CONTENT

1 Introduction

Monopoly is a world-renowned economics-themed board game. Players roll dice to move around the game board, buying properties, paying rent and getting chances. In this article, we create a SQL database for Monopoly, which automatically updates by inserting one row in one table. Meanwhile, it provides a review of the previous move and gives each player a final score separately.

This database is a simplified version of Monopoly, with only 16 squares on the game board. In reality, we can have bigger maps with more squares. There are also some differences in the rules. In this database, if a player lands on a property, he has to buy it unless he cannot afford it, whereas in reality, buying or not buying is part of players' game strategy. Players can consider not purchasing the property even though they have enough money in their bank account.

Figure 1. The monopoly board the database based on

2 Entity Relationship Diagram for Database

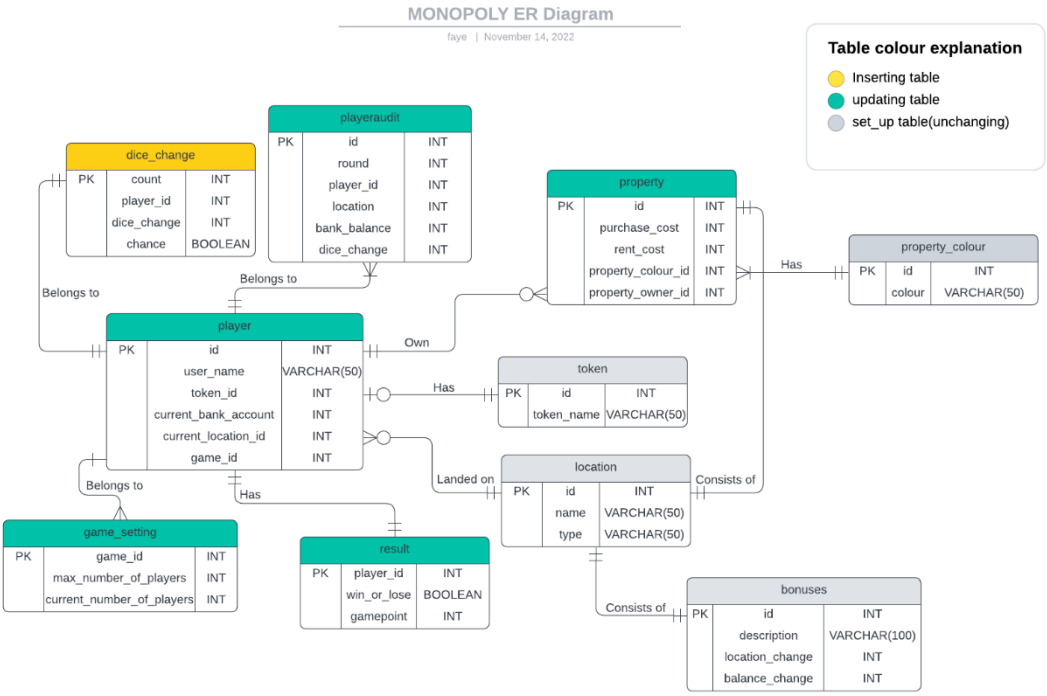

Figure 2. monopoly ER diagram

The design of ER diagram should cover all aspects of our monopoly game. And it would also be nice if it is flexible enough to upgrade to a bigger and more complex monopoly game. Here we have 10 tables, which can be divided into three categories in our monopoly ER diagram.

2.1 Inserting table

The game is played via manually inserting into ***Dice*** table. All attributes in the table are mandatory. The *count* attribute is unique and used to record number of times players rolling. The *dice_change* attribute is equal to the dice number that player rolls. Not all players have the chance to take a move, we need a Boolean type *chance* attribute to record if a player can take a move. For those who are in jail and need a 6 to go out, we'll set their' *chance* to 0.

Table 1. dice_change table setting

dice_change		
PK	count	INT
	player_id	INT
	dice_change	INT
	chance	BOOLEAN

2.2 Changing table

Changing table means tables will change after insert. To start a new monopoly game, we should have a ***game_setting*** table. All attributes in the table are mandatory. With different players and different game starting time, the game should have a unique ID to indicates that game. Thus, we add *game_id* attribute as the primary key for this table. However, the dataset to be tested is limited, we leave this table unlinked and set the value of *game_id* to 1. If the dataset is expanded and more games are recorded in this batabase, we need to connect it to the *player* table and *result* table. The *max_number_of_players* attribute counts the number of players in the current game. If someone goes bankrupt, the value of *current_number_of_players* attribute would automatically minus 1.

Table 2. game_setting table setting

game_setting		
PK	game_id	INT
	max_number_of_players	INT
	current_number_of_players	INT

The ***player*** table is where we store player's game information. All attributes in the table are mandatory. The value of *id*, *user_name* and *token_id* attributes remain unchanged throughout the game. Player's id and *token_id* should be unique. We don't want one token linked to multiple players. The value of *current_bank_account* and *current_location_id* attribute update automatically via player's each move. At the start of the game, the default value for each player's *current_bank_account* and *current_location_id* is 0.

Table 3. player table setting

player		
PK	id	INT
	user_name	VARCHAR(50)
	token_id	INT
	current_bank_account	INT
	current_location_id	INT
	game_id	INT

Games are often competitive in nature, so we need a table called ***result*** to record each player's score. The *win_or_lose* attribute is Boolean type. In this game, players only have two result states, WIN or LOSE, and only the player who remains to the end WIN. The *gamepoint* attribute records each player's score. Here we set up that a player's score = the number of players in the game - the number of players left after that player goes bankrupt. For example, if Faye is the third player to go bankrupt in a 6-player monopoly, she will score 3 points and the player left at the end will score 6 points. This table is automatically insert as the game progresses. Again, because of the limited dataset, *game_id* is not added here to indicate which game these players are playing. As the dataset expands, this table will add a *game_id* column, which will be used as a compound primary key along with *player_id*, and connect to the *game_setting* table.

