Introduction

This report details the steps taken to model the gameplay of a simplified version of Monopoly. The data definition and manipulation requirements of the problem domain have been expressed in terms of an entity-relationship diagram. The report also goes in-depth about the design choices made in the said diagram and the corresponding conversion of the same to a Relational Database schema.

Monopoly: Modelling the gameplay

1. Entity Relationship Diagram

In the following Entity-Relationship diagram, we identify the different entities that make up the gameplay and establish relationships to extract an accurate Relational Database schema that will aid in implementing the said schema.

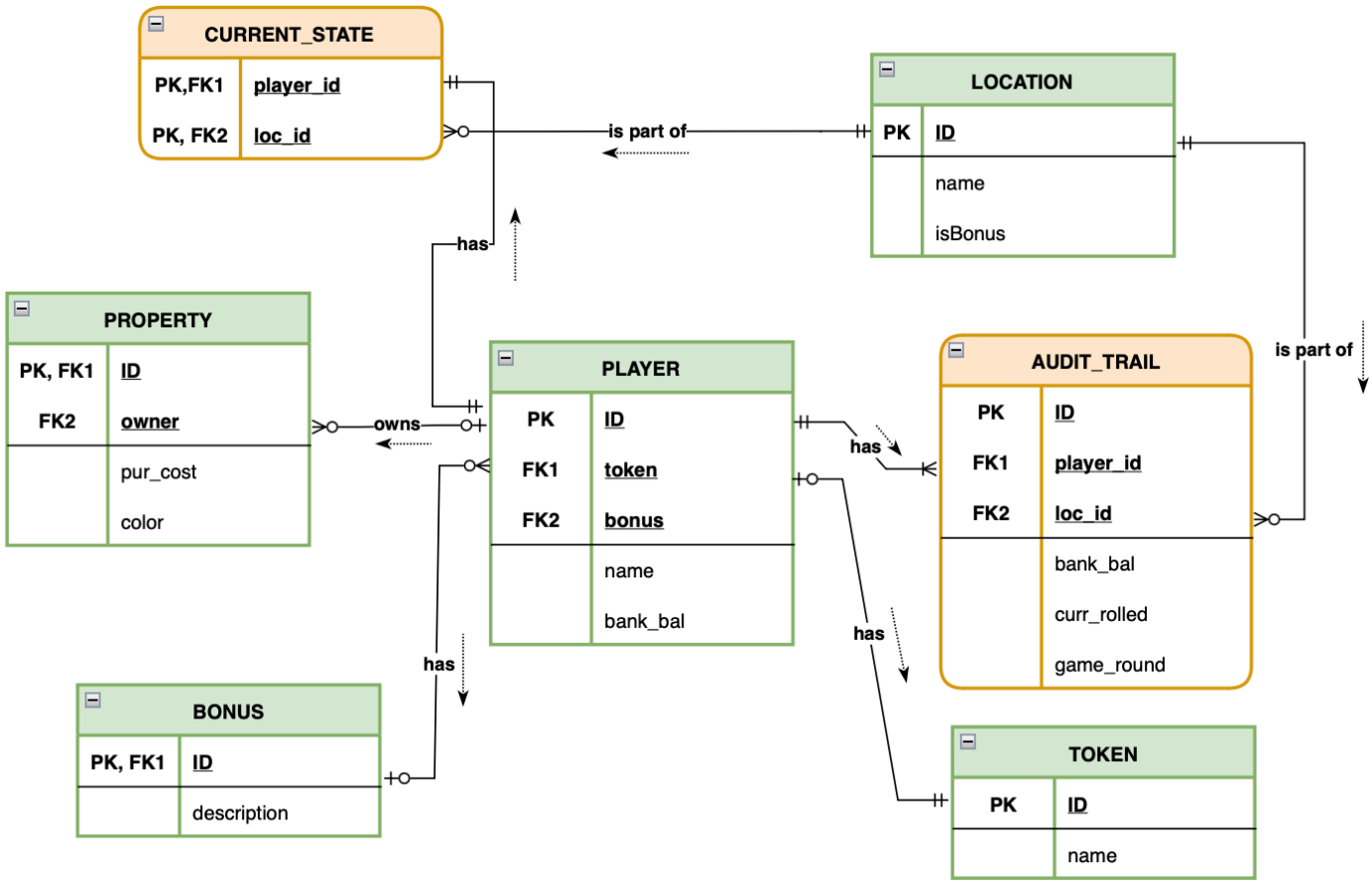


Fig 1: Entity-Relationship Diagram for Monopoly

2. Entity Relationship Diagram – Design Choices

There are strong and weak entities identified for the ER model. The strong entities are as follows:

- **Token**
  This entity identifies the token used by each player. Every player needs to have a distinct token while playing the game.
- **Player**
  This entity represents the players participating in the game. Since every player must have one token, there is a mandatory **one-and-only-one** relationship with the token. On the other hand, a token can

THUSHARA THULASIDAS NAIR (11078927)

either be owned by zero (if there are a lesser number of players than tokens) or one player at most, which establishes a **zero-or-many** relation for the token with the player.

- **Location**
  Every spot on the Monopoly board is represented by the **Location** entity. To have a better separation of concerns, additional information regarding each spot is provided as part of the **Bonus** or **Property** entities since every spot on the board is either a bonus or a property.
- **Bonus**
  The **Bonus** entity represents those locations on the board that are either Chances, Corners or Community chests. Since every player can have at most one bonus at any given time in the game, there is a **zero-or-one** relation with the **Bonus** entity, but the same bonus can be used by multiple players throughout the game which makes the relationship **zero-or-many.**
- **Property**
  Every property can have one owner at most, making the relationship **zero-or-one**. Every player can own zero or multiple properties throughout the game making its relationship with **Property** of the **zero-or-many** type.

