Section 1: Entity Relationship Diagram

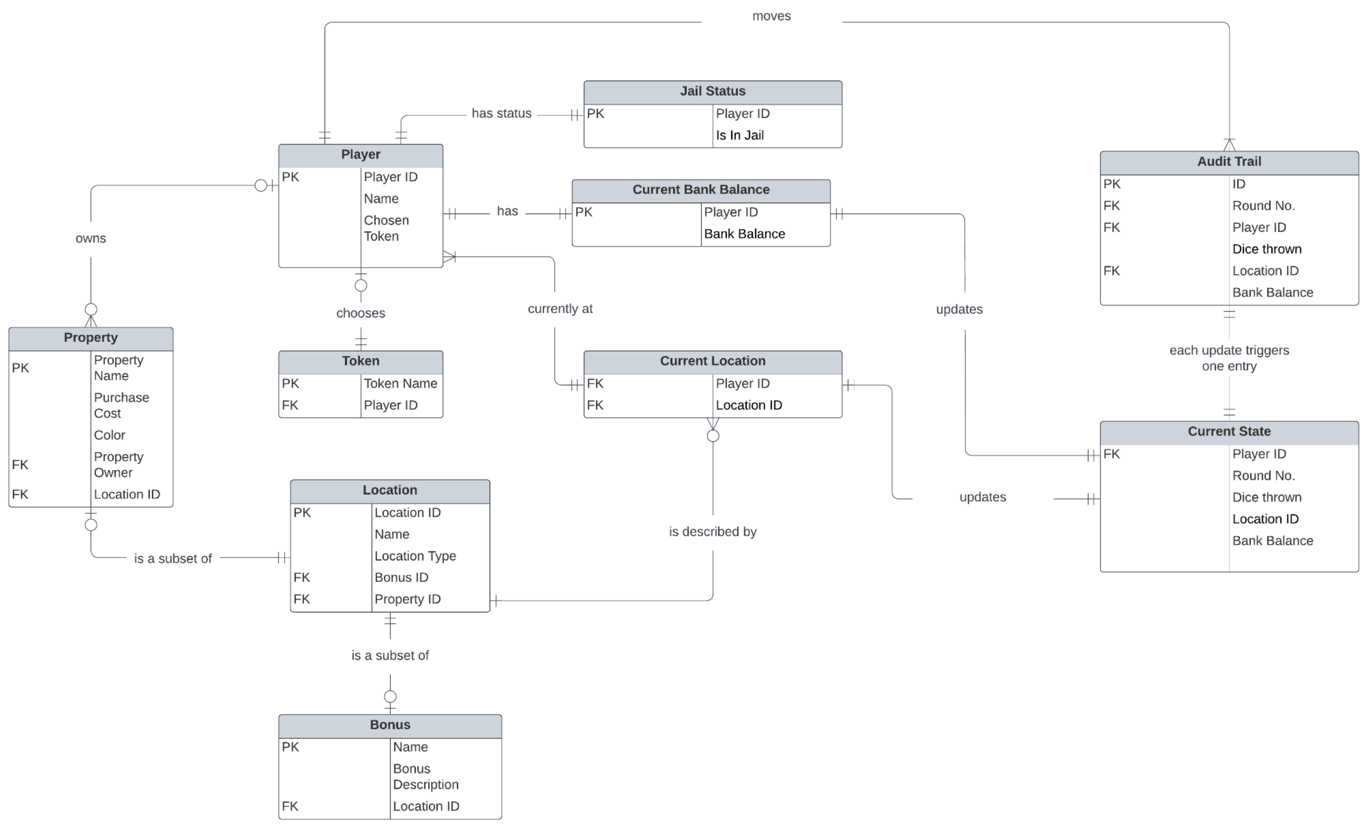


Figure 1: Entity Relationship Diagram for Monopolee Game Play

Section 2: Short Discussion detailing design choices

Figure 1 above illustrates the ER Diagram that has been created to model the Monopolee game play. Entities identified were Player, Location, Property, Bonus, Token and Jail Status, while the Current Bank Balance, Current Location and Current State tables were created to record the latest state of each player after a round was played. These tables were then used to trigger new entries in the Audit Trail table.

Main Tables

The table player is used to keep a record of all players in the game, along with their token of choice. Each player **must choose only one** token to use throughout the game, while each token can be used by **none or at most one** player in one game. This results in a one-to-one relationship between players and tokens.

The Location table is created as a record of all the locations on the board and the type of location, whether it is a '**Bonus**' or a '**Property**'. This table can then be linked to the Property or Bonus tables to get the description of a particular Bonus and the attributes associated with each Property, such as the purchase cost, color and the owner of the property if any. This results in a **one-to-one relationship between Location and Property**, as well as between **Location and Bonus,** where each property/bonus must belong to location, but a particular location can be either property or bonus. The Property table can also be linked to the Player table in the form of an **ownership relationship**, that is, **each player may own zero or many properties**, while each property can only be owned by one player, resulting in a **one-to-many relationship**.

Updating gameplay

Throughout the game, each player will experience **changes in location and/or property owned and bank balance**. The table **Current Location** is created to reflect the changes in location for a player after they finish a round, while the table **Current Bank Balance** is used to store the latest amount of bank balance for all players after a player finishes his/her round, (this includes any payment received by other players from the current player in the form of rent or bonus). **The Current State** table is created to store all the updated attributes of a particular player after he/she finishes a round, which is then used to trigger the audit trail table. The audit trail table records all updates in location and bank balance for a particular player after they finish a round. The **Jail Status** table is created to record the jail status of a particular player.

Section 3.1: Relational Database Schema

player			
Variable Name	**player_id**	**player_name**	**chosen_token**
Type	INT	VARCHAR(30)	VARCHAR(30)
Constraint	NOT NULL	NOT NULL	NOT NULL
Default	none	none	none
Key	PRIMARY KEY, AUTO INCREMENT		FOREIGN KEY REFERENCES TOKEN

token	
Variable Name	**token_name**
Type	VARCHAR(30)
Constraint	NOT NULL
Default	none
Key	PRIMARY KEY

location			
Variable Name	**location_id**	**location_name**	**location_type**
Type	INT	VARCHAR(50)	VARCHAR(30)
Constraint	NOT NULL	NOT NULL	NOT NULL
Default	none	none	none
Key	PRIMARY KEY, AUTO INCREMENT		

property					
Variable Name	**property_name**	**property_id**	**purchase_cost**	**color**	**property_owner**
Type	VARCHAR(30)	INT	INT	VARCHAR(30)	INT
Constraint	NOT NULL	NOT NULL	NOT NULL	NOT NULL	none
Default	none	none	none	none	none
Key	PRIMARY KEY	FOREIGN KEY REFERENCES LOCATION			FOREIGN KEY REFERENCES PLAYER

is in jail		
Variable Name	**player_id**	**is_in_jail**
Type	INT	BOOLEAN
Constraint	NOT NULL	NOT NULL
Default	none	FALSE
Key	PRIMARY KEY, FOREIGN KEY REFERENCES PLAYER	

current bank balance		
Variable Name	**player_id**	**bank_balance**
Type	INT	INT
Constraint	NOT NULL	NOT NULL
Default	none	0
Key	PRIMARY KEY, FOREIGN KEY REFERENCES PLAYER	

current location		
Variable Name	**player_id**	**location_id**
Type	INT	INT
Constraint	NOT NULL	NOT NULL
Default	none	none
Key	PRIMARY KEY, FOREIGN KEY REFERENCES PLAYER	FOREIGN KEY REFERENCES LOCATION

current state					
Variable Name	**player_id**	**round_no**	**dice_thrown**	**location_id**	**bank_balance**
Type	INT	INT	INT	INT	INT
Constraint	NOT NULL	NOT NULL	NOT NULL	NOT NULL	none
Default	none	none	none	none	0
Key	PRIMARY KEY, FOREIGN KEY REFERENCES PLAYER			FOREIGN KEY REFERENCES LOCATION	

audit trail					
Variable Name	**player_id**	**round_no**	**dice_thrown**	**location_id**	**bank_balance**
Type	INT	INT	INT	INT	INT
Constraint	NOT NULL	NOT NULL	NOT NULL	NOT NULL	none
Default	none	none		none	0
Key	FOREIGN KEY REFERENCES PLAYER			FOREIGN KEY REFERENCES LOCATION	

Section 3.2: Relational Database Schema Discussion

Main Tables

The player table has a primary key player_id, which is set to not null and auto increment, such that any additional player added into the table in the future will be assigned a unique primary key. Player name variable is also set to not null, such that we get a name that can be displayed in the scoreboard. The chosen_token variable also has a not null constraint, such that it does not violate the cardinality requirement, which is that *each player must choose one token*. The token table has its name as a primary key which is set to not null.