Table 4. result table setting

result		
PK	player_id	INT
	win_or_lose	BOOLEAN
	gamepoint	INT

We also expect to learn from previous experience, and analyse past mistakes. Thus, we create the ***playeraudit*** table, which is like a web browsing history, it records the *location*, *bank_balance*, *dice_change* of each player after each action. To simplify the code, player dices a 6 to go out the jail is not recorded in this table. Here we use the *round* attribute to count all player's movement. A *round* of the game is defined as each players taking their next turn. This table is automatically insert as the game progresses. All attributes are mandatory.

Table 5. playeraudit table setting

playeraudit		
PK	id	INT
	round	INT
	player_id	INT
	location	INT
	bank_balance	INT
	dice_change	INT

2.3 Set-up table(unchanging table)

Of course we need some default settings to execute our monopoly. Here *token*, *location*, *bonuses* and *property_colour* table are unchanging tables used to set up the game.

The ***token*** table describers the token name, each token name has a unique id. All attributes are mandatory and entered before the game starts. The creation of this table is for further dataset expansion. If more new games are run, it is inevitable that different players will choose the same token. However, the *second normal form* requires us

that each table should describe one entity, and every column in that table should described that entity. Apparently, *token* itself is an independent entity that needs a new table, otherwise it will repeat many times in *player* table when the datasets expansion.

Table 6. token table setting

token		
PK	id	INT
	token_name	VARCHAR(50)

Since there are 16 squares on our monopoly board, we need 16 different ids for each location. So we create the ***location*** table. All the locations are divided into two types according to the game setting, property and bonus. The type attribute is used to store the type of each location. All data in this table are pre-inserted before the game.

Table 7. location table setting

location		
PK	id	INT
	name	VARCHAR(50)
	type	VARCHAR(50)

In order to satisfy the third norm form and reduce the space taken up by the database, we create separate tables for properties and bonuses. The ids in these two tables are also the foreign key from the *location* table while being the primary key, meaning that they can use the ids from the *location* table directly.

In the ***bonuses*** table, the *description* attribute displays each bonus's meaning. We can conclude that all bonus will relate to player's location change or/and balance change. For easier auto-manipulation, we put values into the location_change and balance_change attributes in advance.

Table 8. bonuses table setting

bonuses		
PK	id	INT
	description	VARCHAR(100)
	location_change	INT
	balance_change	INT

When player lands on a ***property*** type location, he will purchase the house or pay the rent bill. Thus, we need the *purchase_cost* and *rent_cost* attributes in the *property* table. The *property_owner_id* indicates the owner of the property, it should be NULL at the beginning as no one purchase the property. The *property_colour_id* attribute indicates the colour of each property.

Table 9. property table setting

property		
PK	id	INT
	purchase_cost	INT
	rent_cost	INT
	property_colour_id	INT
	property_owner_id	INT

Apparently for the 2nd norm form, we need a new table called ***property_colour*** to describe the colour type. The id attributes are unique primary key to each colour.

Table 10. property_colour table setting

property_colour		
PK	id	INT
	colour	VARCHAR(50)

2.4 logistic relations via tables

- ✧ In a separate game, each player should have a unique token, and a token can only be owned by one player.
- ✧ Each player can only land on one location at once, but each location can have multiple players on it.
- ✧ Each player can have multiple lines of player-audit records, and each line of playeraudit only relates to one player.
- ✧ Each player can only dice once each time, and one dice record only relates to one player.
- ✧ Each player only has one line of result for each new game, and one line of result only associates with one player.
- ✧ Each location should have a unique id, each bonus/property should link to one id.
- ✧ Each property should have one colour. And multiple properties can have the same colour.
- ✧ If the dataset is expanded in the future, game_setting table should link with result table. Thus, each player in each game should have one record.

3 Physical model and table creation

Based on the ER diagram above, we created the physical schema(EER diagram) in MySQL. Code to generate our database is attached in the Appendix.