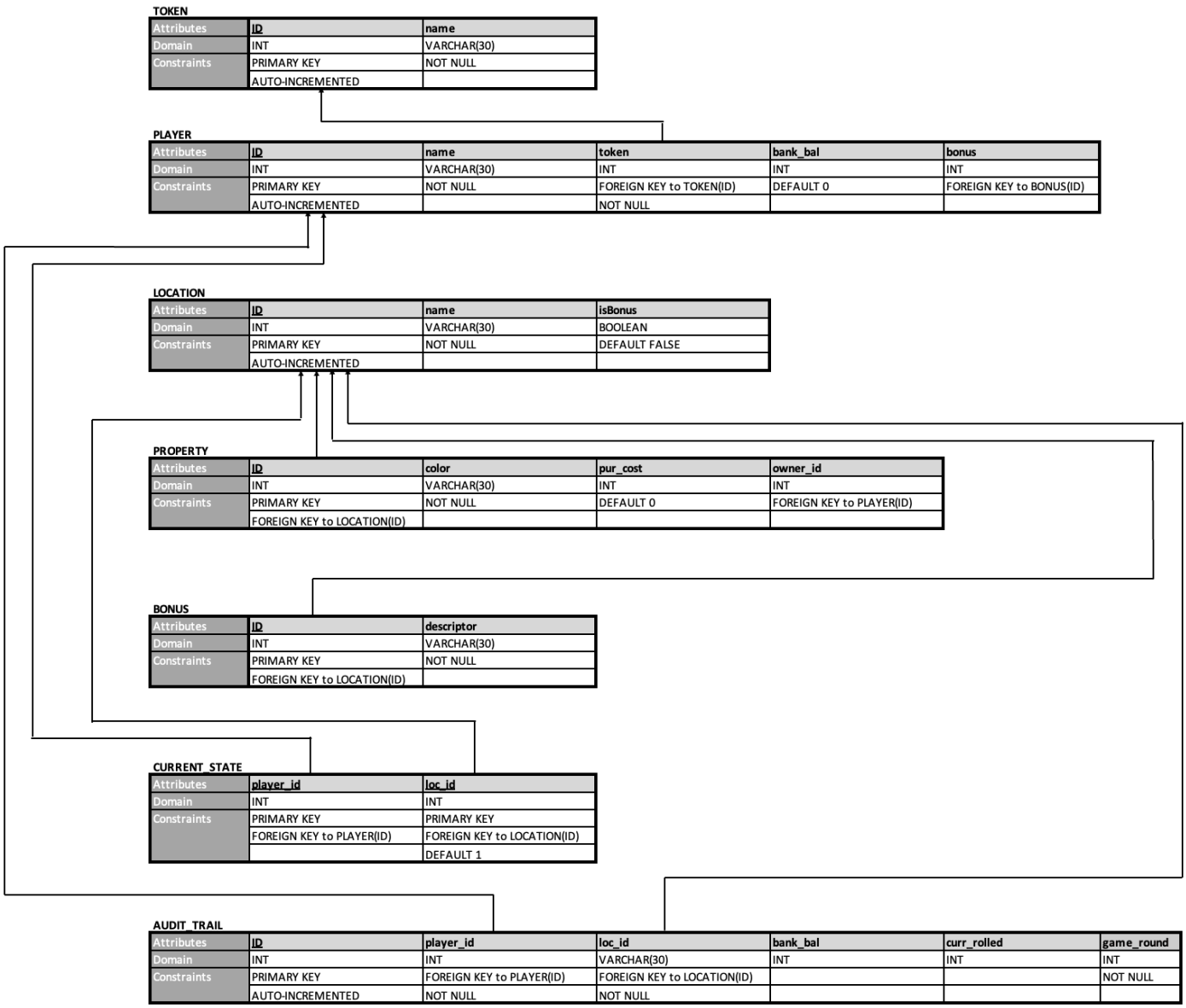
Current State and **Audit Trial** are **weak entities** because they cease to exist without the **Player** and **Location** entities since only when a player participates, will there exist a state for that player. Their relations are described further as follows:

- **Current state**
  This entity represents the current state that a player is in. Since every player has one state and every state is unique to a player, the relation is **one-and-only-one** on both ends. A location can be part of multiple states for different players, but every state can have only one location since a player can only be present at one location at any given time which makes the relationship between Location and Current State **one-to-many** (**one-and-only-one** combined with **zero-or-many**)
- **Audit trail**
  The **Audit trail** entity resembles the **Current State** entity, the difference being every state of every player throughout the game is recorded as part of this entity. A player will have at least one state as the game progresses however one record from **Audit trial** cannot have multiple players since it represents the state of a player at any given time in the game. Hence the relations established are **one-and-only-one** and **one-or-many** on either of the two sides, respectively. A location might or might not be occupied by multiple players as the game progresses however, at any point in the game, a state will be associated with only one player and their corresponding location. This makes the relationship between Location and Audit Trail of the **one-to-many** (**one-and-only-one** combined with **zero-or-many**) type.

3. Converting ER Diagram to a Relational Database Schema

3.1 Relational Database Schema

The Relational Database schema for the above ER diagram can be represented as follows:



TOKEN

Attributes	ID	name
Domain	INT	VARCHAR(30)
Constraints	PRIMARY KEY	NOT NULL
	AUTO-INCREMENTED	

PLAYER

Attributes	ID	name	token	bank_bal	bonus
Domain	INT	VARCHAR(30)	INT	INT	INT
Constraints	PRIMARY KEY	NOT NULL	FOREIGN KEY to TOKEN(ID)	DEFAULT 0	FOREIGN KEY to BONUS(ID)
	AUTO-INCREMENTED		NOT NULL		

LOCATION

Attributes	ID	name	isBonus
Domain	INT	VARCHAR(30)	BOOLEAN
Constraints	PRIMARY KEY	NOT NULL	DEFAULT FALSE
	AUTO-INCREMENTED		

PROPERTY

Attributes	ID	color	pur_cost	owner_id
Domain	INT	VARCHAR(30)	INT	INT
Constraints	PRIMARY KEY	NOT NULL	DEFAULT 0	FOREIGN KEY to PLAYER(ID)
	FOREIGN KEY to LOCATION(ID)			

BONUS

Attributes	ID	descriptor
Domain	INT	VARCHAR(30)
Constraints	PRIMARY KEY	NOT NULL
	FOREIGN KEY to LOCATION(ID)	

CURRENT_STATE

Attributes	player_id	loc_id
Domain	INT	INT
Constraints	PRIMARY KEY	PRIMARY KEY
	FOREIGN KEY to PLAYER(ID)	FOREIGN KEY to LOCATION(ID)
		DEFAULT 1

AUDIT_TRAIL

Attributes	ID	player_id	loc_id	bank_bal	curr_rolled	game_round
Domain	INT	INT	VARCHAR(30)	INT	INT	INT
Constraints	PRIMARY KEY	FOREIGN KEY to PLAYER(ID)	FOREIGN KEY to LOCATION(ID)			NOT NULL
	AUTO-INCREMENTED	NOT NULL	NOT NULL			

Fig 2: Relational Database Schema for Monopoly

- Even though a token can be uniquely identified by the name, an auto-incremental **ID** has been added to the **Token** table as the primary key because the *Integer* data type tends to perform faster than the *String* data type as foreign keys. This table is also a **static** table because the contents do not change for any game (provided the number/name of existing tokens do not change)
- For the **Player** table, an auto-incremental **ID** is used as a primary key which can uniquely identify any player. The **ID** attribute from the **Token** entity is used as a foreign key to identifying the token with which the player is associated.
- In the **Location** table, each spot is unique in terms of the auto-incremental ID. This entity also has a **name** and an **isBonus** attribute which give the name of the spot and identifies if the spot is a bonus or not, respectively.

