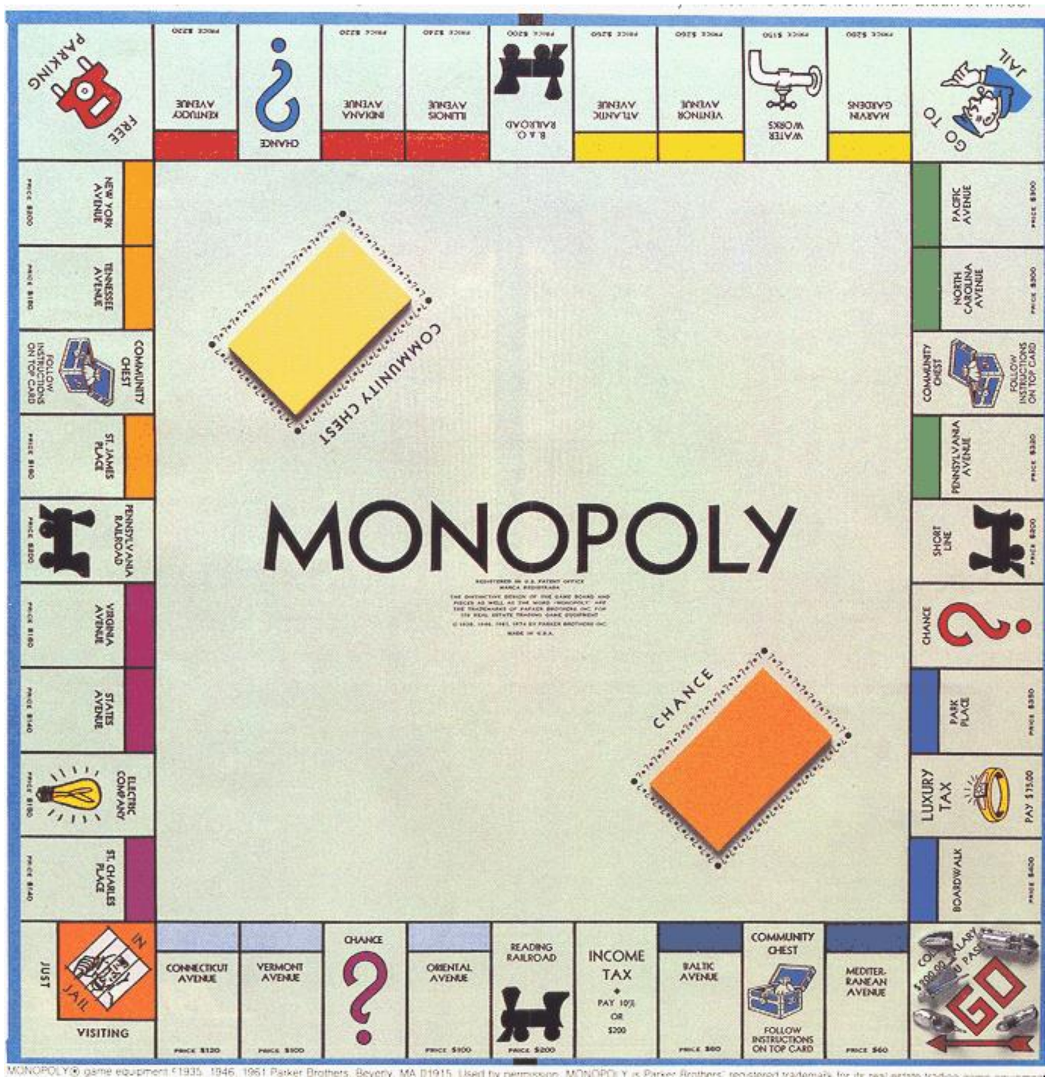
The Monopoly Database



By: Tadd Bindas

December 8th, 2016

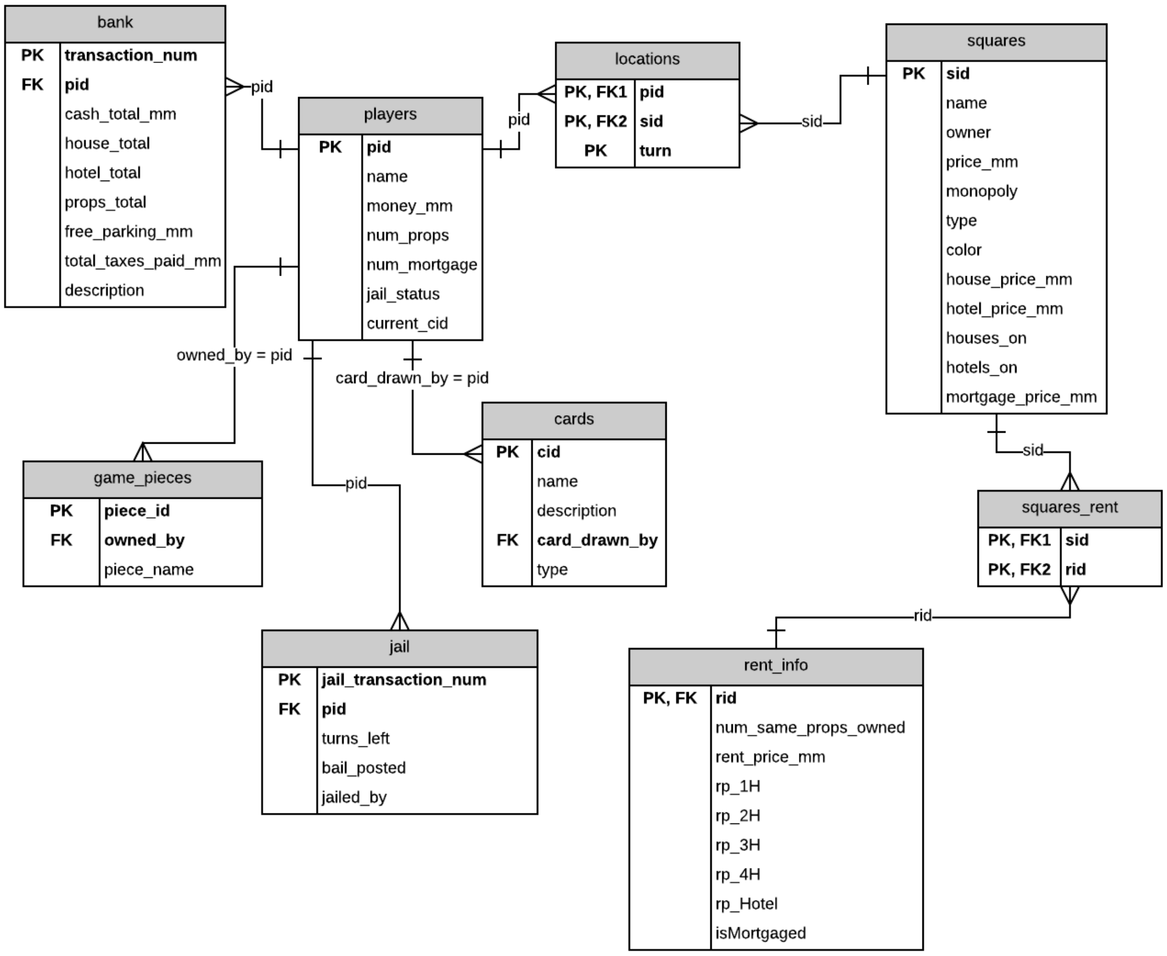
Table of Contents

Executive Summary

A complaint that many players of the game "Monopoly" have is that it is very hard to track the different elements of the game. No one player knows how many properties that the other players have, nor the amount of cash that each player possesses.

This database will be used to help the games proceed without having to constantly stop to ask who owns what property, and how much money each player has. This database might even be so useful that it will prevent players from flipping the board and end up aiding in successfully finishing a game.

Entity-Relation Diagram



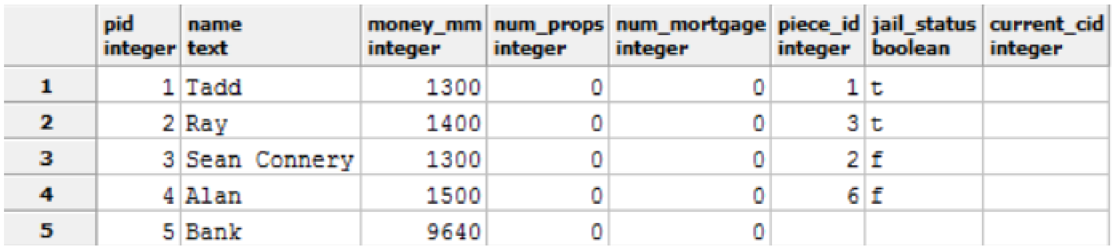
Players Table

Keeps track of the players of the game, their money, their number of properties owned and mortgaged, their game piece owned, jail status, and if they possess a get out of jail free card.

```
create table players
(
   pid integer not null unique,
   name text not null,
   money_mm integer not null,
   num_props integer not null,
   num_mortgage integer not null,
   piece_id integer,
   jail_status boolean,
   current_cid integer,
