norman	Thimble	200	NULL	3

Action performed:
Initial location - Chance 1
Update location - Community Chest 1
Rule - For winning a beauty contest, Add £100 to balance.

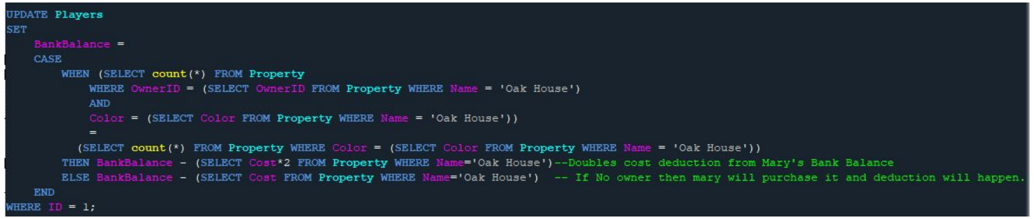
Key Aspects:

1. **Trigger - UpdateAudit** is already running.
2. **Trigger - Community Chest activates** before update of BonusID. If the trigger condition is met (i.e., if the BonusID is being set to 'Community Chest 1'), the trigger's action block will be executed.

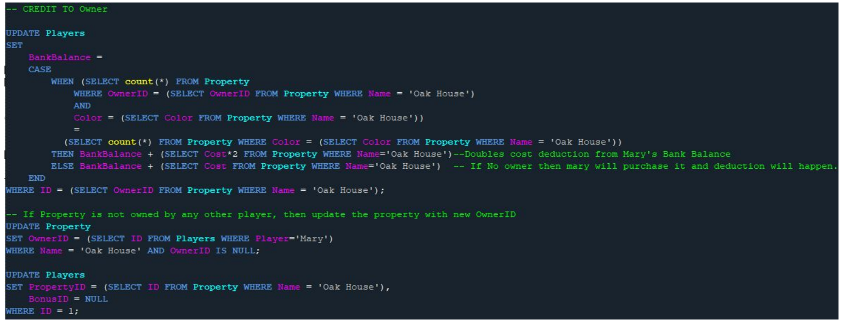
q7.sql

Simulation of G7 - Mary rolls a 6, and then a 5

Query

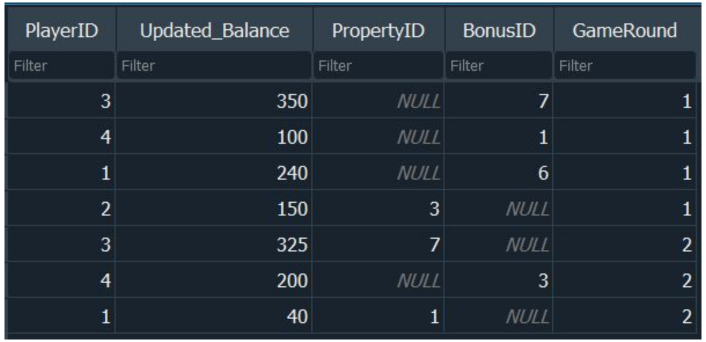


```sql
UPDATE Players
SET
    BankBalance =
    CASE
        WHEN (SELECT count(*) FROM Property
            WHERE OwnerID = (SELECT OwnerID FROM Property WHERE Name = 'Oak House')
            AND
            Color = (SELECT Color FROM Property WHERE Name = 'Oak House'))
            =
          (SELECT count(*) FROM Property WHERE Color = (SELECT Color FROM Property WHERE Name = 'Oak House'))
        THEN BankBalance - (SELECT Cost*2 FROM Property WHERE Name='Oak House')--Doubles cost deduction from Mary's Bank Balance
        ELSE BankBalance - (SELECT Cost FROM Property WHERE Name='Oak House')  -- If No owner then mary will purchase it and deduction will happen.
    END
WHERE ID = 1;
```