- The **Location** table has a **parent-child relationship** with the **Bonus** and **Property** tables. This separation is done because Bonuses and Properties, although locations on the table, behave differently when a player lands on either of the two. Having separate child tables that link back to the **Location** table through the **ID** attribute allows for clean data manipulation throughout the game.
- The **Property** table has the **ID** attribute from **Location** as the primary and foreign keys. There were 2 approaches in order to identifying the properties that a player owns:
    - Adding list of properties to Player table violates the **1NF** normal form where attribute values are expected to be atomic in nature. In order to combat this, we can split the records in the **Player** table so that there is one record for each player-property combination. Not only will this introduce unwanted redundancy and issues with updating the table because of duplicate players, but it also violates the **2NF** normal form since the property becomes partially dependent on the player.
    - Adding the player's ID to the Property table – Applying **1NF and 2NF normalization**, we can add the **ID** attribute from the **Player** table as a foreign key to the **Property** table where if a property is not owned by a player, this attribute can simply be null.
- Similar to the **Property** table, the **Bonus** table also uses the **ID** attribute from the **Location** table as the primary key and the foreign key. Conforming to **1NF normal form**, the Bonus table cannot keep track of the players that have the bonus because this will result in a violation of **atomicity** since multiple players can have the same bonus. Hence, a **bonus** attribute has been added to the **Player** table to keep track of the current bonus of the player.
- In the **Current State** table, the foreign keys **ID** from the **Player** table (player_id) and **ID** from the **Location** table (loc_id) together form a **composite primary key**. This choice has been made because the current state of any player can be uniquely identified by the player's ID and the location where the player is currently. This table can also help in keeping track of any bonuses that a player might have since the **loc_id** attribute is the same ID being used in the **Bonus** table. This allows the relationship between the Bonus and Player tables to remain just logical in nature i.e., the relationship is not implemented in the actual schema.
- The **Audit trail** table holds all states that the players had throughout the game. However, the **ID** attributes from both **Player** and **Locations** tables are not used as part of the primary key since a player can have multiple states as the game progresses and some states can be at a location that the player had previously landed on. Instead, **Audit Trail** has an auto-incremental ID of its own and uses the player's id and corresponding location ID as foreign keys. This table also stores the **bank balance**, the **number that the player rolled** and the **round for gameplay** for the player for that state.

3.2 SQL Queries for the initial population of the schema

Table	Operation	SQL Query
TOKEN	Creation	`CREATE TABLE TOKEN (` `  ID INT AUTO_INCREMENT,` `  name VARCHAR(30) NOT NULL,` `  PRIMARY KEY (ID)` `);`
	Insertion	`INSERT INTO TOKEN(name)` `VALUES` `  ("dog"),` `  ("car"),` `  ("battleship"),`

		("top hat"), ("thimble"), ("boot");
PLAYER	Creation	CREATE TABLE PLAYER (ID INT AUTO_INCREMENT, name VARCHAR(30) NOT NULL, token INT NOT NULL, bank_bal INT DEFAULT 0, bonus INT, PRIMARY KEY (ID), FOREIGN KEY (token) REFERENCES TOKEN(ID), FOREIGN KEY (bonus) REFERENCES BONUS(ID));
	Insertion	INSERT INTO PLAYER(name, token, bank_bal) VALUES ("Mary", (SELECT ID FROM TOKEN WHERE name="Battleship"), 190), ("Bill", (SELECT ID FROM TOKEN WHERE name="Dog"), 500), ("Jane", (SELECT ID FROM TOKEN WHERE name="Car"), 150), ("Norman", (SELECT ID FROM TOKEN WHERE name="Thimble"), 250);
LOCATION	Creation	CREATE TABLE LOCATION (ID INT AUTO_INCREMENT, name VARCHAR(30) NOT NULL, isBonus BOOLEAN DEFAULT FALSE, PRIMARY KEY (ID));
	Insertion	INSERT INTO LOCATION(name, isBonus) VALUES ("GO", TRUE), ("Kilburn", FALSE), ("Chance 1", TRUE), ("Uni Place", FALSE), ("Jail", TRUE), ("Victoria", FALSE), ("Community Chest 1", TRUE), ("Piccadilly", FALSE), ("Free Parking", TRUE), ("Oak House", FALSE), ("Chance 2", TRUE), ("Owens Park", FALSE), ("Go to Jail", TRUE), ("AMBS", FALSE), ("Community Chest 2", TRUE), ("Co-Op", FALSE);
BONUS	Creation	CREATE TABLE BONUS (ID INT, description VARCHAR(200) NOT NULL, PRIMARY KEY (ID),