primary key (pid)
);
```

Dependencies:

pid ← name, money_mm, num_props, num_mortgage, piece_id, jail_status, current_cid



	pid integer	name text	money_mm integer	num_props integer	num_mortgage integer	piece_id integer	jail_status boolean	current_cid integer
1	1	Tadd	1300	0	0	1	t	
2	2	Ray	1400	0	0	3	t	
3	3	Sean Connery	1300	0	0	2	f	
4	4	Alan	1500	0	0	6	f	
5	5	Bank	9640	0	0			

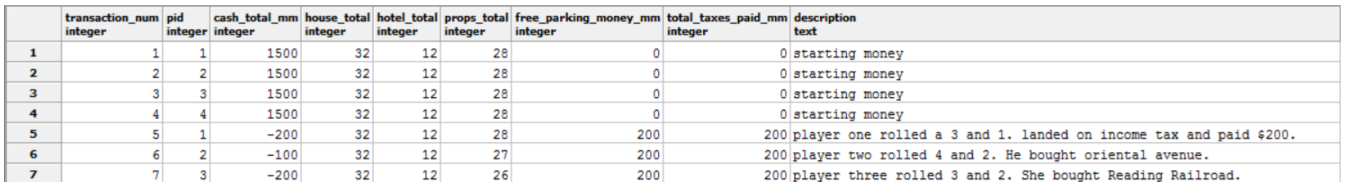
Bank Table

Keeps track of each transaction that goes through the bank. This helps make it easier to figure out what goes on so no player can casually take money from the bank without the others looking. This table has a player's id, the cash that was exchanged, the total number of houses, hotels, and properties the bank owns, the total taxes paid, the free parking available, and a description of the transaction.

```
create table bank
(
   transaction_num integer not null unique,
   pid integer not null references players(pid),
   cash_total_mm integer not null,
   house_total integer not null,
   hotel_total integer not null,
   props_total integer not null,
   free_parking_money_mm integer not null,
   total_taxes_paid_mm integer not null,
   description text not null,
primary key (transaction_num)
);
```