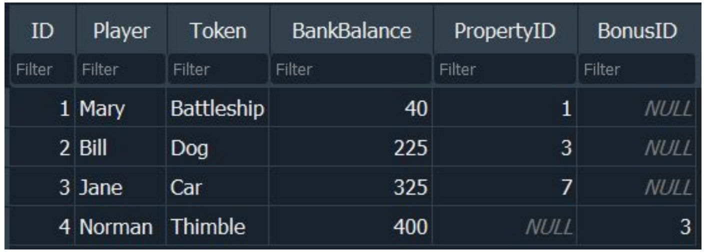
```sql
-- CREDIT TO Owner

UPDATE Players
SET
    BankBalance =
    CASE
        WHEN (SELECT count(*) FROM Property
            WHERE OwnerID = (SELECT OwnerID 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 BankBalance + (SELECT Cost*2 FROM Property WHERE Name='Oak House')--Doubles cost deduction from Mary's Bank Balance
        ELSE BankBalance + (SELECT Cost FROM Property WHERE Name='Oak House')  -- If No owner then mary will purchase it and deduction will happen.
    END
WHERE ID = (SELECT OwnerID FROM Property WHERE Name = 'Oak House');

-- If Property is not owned by any other player, then update the property with new OwnerID
UPDATE Property
SET OwnerID = (SELECT ID FROM Players WHERE Player='Mary')
WHERE Name = 'Oak House' AND OwnerID IS NULL;

UPDATE Players
SET PropertyID = (SELECT ID FROM Property WHERE Name = 'Oak House'),
    BonusID = NULL
WHERE ID = 1;
```

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
Filter	Filter	Filter	Filter	Filter
3	350	NULL	7	1
4	100	NULL	1	1
1	240	NULL	6	1
2	150	3	NULL	1
3	325	7	NULL	2
4	200	NULL	3	2
1	40	1	NULL	2

Players table



ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	40	1	NULL
2	Bill	Dog	225	3	NULL
3	Jane	Car	325	7	NULL
4	Norman	Thimble	400	NULL	3

Action performed:

Initial location - Jail

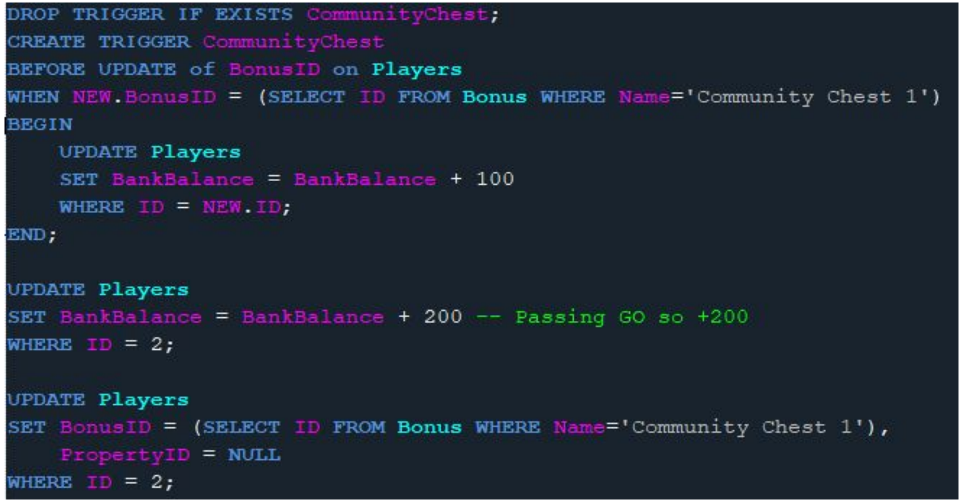
Update location - Oak House

Rule - Mary comes out from Jail on 6, and lands on Oak house after second roll. Oak House(orange) is already owned by P4 and all other properties are owned by P4. hence double rent will be deducted. Deduct £200.

q8.sql

Simulation of G8 - Bill rolls a 6, and then a 3

Query

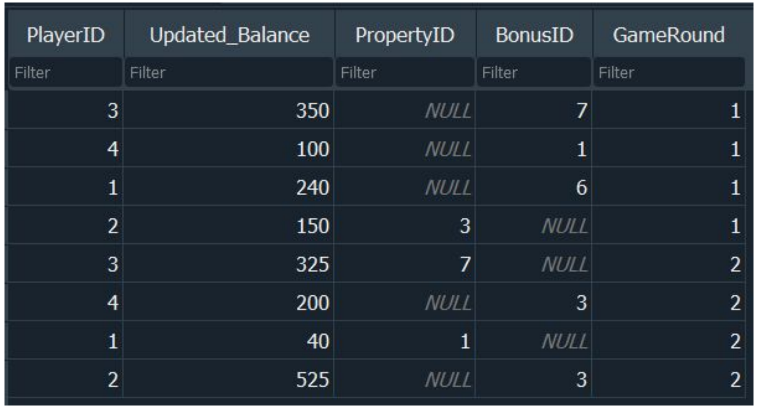


```
DROP TRIGGER IF EXISTS CommunityChest;
CREATE TRIGGER CommunityChest
BEFORE UPDATE of BonusID on Players
WHEN NEW.BonusID = (SELECT ID FROM Bonus WHERE Name='Community Chest 1')
BEGIN
    UPDATE Players
    SET BankBalance = BankBalance + 100
    WHERE ID = NEW.ID;
END;

UPDATE Players
SET BankBalance = BankBalance + 200 -- Passing GO so +200
WHERE ID = 2;

UPDATE Players
SET BonusID = (SELECT ID FROM Bonus WHERE Name='Community Chest 1'),
    PropertyID = NULL
WHERE ID = 2;
```