		FOREIGN KEY (ID) REFERENCES Location(ID));
	Insertion	INSERT INTO BONUS VALUES ((SELECT ID FROM LOCATION WHERE name="GO"), "Collect £200"), ((SELECT ID FROM LOCATION WHERE name="Chance 1"), "Pay each of the other players £50"), ((SELECT ID FROM LOCATION WHERE name="Jail"), "In Jail"), ((SELECT ID FROM LOCATION WHERE name="Community Chest 1"), "For winning a Beauty Contest, you win £100"), ((SELECT ID FROM LOCATION WHERE name="Free Parking"), "No action"), ((SELECT ID FROM LOCATION WHERE name="Chance 2"), "Move forward 3 spaces"), ((SELECT ID FROM LOCATION WHERE name="Go to Jail"), "Go to Jail, do not pass GO, do not collect £200"), ((SELECT ID FROM LOCATION WHERE name="Community Chest 2"), "Your library books are overdue. Pay a fine of £30");
PROPERTY	Creation	CREATE TABLE PROPERTY (ID INT, owner_id INT, color VARCHAR(30) NOT NULL, pur_cost INT DEFAULT 0, PRIMARY KEY (ID), FOREIGN KEY (ID) REFERENCES Location(ID), FOREIGN KEY (owner_id) REFERENCES PLAYER(ID));
	Insertion	INSERT INTO PROPERTY VALUES ((SELECT ID FROM LOCATION WHERE name="Kilburn"), NULL, "Yellow", 120), ((SELECT ID FROM LOCATION WHERE name="Uni Place"), (SELECT ID FROM PLAYER WHERE name="Mary"), "Yellow", 100), ((SELECT ID FROM LOCATION WHERE name="Victoria"), (SELECT ID FROM PLAYER WHERE name="Bill"), "Green", 75), ((SELECT ID FROM LOCATION WHERE name="Piccadilly"), NULL, "Green", 35), ((SELECT ID FROM LOCATION WHERE name="Oak House"), (SELECT ID FROM PLAYER WHERE name="Norman"), "Orange", 100), ((SELECT ID FROM LOCATION WHERE name="Owens Park"), (SELECT ID FROM PLAYER WHERE name="Norman"), "Orange", 30), ((SELECT ID FROM LOCATION WHERE name="AMBS"), NULL, "Blue", 400), ((SELECT ID FROM LOCATION WHERE name="Co-Op"), (SELECT ID FROM PLAYER WHERE name="Jane"), "Blue", 30);
CURRENT_STATE	Creation	CREATE TABLE CURRENT_STATE (player_id INT, loc_id INT DEFAULT 1, PRIMARY KEY (player_id, loc_id),

		FOREIGN KEY (player_id) REFERENCES PLAYER(ID), FOREIGN KEY (loc_id) REFERENCES LOCATION(ID));
	Insertion	INSERT INTO CURRENT_STATE VALUES ((SELECT ID FROM PLAYER WHERE name="Mary"), (SELECT ID FROM Location WHERE name="Free Parking")), ((SELECT ID FROM PLAYER WHERE name="Bill"), (SELECT ID FROM Location WHERE name="Owens Park")), ((SELECT ID FROM PLAYER WHERE name="Jane"), (SELECT ID FROM Location WHERE name="AMBS")), ((SELECT ID FROM PLAYER WHERE name="Norman"), (SELECT ID FROM Location WHERE name="Kilburn"));
AUDIT_TRAIL	Creation	CREATE TABLE AUDIT_TRAIL (ID INT AUTO_INCREMENT, player_id INT NOT NULL, loc_id INT NOT NULL, bank_bal INT DEFAULT 0, curr_rolled INT, game_round INT NOT NULL, PRIMARY KEY (ID), FOREIGN KEY (player_id) REFERENCES PLAYER(ID), FOREIGN KEY (loc_id) REFERENCES LOCATION(ID));
	Insertion	INSERT INTO AUDIT_TRAIL(player_id, loc_id, bank_bal, game_round) VALUES (1, (SELECT loc_id FROM CURRENT_STATE WHERE player_id=1), (SELECT bank_bal FROM PLAYER WHERE ID=1), 0), (2, (SELECT loc_id FROM CURRENT_STATE WHERE player_id=2), (SELECT bank_bal FROM PLAYER WHERE ID=2), 0), (3, (SELECT loc_id FROM CURRENT_STATE WHERE player_id=3), (SELECT bank_bal FROM PLAYER WHERE ID=3), 0), (4, (SELECT loc_id FROM CURRENT_STATE WHERE player_id=4), (SELECT bank_bal FROM PLAYER WHERE ID=4), 0);

Table 1: SQL queries for initial creation & population of the tables

The state of the players after simulating the initial state is as follows:

Player	Token	Balance	Current_location	Properties owned
Bill	dog	500	Owens Park	Victoria
Jane	car	150	AMBS	Co-Op
Mary	battleship	190	Free Parking	Uni Place
Norman	thimble	250	Kilburn	Oak House,Owens Park

Fig 3: Initial state of the players

4. Actual Gameplay

4.1 Stored procedures & views used for automating the gameplay

We make use of 5 stored procedures and 1 view to automate and view the gameplay, respectively. The procedures are as follows:

- update_player_pos(…)

  This is the root procedure that is invoked when a player rolls a die. This procedure will be responsible for identifying the final position of the player after the die is rolled and perform subsequent actions, accordingly.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `update_player_pos`(IN p_name VARCHAR(30),
                                                                IN rolled_num INT,
                                                                IN round_num INT)
BEGIN
    DECLARE p_id INT;
    DECLARE curr_loc INT;
    DECLARE curr_loc_name VARCHAR (30);
    DECLARE is_bonus BOOLEAN;
    DECLARE next_pos INT;
    DECLARE last_played INT;
    DECLARE continue_flow BOOLEAN;
    DECLARE player_bonus INT;

    SET continue_flow = TRUE;

    -- Helper procedure to get the current location details of the player
    CALL get_location_details(p_name, p_id, curr_loc, curr_loc_name);

    -- Get the previous state of the player from Audit table
    SELECT MAX(ID) INTO last_played FROM AUDIT_TRAIL WHERE player_id = p_id;
    IF curr_loc_name = "Jail" THEN
        -- How to proceed if a person is in jail
        CALL continue_if_in_jail(last_played, p_id, rolled_num, continue_flow);
    ELSE
        -- Update audit trial with the rolled number
        UPDATE AUDIT_TRAIL SET curr_rolled = rolled_num WHERE ID = last_played;
    END IF;

    IF continue_flow THEN
        SET next_pos = curr_loc + rolled_num;

        -- If the Player passes GO, update the bank balance by 200 and bonus by 1
        -- indicating the player has received a bonus during this gameplay
        SELECT bonus INTO player_bonus FROM PLAYER_ WHERE ID = p_id;
        IF player_bonus IS NULL THEN
            IF next_pos > 17 THEN
                UPDATE PLAYER
                SET bank_bal = bank_bal + 200, bonus = 1
                WHERE name = p_name;
            END IF;
        END IF;