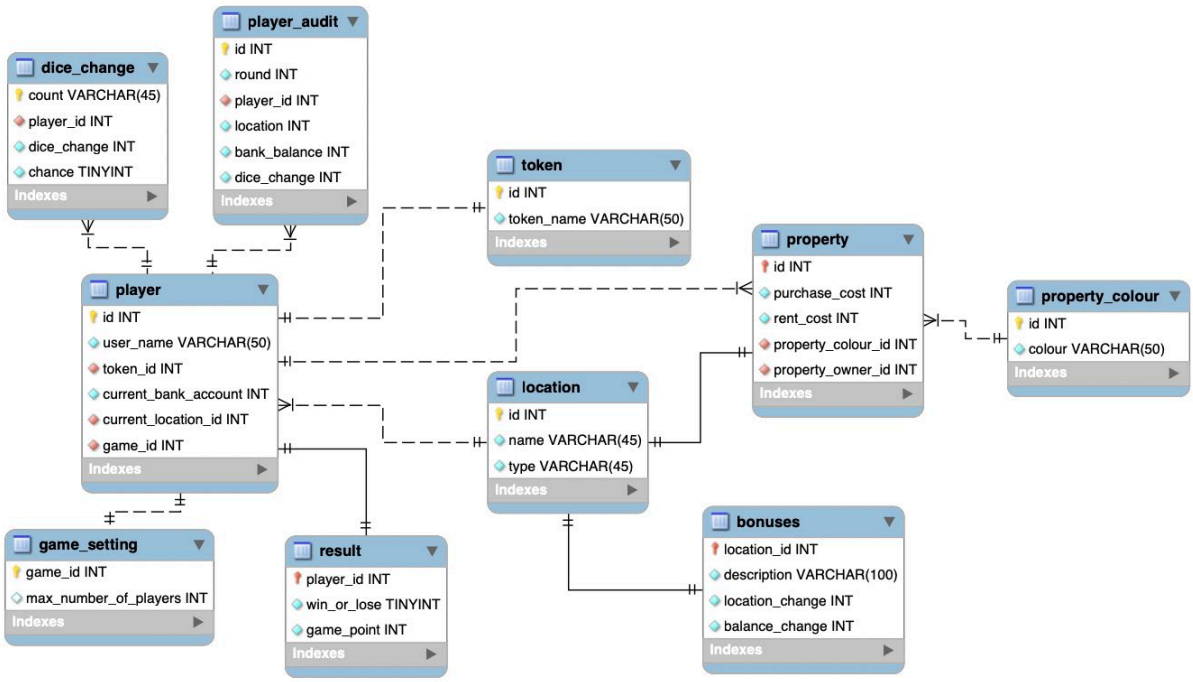

Figure 3. EER Diagram from MySQL

3.1 Location

Because of the presence of the foreign key, the tables need to be inserted in a certain order. Firstly, the location

table. All attributes are not null. The id is allocated based on this figure:

Figure 4. Location id

Then we use these SQL statements to insert data into the *location* table:

```
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('1', 'go', 'bonus');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('2', 'kilburn', 'property');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('3', 'chance_1', ' bonus ');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('4', 'uni_place', 'property');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('5', 'in_jail', 'bonus');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('6', 'victoria', 'property');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('7', 'community_chest_1', ' bonus
');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('8', 'piccadilly', 'property');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('9', 'free_parking', 'bonus');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('10', 'oak_house', 'property');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('11', 'chance_2', 'bonus');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('12', 'go_to_jail', 'bonus');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('13', 'go_to_jail', 'bonus');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('14', 'ambs', 'property');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('15', 'community_chest_2', 'bonus
');
INSERT INTO `monopoly`.`location` (`id`, `name`, `type`) VALUES ('16', 'co_op', 'property');
```

Table we get:

id	name	type
1	go	bonus
2	kilburn	property
3	chance_1	bonus
4	uni_place	property
5	in_jail	bonus
6	victoria	property
7	community_chest_1	bonus
8	piccadilly	property
9	free_parking	bonus
10	oak_house	property
11	chance_2	bonus
12	owens_park	property
13	go_to_jail	bonus
14	ambs	property
15	community_chest_2	bonus
16	co_op	property

Table 11. location values

3.2 Token

Then, the *token* table. All attributes are not null.

SQL statements for inserting data:

```
INSERT INTO `monopoly`.`token` (`id`, `token_name`) VALUES ('1', 'dog');
INSERT INTO `monopoly`.`token` (`id`, `token_name`) VALUES ('2', 'car');
INSERT INTO `monopoly`.`token` (`id`, `token_name`) VALUES ('3', 'battleship');
INSERT INTO `monopoly`.`token` (`id`, `token_name`) VALUES ('4', 'top_hat');
INSERT INTO `monopoly`.`token` (`id`, `token_name`) VALUES ('5', 'thimble');
INSERT INTO `monopoly`.`token` (`id`, `token_name`) VALUES ('6', 'boot');
```

Table we get:

id	token_name
1	dog
2	car
3	battleship
4	top_hat
5	thimble
6	boot

Table 12. token values

3.3 Propertycolour

The *propertycolour* table. All attributes are not null.

SQL statements for inserting data:

```
INSERT INTO `monopoly`.`propertycolour` (`id`, `colour`) VALUES ('1', 'orange');
INSERT INTO `monopoly`.`propertycolour` (`id`, `colour`) VALUES ('2', 'blue');
INSERT INTO `monopoly`.`propertycolour` (`id`, `colour`) VALUES ('3', 'yellow');
INSERT INTO `monopoly`.`propertycolour` (`id`, `colour`) VALUES ('4', 'green');
```

Table we get:

id	colour
1	orange
2	blue
3	yellow
4	green

Table 13. propertycolour values

3.4 Property

The *property* table. All attributes are not null, except the *property_owner_id* attribute. When a player makes a purchase, the database will automatically update this attribute.

SQL statements for inserting data:

```
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`) VALUE
S ('2', '120', '120', '3');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`, `prop
erty_owner_id`) VALUES ('4', '100', '100', '3', '1');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`, `prop
erty_owner_id`) VALUES ('6', '75', '75', '4', '2');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`) VALUE
S ('8', '35', '35', '4');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`, `prop
erty_owner_id`) VALUES ('10', '100', '100', '1', '4');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`, `prop
erty_owner_id`) VALUES ('12', '30', '30', '1', '4');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`) VALUE
S ('14', '400', '400', '2');
INSERT INTO `monopoly`.`property` (`id`, `purchase_cost`, `rent_cost`, `property_colour_id`, `prop
erty_owner_id`) VALUES ('16', '30', '30', '2', '3');
```

Table we get:

id	purchase_cost	rent_cost	property_colour_id	property_owner_id
2	120	120	3	NULL
4	100	100	3	1
6	75	75	4	2
8	35	35	4	NULL
10	100	100	1	4
12	30	30	1	4
14	400	400	2	NULL
16	30	30	2	3

Table 14. property values

3.5 Bonuses

The *bonuses* table. All attributes are not null.