The location table has location_id which is set to not null as the primary key. This is set with auto increment, such that the location_ids are set in an increasing order from the first position 'GO' on the board, with location_id = 1 to the last position on the board, 'Co-op' with location_id = 16. Setting the location_id in this way is crucial in updating the players' moves in the gameplay.

The property table has its name as the primary key, which is set to not null. Other attributes in this table that also have the not null constraint are; property_id, purchase_cost and color. The property_id is a foreign key that corresponds to the location_id in the location table. In order to be consistent with the requirement that all property is a location, this attribute is also set to not null. Purchase cost of a property is set to not null to be in line with the concept of property specified in the requirement, whereby whenever a player lands on a property they must either buy/rent it depending on the ownership status of the property. The color of a property has a not null constraint to reflect the fact that the color of the property can affect the amount of rent paid as per the requirement. Lastly, property_owner is a foreign key that points to the player table, to identify which player owns the property. In this current design, the property table is normalised at the 1st Normal Form. To normalise it at 2nd Normal Form, another table, property_owned is created to keep a record of property_id and property_owners, such that both property and property_owned tables have non-key columns that are dependent on the primary key.

Update Tables

The **current_bank_balance** table has player_id as both its primary key and foreign key, and a bank balance with default 0. This is to model the fact that players will start at 0 balance in their account before they receive any money. This attribute is set to not null as balance a player has during the game is a crucial part of Monopolee. The **current_location** table also has player_id as both its primary key and foreign key, which points to the player table, and location_id which is a foreign key that points to the location table. The location_id in the current_location table identifies the current location a particular player is at at any point in the game play. This attribute has a not null constraint as a player should always be positioned somewhere on the board during the game. If this value is missing, it will affect the updates to the player. Lastly, the **current_state** table has the player_id as both the primary and foreign key which can be linked to the player table.

The is_in_jail table updates the jail status for each player after they finish a round. This table has player_id as both primary key and foreign key which can be linked back to the player table. The is_in_jail attribute is set with type boolean and default false, to model for the initial status of players that are not in jail. Throughout the game, if a player lands on 'Go to Jail', the status on this table will be updated accordingly.

3.3 Initial State of the Game

1. **SQL command for the player table:**

```
CREATE TABLE player
(
player_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
player_name VARCHAR(30),
chosen_token VARCHAR(30)
);
INSERT INTO player(
player_name,
chosen_token
)
VALUES
('Mary','Battleship'),
('Bill','Dog'),
('Jane','Car'),
('Norman','Thimble');
```

2. **SQL command for the token table:**

```
CREATE TABLE token
(
token_name VARCHAR(30) NOT NULL PRIMARY KEY
);
INSERT INTO token (
token_name
) VALUES
('Dog'),
('Car'),
('Battleship'),
('Top hat'),
('Thimble'),
('Boot');
```

3. SQL command for location table

```
CREATE TABLE location
(location_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
location_name VARCHAR(50),
location_type VARCHAR(30));

INSERT INTO location
(location_name,location_type)
VALUES ('GO','Bonus'),
('Kilburn','Property'),
('Chance 1','Bonus'),
('Uni Place','Property'),
('In Jail','Bonus'),
('Victoria','Property'),
('Community Chest 1','Bonus'),
('Piccadilly','Property'),
('Free Parking','Bonus'),
('Oak House','Property'),
('Chance 2','Bonus'),
('Owens Park','Property'),
('Go To Jail','Bonus'),
('AMBS','Property'),
('Community Chest 2','Bonus'),
('Co-op','Property');
```

4. SQL Command for property table

```
CREATE TABLE property
(property_id INT NOT NULL,
property_name VARCHAR(50) NOT NULL PRIMARY KEY,
purchase_cost INT,
color VARCHAR(30),
property_owner INT DEFAULT 0,
FOREIGN KEY(property_id) REFERENCES location(location_id)
);
```

```
INSERT INTO property(
property_id,
 property_name,
 purchase_cost,
 color,
 property_owner
 )
VALUES
    (10,'Oak House',100,'Orange',4),
    (12,'Owens Park',30,'Orange',4),
    (14,'AMBS',400,'Blue',0),
    (16,'Co-op',30,'Blue',3),
    (2,'Kilburn',120,'Yellow',0),
    (4,'Uni Place',100,'Yellow',1),
    (6,'Victoria',75,'Green',2),
    (8,'Piccadilly',35,'Green',0);
```

5. SQL command for bonus table

```
CREATE TABLE bonus(
bonus_name VARCHAR(30) PRIMARY KEY,
bonus_desc VARCHAR(100),
bonus_id INT NOT NULL,
FOREIGN KEY (bonus_id) REFERENCES location(location_id)
);
 INSERT INTO bonus(
    bonus_name,
    bonus_desc,
    bonus_id
    )
    VALUES
    ('Chance 1','Pay each of the players £50',3),
    ('Chance 2','Move forward 3 spaces',11),
    ('Community Chest 1','For winning a Beauty Contest, you win £100',7),
    ('Community Chest 2','Your library books are overdue. Play a fine of
£30',15),
    ('In Jail','No action',5)
    ('Free Parking','No action',9),
    ('Go to Jail','Go to Jail, do not pass GO, do not collect £200',13),
    ('GO','Collect £200',1);
```

6. SQL command for is_in_jail (Jail Status) table

```
CREATE TABLE is_in_jail
(player_id INT NOT NULL PRIMARY KEY,
is_in_jail BOOLEAN DEFAULT FALSE,
FOREIGN KEY (player_id) REFERENCES player(player_id)
);
INSERT INTO is_in_jail(
player_id
) VALUES (1),(2),(3),(4);
```

Default jail status is false.

7. SQL command for current bank balance

```
CREATE TABLE current_bank_balance
(player_id INT NOT NULL PRIMARY KEY,
bank_balance INT NOT NULL DEFAULT 0,
FOREIGN KEY (player_id) REFERENCES player(player_id)
) ;

INSERT INTO current_bank_balance(
player_id,
bank_balance)
VALUES
(1,190),
(2,500),
(3,150),
(4,250);
```

8. SQL command for current location

```
CREATE TABLE current_location(
player_id INT NOT NULL PRIMARY KEY,
location_id INT NOT NULL.
FOREIGN KEY (player_id) REFERENCES player(player_id)
);

INSERT INTO current_location(
player_id,
location_id
) VALUES
(1,9),
(2,12),
(3,14),
(4,2);
```

9. SQL command for current_state

```
CREATE TABLE current_state(
 player_id INT NOT NULL PRIMARY KEY,
 round_no INT,
 dice_thrown INT,
 location_id INT,
 bank_balance INT,
 FOREIGN KEY (player_id) REFERENCES player(player_id)
 );

 INSERT INTO current_state (
 player_id,
 round_no,
 dice_thrown,
 location_id,
 bank_balance
 ) VALUES
 (1,0,0,9,190),
 (2,0,0,12,500),
 (3,0,0,14,150),
 (4,0,0,2,250);
```

Round no and dice thrown is set to 0 as this is the initial state of the players.