        -- Update current state and Audit trail for the new position of the player
        IF next_pos > 16 THEN
```

```
            SET next_pos = next_pos%16;
        END IF;
        UPDATE CURRENT_STATE SET loc_id = next_pos WHERE player_id = p_id;
        INSERT INTO AUDIT_TRAIL(player_id, loc_id, bank_bal, game_round)
        VALUES (p_id, next_pos, (SELECT bank_bal FROM PLAYER WHERE ID = p_id), round_num);

        -- If rolled number is 6, only the position changes
        -- Bonus or property-related actions to be done only on the next roll
        IF rolled_num != 6 THEN
            SELECT isBonus INTO is_bonus FROM LOCATION WHERE ID = next_pos;
            SELECT bonus INTO player_bonus FROM PLAYER WHERE ID = p_id;
            IF is_bonus THEN
                IF player_bonus IS NULL THEN
                    CALL do_bonus_task(p_id, next_pos, round_num);
                END IF;
            ELSE
                CALL own_or_rent_property(p_id, next_pos);
            END IF;
            -- make the bonus null because this gameplay has ended; the player can collect
another bonus in the next gameplay
            UPDATE PLAYER SET bonus = NULL WHERE name = p_name;
        END IF;
    END IF;

END
```

Code block 1: update_player_pos(...)

- get_location_details(…)

This is a helper procedure used to retrieve location-related details given the player's name. This procedure is used in update_player_pos() (refer to Code block 1) to fetch the player's current location before updating the next location as per the rolled number.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_location_details`(IN p_name VARCHAR(30),
                                        OUT p_id INT,
                                        OUT location_id INT,
                                        OUT location_name VARCHAR(30))
BEGIN
    SELECT ID INTO p_id FROM PLAYER WHERE name = p_name ;
    SELECT loc_id INTO location_id FROM CURRENT_STATE WHERE player_id = p_id;
    SELECT name INTO location_name FROM LOCATION WHERE ID = location_id;
END
```

Code block 2: get_location_details(...)

- continue_if_in_jail(….)

We need to consider several edge cases when a player, who is in jail, rolls a die. *continue_if_in_jail(….)* is responsible for analyzing past states of the player and using that to identify the process flow if the player is in jail. This procedure is called in update_player_pos() (*refer Code block 1)*

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `continue_if_in_jail`(IN latest_play INT,
                                                                  IN player INT,
                                                                  IN rolled_num INT,
                                                                  OUT continue_flow BOOLEAN)
```

```
BEGIN

    DECLARE prev_pos_name VARCHAR(30);
    DECLARE prev_rolled INT;

    SET continue_flow = FALSE;

    -- Getting the location name of the player before they landed in Jail
    SELECT name into prev_pos_name FROM LOCATION WHERE ID = (
        SELECT loc_id FROM AUDIT_TRAIL WHERE ID = (
            SELECT MAX(ID) FROM AUDIT_TRAIL
            WHERE ID != latest_play AND player_id = player
            )
    );

    -- Check if the player landed 'In Jail' due to the 'Go To Jail' bonus
    IF prev_pos_name != "Go to Jail" THEN
        SET continue_flow = TRUE;
    ELSE
        -- Check if the player had previously rolled a 6
        SELECT curr_rolled INTO prev_rolled FROM AUDIT_TRAIL WHERE ID = latest_play;

        -- If the player had previously rolled a six, they can get out of jail
        IF prev_rolled = 6 THEN
            SET continue_flow = TRUE;
        END IF;
    END IF;
    UPDATE AUDIT_TRAIL SET curr_rolled = rolled_num WHERE ID = latest_play;

END
```

Code block 3: continue_if_in_jail(….)

- do_bonus_task(..)

In Monopoly, a player can either land on a bonus or a property. If a player lands on a bonus, this procedure defines SQL queries to be executed for each bonus. This procedure gets called to do the action described by the bonus (*refer to Code block 1)*.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `do_bonus_task`(IN player INT,
                            IN curr_loc INT,
                            IN round_num INT)
BEGIN

    DECLARE num_ids INT;
    DECLARE last_id INT;

    CASE curr_loc
        WHEN 1 THEN
            UPDATE PLAYER SET bank_bal = bank_bal + 200 WHERE ID = player;
            SELECT MAX(ID) INTO last_id FROM AUDIT_TRAIL WHERE player_id = player;
            UPDATE AUDIT_TRAIL SET bank_bal = bank_bal + 200 WHERE ID = last_id;

        WHEN 3 THEN
```

```
            SELECT COUNT(ID) INTO num_ids FROM PLAYER WHERE ID != player;
            UPDATE PLAYER SET bank_bal = bank_bal + 50 WHERE ID != player;
            UPDATE PLAYER SET bank_bal = bank_bal - 50*num_ids WHERE ID = player;
            SELECT MAX(ID) INTO last_id FROM AUDIT_TRAIL WHERE player_id = player;
            UPDATE AUDIT_TRAIL SET bank_bal = bank_bal - 50*num_ids WHERE ID = last_id;

        WHEN 7 THEN
            UPDATE PLAYER SET bank_bal = bank_bal + 100 WHERE ID = player;
            SELECT MAX(ID) INTO last_id FROM AUDIT_TRAIL WHERE player_id = player;
            UPDATE AUDIT_TRAIL SET bank_bal = bank_bal + 100 WHERE ID = last_id;

        WHEN 11 THEN
            -- If we move 3 spaces from Chance 2, we land at AMBS
            UPDATE CURRENT_STATE SET loc_id = 14 WHERE player_id = player;
            INSERT INTO AUDIT_TRAIL(player_id, loc_id, bank_bal, game_round)
                VALUES (player, 14, (SELECT bank_bal FROM PLAYER WHERE ID = player), round_num);
            CALL own_or_rent_property(player, 14);

        WHEN 13 THEN
            UPDATE CURRENT_STATE SET loc_id = 5 WHERE player_id = player;
            INSERT INTO AUDIT_TRAIL(player_id, loc_id, bank_bal, game_round)
                VALUES (player, 5, (SELECT bank_bal FROM PLAYER WHERE ID = player), round_num);

        WHEN 15 THEN
            UPDATE PLAYER SET bank_bal = bank_bal - 30 WHERE ID = player;
            SELECT MAX(ID) INTO last_id FROM AUDIT_TRAIL WHERE player_id = player;
            UPDATE AUDIT_TRAIL SET bank_bal = bank_bal - 30 WHERE ID = last_id;

        ELSE
            -- Do Nothing
            SELECT '';
    END CASE;
END
```

Code block 4: do_bonus_task(….)