SQL statements for inserting data:

```
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('1', ' Collect £200', '0', '0');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('3', ' Pay each of the other players £50', '0', '-50');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('5', 'In jail, but no action', '0', '0');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('7', ' For winning a Beauty Contest, you win £100 ', '0', '100');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('9', 'no action', '0', '0');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('11', ' Your library books are overdue. Play a fine of £30', '3', '0');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('13', ' Go to Jail, do not pass GO, do not collect £200', '8', '0');
INSERT INTO `monopoly`.`bonuses` (`id`, `description`, `location_change`, `balance_change`) VALUES ('15', ' Your library books are overdue. Play a fine of £30', '0', '-30');
```

Table we get:

id	description	location_change	balance_change
1	Collect 200	0	0
3	Pay each of the other player 50	0	-50
5	In_jail, but no action	0	0
7	For winning a Beauty Contest, you win 100	0	100
9	no cation	0	0
11	Move three steps forword	3	0
13	Go to jail, do not pass Go, do not collect 200	8	0
15	You library books are overdue. Pay a fine of 30	0	-30
NULL	NULL	NULL	NULL

Table 15. bonuses values

3.6 Game_setting

The *game_setting* table. All attributes are not null.

SQL statements for inserting data:

```
INSERT INTO `monopoly`.`gamesetting` (`game_id`, `max_number_of_players`, `current_player_number`) VALUES ('1', '4', '4');
```

Table we get:

game_id	max_number_of_players	current_player_number
1	4	4

Table 16. game_setting values

3.7 Player

The *player* table. All attributes are not null.

SQL statements for inserting data:

```
INSERT INTO `monopoly`.`player` (`id`, `user_name`, `token_id`, `current_bank_account`, `current_location_id`) VALUES ('1', 'Mary', '3', '190' , '9');
INSERT INTO `monopoly`.`player` (`id`, `user_name`, `token_id`, `current_bank_account`, `current_location_id`) VALUES ('2', 'Bill', '1', '500' , '12');
INSERT INTO `monopoly`.`player` (`id`, `user_name`, `token_id`, `current_bank_account`, `current_location_id`) VALUES ('3', 'Jane', '2', '150' , '14');
INSERT INTO `monopoly`.`player` (`id`, `user_name`, `token_id`, `current_bank_account`, `current_location_id`) VALUES ('4', 'Norman', '5', '400' , '7');
```

Table we get:

id	user_name	token_id	current_bank_account	current_location_id	game_id
1	Mary	3	190	9	1
2	Bill	1	500	12	1
3	Jane	2	150	14	1
4	Norman	5	250	2	1

Table 17. player values

4 Automatic update settings

Monopoly runs based on the number of points a player rolls and the location of the player lands on. In oue monopoly database, location is divided into two types: bonus and property. Therefore we need two separate procedures, one that is executed when a player lands on the bonus type location and one that is executed when a player lands on the property type location. when a player goes bankrupt, he automatically quits the game and gets his result, so we also need a bankruptcy liquidation procedure. Apparently procedures are made up of a number of functions.

A good flowchart before starting code often makes things easier. So we have the flowchart diagram below, where the diamond represents decision and the rectangle represents process.