10. SQL command for audit trail table

```
CREATE TABLE audit_trail(
round_no INT ,
player_id INT,
dice_throw INT,
location_id INT,
bank_balance INT
 );
```

Entries into this table will be triggered after the current_state table is updated

4.1 SQL Commands for Gameplay Updates

4.1.1 Stored procedure 1: Updating players' location

Firstly, a stored procedure is created to update the current location of players based on dice thrown by them

```
DELIMITER /
 CREATE PROCEDURE new_location(IN param_player_id INT, IN param_dice_thrown INT,
                               IN param_round_no INT, OUT c_player_id INT,
                               OUT new_location_id INT, OUT c_dice_thrown INT,
                               OUT c_round_no INT)
 BEGIN
    DECLARE in_jail BOOLEAN DEFAULT FALSE;
    DECLARE location INT;
    DECLARE n_balance INT;
    DECLARE loc_type INT;
SELECT is_in_jail INTO @in_jail FROM is_in_jail
      WHERE player_id = param_player_id;
SELECT location_id INTO @location FROM current_location
      WHERE player_id = param_player_id;
SELECT bank_balance INTO @n_balance FROM current_bank_balance bb
      WHERE bb.player_id = param_player_id;
SELECT location_type INTO @loc_type FROM current_location
      WHERE player_id = param_player_id;

    SET c_round_no = param_round_no;
    SET c_player_id = param_player_id;
    SET c_dice_thrown = param_dice_thrown;

--to check whether player is in jail or not, if in_jail, player will stay at in
--jail location, otherwise, they get their location updated
    IF @in_jail = 1 THEN SET new_location_id = @location;
             ELSE SET new_location_id = @location + param_dice_thrown ;
             END IF;

--since the location is updated arithmetically, for cases where adding param_dice_thrown
--increases location_id beyond max location_id (which is 16), 16 is deducted, such that
--player moves around the board in a circle
      IF new_location_id > 16 THEN SET new_location_id = new_location_id - 16;
             ELSE SET new_location_id = new_location_id;
        END IF;
```

```
--to account for cases where player lands on Go to Jail,
--player’s in_jail status gets updated:
      IF new_location_id = 13 THEN SET @in_jail = 1;
            END IF;
--player’s new location gets updated to In Jail, at location_id = 5
    IF new_location_id = 13 THEN SET new_location_id = 5;
            END IF;
--to change in jail status to 0 for cases where current player was in jail, and throws a 6
    IF @in_jail = 1 AND param_dice_thrown = 6 THEN SET @in_jail = 0;
            END IF;
--to account for Chance 2, where player moves forward 3 spaces
    IF new_location_id = 11 THEN SET new_location_id = new_location_id + 3;
            END IF;
--to account for cases where players passes GO
IF (@location > new_location_id AND @in_jail = 0) THEN SET @n_balance = @n_balance + 200;
ELSE SET @n_balance = @n_balance;
END IF;
--updating current_location table to reflect current player’s new location
UPDATE current_location
SET location_id = new_location_id WHERE player_id = param_player_id;
--updating is_in_jail table to reflect current player’s jail status
UPDATE is_in_jail
SET is_in_jail = @in_jail WHERE player_id = param_player_id;
--updating current_bank_balance table to reflect current player’s new bank balance
UPDATE
current_bank_balance
SET bank_balance = @n_balance WHERE player_id = param_player_id;

END /
DELIMITER ;
```

4.1.2 Stored procedure 2: Updating bank balance for other players when someone else lands on Chance 1: pay each of the other players £50.

```
DELIMITER /
 CREATE PROCEDURE bonus(IN param_player_id INT)
 BEGIN
      UPDATE current_bank_balance
   SET bank_balance = bank_balance + 50 WHERE player_id <> param_player_id;
END /
DELIMITER ;
```

4.1.3 Stored procedure 3: Accounting for cases where double rent is charged

To find out whether someone owns both properties of the same colour, an IF statement is run to check whether the distinct count of the property owner for a particular colour is equal to 1. (*Note that the default value for property_owner is 0 to denote that no one owns that property*). The count distinct property_owner checks whether there is only 1 distinct value across all properties of the same colour. The sum property_owner <> 0 ensures that someone indeed owns the property. (*This is because, for cases where both properties of a particular color have no owner, they both will have property_owner = 0, thus, will meet the first condition of distinct count property_owner = 1, but we wouldn't want to double the purchase_cost for such cases).* When both conditions are met, the multiplier for rent is set to 2, otherwise it is set to 1.

This multiplier is later used to update the rent payable amount in the play_update stored procedure.

```
DELIMITER /
--outputs multiplier based on the colour of the property and whether the same owner owns it
CREATE PROCEDURE double_rent(OUT o_multiplier INT,OUT b_multiplier INT,
                             OUT y_multiplier INT,OUT g_multiplier INT)
BEGIN
   DECLARE l_orange INT;
   DECLARE s_orange INT;
   DECLARE l_blue INT;
   DECLARE l_yellow INT;
   DECLARE l_green INT;
   DECLARE s_blue INT;
   DECLARE s_yellow INT;
   DECLARE s_green INT;
SELECT COUNT(DISTINCT property_owner) INTO @l_orange FROM property WHERE color = 'Orange';
SELECT SUM(property_owner) INTO @s_orange FROM property WHERE color = 'Orange';
SELECT COUNT(DISTINCT property_owner) INTO @l_blue FROM property WHERE color = 'Blue';
SELECT SUM(property_owner) INTO @s_blue FROM property WHERE color = 'Blue';
SELECT COUNT(DISTINCT property_owner) INTO @l_yellow FROM property WHERE color = 'Yellow';
SELECT SUM(property_owner) INTO @s_yellow FROM property WHERE color = 'Yellow';
SELECT COUNT(DISTINCT property_owner) INTO @l_green FROM property WHERE color = 'Green';
SELECT SUM(property_owner) INTO @s_green FROM property WHERE color = 'Green';

IF (@l_orange = 1 AND @s_orange<>0) THEN SET o_multiplier=2;
 ELSE SET o_multiplier=1;
 END IF;

 IF (@l_blue = 1 AND @s_blue <>0) THEN SET b_multiplier=2;
 ELSE SET b_multiplier=1;
 END IF;
 IF (@l_yellow = 1 AND @s_yellow <>0) THEN SET y_multiplier=2;
 ELSE SET y_multiplier=1;
 END IF;

 IF (@l_green = 1 AND @s_green <> 0) THEN SET g_multiplier=2;
 ELSE SET g_multiplier=1;
 END IF;

 END /
DELIMITER ;
```

4.1.3 Stored procedure 3: to update changes to the tables after each move

```
DELIMITER /
CREATE PROCEDURE play_update (IN param_player_id INT,IN dice_thrown INT,
                              OUT new_bank_balance INT, OUT new_owner INT,
                              OUT rent_earnings INT, OUT multiplier INT)
BEGIN
    DECLARE l_location INT;
    DECLARE l_balance INT;
    DECLARE l_property_cost INT;
    DECLARE l_prop_owner INT;
    DECLARE l_loc_type INT;
    DECLARE l_player_count INT;
    DECLARE addition INT;
    DECLARE l_color INT;

SELECT location_id INTO @l_location FROM current_location
   WHERE player_id = param_player_id;
SELECT bank_balance INTO @l_balance FROM current_bank_balance bb
   WHERE bb.player_id = param_player_id;
SELECT purchase_cost INTO @l_property_cost FROM property
   WHERE property_id = @l_location;
SELECT property_owner INTO @l_prop_owner FROM property
   WHERE property_id = @l_location;
SELECT location_type INTO @l_loc_type FROM location
   WHERE location_id = @l_location;
SELECT COUNT(*) INTO @l_player_count FROM current_location WHERE player_id IS NOT NULL;
SELECT color INTO @l_color FROM property WHERE property_id = @l_location;
```

```
-- to account for multiplier of rent by color
CALL double_rent(@o_multiplier,@b_multiplier,@y_multiplier,@g_multiplier);
IF @l_color = 'Orange' THEN SET multiplier = @o_multiplier;
    ELSEIF @l_color = 'Blue' THEN SET multiplier = @b_multiplier;
    ELSEIF @l_color = 'Yellow' THEN SET multiplier = @y_multiplier;
    ELSEIF @l_color = 'Green' THEN SET multiplier = @g_multiplier;
   END IF;

-- to account for rent earnings
IF (@l_prop_owner <> 0 AND @l_prop_owner <> param_player_id AND @l_loc_type = 'Property' AND
dice_thrown <>6)
      THEN SET rent_earnings = @l_property_cost*multiplier;
      ELSE SET rent_earnings = 0;
END IF;

-- if dice is not 6, update bank balance and property ownership, else no updates
-- if property owned by self, no updates on bank balance
IF (@l_loc_type = 'Property' AND @l_prop_owner = param_player_id AND dice_thrown <> 6 )
      THEN SET new_bank_balance = @l_balance;
-- if property doesn’t have owner, deduct purchase_cost
ELSEIF (@l_loc_type = 'Property' AND @l_prop_owner = 0 AND dice_thrown <>6 )
      THEN SET new_bank_balance = @l_balance - @l_property_cost;
-- if property owned by someone other then current player, deduct purchase_cost*multiplier
ELSEIF (@l_loc_type = 'Property' AND @l_prop_owner <> 0 AND @l_prop_owner != param_player_id
AND dice_thrown <>6 )
      THEN SET new_bank_balance = @l_balance - (@l_property_cost*multiplier);
-- if bonus at Chance 1, minus £50*count of players in the game
ELSEIF (@l_loc_type = 'Bonus' AND @l_location = 3 AND dice_thrown <>6 )
      THEN SET new_bank_balance = @l_balance - ((@l_player_count-1)*50);
-- if bonus at Community Chest 1, earn £100
ELSEIF (@l_loc_type = 'Bonus' AND @l_location = 7 AND dice_thrown <>6 )
      THEN SET new_bank_balance = @l_balance + 100;
-- if bonus at Community Chest 2, pay library fine of £30
ELSEIF (@l_loc_type = 'Bonus' AND @l_location = 15 AND dice_thrown <>6)
      THEN SET new_bank_balance = @l_balance - 30;
ELSE SET new_bank_balance = @l_balance;
  END IF;
```

```
-- to account for payment of 50 to other players when land on Chance 1
IF (@l_location = 3 AND dice_thrown <>6) THEN CALL bonus(param_player_id);
END IF;

-- to update owner status
-- if property owned by self, no actions
IF (@l_loc_type = 'Property' AND @l_prop_owner = param_player_id AND dice_thrown <>6)
       THEN SET new_owner = @l_prop_owner;
-- if property doesn’t have owner, update owner as current player
ELSEIF (@l_loc_type = 'Property' AND @l_prop_owner = 0 AND dice_thrown <>6)
       THEN SET new_owner = param_player_id;
-- if property has owner, no actions
ELSEIF (@l_loc_type = 'Property' AND @l_prop_owner <> 0 AND @l_prop_owner != param_player_id
AND dice_thrown <>6)
       THEN SET new_owner = @l_prop_owner;
-- if rolls a 6, no action
ELSEIF (@l_loc_type = 'Bonus' AND dice_thrown <>6) THEN SET new_owner = 0;
ELSE SET new_owner = @l_prop_owner;
END IF;

UPDATE property
       SET property_owner = new_owner WHERE property_id = @l_location;
UPDATE current_bank_balance
       SET bank_balance = bank_balance + rent_earnings WHERE player_id = @l_prop_owner;
UPDATE current_bank_balance
       SET bank_balance = new_bank_balance WHERE player_id = param_player_id;

END /
DELIMITER ;
```