- own_or_rent_property(….)

Similar to bonuses, certain actions need to be taken if a player lands on a property depending on whether the property has an owner or not. This procedure will take the necessary checks and steps to either allow the player to own the property or pay rent in case a prior owner exists(*refer to Code block 1*).

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `own_or_rent_property`(IN player_id INT,
                                                                   IN curr_loc INT)
BEGIN

    DECLARE property_id INT;
    DECLARE curr_owner INT;
    DECLARE property_color VARCHAR(30);
    DECLARE cost INT;
    DECLARE number_of_owners INT;
    DECLARE deduction INT;
    DECLARE latest_id INT;
```

```
    DECLARE own_id INT;
    DECLARE own_loc_id INT;
    DECLARE own_loc_name VARCHAR(30);

    -- Get property details
    SELECT ID, owner_id, color, pur_cost INTO property_id, curr_owner, property_color, cost FROM
PROPERTY WHERE ID = curr_loc;
    SET deduction = cost;

    -- Helper procedure to get the current location details of the player
    CALL get_location_details((SELECT name FROM PLAYER WHERE ID = curr_owner), own_id,
own_loc_id, own_loc_name);

    -- Get the ID of the latest state; to be used for updations at a later stage
    SELECT MAX(ID) INTO latest_id FROM AUDIT_TRAIL WHERE player_id = player_id;

    -- Check if the property is owned by anyone
    IF curr_owner IS NOT NULL THEN
        -- Check that the current player is not the owner
        IF curr_owner != player_id THEN
            -- Check if the owner owns all other properties of the same colour
            SELECT COUNT(DISTINCT COALESCE(owner_id,'dummy_for_null')) INTO number_of_owners
            FROM PROPERTY
            WHERE color = property_color;

            -- Paying double rent if all the properties are of the same colour
            -- are owned by the same owner
            IF number_of_owners = 1 THEN
                SET deduction = 2*cost;
            END IF;

            -- Update the bank balances of both the owner and
            -- the player who has landed
            UPDATE PLAYER SET bank_bal = bank_bal - deduction WHERE ID = player_id;
            UPDATE PLAYER SET bank_bal = bank_bal + deduction WHERE ID = curr_owner;

            -- Update bank balance for the latest state of the player
            UPDATE AUDIT_TRAIL SET bank_bal = bank_bal - deduction WHERE ID = latest_id;

        END IF;
    ELSE
        -- Buy the property if it does not have an owner
        UPDATE PLAYER SET bank_bal = bank_bal - deduction WHERE ID = player_id;
        UPDATE PROPERTY SET owner_id = player_id WHERE ID = property_id;

        -- Update bank balance For the latest state of the player
        UPDATE AUDIT_TRAIL SET bank_bal = bank_bal - deduction WHERE ID = latest_id;

    END IF;
END
```

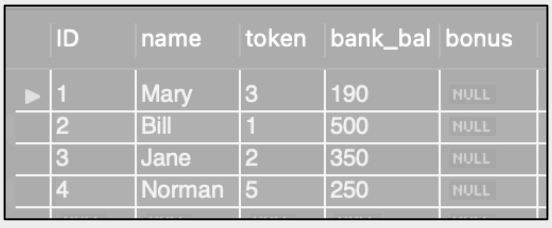
Code block 5: own_or_rent_property(....)

Apart from the above 5 procedures, we have created a view to display the status of the players after every round. The view shows the player details along with their current bank balance and the properties owned by the players towards the end of the round.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `game_view` AS
    SELECT
        `pj`.`Player` AS `Player`,
        `pj`.`Token` AS `Token`,
        `pj`.`Balance` AS `Balance`,
        `pj`.`LOCATION` AS `Current_location`,
        GROUP_CONCAT(`loc`.`name` SEPARATOR ',') AS `Properties owned`
    FROM
        ((SELECT
            `p`.`name` AS `Player`,
                `t`.`name` AS `Token`,
                `p`.`bank_bal` AS `Balance`,
                `l`.`name` AS `LOCATION`,
                `pp`.`ID` AS `Property`
        FROM
            ((((`PLAYER` `P`
        JOIN `TOKEN` `T`)
        JOIN `LOCATION` `L`)
        JOIN `CURRENT_STATE` `CSD`)
        JOIN `PROPERTY` `PP`)
        WHERE
            ((`p`.`token` = `t`.`ID`)
                AND (`p`.`ID` = `csd`.`player_id`)
                AND (`l`.`ID` = `csd`.`loc_id`)
                AND (`pp`.`owner_id` = `p`.`ID`))) `PJ`
        LEFT JOIN `LOCATION` `Loc` ON ((`pj`.`PROPERTY` = `loc`.`ID`)))
    GROUP BY `pj`.`Player`
```

Code block 6: game_view

4.2 Monopoly

Round	Game Step/SQL Query	Game view
1	**Jane rolls a 3** `CALL update_player_pos('Jane', 3, 1);` `SELECT * from PLAYER;` `SELECT * from CURRENT_STATE;` `SELECT * FROM AUDIT_TRAIL;`	*[Jane moves to 'GO'; Gets a 200 £ bonus]*  *Fig 4: Player Table for G1*

ID	name	token	bank_bal	bonus
1	Mary	3	190	NULL
2	Bill	1	500	NULL
3	Jane	2	350	NULL
4	Norman	5	250	NULL

```
select * from
PROPERTY;
SELECT * FROM
game_view;
```

player_id	loc_id
3	1
4	2
1	9
2	12

Fig 5: Current_state table for G1

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	NULL	0
2	2	12	500	NULL	0
3	3	14	150	3	0
4	4	2	250	NULL	0
5	3	1	350	NULL	1

Fig 6: Audit_trail table for G1

Player	Token	Balance	Current_location	Properties owned
Bill	dog	500	Owens Park	Victoria
Jane	car	350	GO	Co-Op
Mary	battleship	190	Free Parking	Uni Place
Norman	thimble	250	Kilburn	Oak House,Owens Park

Fig 7: Game view for G1

Norman rolls a 1

```
CALL
update_player_pos('Nor
man', 1, 1);

SELECT * from PLAYER;
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM
game_view;
```