Dependencies:

transaction_num ← pid, cash, house_total, hotel_total, props_total, free_parking_money, total_taxes_paid, description



	transaction_num integer	pid integer	cash_total_mm integer	house_total integer	hotel_total integer	props_total integer	free_parking_money_mm integer	total_taxes_paid_mm integer	description text
1	1	1	1500	32	12	28	0	0	starting money
2	2	2	1500	32	12	28	0	0	starting money
3	3	3	1500	32	12	28	0	0	starting money
4	4	4	1500	32	12	28	0	0	starting money
5	5	1	-200	32	12	28	200	200	player one rolled a 3 and 1. landed on income tax and paid $200.
6	6	2	-100	32	12	27	200	200	player two rolled 4 and 2. He bought oriental avenue.
7	7	3	-200	32	12	26	200	200	player three rolled 3 and 2. She bought Reading Railroad.

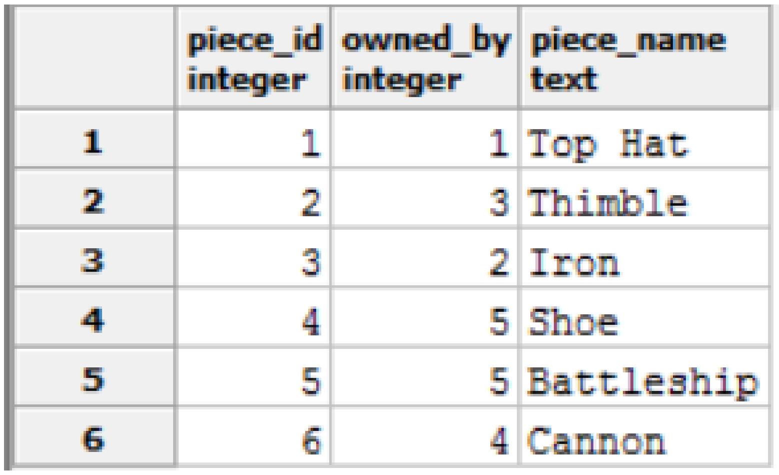
Game Pieces Table

This table keeps track of the game pieces, who owns what piece, and the names of each piece.

```
create table game_pieces
(
    piece_id integer not null,
    owned_by integer references Players(pid),
    piece_name text not null,
primary key (piece_id, owned_by)
);
```

Dependencies:

Piece_id ←owned_by, piece_name



	piece_id integer	owned_by integer	piece_name text
1	1	1	Top Hat
2	2	3	Thimble
3	3	2	Iron
4	4	5	Shoe
5	5	5	Battleship
6	6	4	Cannon

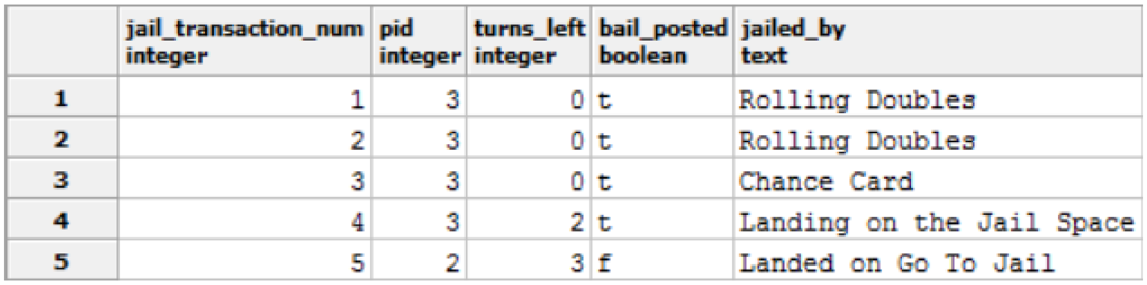
The Jail Table

The jail table keeps track of each jail transactiol. In each transaction, the player entering, his/her turns left, if they posted bail and how they were jailed are tracked.

```
create table jail
(
    jail_transaction_num integer not null unique,
    pid integer not null references Players(pid),
    turns_left integer not null,
    bail_posted boolean not null,
    jailed_by text not null,
primary key (jail_transaction_num)
);
```

Dependencies:

Jail_transaction_num🡐pid, turns_left, bail_posted, jailed_by



	jail_transaction_num integer	pid integer	turns_left integer	bail_posted boolean	jailed_by text
1	1	3	0	t	Rolling Doubles
2	2	3	0	t	Rolling Doubles
3	3	3	0	t	Chance Card
4	4	3	2	t	Landing on the Jail Space
5	5	2	3	f	Landed on Go To Jail