4.1.5 Stored procedure 5: to update the current state table

```
DELIMITER /
CREATE PROCEDURE update_current_state (IN param_player_id INT)
BEGIN
UPDATE current_state
SET
    round_no = @c_round_no,
    dice_thrown = @c_dice_thrown,
    location_id = @new_location_id,
    bank_balance = @new_bank_balance
WHERE player_id = param_player_id;
END/
DELIMITER ;
```

4.1.6 Stored procedure 6: To call all the above stored procedures to update the relevant tables

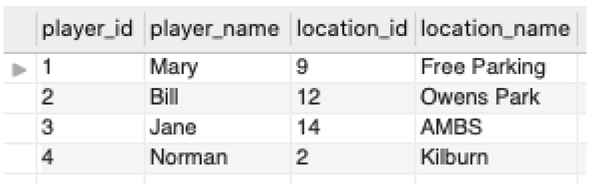
```
DELIMITER /
CREATE PROCEDURE game_play(IN param_player_id INT, IN param_dice_thrown INT,
                           IN param_round_no INT)
BEGIN
CALL new_location(param_player_id,param_dice_thrown,param_round_no,@c_player_id,
                   @new_location_id,@c_dice_thrown,@c_round_no );
CALL play_update (param_player_id,param_dice_thrown,@new_bank_balance,@new_owner,
                   @rent_earnings,@multiplier);
CALL update_current_state(param_player_id);
END /
DELIMITER ;
```

4.1.7 Trigger to update audit trail table

```
DELIMITER $$
CREATE TRIGGER audit_trail
AFTER UPDATE ON current_state FOR EACH ROW
BEGIN
INSERT INTO audit_trail (round_no,player_id,dice_throw,location_id,bank_balance)
VALUES (@c_round_no,@c_player_id,@c_dice_thrown,@new_location_id,@new_bank_balance);
END$$
DELIMITER ;
```

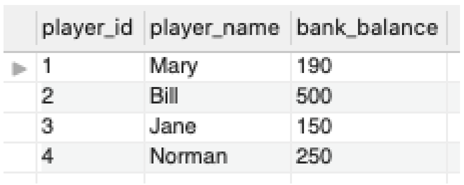
4.2 Round 1

The following are current state of the tables:



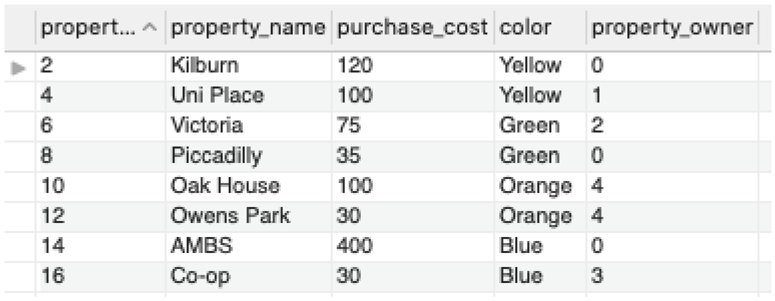
player_id	player_name	location_id	location_name
1	Mary	9	Free Parking
2	Bill	12	Owens Park
3	Jane	14	AMBS
4	Norman	2	Kilburn

Figure 4.2.1: Players' current location before Round 1



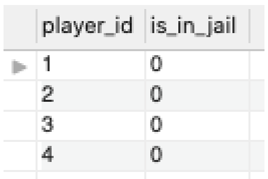
player_id	player_name	bank_balance
1	Mary	190
2	Bill	500
3	Jane	150
4	Norman	250

Figure 4.2.2: current_bank_balance before Round 1



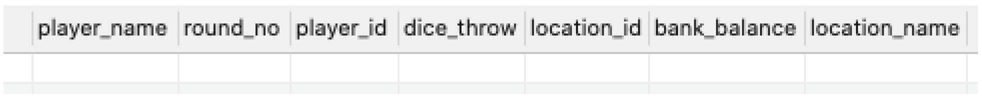
propert...	property_name	purchase_cost	color	property_owner
2	Kilburn	120	Yellow	0
4	Uni Place	100	Yellow	1
6	Victoria	75	Green	2
8	Piccadilly	35	Green	0
10	Oak House	100	Orange	4
12	Owens Park	30	Orange	4
14	AMBS	400	Blue	0
16	Co-op	30	Blue	3

Figure 4.2.3: property table before Round 1



player_id	is_in_jail
1	0
2	0
3	0
4	0

Figure 4.2.4: Jail status before Round 1



player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name

Figure 4.2.5: Audit trail table before Round 1.

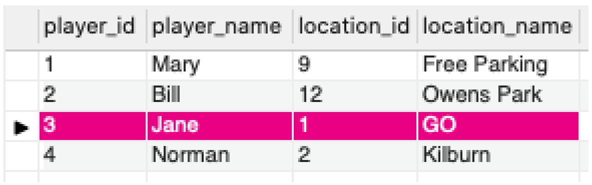
All the tables have been populated with data to reflect the current state of the players. An empty audit trail table is created to prepare for triggers to insert data at the end of every round for each player.

G1: Jane Rolls a 3 Table Updates

Query:

```
CALL game_play(3,3,1);

select
   cl.*,
   l.location_name
from current_location cl
left join location l
   on cl.location_id = l.location_id;

select * from current_bank_balance;

select
   p.player_name,
   au.*,
   l.location_name
from audit_trail au
left join player p
   on p.player_id = au.player_id
left join location l
   on l.location_id = au.location_id;
```

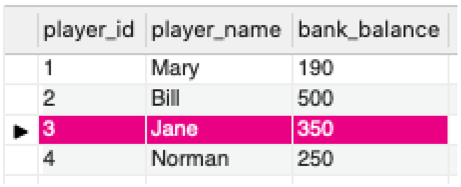
When Jane rolls a 3, she moves to GO, with location_id = 1, from her previous location AMBS, with location_id = 14. This gets updated in the current_location table, and can be observed in Figure 4.2.6



	player_id	player_name	location_id	location_name
	1	Mary	9	Free Parking
	2	Bill	12	Owens Park
▶	3	Jane	1	GO
	4	Norman	2	Kilburn