[Norman moves to Chance 1; pays 50 each to other players]

ID	name	token	bank_bal	bonus
1	Mary	3	240	NULL
2	Bill	1	550	NULL
3	Jane	2	400	NULL
4	Norman	5	100	NULL

Fig 8: Player table for G2

player_id	loc_id
3	1
4	3
1	9
2	12
NULL	NULL

Fig 9: Current_state table for G2

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	NULL	0
2	2	12	500	NULL	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	NULL	1
6	4	3	100	NULL	1

Fig 9: Audit_trail table for G2

Player	Token	Balance	Current_location	Properties owned
Bill	dog	550	Owens Park	Victoria
Jane	car	400	GO	Co-Op
Mary	battleship	240	Free Parking	Uni Place
Norman	thimble	100	Chance 1	Oak House,Owens Park

Fig 10: Game view for G2

Mary rolls a 4

```
CALL
update_player_pos('Mar
y', 4, 1);

SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
select * from
PROPERTY;
SELECT * FROM
game_view;
```

[Mary lands on 'Go to Jail'; lands in jail]

player_id	loc_id
3	1
4	3
1	5
2	12
NULL	NULL

Fig 11: Current_state table for G3

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
2	2	12	500	NULL	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	NULL	1
6	4	3	100	NULL	1
7	1	13	240	NULL	1
8	1	5	240	NULL	1

Fig 12: Audit_trail table for G3

Player	Token	Balance	Current_location	Properties owned
Bill	dog	550	Owens Park	Victoria
Jane	car	400	GO	Co-Op
Mary	battleship	240	Jail	Uni Place
Norman	thimble	100	Chance 1	Oak House,Owens Park

Fig 13: Game view for G3

Bill rolls a 2

[Bill reached 'AMBS'; Buys 'AMBS']

```
CALL
update_player_pos('Bil
l', 2, 1);

SELECT * from PLAYER;
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
select * from
PROPERTY;
SELECT * FROM
game_view;
```

ID	name	token	bank_bal	bonus
1	Mary	3	240	NULL
2	Bill	1	150	NULL
3	Jane	2	400	NULL
4	Norman	5	100	NULL

Fig 14: Player table for G4

ID	owner_id	color	pur_cost
4	1	Yellow	100
6	2	Green	75
8	NULL	Green	35
10	4	Orange	100
12	4	Orange	30
14	2	Blue	400
16	3	Blue	30

Fig 15: Property table for G4

player_id	loc_id
3	1
4	3
1	5
2	14

Fig 16: Current_state table for G4

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	NULL	1
6	4	3	100	NULL	1
7	1	13	240	NULL	1
8	1	5	240	NULL	1
9	2	14	150	NULL	1

Fig 17: Audit_trail table for G4

Player	Token	Balance	Current_location	Properties owned
Bill	dog	150	AMBS	Victoria,AMBS
Jane	car	400	GO	Co-Op
Mary	battleship	240	Jail	Uni Place
Norman	thimble	100	Chance 1	Oak House,Owens Park

Fig 18: Game view for G4

```
SELECT * FROM
game_view;
```

Player	Token	Balance	Current_location	Properties owned
Bill	dog	150	AMBS	Victoria,AMBS
Jane	car	400	GO	Co-Op
Mary	battleship	240	Jail	Uni Place
Norman	thimble	100	Chance 1	Oak House,Owens Park

Fig 19: Game view after Round 1

2 **Jane rolls a 5**

```
CALL
update_player_pos('Jane'
, 5, 2);

SELECT * from PLAYER;
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM game_view;
```

[*Jane reaches 'Victoria'; Pays rent to the owner Bill]*

ID	name	token	bank_bal	bonus
1	Mary	3	240	NULL
2	Bill	1	225	NULL
3	Jane	2	325	NULL
4	Norman	5	100	NULL

Fig 20: Player table for G5

player_id	loc_id
4	3
1	5
3	6
2	14

Fig 21: Current_state table for G5

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	5	1
6	4	3	100	NULL	1
7	1	13	240	NULL	1
8	1	5	240	NULL	1
9	2	14	150	NULL	1
10	3	6	325	NULL	2
NULL	NULL	NULL	NULL	NULL	NULL

Fig 22: Audit_trail table for G5

Player	Token	Balance	Current_location	Properties owned
Bill	dog	225	AMBS	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	240	Jail	Uni Place
Norman	thimble	100	Chance 1	Oak House,Owens Park

Fig 23: Game view for G5

Norman rolls a 4

```
CALL
update_player_pos('Norma
n', 4, 2);

SELECT * from PLAYER;
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM game_view;
```

[Norman reaches 'Community Chest 1; Gets a 100 £ bonus]

ID	name	token	bank_bal	bonus
1	Mary	3	240	NULL
2	Bill	1	225	NULL
3	Jane	2	325	NULL
4	Norman	5	200	NULL

Fig 24: Player table for G6

player_id	loc_id
1	5
3	6
4	7
2	14

Fig 25: Current_state table for G6

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	5	1
6	4	3	100	4	1
7	1	13	240	NULL	1
8	1	5	240	NULL	1
9	2	14	150	NULL	1
10	3	6	325	NULL	2
11	4	7	200	NULL	2

Fig 26: Audit_trail table view for G6

Player	Token	Balance	Current_location	Properties owned
Bill	dog	225	AMBS	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	240	Jail	Uni Place
Norman	thimble	200	Community Chest 1	Oak House,Owens Park

Fig 27: Game view for G6

Mary rolls a 6, and then a 5

```
CALL
update_player_pos('Mary'
, 6, 2);

SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM game_view;


CALL
update_player_pos('Mary'
, 5, 2);

SELECT * from PLAYER;
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM game_view;
```

[Mary can move out of jail after the next roll]

player_id	loc_id
1	5
3	6
4	7
2	14

Fig 27: Current_state table for G7 (no updates)

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	5	1
6	4	3	100	4	1
7	1	13	240	NULL	1
8	1	5	240	6	1
9	2	14	150	NULL	1
10	3	6	325	NULL	2
11	4	7	200	NULL	2
NULL	NULL	NULL	NULL	NULL	NULL

Fig 28: Audit_trail table for G7

Player	Token	Balance	Current_location	Properties owned
Bill	dog	225	AMBS	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	240	Jail	Uni Place
Norman	thimble	200	Community Chest 1	Oak House,Owens Park