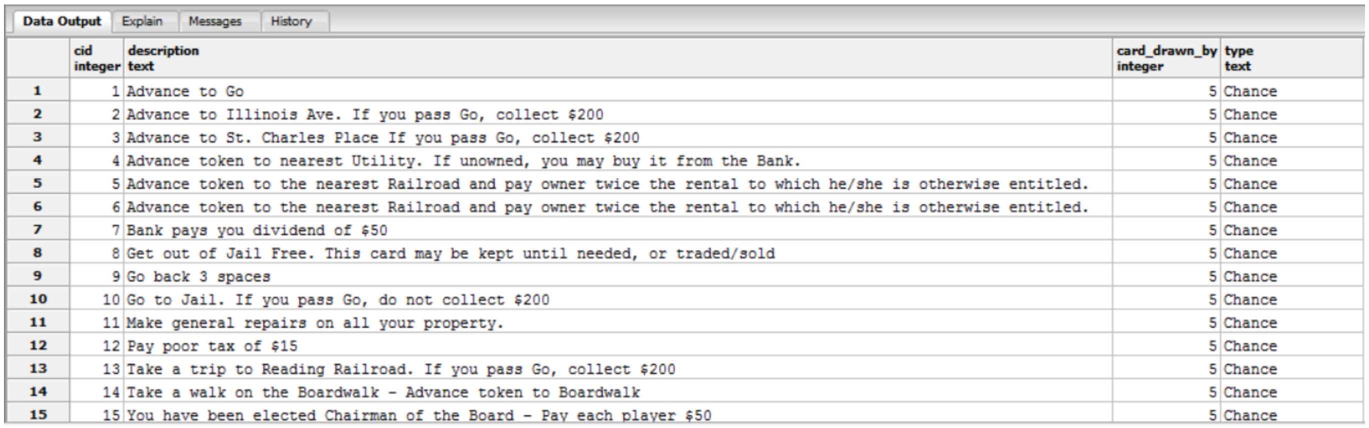
The Cards Table

The cards table stores the information of each card that is drawn during the game. Each card's description, type (Chance or Community Chest), and who drew each card are stored.

```
create table cards
(
cid integer not null unique,
description text,
card_drawn_by integer not null references players(pid),
type text not null,
primary key (cid)
);
```

Dependencies:

cid← description, card_drawn_by, type



	cid integer	description text	card_drawn_by integer	type text
1	1	Advance to Go	5	Chance
2	2	Advance to Illinois Ave. If you pass Go, collect $200	5	Chance
3	3	Advance to St. Charles Place If you pass Go, collect $200	5	Chance
4	4	Advance token to nearest Utility. If unowned, you may buy it from the Bank.	5	Chance
5	5	Advance token to the nearest Railroad and pay owner twice the rental to which he/she is otherwise entitled.	5	Chance
6	6	Advance token to the nearest Railroad and pay owner twice the rental to which he/she is otherwise entitled.	5	Chance
7	7	Bank pays you dividend of $50	5	Chance
8	8	Get out of Jail Free. This card may be kept until needed, or traded/sold	5	Chance
9	9	Go back 3 spaces	5	Chance
10	10	Go to Jail. If you pass Go, do not collect $200	5	Chance
11	11	Make general repairs on all your property.	5	Chance
12	12	Pay poor tax of $15	5	Chance
13	13	Take a trip to Reading Railroad. If you pass Go, collect $200	5	Chance
14	14	Take a walk on the Boardwalk - Advance token to Boardwalk	5	Chance
15	15	You have been elected Chairman of the Board - Pay each player $50	5	Chance

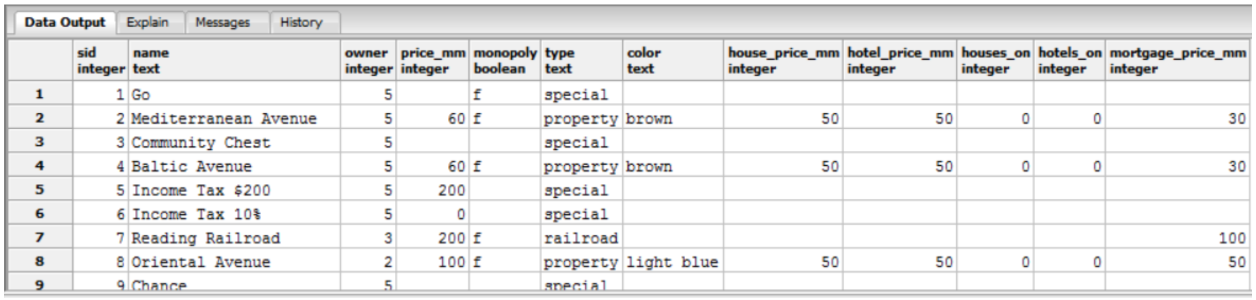
The Squares Table

The squares table is used to store the information per each square. The name, owner (owner 5 is the bank), their price in monopoly money, if there is a monopoly on it, the type of square, color, price of houses and hotels, number of houses or hotels on, and lastly the mortgage price are stored. The types of square are Special, Property, Railroad, and Utility. The squares of jail and income tax have two separate square ids because each square has two separate actions.

```
create table squares
(
    sid integer not null unique,
    name text not null,
    owner integer not null,
    price_mm integer,
    monopoly boolean,
    type text not null,
    color text,
    house_price_mm integer,
    hotel_price_mm integer,
    houses_on integer,
    hotels_on integer,
    mortgage_price_mm integer,
primary key (sid)
);
```

Dependencies:

Sid← name, owner, price_mm, monopoly, type, color, house_price_mm, hotel_price_mm, houses_on, hotels_on, mortgage_price_mm


Data Output | Explain | Messages | History

	sid integer	name text	owner integer	price_mm integer	monopoly boolean	type text	color text	house_price_mm integer	hotel_price_mm integer	houses_on integer	hotels_on integer	mortgage_price_mm integer
1	1	Go	5		f	special						
2	2	Mediterranean Avenue	5	60	f	property	brown	50	50	0	0	30
3	3	Community Chest	5			special						
4	4	Baltic Avenue	5	60	f	property	brown	50	50	0	0	30
5	5	Income Tax $200	5	200		special						
6	6	Income Tax 10%	5	0		special						
7	7	Reading Railroad	3	200	f	railroad						100
8	8	Oriental Avenue	2	100	f	property	light blue	50	50	0	0	50
9	9	Chance	5			special						