Figure 4.2.6: updated location after Jane moves 3

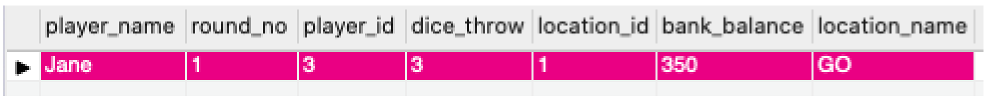
Since she landed on GO, she gets to collect £200, so her bank balance gets updated to £350 (Figure 4.2.7) , from her previous balance of £150 (Figure 4.2.2).



	player_id	player_name	bank_balance
	1	Mary	190
	2	Bill	500
▶	3	Jane	350
	4	Norman	250

Figure 4.2.7: current_bank_balance table after Jane moves 3

Her updated location and bank balance at the end of Round 1 gets inserted into the audit_trail table in the Figure 4.2.8 below;



	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
▶	Jane	1	3	3	1	350	GO

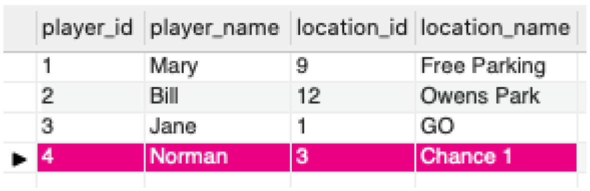
Figure 4.2.8: audit trail table after Jane moves 3

G2: Norman rolls a 1 Table Updates

Query:

```
CALL game_play(4,1,1);
select
   cl.*,
   l.location_name
from current_location cl
   left join location l
      on cl.location_id = l.location_id;

select * from current_bank_balance;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

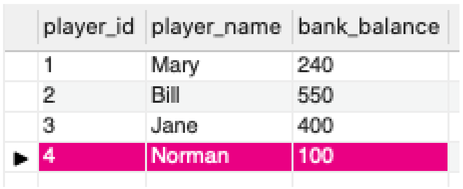
When Norman rolls a 1, he moves from Kilburn (Figure 4.2.1) to Chance 1 (Figure 4.2.9). At this location, Norman needs to pay the other players £50 each.



	player_id	player_name	location_id	location_name
	1	Mary	9	Free Parking
	2	Bill	12	Owens Park
	3	Jane	1	GO
▶	4	Norman	3	Chance 1

Figure 4.2.9: updated current location after Norman moves 1

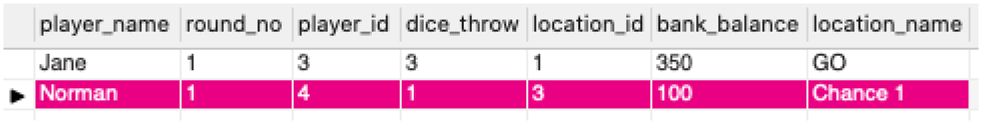
As seen in Figure 4.2.10 below, Norman's bank balance is reduced to £100, while the other players' bank balance increased by £50 each.



	player_id	player_name	bank_balance
	1	Mary	240
	2	Bill	550
	3	Jane	400
▶	4	Norman	100

Figure 4.2.10: Players' updated bank balance after Norman moves 1

The following is the new record inserted into the audit trail table after Norman moves 1 position:



	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
	Jane	1	3	3	1	350	GO
▶	Norman	1	4	1	3	100	Chance 1

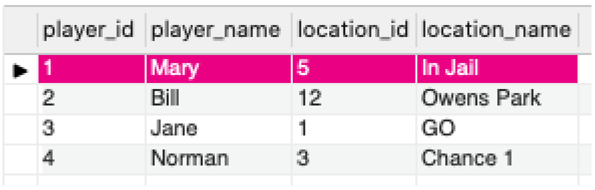
Figure 4.2.11: audit_trail table after Norman moves 1

G3: Mary rolls a 4 Table Updates

Query:

```
CALL game_play(1,4,1);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

Select * from is_in_jail;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

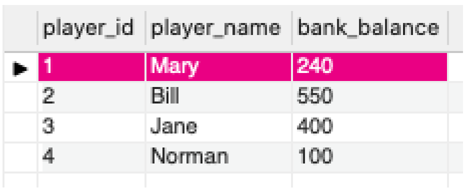
When Mary rolls a 4, she moves from *Free Parking* to *Go to Jail.* Mary is now in jail. As specified in the first stored procedure, (at 4.1.1), Mary's location gets updated to *In Jail,* at location_id = 5.



player_id	player_name	location_id	location_name
1	Mary	5	In Jail
2	Bill	12	Owens Park
3	Jane	1	GO
4	Norman	3	Chance 1

Figure 4.2.12: updated location after Mary rolls a 4

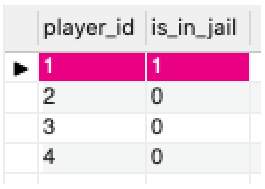
There are no changes to the players' bank balances:



player_id	player_name	bank_balance
1	Mary	240
2	Bill	550
3	Jane	400
4	Norman	100

Figure 4.2.13: Updated bank balance after Mary rolls a 4

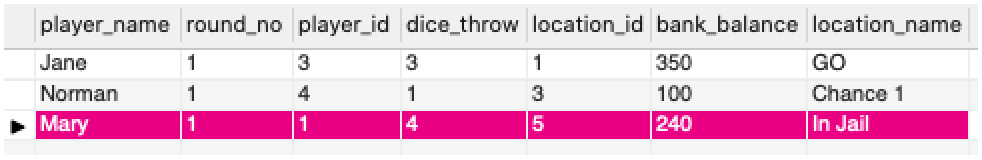
And Mary's jail status gets updated in the is_in_jail table. This will prevent any location updates for Mary in the next rounds, until she throws a 6.



player_id	is_in_jail
1	1
2	0
3	0
4	0

Figure 4.2.14: is_in_jail table after Mary rolls a 4

Mary’s latest location status gets recorded in the audit trail table as shown below:



	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
	Jane	1	3	3	1	350	GO
	Norman	1	4	1	3	100	Chance 1
▶	Mary	1	1	4	5	240	In Jail

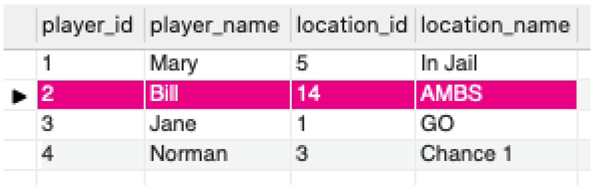
Figure 4.2.15: audit_trail table after Mary rolls a 4

G4: Bill rolls a 2 Table Updates

Query

```
CALL game_play(2,2,1);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

Select * from is_in_jail;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

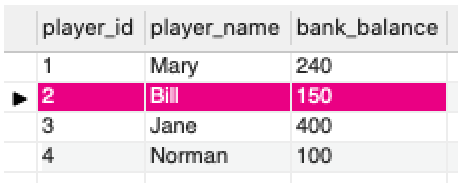
After rolling a 2, Bill lands on *AMBS* from *Owens Park,* as can be observed in the updated location below:



	player_id	player_name	location_id	location_name
	1	Mary	5	In Jail
▶	2	Bill	14	AMBS
	3	Jane	1	GO
	4	Norman	3	Chance 1

Figure 4.2.16: updated location after Bill rolls a 2

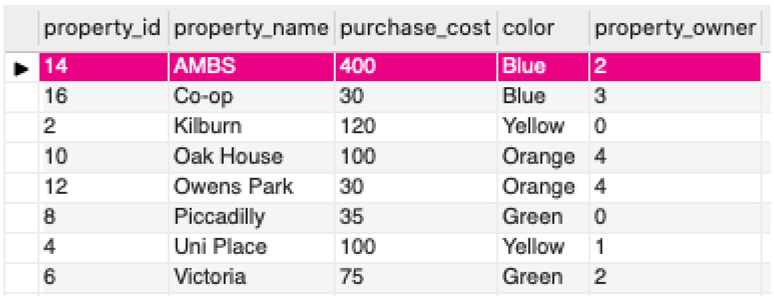
Since no one owns *AMBS,* he needs to pay £400 to purchase it. This gets reflected in the current_bank_balance table where Bill's bank balance gets reduced from £550 (Figure 4.2.13) to £150, as seen below:



	player_id	player_name	bank_balance
	1	Mary	240
▶	2	Bill	150
	3	Jane	400
	4	Norman	100

Figure 4.2.17: current_bank_balance table after Bill rolls a 2

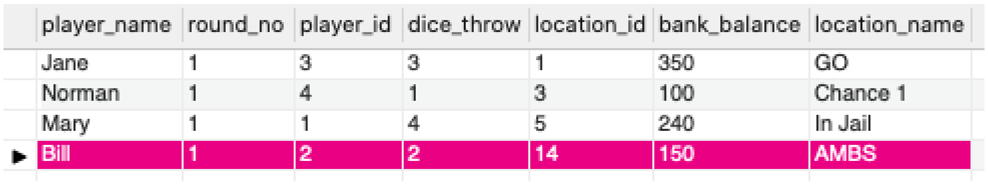
Since no one owns *AMBS* previously, the property table gets updated to reflect that Bill is now the new owner of *AMBS.*



	property_id	property_name	purchase_cost	color	property_owner
▶	14	AMBS	400	Blue	2
	16	Co-op	30	Blue	3
	2	Kilburn	120	Yellow	0
	10	Oak House	100	Orange	4
	12	Owens Park	30	Orange	4
	8	Piccadilly	35	Green	0
	4	Uni Place	100	Yellow	1
	6	Victoria	75	Green	2

Figure 4.2.18: property table after Bill rolls a 2

His updated state gets recorded in the audit trail table at the end of his round:



	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
	Jane	1	3	3	1	350	GO
	Norman	1	4	1	3	100	Chance 1
	Mary	1	1	4	5	240	In Jail
▶	Bill	1	2	2	14	150	AMBS

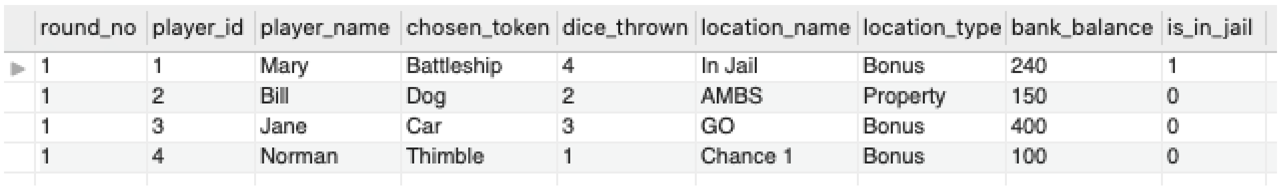
Figure 4.2.19: audit_trail table after Bill rolls a 2

Updated GameView after Round 1

SQL Query for Game View Update