Fig 29: Game view for G7

[Mary reaches 'Oak House'; Pays double the rent since Norman owns all orange-colored properties]

ID	name	token	bank_bal	bonus
1	Mary	3	40	NULL
2	Bill	1	225	NULL
3	Jane	2	325	NULL
4	Norman	5	400	NULL
NULL	NULL	NULL	NULL	NULL

Fig 30: Player table for G7

player_id	loc_id
3	6
4	7
1	10
2	14

Fig 31: Current_state table for G7

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	5	1
6	4	3	100	4	1
7	1	13	240	NULL	1
8	1	5	240	5	1
9	2	14	150	NULL	1
10	3	6	325	NULL	2
11	4	7	200	NULL	2
12	1	10	40	NULL	2

Fig 32: Audit_trail table for G7

Player	Token	Balance	Current_location	Properties owned
Bill	dog	225	AMBS	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	40	Oak House	Uni Place
Norman	thimble	400	Community Chest 1	Oak House,Owens Park

Fig 33: Game view for G7

Bill rolls a 6, and then a 3

```
CALL
update_player_pos('Bill'
, 6, 2);

SELECT * from PLAYER;
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM game_view;


CALL
update_player_pos('Bill'
, 3, 2);
```

[Bill reaches 'Uni place'; Does not pay rent because he rolled a 6 but collects 200 £ for crossing 'GO']

ID	name	token	bank_bal	bonus
1	Mary	3	40	NULL
2	Bill	1	425	1
3	Jane	2	325	NULL
4	Norman	5	400	NULL

Fig 34: Player table for G8

```
SELECT * from
CURRENT_STATE;
SELECT * FROM
AUDIT_TRAIL;
SELECT * FROM game_view;
```

player_id	loc_id
2	4
3	6
4	7
1	10
NULL	NULL

Fig 35: Current_state table for G8

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	5	1
6	4	3	100	4	1
7	1	13	240	NULL	1
8	1	5	240	5	1
9	2	14	150	6	1
10	3	6	325	NULL	2
11	4	7	200	NULL	2
12	1	10	40	NULL	2
13	2	4	425	NULL	2
NULL	NULL	NULL	NULL	NULL	NULL

Fig 36: Audit_trail table for G8

Player	Token	Balance	Current_location	Properties owned
Bill	dog	425	Uni Place	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	40	Oak House	Uni Place
Norman	thimble	400	Community Chest 1	Oak House,Owens Park

Fig 37: Game view for G8

[Bill reaches 'Community Chest 1'; Does not get a bonus because he had a bonus from 'GO' in the same gameplay]

player_id	loc_id
3	6
2	7
4	7
1	10
NULL	NULL

Fig 38: Current_state for G8

ID	player_id	loc_id	bank_bal	curr_rolled	game_round
1	1	9	190	4	0
2	2	12	500	2	0
3	3	14	150	3	0
4	4	2	250	1	0
5	3	1	350	5	1
6	4	3	100	4	1
7	1	13	240	NULL	1
8	1	5	240	5	1
9	2	14	150	6	1
10	3	6	325	NULL	2
11	4	7	200	NULL	2
12	1	10	40	NULL	2
13	2	4	425	3	2
14	2	7	425	NULL	2

Fig 39: Audit_trail for G8

Player	Token	Balance	Current_location	Properties owned
Bill	dog	425	Community Chest 1	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	40	Oak House	Uni Place
Norman	thimble	400	Community Chest 1	Oak House,Owens Park

Fig 40: Game view for G8

```
SELECT * FROM
game_view;
```

Player	Token	Balance	Current_location	Properties owned
Bill	dog	425	Community Chest 1	Victoria,AMBS
Jane	car	325	Victoria	Co-Op
Mary	battleship	40	Oak House	Uni Place
Norman	thimble	400	Community Chest 1	Oak House,Owens Park

Fig 41: Game view after Round 2

Table 2: SQL queries & screenshots of the Gameplay

4.3 Assumptions

Some assumptions have been made to facilitate the modelling of the gameplay. These assumptions have been put into place to combat the ambiguity of some requirements in the description.

- If a player crosses 'GO' by rolling a 6, we assume that the player gets a 200£ bonus even though the final position of the player is decided only after the immediate next roll. After the immediate next roll, if the player lands on another bonus, it has no effect since the player already collected a bonus (200 £ collected early).
- If a player lands on the 'In Jail' location due to dice rolling **i.e.,** the player was not sent to jail due to the 'Go to Jail' bonus, the player does not need a 6 to get out. Only in cases where the player was sent to jail from the 'Go to Jail' corner, the player needs a 6 to get out.
- We assume when a game starts, the CURRENT_STATE and AUDIT_TRIAL tables have been populated with the player's first state, i.e., the player at location 'GO'.

- While auditing, any changes in other players' bank balances caused due to a bonus or a property being used by the current player, are not reflected in the AUDIT_TRIAL table right away. This gets reflected when the other players take their turn. This helps maintain the truth and actual order of events while Auditing.

5. Bibliography

- *diagrams.net - free flowchart maker and diagrams online*. (n.d.). https://app.diagrams.net/?libs=er
- *House Rules | Monopoly Wiki | Fandom*. (n.d.). Monopoly Wiki. https://monopoly.fandom.com/wiki/House_Rules
- *ACID Properties in DBMS - javatpoint*. (n.d.). www.javatpoint.com. https://www.javatpoint.com/acid-properties-in-dbms
- *MySQL :: MySQL and Windows :: 5.10 Debugging Stored Procedures and Functions*. (n.d.). https://dev.mysql.com/doc/mysql-windows-excerpt/8.0/en/visual-studio-debugger.html
- Claria, F. (2018, November 16). *Referential Constraints and Foreign Keys in MySQL*. LearnSQL.com. https://learnsql.com/blog/referential-constraints-foreign-keys-mysql/
- *User Defined Functions in MySQL*. (2021, July 9). Dot Net Tutorials. https://dotnettutorials.net/lesson/user-defined-functions-in-mysql/
- Rayan, J. C. (2015, February 20). *Data Modelling using ERD with Crow Foot Notation*. CodeProject. https://www.codeproject.com/Articles/878359/Data-Modelling-using-ERD-with-Crow-Foot-Notation