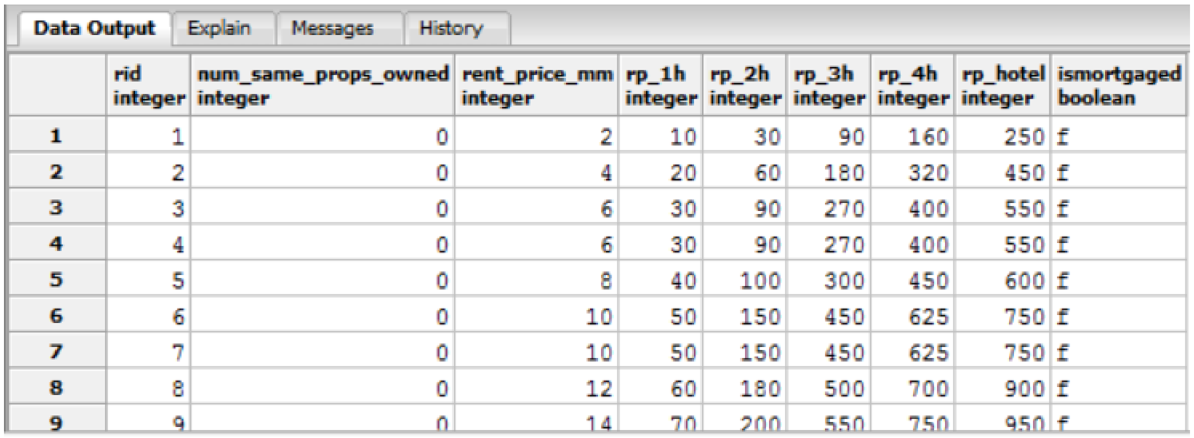
The Rent Info Table

The Rent Info table contains information on the rents per each square that is a property, a railroad, or a utility. The normal rent, number of same properties owned (for calculating railroad and utility rents), rent for when different numbers of houses and hotels are on each space, and if the space is mortgaged are all included.

```
create table rent_info
(
    rid integer not null unique,
    num_same_props_owned integer not null,
    rent_price_mm integer not null,
    rp_1H integer,
    rp_2H integer,
    rp_3H integer,
    rp_4H integer,
    rp_Hotel integer,
    isMortgaged boolean not null,
primary key (rid)
);
```

Dependencies:

Rid←num_same_props_owned, rent_price_mm, rp1H, rp_2H, rp_3H, rp_4H, rp_Hotel, isMortgaged



Data Output | Explain | Messages | History

	rid integer	num_same_props_owned integer	rent_price_mm integer	rp_1h integer	rp_2h integer	rp_3h integer	rp_4h integer	rp_hotel integer	ismortgaged boolean
1	1	0	2	10	30	90	160	250	f
2	2	0	4	20	60	180	320	450	f
3	3	0	6	30	90	270	400	550	f
4	4	0	6	30	90	270	400	550	f
5	5	0	8	40	100	300	450	600	f
6	6	0	10	50	150	450	625	750	f
7	7	0	10	50	150	450	625	750	f
8	8	0	12	60	180	500	700	900	f
9	9	0	14	70	200	550	750	950	f

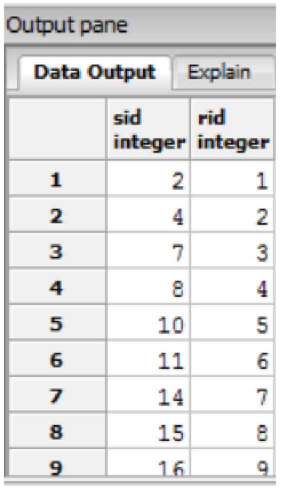
Square Rent Table

This table is the associative entity between squares and rent_info. It contains the sid and the rid.

```
create table squares_rent
(
    sid integer not null references squares(sid),
    rid integer not null references rent_info,
primary key (sid, rid)
);
```

Dependencies:

sid, rid ←



	sid integer	rid integer
1	2	1
2	4	2
3	7	3
4	8	4
5	10	5
6	11	6
7	14	7
8	15	8
9	16	9

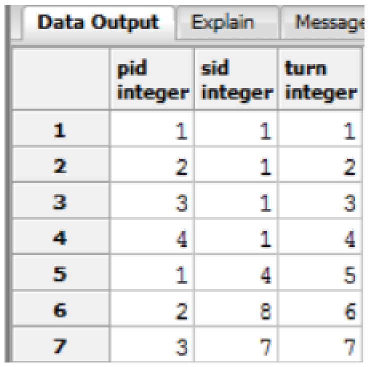
Locations Table

The locations table contains information that links together players and squares. This table shows where each player is on the board and what turn they moved to that location.

```
create table locations
(
    pid  integer not null references Players(pid),
    sid  integer not null references Squares(sid),
    turn integer not null,
primary key (pid, sid, turn)
);
```

Dependency:

pid, sid, turn ←



	pid integer	sid integer	turn integer
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	1	4	5
6	2	8	6
7	3	7	7

Views:

Brown/Light-Blue/Pink/Orange/Red/Yellow/Green/BlueSpacesView

Views that consolidate all the property information based on their colors

```
create view BrownSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='brown' and sr.rid=r.rid and sr.sid = s.sid;


create view LightBlueSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='light blue' and sr.rid=r.rid and sr.sid = s.sid;


create view PinkSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='pink' and sr.rid=r.rid and sr.sid = s.sid;
create view OrangeSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='orange' and sr.rid=r.rid and sr.sid = s.sid;


create view RedSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='red' and sr.rid=r.rid and sr.sid = s.sid;

create view YellowSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='yellow' and sr.rid=r.rid and sr.sid = s.sid;

create view GreenSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='green' and sr.rid=r.rid and sr.sid = s.sid;

create view BlueSpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type, s.color,
        s.house_price_mm, s.hotel_price_mm, s.houses_on, s.hotels_on, s.mortgage_price_mm,
        r.num_same_props_owned, r.rent_price_mm, r.rp_1H, r.rp_2H, r.rp_3H, r.rp_4H, r.rp_Hotel
        from rent_info r, squares s, squares_rent sr
        where color='blue' and sr.rid=r.rid and sr.sid = s.sid;
```

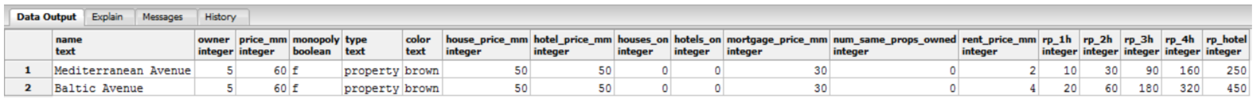
RR/Utility/SpecialSpacesView

Views that consolidate all railroad/utility/special space information

```
create view RRSpacesView as
            select s.name, s.owner, s.price_mm, s.monopoly, s.type,
            r.num_same_props_owned, r.rent_price_mm
            from rent_info r, squares s, squares_rent sr
            where type='railroad' and sr.rid=r.rid and sr.sid = s.sid;
```

```
create view UtilitySpacesView as
        select s.name, s.owner, s.price_mm, s.monopoly, s.type,
        r.num_same_props_owned, r.rent_price_mm
        from rent_info r, squares s, squares_rent sr
        where type='utility' and sr.rid=r.rid and sr.sid = s.sid;


create view SpecialSpacesView as
        select s.name, s.owner, s.type
        from  squares s
        where type='special';
```

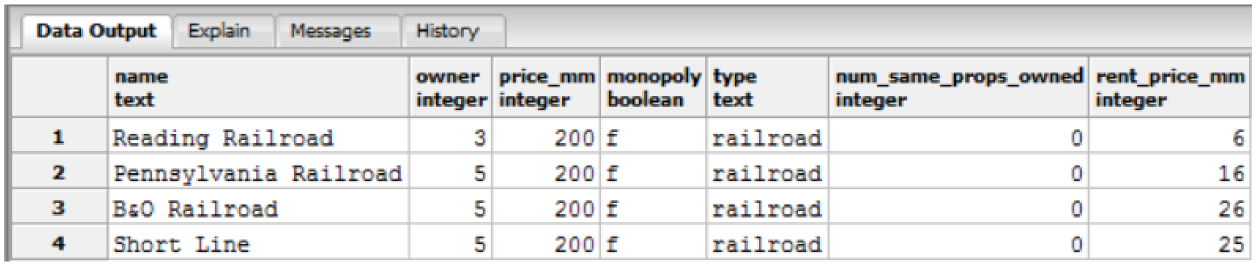
*BrownSpacesView



Data Output | Explain | Messages | History

	name text	owner integer	price_mm integer	monopoly boolean	type text	color text	house_price_mm integer	hotel_price_mm integer	houses_on integer	hotels_on integer	mortgage_price_mm integer	num_same_props_owned integer	rent_price_mm integer	rp_1h integer	rp_2h integer	rp_3h integer	rp_4h integer	rp_hotel integer
1	Mediterranean Avenue	5	60	f	property	brown	50	50	0	0	30	0	2	10	30	90	160	250
2	Baltic Avenue	5	60	f	property	brown	50	50	0	0	30	0	4	20	60	180	320	450

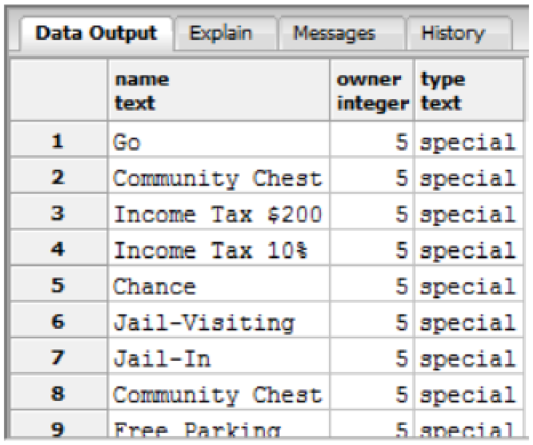
*RRSpacesView



Data Output | Explain | Messages | History

	name text	owner integer	price_mm integer	monopoly boolean	type text	num_same_props_owned integer	rent_price_mm integer
1	Reading Railroad	3	200	f	railroad	0	6
2	Pennsylvania Railroad	5	200	f	railroad	0	16
3	B&O Railroad	5	200	f	railroad	0	26
4	Short Line	5	200	f	railroad	0	25

*SpecialSpacesView



Data Output | Explain | Messages | History

	name text	owner integer	type text
1	Go	5	special
2	Community Chest	5	special
3	Income Tax $200	5	special
4	Income Tax 10%	5	special
5	Chance	5	special
6	Jail-Visiting	5	special
7	Jail-In	5	special
8	Community Chest	5	special
9	Free Parking	5	special