```
CREATE VIEW gameView AS
SELECT
       cs.round_no,
       cs.player_id,
       p.player_name,
       p.chosen_token,
       cs.dice_thrown,
       l.location_name,
       l.location_type,
       cs.bank_balance,
       ij.is_in_jail

FROM current_state cs
       LEFT JOIN player p
              ON cs.player_id = p.player_id
       LEFT JOIN location l
              ON l.location_id = cs.location_id
       LEFT JOIN is_in_jail ij
              ON ij.player_id = cs.player_id;

SELECT * FROM gameView
```

gameView after Round 1



	round_no	player_id	player_name	chosen_token	dice_thrown	location_name	location_type	bank_balance	is_in_jail
▶	1	1	Mary	Battleship	4	In Jail	Bonus	240	1
	1	2	Bill	Dog	2	AMBS	Property	150	0
	1	3	Jane	Car	3	GO	Bonus	400	0
	1	4	Norman	Thimble	1	Chance 1	Bonus	100	0

Figure 4.2.20: Game Play Round 1 View

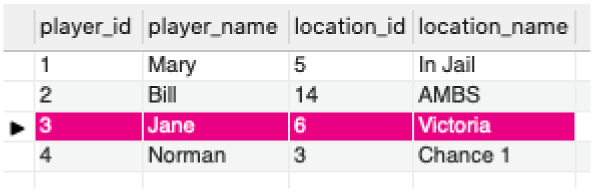
4.3 Round 2

G5 Jane rolls a 5 Table Updates

Query

```
CALL game_play(3,5,2);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

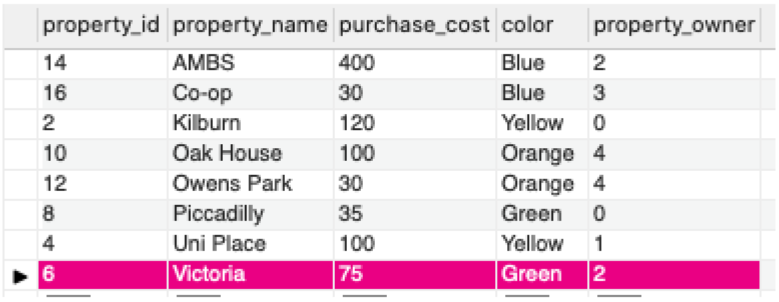
After rolling a 5, Jane moves to *Victoria,* from her previous location at *GO*.



player_id	player_name	location_id	location_name
1	Mary	5	In Jail
2	Bill	14	AMBS
3	Jane	6	Victoria
4	Norman	3	Chance 1

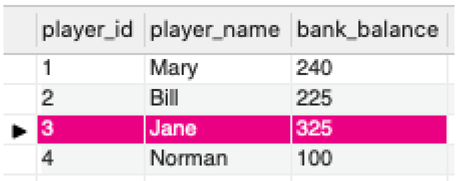
Figure 4.3.1: Current location after Jane rolls a 5

Since Victoria is owned by Bill (as seen in 4.3.2), Jane will have to pay Bill £75. This gets updated in the current_bank_balance table in Figure 4.3.3, where Jane's balance is reduced by £75, while Bill's balance gets increased by £75.



property_id	property_name	purchase_cost	color	property_owner
14	AMBS	400	Blue	2
16	Co-op	30	Blue	3
2	Kilburn	120	Yellow	0
10	Oak House	100	Orange	4
12	Owens Park	30	Orange	4
8	Piccadilly	35	Green	0
4	Uni Place	100	Yellow	1
6	Victoria	75	Green	2

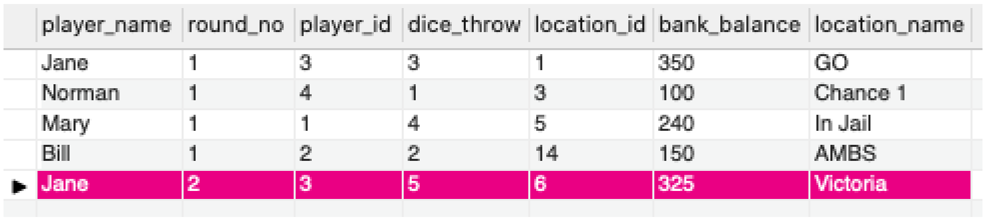
Figure 4.3.2: Current state of property table



player_id	player_name	bank_balance
1	Mary	240
2	Bill	225
3	Jane	325
4	Norman	100

Figure 4.3.3: current_bank_balance table after Jane rolls a 5

Updated audit_Trail table:



	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
	Jane	1	3	3	1	350	GO
	Norman	1	4	1	3	100	Chance 1
	Mary	1	1	4	5	240	In Jail
	Bill	1	2	2	14	150	AMBS
▶	Jane	2	3	5	6	325	Victoria

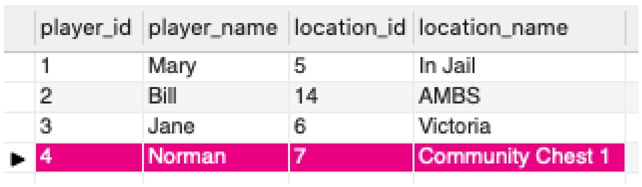
Figure 4.3.4: audit_trail table after Jane rolls a 5