Action performed:

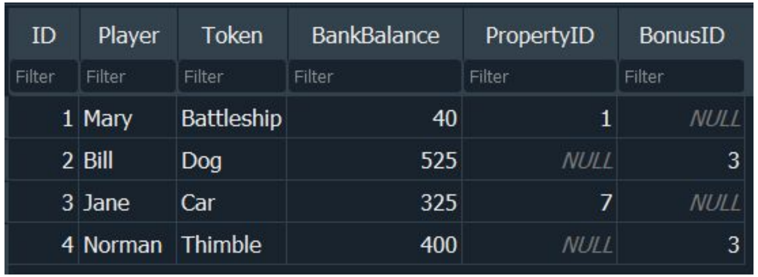
Initial location - AMBS
Update location - Community Chest 1
Rule - After rolling a 6, whatever location you land has no effect. Roll again. After 3 steps Bill lands on Community Chest 1 for which he wins a beauty contest, Add £100 to balance.

Audit table



PlayerID	Updated_Balance	PropertyID	BonusID	GameRound
Filter	Filter	Filter	Filter	Filter
3	350	NULL	7	1
4	100	NULL	1	1
1	240	NULL	6	1
2	150	3	NULL	1
3	325	7	NULL	2
4	200	NULL	3	2
1	40	1	NULL	2
2	525	NULL	3	2

Players table



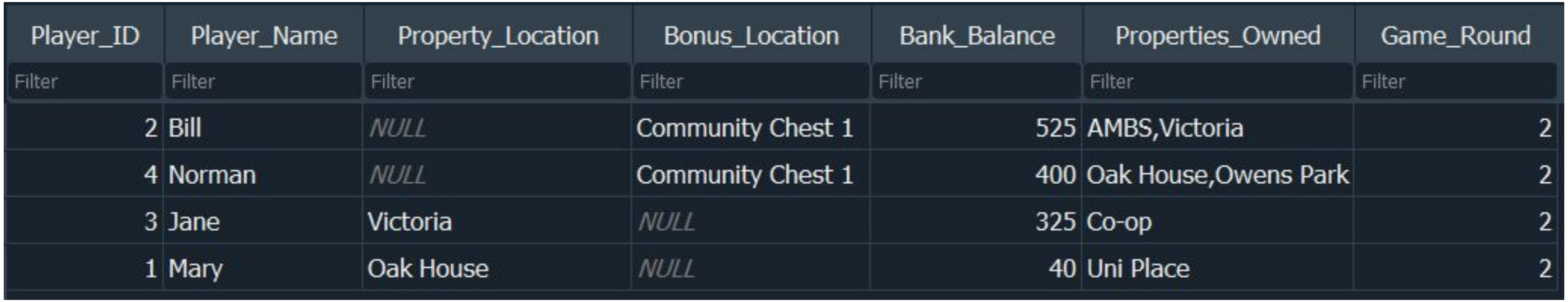
ID	Player	Token	BankBalance	PropertyID	BonusID
Filter	Filter	Filter	Filter	Filter	Filter
1	Mary	Battleship	40	1	NULL
2	Bill	Dog	525	NULL	3
3	Jane	Car	325	7	NULL
4	Norman	Thimble	400	NULL	3

Key Aspects:

1. **Trigger - UpdateAudit** is already running.
2. **Trigger - Community Chest, which is already running.**

State after Gameplay round 2 (Final View)

Figure 4 : Leaderboard table (ORDER BY Balance DESC)



Player_ID	Player_Name	Property_Location	Bonus_Location	Bank_Balance	Properties_Owned	Game_Round
Filter	Filter	Filter	Filter	Filter	Filter	Filter
2	Bill	NULL	Community Chest 1	525	AMBS,Victoria	2
4	Norman	NULL	Community Chest 1	400	Oak House,Owens Park	2
3	Jane	Victoria	NULL	325	Co-op	2
1	Mary	Oak House	NULL	40	Uni Place	2

Key Aspects:

1. This table is generated from **view.sql** which shows the **balance, current location of each player (property or bonus) and all properties owned** with round value as 2.