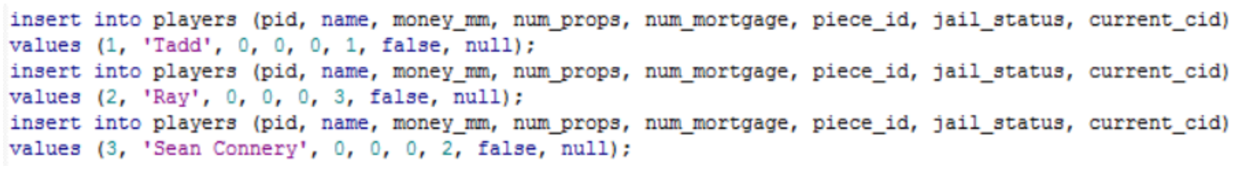
Triggers

changeMoney trigger:

This trigger will update the player's money based on each bank transaction the banker records

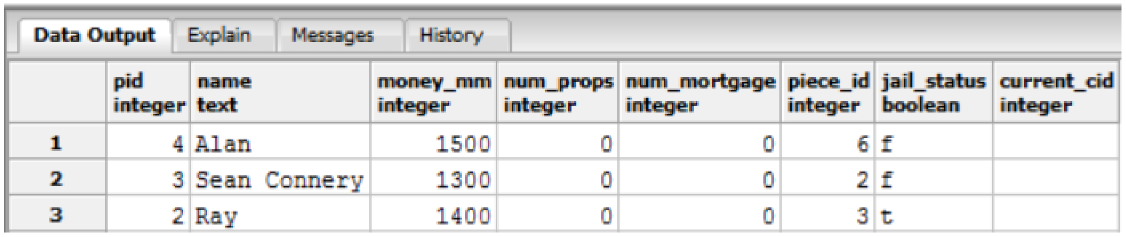
```
create or replace function changeMoney() returns trigger as
$$
begin
    if new.pid is not null then
        update players
        set money_mm = (money_mm + new.cash_total_mm)
        where pid = new.pid;
        update players
        set money_mm = (money_mm - new.cash_total_mm)
        where pid = 5;
    else
        rollback;
    end if;

    return new;
end;
$$
language plpgsql;

create trigger changeMoney
before insert on bank
for each row
execute procedure changeMoney();
```

*changeMoney Trigger insert statements



```
insert into players (pid, name, money_mm, num_props, num_mortgage, piece_id, jail_status, current_cid)
values (1, 'Tadd', 0, 0, 0, 1, false, null);
insert into players (pid, name, money_mm, num_props, num_mortgage, piece_id, jail_status, current_cid)
values (2, 'Ray', 0, 0, 0, 3, false, null);
insert into players (pid, name, money_mm, num_props, num_mortgage, piece_id, jail_status, current_cid)
values (3, 'Sean Connery', 0, 0, 0, 2, false, null);
```

*changeMoney Trigger Output



Data Output | Explain | Messages | History

	pid integer	name text	money_mm integer	num_props integer	num_mortgage integer	piece_id integer	jail_status boolean	current_cid integer
1	4	Alan	1500	0	0	6	f	
2	3	Sean Connery	1300	0	0	2	f	
3	2	Ray	1400	0	0	3	t	

goToJail Trigger

This trigger will automatically place the player in jail if they land on the go to jail space

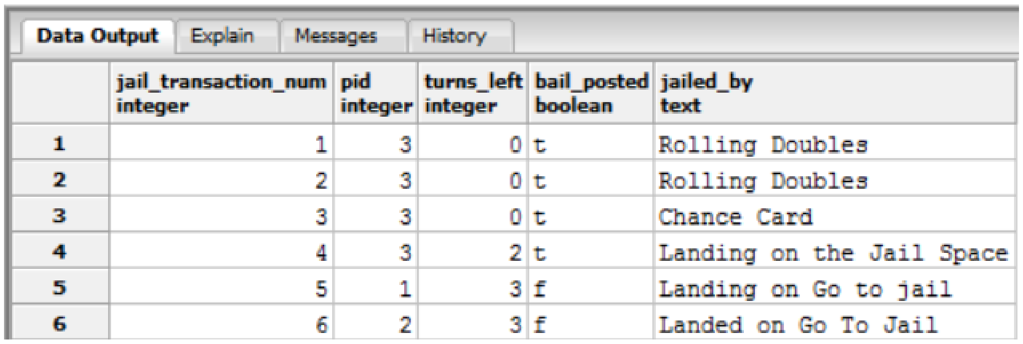
```
create or replace function goToJail() returns trigger as
$$
begin
        if (new.pid is not null) and new.sid = 33 then
            new.sid = 13;
            update players
            set jail_status = true
            where pid = new.pid;
            insert into jail (jail_transaction_num, pid, turns_left, bail_posted, jailed_by)
            values((select max(jail_transaction_num) from jail)+1, new.pid, 3, false, 'Landed on
Go To Jail');
                end if;
        return new;
end;
$$
language plpgsql;


create trigger jailed
before insert on locations
for each row
execute procedure goToJail();
```

*goToJail() output from jail table if fed

Insert into locations(pid, sid, turn)

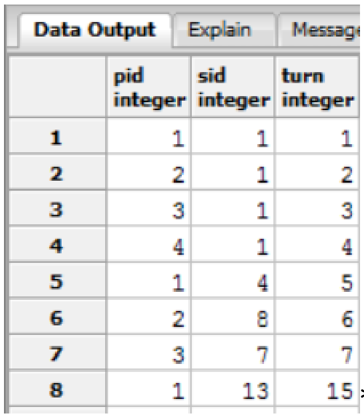
Values(1, 33, 15);