G6 Norman rolls a 4 Table Updates

Query

```
CALL game_play(4,4,2);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

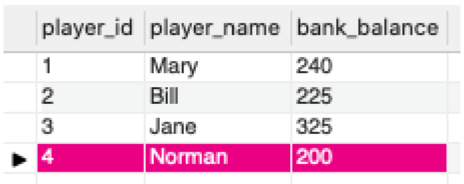
After rolling a 4, Norman moves to *Community Chest 1.*



	player_id	player_name	location_id	location_name
	1	Mary	5	In Jail
	2	Bill	14	AMBS
	3	Jane	6	Victoria
▶	4	Norman	7	Community Chest 1

Figure 4.3.5: current_location table after Norman rolls a 4

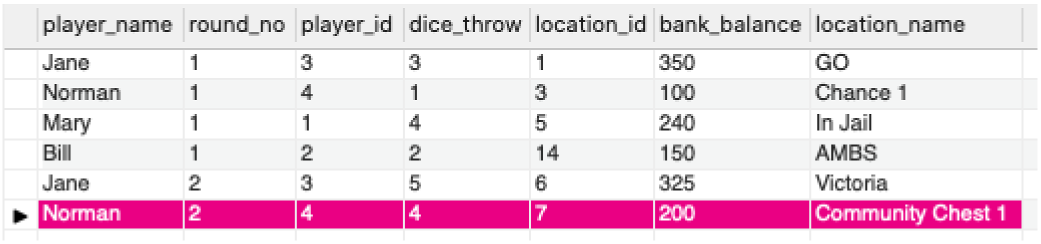
At this location, Norman gets £100 for winning a beauty contest. His bank balance gets updated from £100 to £200 as seen in Figure 4.3.6



	player_id	player_name	bank_balance
	1	Mary	240
	2	Bill	225
	3	Jane	325
▶	4	Norman	200

Figure 4.3.6: current_bank_balance table after Norman rolls a 4

His new updates gets recorded in the audit_trail table, as seen in 4.3.7:



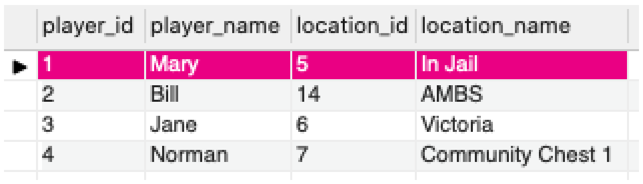
	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
	Jane	1	3	3	1	350	GO
	Norman	1	4	1	3	100	Chance 1
	Mary	1	1	4	5	240	In Jail
	Bill	1	2	2	14	150	AMBS
	Jane	2	3	5	6	325	Victoria
▶	Norman	2	4	4	7	200	Community Chest 1

Figure 4.3.7: audit_trail table after after Norman rolls a 4

G7 Mary rolls a 6, then a 5 Table Updates
Query

```
CALL game_play(1,6,2);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

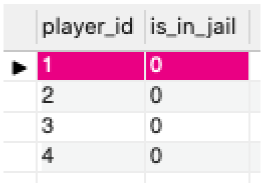
After rolling a 6, Mary gets released from prison. Her location remains at In Jail, as the logic applied was that if a player is in jail, they need to roll a 6 to get out, and immediately roll again. Only the dice thrown after first rolling a 6 will move players who were in jail.



player_id	player_name	location_id	location_name
1	Mary	5	In Jail
2	Bill	14	AMBS
3	Jane	6	Victoria
4	Norman	7	Community Chest 1

Figure 4.3.8: current_location table after Mary rolls a 6

Notice that her is_in_jail status has been changed to 0. This will enable Mary's moves after the next throw to be updated into the tables.



player_id	is_in_jail
1	0
2	0
3	0
4	0

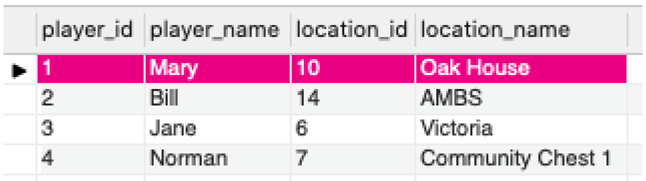
Figure 4.3.9: is_in_jail table after Mary rolls a 6

Mary rolls a 5

Query

```
CALL game_play(1,5,2);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

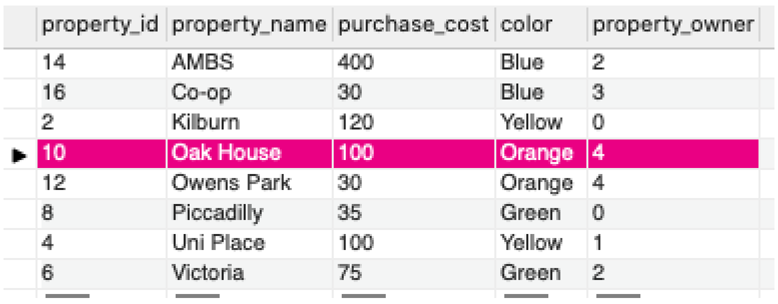
After being released from prison, Mary rolled a 5. This moves her from *In Jail* to Oak House, as can be seen in the following Figure 4.3.10



	player_id	player_name	location_id	location_name
▶	1	Mary	10	Oak House
	2	Bill	14	AMBS
	3	Jane	6	Victoria
	4	Norman	7	Community Chest 1

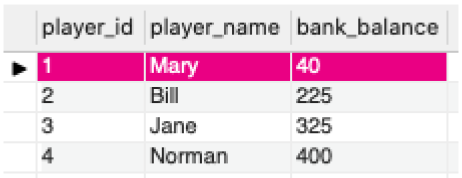
Figure 4.3.10: Players current location after Mary rolls a 5

Oak House is owned by Norman as observed in the following Figure 4.3.11. As can be seen in the same figure, Norman owns both properties in the Orange colour group. Thus, Mary will have to pay double rent to Norman. This is reflected in updated current_bank_balance in Figure 4.3.12, where Mary's balance dropped from £240 to £40, and Norman's balance increased from £200 to £400.



	property_id	property_name	purchase_cost	color	property_owner
	14	AMBS	400	Blue	2
	16	Co-op	30	Blue	3
	2	Kilburn	120	Yellow	0
▶	10	Oak House	100	Orange	4
	12	Owens Park	30	Orange	4
	8	Piccadilly	35	Green	0
	4	Uni Place	100	Yellow	1
	6	Victoria	75	Green	2

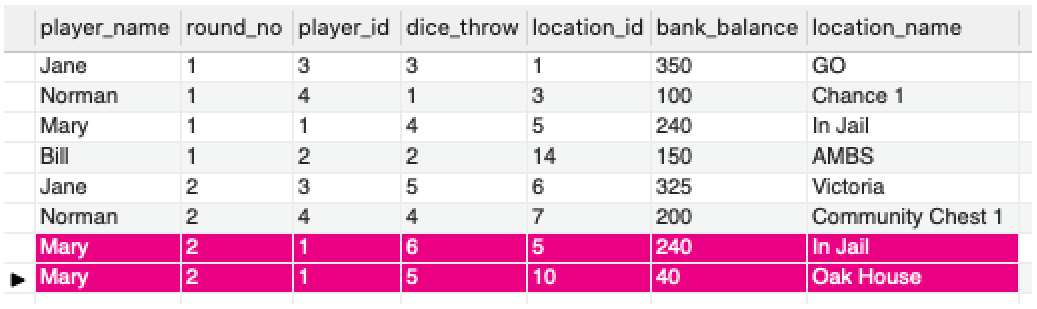
Figure 4.3.11: property table after Mary rolls a 5



	player_id	player_name	bank_balance
▶	1	Mary	40
	2	Bill	225
	3	Jane	325
	4	Norman	400

Figure 4.3.12: current_bank_balance table after Mary rolls a 5

Updated audit_Trail table:



player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
Jane	1	3	3	1	350	GO
Norman	1	4	1	3	100	Chance 1
Mary	1	1	4	5	240	In Jail
Bill	1	2	2	14	150	AMBS
Jane	2	3	5	6	325	Victoria
Norman	2	4	4	7	200	Community Chest 1
Mary	2	1	6	5	240	In Jail
Mary	2	1	5	10	40	Oak House

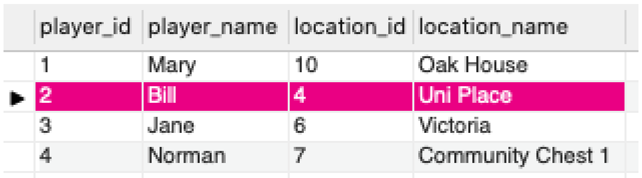
Figure 4.3.13: audit_trail table after Mary rolls a 5

G8 Bill rolls a 6, then a 3 Table Updates

Query

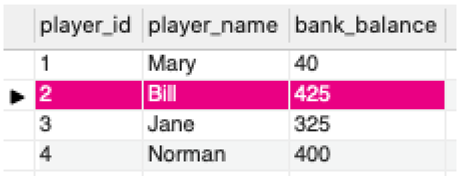
```
CALL game_play(2,6,2);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

After rolling a 6, Bill moves from *AMBS* to *Uni Place,* and passes *GO* in the process. Since he passed *GO,* he gets to collect £200. His bank balance increased to £425 from £225 (Figure 4.3.15). Even though *Uni Place* has an owner (Figure 4.3.16), Bill does not have to pay rent as he gets another turn for rolling a 6.