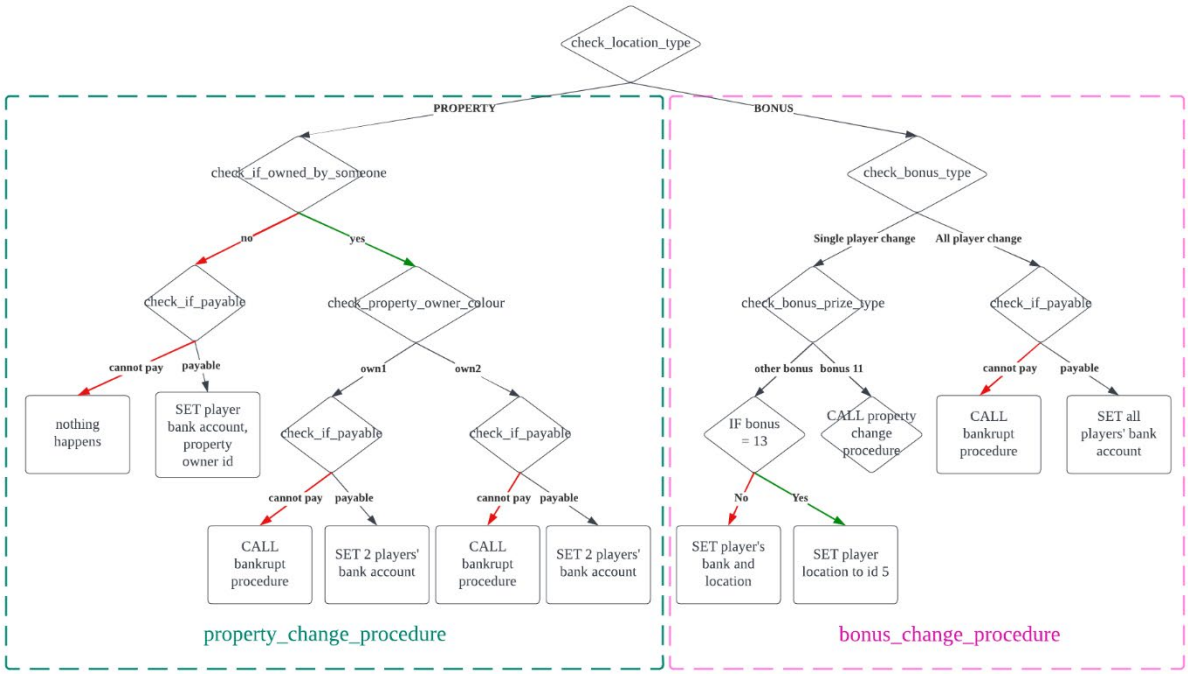

Figure 5. monopoly automatic updating flowchart

4.1 *Bonus_change* Procedure

The program checks what types of bonus are at the first. Bonus 3 relates to all players' accounts; other bonuses only relate to the player who lands on that. If a player lands on Bonus 3, the procedure is going to check if the player can pay 150, if not, it will call bankrupt procedure. If a player lands on other bonus, the procedure will automatically add or minus it's bank account or location id.

Tips:

- ✧ Bonus 11 refers to moving forward three steps to AMBS, so here we call the property_change procedure directly.
- ✧ Bonus 13 refers to going to jail, so here we set player's current location id to 5(in jail).

SQL statements for bonus_change PROCEDURE:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `bonus_change_procedure`(location_id INT, p
        layer_id INT)
BEGIN
    DECLARE player_current_bank_account INT;
```

```
    DECLARE location_change INT;
    DECLARE bank_account_change INT;

    SET player_current_bank_account = (SELECT current_bank_account
                                       FROM monopoly.player
                                       WHERE player.id = player_id);

    IF check_bonus_type(location_id) THEN

        IF check_if_payable(player_current_bank_account,150) THEN
            UPDATE player
            SET current_bank_account = current_bank_account + 50
            WHERE id <> player_id;

            UPDATE player
            SET current_bank_account = current_bank_account - 150,
                current_location_id = location_id
            WHERE id = player_id;
        ELSE
            CALL bankrupt_procedure(player_id);
        END IF;

    ELSE

        IF check_bonus_prize_type(location_id) THEN
            CALL property_change_procedure(14, player_id);
        ELSE
            IF location_id = 13 THEN
               SET location_id = 5;

               UPDATE player
               SET current_location_id = 5
               WHERE id = player_id;
            ELSE
                SET location_change = (SELECT location_change
                                       FROM bonuses
                                       WHERE id = location_id);
                IF location_change IS NULL THEN SET location_change = 0;
                END IF;

                SET bank_account_change = (SELECT balance_change
                                           FROM bonuses
                                           WHERE id = location_id);
                IF bank_account_change IS NULL THEN SET bank_account_change = 0;
                END IF;
```

```

                UPDATE player
                SET current_location_id = location_id + location_change,
                    current_bank_account = current_bank_account + bank_account_change
                WHERE id = player_id;
            END IF;

        END IF;

    END IF;

END
```

4.2 *Property_change* Procedure

The program checks if the property belongs to someone first. If it has an owner, then check if the owner have another same colour property. If the property is independent, then check if the player can afford the purchase price. If he has enough money, then update the value of *current_bank_account* in his *player* table and the *propert_owner_id* of the house in the *property* table.

SQL statements for property_change PROCEDURE:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `property_change_procedure`(location_id INT
        , player_id INT)
BEGIN
    DECLARE rent_to_pay INT;
    DECLARE player_current_bank_account INT;
    DECLARE owner_id INT;
    DECLARE house_prise INT;

    SET player_current_bank_account = (SELECT current_bank_account
                                       FROM monopoly.player
                                       WHERE player.id = player_id);

    IF check_if_owned_by_someone(location_id) THEN
        SET owner_id = (SELECT property_owner_id
                        FROM property
                        WHERE id = location_id);
        IF check_property_owner_colour(location_id) THEN
           SET rent_to_pay = 2 * (SELECT rent_cost
                                  FROM monopoly.property
                                  WHERE monopoly.property.id = location_id);
            IF check_if_payable(player_current_bank_account, rent_to_pay) THEN
                UPDATE player
                SET current_bank_account = current_bank_account - rent_to_pay
                WHERE id = player_id;
```

```
                UPDATE player
                SET current_bank_account = current_bank_account + rent_to_pay
                WHERE id = owner_id;
            ELSE
                CALL bankrupt_procedure(player_id);
            END IF;

        ELSE
            SET rent_to_pay = (SELECT rent_cost
                               FROM monopoly.property
                               WHERE monopoly.property.id = location_id);

            IF check_if_payable(player_current_bank_account, rent_to_pay) THEN
                UPDATE player
                SET current_bank_account = current_bank_account - rent_to_pay
                WHERE id = player_id;

                UPDATE player
                SET current_bank_account = current_bank_account + rent_to_pay
                WHERE id = owner_id;
            ELSE
                CALL bankrupt_procedure(player_id);
            END IF;
        END IF;
    ELSE
        SET house_prise = (SELECT purchase_cost
                           FROM monopoly.property
                           WHERE property.id = location_id);
        IF check_if_payable(player_current_bank_account, house_prise) THEN
            UPDATE player
            SET current_bank_account = current_bank_account - house_prise
            WHERE id = player_id;

            UPDATE property
            SET property_owner_id = player_id
            WHERE id = location_id;
        END IF;
    END IF;

END
```

4.3 *Bankrupt* Procedure

Bankruptcy is repeated several times in the flowchart, it is best to make it a procedure to avoid duplicate and save

code lines. A player's bankruptcy is associated with two tables: *gamesetting* and *result*. We first update the *current_number_of_player* in the *gamesetting* table, then insert a row for that player's *id*, *win_or_lose* and *game_point* in the result table. and score.