Normalization

First Normal Form (1NF)	Second Normal Form (2NF)	Third Normal Form (3NF)
• **Bonus, Property, Token, Players,** and **Audit** tables follow 1NF. • The tables have a primary key and attributes with atomic values, ensuring that each cell contains only a single value.	• **Bonus, Property, Token, Players** and **Audit** tables follow 2NF. • There are no partial dependencies within these tables. All non key attributes depend entirely on the primary key.	• **Bonus, Property, Token, Players** and **Audit** tables follow 3NF. • There are no transitive dependencies and all non key attributes depend solely on the primary key and not on other non-key attributes.

CONCLUSION

Technical Implementation - DB Browser and SQLite was used to implement the project.

SQL Usage - Key SQL concepts are used while designing the game (DDL/DML commands, Views, Aggregate functions, Triggers, Group By, Order By, SET operators, Nested queries).

Challenges Faced - **"Foreign Key Constraint"** was faced while populating the tables.

- Stored Procedures cannot be used in sqlite.

Database Design - While designing the tables, **property and bonus** are added as an individual columns to display if the player is currently on a property or on a bonus. Relationships and Constraints are used while establishing the relations between the tables. The redundancy was handled with normalization.

Benefits - All the key concepts were used while designing the game. Database design and SQL skills were improved as all concepts were used collectively. Views can simplify complex joins between multiple tables. We can create views that represent common join operations, making it easier to work with related data. Triggers were used to automate the actions while validating the logic.

Future Enhancements

The game was designed considering lot of pre defined factors which can be improved by automation.

1. **Automating Dice Roll**
   1.1. By using the **random** module in python.
   1.2. By creating an application with a button to roll a dice which can communicate with the game logic through API calls.
2. **Automatic Locations on Boards**
   2.1. Loading the locations (properties and bonus) of the Monopoly board is an essential part of simulating a Monopoly game. We can represent the board locations using data structures like **lists or dictionaries in Python.**
3. **Achieving Higher Level of Normalization**
   3.1. Splitting the attributes into their own tables would help achieve higher levels of normalization, reducing data redundancy and improving the structure of database. The goal is to minimize redundancy and improve data integrity, making it easier to query the database.

Future Enhancements

1. **Board Location Dictionary**
   The dictionary represents the Monopoly board, where keys are the position on the board and values are the locations.

2. **Players Dictionary**
   The 'players' dictionary stores the players in the game. Each player is represented their name as key, and their current position on board as the value.

3. **Rolling the Dice**
   The roll_dice() function simulates rolling a dice by generating random number between 1 to 6. Module used is **random**.

4. **Simulating Player's Turn**
   Function player_turn() takes player as an argument. We roll the dice using the above function and update the player's position while ensuring that the position wraps around the board if it exceeds the maximum position.

```python
#module used to generate random number for die roll
import random

# Define the Monopoly board locations
board_locations = {
    0: "Go",
    1: "Kilburn",
    2: "Chance1",
    3: "Uni Place",
    4: "In Jail",
    5: "Victoria",
    6: "Community Chest 1",
    7: "Piccadilly",
    8: "Free Parking",
    9: "Oak House",
    10: "Chance 2",
    11: "Owens Park",
    12: "Go to Jail",
    13: "AMBS",
    14: "Community Chest 2",
    15: "Co-op"
}
#Define initial location of players
players = {
    "Mary": 8,
    "Bill": 11,
    "Jane": 13,
    "Norman":1
}

# Function to roll the dice
def roll_dice():
    return random.randint(1, 6)

# Simulate a player's turn
def player_turn(player):
    dice_roll = roll_dice()
    print(f"{player} rolls a {dice_roll}")
    players[player] = (players[player] + dice_roll) % len(board_locations)
    location = board_locations[players[player]]
    print(f"{player} lands on {location}")

# Simulate multiple player turns
num_turns = 2
for _ in range(num_turns):
    for player in players:
        player_turn(player)
```

Simulation using python code

OUTPUT:

```
Mary rolls a 6
Mary lands on Community Chest 2
Bill rolls a 6
Bill lands on Kilburn
Jane rolls a 1
Jane lands on Community Chest 2
Norman rolls a 4
Norman lands on Victoria

Mary rolls a 4
Mary lands on Chance1
Bill rolls a 1
Bill lands on Chance1
Jane rolls a 1
Jane lands on Co-op
Norman rolls a 4
Norman lands on Oak House
```

MANCHESTER
1824
The University of Manchester

THANK YOU

DATA 70141 - Understanding Databases

Abhishek Kumar
11358614