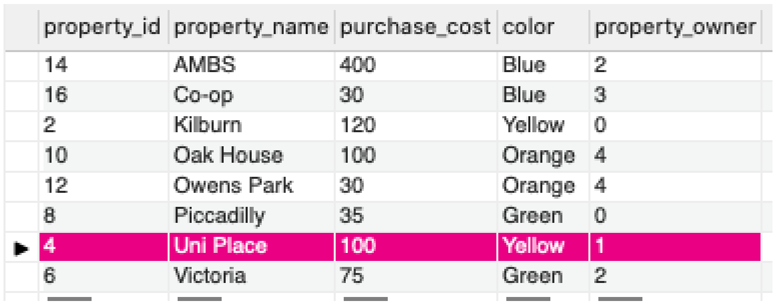
	player_id	player_name	location_id	location_name
	1	Mary	10	Oak House
▶	2	Bill	4	Uni Place
	3	Jane	6	Victoria
	4	Norman	7	Community Chest 1

Figure 4.3.14: current_location table after Bill rolls a 6



	player_id	player_name	bank_balance
	1	Mary	40
▶	2	Bill	425
	3	Jane	325
	4	Norman	400

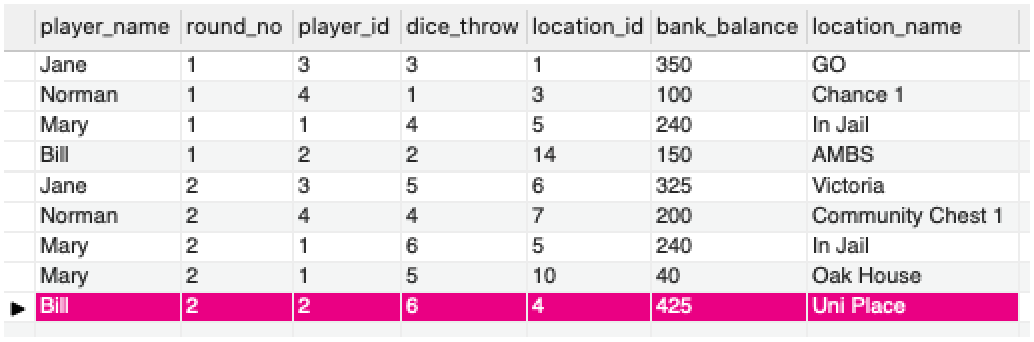
Figure 4.3.15: current_bank_balance table after Bill rolls a 6



	property_id	property_name	purchase_cost	color	property_owner
	14	AMBS	400	Blue	2
	16	Co-op	30	Blue	3
	2	Kilburn	120	Yellow	0
	10	Oak House	100	Orange	4
	12	Owens Park	30	Orange	4
	8	Piccadilly	35	Green	0
▶	4	Uni Place	100	Yellow	1
	6	Victoria	75	Green	2

Figure 4.3.16: property table after Bill rolls a 6

Updated audit_Trail table:



	player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
	Jane	1	3	3	1	350	GO
	Norman	1	4	1	3	100	Chance 1
	Mary	1	1	4	5	240	In Jail
	Bill	1	2	2	14	150	AMBS
	Jane	2	3	5	6	325	Victoria
	Norman	2	4	4	7	200	Community Chest 1
	Mary	2	1	6	5	240	In Jail
	Mary	2	1	5	10	40	Oak House
▶	Bill	2	2	6	4	425	Uni Place

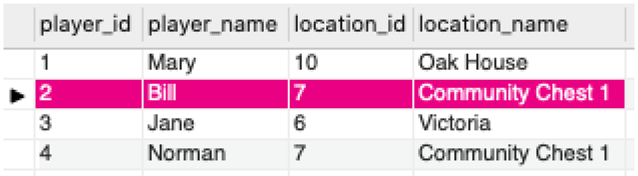
Figure 4.3.17: audit_trail table after Bill rolls a 6

G8 Bill rolls a 3 Table Updates

Query

```
CALL game_play(2,3,2);
select
   cl.*,
   l.location_name
from current_location cl
      left join location l
            on cl.location_id = l.location_id;

select * from current_bank_balance;

select
      p.player_name,
      au.*,
      l.location_name
from audit_trail au
      left join player p on p.player_id = au.player_id
      left join location l on l.location_id = au.location_id;
```

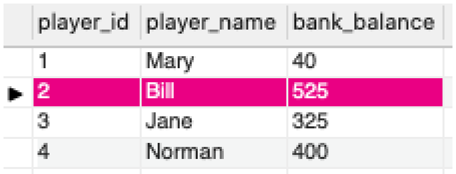
After rolling a 3, Bill now lands on *Community Chest 1* as can be observed in the updated table in Figure 4.3.18:



	player_id	player_name	location_id	location_name
	1	Mary	10	Oak House
▶	2	Bill	7	Community Chest 1
	3	Jane	6	Victoria
	4	Norman	7	Community Chest 1

Figure 4.3.18: current_location table after Bill rolls a 3

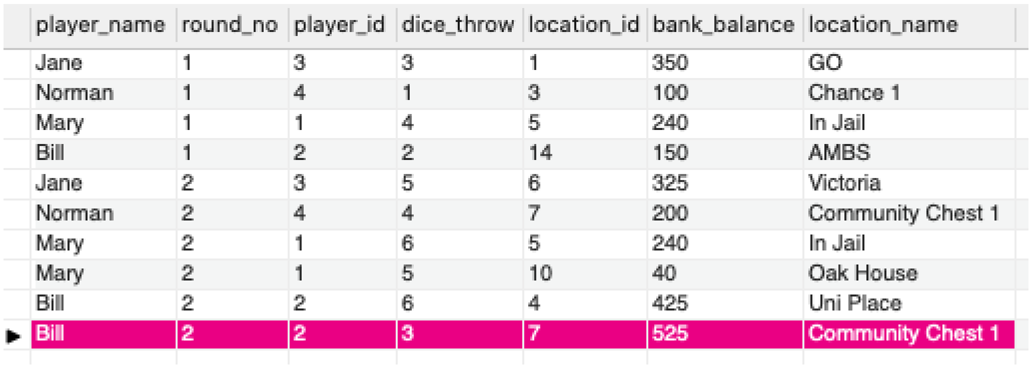
He wins £100 for landing on *Community Chest 1,*



	player_id	player_name	bank_balance
	1	Mary	40
▶	2	Bill	525
	3	Jane	325
	4	Norman	400

Figure 4.3.19: current_bank_balance table after Bill rolls a 3

Updated audit_Trail table:



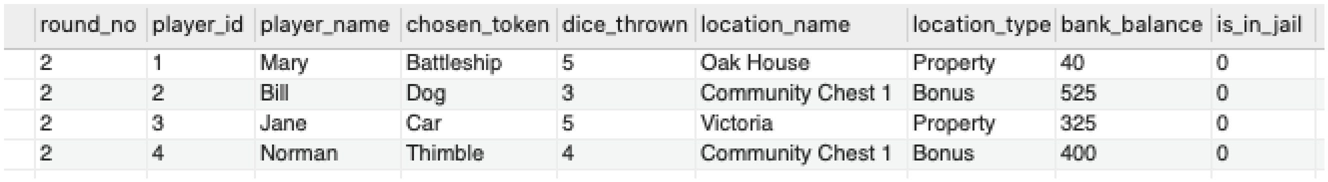
player_name	round_no	player_id	dice_throw	location_id	bank_balance	location_name
Jane	1	3	3	1	350	GO
Norman	1	4	1	3	100	Chance 1
Mary	1	1	4	5	240	In Jail
Bill	1	2	2	14	150	AMBS
Jane	2	3	5	6	325	Victoria
Norman	2	4	4	7	200	Community Chest 1
Mary	2	1	6	5	240	In Jail
Mary	2	1	5	10	40	Oak House
Bill	2	2	6	4	425	Uni Place
Bill	2	2	3	7	525	Community Chest 1

Figure 4.3.20: audit_trail table after Bill rolls a 3

Updated GameView after Round 1

SQL Query for Game View Update

```
SELECT * FROM gameView;
```



round_no	player_id	player_name	chosen_token	dice_thrown	location_name	location_type	bank_balance	is_in_jail
2	1	Mary	Battleship	5	Oak House	Property	40	0
2	2	Bill	Dog	3	Community Chest 1	Bonus	525	0
2	3	Jane	Car	5	Victoria	Property	325	0
2	4	Norman	Thimble	4	Community Chest 1	Bonus	400	0

Figure 4.3.21:Game Play View at the end of Round 2