An example: if player 2 goes bankrupt, after calling bankrupt_procedure(2), we can have these two tables:

game_id	max_number_of_players	current_player_number
1	4	3
NULL	NULL	NULL

Table 18. *game_setting* table's update after one player bankrupt

player_id	win_or_lose	game_point
2	0	1
NULL	NULL	NULL

Table 19. *result* table's update after one player bankrupt

SQL statements for bankrupt PROCEDURE:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `bankrupt_procedure`(player_id INT)
BEGIN
    DECLARE current_number_of_player INT;
    DECLARE game_point_player_get INT;  0020

    SET current_number_of_player = (SELECT current_player_number
                                    FROM  monopoly.gamesetting
                                    WHERE monopoly.gamesetting.game_id = 1) - 1;

    SET game_point_player_get = @playernum - current_number_of_player;

    UPDATE gamesetting
    SET current_player_number = current_number_of_player
    WHERE monopoly.gamesetting.game_id = 1;

    INSERT INTO result
    VALUES(player_id,0,game_point_player_get);
END
```

4.4 Functions

✧ SQL statements for check_bonus_prize_type FUNCTION

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_bonus_prize_type`(location_id INT) RETURNS int
    READS SQL DATA
BEGIN
   IF location_id = 11 THEN
       RETURN 1;
   ELSE
       RETURN 0;
   END IF;
END
```

✧ SQL statements for check_bonus_type FUNCTION

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_bonus_type`(location_id INT) RETURNS int
    READS SQL DATA
BEGIN
    IF location_id = 3 THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END
```

✧ SQL statements for check_if_owned_by_someone FUNCTION

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_if_owned_by_someone`(location_id INT) RETURNS int
    READS SQL DATA
BEGIN
IF (SELECT property_owner_id
FROM monopoly.property
WHERE monopoly.property.id = location_id) IS NULL THEN
    RETURN 0;
ELSE
    RETURN 1;
END IF;

END
```

✧ SQL statements for check_if_payable FUNCTION

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_if_payable`(
    player_current_account INT,
    money_needs_to_pay INT) RETURNS int
    READS SQL DATA
BEGIN
   IF player_current_account > money_needs_to_pay THEN
```

```
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END
```

✧ SQL statements for check_location_type FUNCTION

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_location_type`(location_id INT) RETURNS int
    READS SQL DATA
BEGIN
    DECLARE location_type TINYINT;

    IF location_id IN (SELECT id FROM monopoly.bonuses) THEN
        SET location_type = 1;
    ELSE
        SET location_type = 0;
    END IF;
RETURN location_type;
END
```

✧ SQL statements for check_property_owner_colour FUNCTION

```
CREATE DEFINER=`root`@`localhost` FUNCTION `check_property_owner_colour`(location_id INT) RETURNS int
    READS SQL DATA
BEGIN
    DECLARE divi TINYINT;
    DECLARE owner_id TINYINT;
    DECLARE another_owner_id TINYINT;

    SET owner_id = (SELECT property_owner_id
                    FROM monopoly.property
                    WHERE monopoly.property.id = location_id);

    IF location_id % 4 = 2 THEN
        SET another_owner_id =  (SELECT property_owner_id
                                 FROM monopoly.property
                                 WHERE monopoly.property.id = location_id + 2);
    ELSE
        SET another_owner_id =  (SELECT property_owner_id
                                 FROM monopoly.property
                                 WHERE monopoly.property.id = location_id - 2);
    END IF;

    IF another_owner_id = owner_id THEN
```

```
        RETURN 1;
    ELSE
        RETURN 0;
     END IF;

    END
```

4.5 After_insert Trigger and global variables setting

After the main procedures are done, we have to string together the new-inserting data and the database. Again, we need a flowchart.

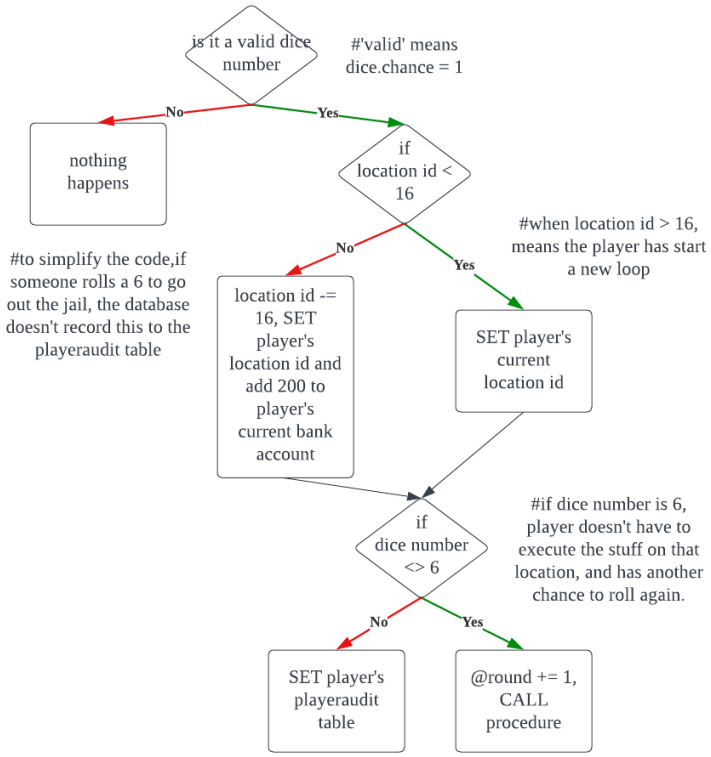

Figure 6. monopoly automatic updating flowchart after inserting

After a player goes to jail, he needs a 6 to go out, so when we are manually inserting the *dice* table, we need to enter a *chance* value. *chance 0* means the player cannot move, and *chance 1* means the player can move. This check should be put at the beginning of our program.

```
SET @playernum = (SELECT count(id) FROM player),
    @round_update = 1,
    @round_count = 0;

DROP TRIGGER IF EXISTS dice_after_insert;

DELIMITER $$

```

```
CREATE TRIGGER dice_after_insert
    AFTER INSERT ON dice
    FOR EACH ROW

BEGIN
   DECLARE dice_number INT;
   DECLARE player_id INT;
   DECLARE dice_chance BOOLEAN;
   DECLARE past_location_id INT;
   DECLARE location_id INT;
   DECLARE final_location_id INT;
   DECLARE final_bank_account INT;

   SET dice_number = NEW.dice_change,
       player_id = NEW.player_id,
       dice_chance = NEW.chance;

   IF dice_chance = 1 THEN
       SET past_location_id = (SELECT current_location_id FROM player
                               WHERE id = player_id);
       SET location_id = past_location_id + dice_number;

       IF location_id > 16 THEN
            SET location_id = location_id - 16;
            UPDATE player
            SET current_bank_account = current_bank_account + 200,
                current_location_id = location_id
            WHERE id = player_id;
       ELSE
           UPDATE player
           SET current_location_id = location_id
           WHERE id = player_id;
       END IF;

       IF dice_number <> 6 THEN
           SET @round_count = @round_count + 1;

           IF check_location_type(location_id) THEN
               CALL bonus_change_procedure(location_id, player_id);
           ELSE
               CALL property_change_procedure(location_id, player_id);
           END IF;
        END IF;

        SET final_location_id = (SELECT current_location_id
```

```
                                FROM player
                                WHERE id = player_id),
            final_bank_account = (SELECT current_bank_account
                                 FROM player
                                 WHERE id = player_id);

        INSERT INTO playeraudit
        VALUES(DEFAULT, @round_update, player_id, final_location_id, final_bank_account
        ,dice_number);
    END IF;

    IF @round_count = 4 THEN
        SET @round_count = 0,
            @round_update = @round_update + 1;
    END IF;


END $$

DELIMITER ;
```

5 Testing

Now, move on to the most exciting part: testing! Before inserting data into dice table, we make a view of beginning player status first.

SQL statements for creating a view:

```
CREATE VIEW beginning_player_status AS
SELECT p.id, p.user_name, p.token_id, l.name AS current_location_name, p.current_bank_ac
        count
FROM monopoly.player p
JOIN location l
ON p.current_location_id = l.id
```

We get the view:

id	user_name	token_id	current_location_name	current_bank_account
1	Mary	3	free_parking	190
2	Bill	1	owens_park	500
3	Jane	2	ambs	150
4	Norman	5	kilburn	250

Table 20.a view of palyer beginning status

Then, we insert data line by line into the *dice* table.

```
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘1’, ‘3’, ‘3’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘2’, ‘4’, ‘1’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘3’, ‘1’, ‘4’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘4’, ‘2’, ‘2’, ‘1’);
```

SQL statements for creating a view:

```
CREATE VIEW round_1_player_status AS
SELECT p.id, p.user_name, p.token_id, l.name AS current_location_name, p.current_bank_account
FROM monopoly.player p
JOIN location l
ON p.current_location_id = l.id
```

We get the view:

id	user_name	token_id	current_location_name	current_bank_account
1	Mary	3	in_jail	240
2	Bill	1	ambs	150
3	Jane	2	go	400
4	Norman	5	chance_1	100

Table 21.a view of palyer status after round one

id	round	player_id	location	bank_balance	dice_change
12	1	3	1	350	3
13	1	4	3	100	1
14	1	1	5	240	4
15	1	2	14	150	2
NULL	NULL	NULL	NULL	NULL	NULL

Table 22.the *playeraudit* table after round one

Addendum: the id attribute in *playeraudit* table is auto-incrementing and I tested a few groups before this, so it starts at 12.

id	purchase_cost	rent_cost	property_colour_id	property_owner_id
2	120	120	3	NULL
4	100	100	3	1
6	75	75	4	2
8	35	35	4	NULL
10	100	100	1	4
12	30	30	1	4
14	400	400	2	2
16	30	30	2	3
NULL	NULL	NULL	NULL	NULL

Table 23.the *property* table after round one(id 14 got a owner 2)

Now, we insert our final data line by line into the *dice* table.

```
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘5’, ‘3’, ‘5’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘6’, ‘4’, ‘4’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘7’, ‘1’, ‘6’, ‘0’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘8’, ‘1’, ‘5’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘9’, ‘2’, ‘6’, ‘1’);
INSERT INTO ‘monopoly’. ‘dice’(‘counts’, ‘player_id’, ‘dice_change’, ‘chance’)VALUES(‘10’, ‘2’, ‘3’, ‘1’);
```

SQL statements for creating a view:

```
   CREATE VIEW final_player_status AS
SELECT p.id, p.user_name, p.token_id, l.name AS current_location_name, p.current_bank_account
FROM monopoly.player p
JOIN location l
ON p.current_location_id = l.id
```

We get the view:

id	user_name	token_id	current_location_name	current_bank_account
1	Mary	3	oak_house	40
2	Bill	1	community_chest_1	525
3	Jane	2	victoria	325
4	Norman	5	community_chest_1	400

Table 24.a view of palyer status after round two

counts	player_id	dice_change	chance
1	3	3	1
2	4	1	1
3	1	4	1
4	2	2	1
5	3	5	1
6	4	4	1
7	1	6	0
8	1	5	1
9	2	6	1
10	2	3	1
NULL	NULL	NULL	NULL

Table 25.a screenshot of *dice* table after insert

id	round	player_id	location	bank_balance	dice_change
12	1	3	1	350	3
13	1	4	3	100	1
14	1	1	5	240	4
15	1	2	14	150	2
16	2	3	6	325	5
17	2	4	7	200	4
18	2	1	10	40	5
19	2	2	4	425	6
20	2	2	7	525	3
NULL	NULL	NULL	NULL	NULL	NULL

Table 26.a screenshot of *playeraudit* table after updating(Being mentioned before, the go-out-jail move didn't record)

Appendix(SQL statements to generate monopoly database)

```sql
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema monopoly
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema monopoly
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `monopoly` DEFAULT CHARACTER SET utf8 ;
USE `monopoly` ;

-- -----------------------------------------------------
-- Table `monopoly`.`token`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`token` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`token` (
  `id` INT NOT NULL,
  `token_name` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- ----------------------------------------$-----------
-- Table `monopoly`.`location`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`location` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`location` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `type` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- Table `monopoly`.`game_setting`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`game_setting` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`game_setting` (
  `game_id` INT NOT NULL,
  `max_number_of_players` INT NULL,
  PRIMARY KEY (`game_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `monopoly`.`player`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`player` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`player` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `user_name` VARCHAR(50) NOT NULL,
  `token_id` INT NOT NULL,
  `current_bank_account` INT NOT NULL,
  `current_location_id` INT NOT NULL,
  `game_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_player_token_idx` (`token_id` ASC) VISIBLE,
  INDEX `fk_player_location1_idx` (`current_location_id` ASC) VISIBLE,
  INDEX `fk_player_game_setting1_idx` (`game_id` ASC) VISIBLE,
  CONSTRAINT `fk_player_token`
    FOREIGN KEY (`token_id`)
    REFERENCES `monopoly`.`token` (`id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_player_location1`
    FOREIGN KEY (`current_location_id`)
    REFERENCES `monopoly`.`location` (`id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_player_game_setting1`
    FOREIGN KEY (`game_id`)
    REFERENCES `monopoly`.`game_setting` (`game_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


```

```
-- -----------------------------------------------------
-- Table `monopoly`.`property_colour`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`property_colour` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`property_colour` (
  `id` INT NOT NULL,
  `colour` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `monopoly`.`property`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`property` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`property` (
  `id` INT NOT NULL,
  `purchase_cost` INT NOT NULL,
  `rent_cost` INT NOT NULL,
  `property_colour_id` INT NOT NULL,
  `property_owner_id` INT NOT NULL,
  INDEX `fk_property_property_colour1_idx` (`property_colour_id` ASC) VISIBLE,
  INDEX `fk_property_player1_idx` (`property_owner_id` ASC) VISIBLE,
  INDEX `fk_property_location1_idx` (`id` ASC) VISIBLE,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_property_property_colour1`
    FOREIGN KEY (`property_colour_id`)
    REFERENCES `monopoly`.`property_colour` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_property_player1`
    FOREIGN KEY (`property_owner_id`)
    REFERENCES `monopoly`.`player` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_property_location1`
    FOREIGN KEY (`id`)
    REFERENCES `monopoly`.`location` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


```

```
-- -----------------------------------------------------
-- Table `monopoly`.`bonuses`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`bonuses` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`bonuses` (
  `location_id` INT NOT NULL,
  `description` VARCHAR(100) NOT NULL,
  `location_change` INT NOT NULL,
  `balance_change` INT NOT NULL,
  INDEX `fk_bonuses_location1_idx` (`location_id` ASC) VISIBLE,
  PRIMARY KEY (`location_id`),
  CONSTRAINT `fk_bonuses_location1`
    FOREIGN KEY (`location_id`)
    REFERENCES `monopoly`.`location` (`id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `monopoly`.`result`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`result` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`result` (
  `player_id` INT NOT NULL,
  `win_or_lose` TINYINT NOT NULL,
  `game_point` INT NOT NULL,
  INDEX `fk_result_player1_idx` (`player_id` ASC) VISIBLE,
  PRIMARY KEY (`player_id`),
  CONSTRAINT `fk_result_player1`
    FOREIGN KEY (`player_id`)
    REFERENCES `monopoly`.`player` (`id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `monopoly`.`player_audit`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`player_audit` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`player_audit` (
```

```
  `id` INT NOT NULL,
  `round` INT NOT NULL,
  `player_id` INT NOT NULL,
  `location` INT NOT NULL,
  `bank_balance` INT NOT NULL,
  `dice_change` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_player_audit_player1_idx` (`player_id` ASC) VISIBLE,
  CONSTRAINT `fk_player_audit_player1`
    FOREIGN KEY (`player_id`)
    REFERENCES `monopoly`.`player` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `monopoly`.`dice_change`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `monopoly`.`dice_change` ;

CREATE TABLE IF NOT EXISTS `monopoly`.`dice_change` (
  `count` VARCHAR(45) NOT NULL,
  `player_id` INT NOT NULL,
  `dice_change` INT NOT NULL,
  `chance` TINYINT NOT NULL,
  INDEX `fk_dice_change_player1_idx` (`player_id` ASC) VISIBLE,
  PRIMARY KEY (`count`),
  CONSTRAINT `fk_dice_change_player1`
    FOREIGN KEY (`player_id`)
    REFERENCES `monopoly`.`player` (`id`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```