*jail table



Data Output | Explain | Messages | History

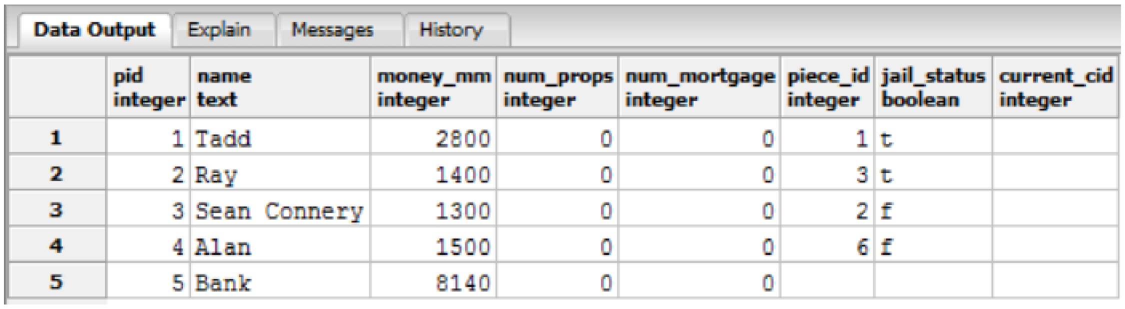
	jail_transaction_num integer	pid integer	turns_left integer	bail_posted boolean	jailed_by text
1	1	3	0	t	Rolling Doubles
2	2	3	0	t	Rolling Doubles
3	3	3	0	t	Chance Card
4	4	3	2	t	Landing on the Jail Space
5	5	1	3	f	Landing on Go to jail
6	6	2	3	f	Landed on Go To Jail



Data Output | Explain | Messag

	pid integer	sid integer	turn integer
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	1	4	5
6	2	8	6
7	3	7	7
8	1	13	15

*locations table



Data Output | Explain | Messages | History

	pid integer	name text	money_mm integer	num_props integer	num_mortgage integer	piece_id integer	jail_status boolean	current_cid integer
1	1	Tadd	2800	0	0	1	t	
2	2	Ray	1400	0	0	3	t	
3	3	Sean Connery	1300	0	0	2	f	
4	4	Alan	1500	0	0	6	f	
5	5	Bank	8140	0	0			

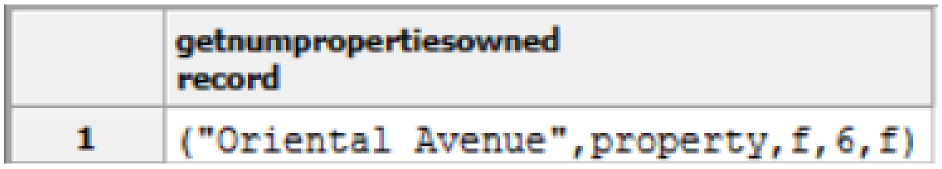
*players table

Stored Procedures

getNumPropertiesOwned Function

This function checks the properties that the player inputted owns

```
create or replace function getNumPropertiesOwned(integer)

returns table(name text, type text, monopoly boolean, rent_price_mm integer, isMortgaged boolean) as

$$

declare

    playerID alias for $1;

begin

    return query

    select s.name, s.type, s.monopoly, r.rent_price_mm, r.isMortgaged

    from squares s, rent_info r, squares_rent sr

    where sr.rid = r.rid

        and sr.sid = s.sid

            and  s.owner = playerID;

end;

$$

language plpgsql;
```

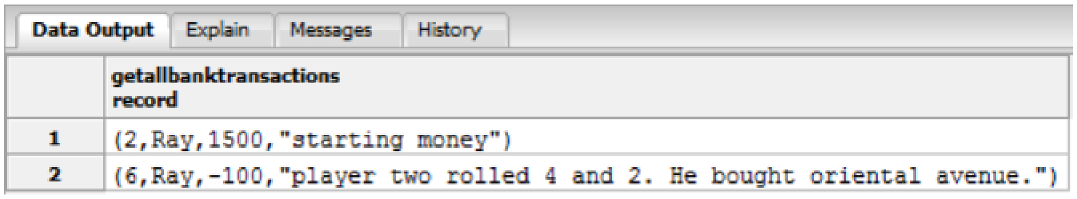


	getnumpropertiesowned record
1	("Oriental Avenue",property,f,6,f)

getAllBankTransactions() Function

This procedure gets all the transactions done by a single player.

```
create or replace function getAllBankTransactions(integer)
returns table(transaction_num integer, name text, cash_total_mm integer, description text) as
$$
declare
    playerID alias for $1;
begin
    return query
    select b.transaction_num, p.name, b.cash_total_mm, b.description
    from  players p
    inner join bank b on b.pid = playerID
    where b.pid=p.pid;
end;
$$
language plpgsql;
```

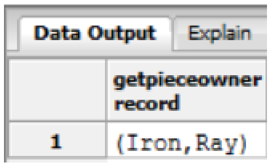


	getallbanktransactions record
1	(2,Ray,1500,"starting money")
2	(6,Ray,-100,"player two rolled 4 and 2. He bought oriental avenue.")

getPieceOwner() Function

This procedure gets the owner of the game piece. The input is a pid.

```
create or replace function getPieceOwner(integer)
returns table(piece_name text, name text) as
$$
declare
    playerID alias for $1;
begin
    return query
    select gp.piece_name, p.name
    from players p
    inner join game_pieces gp on gp.owned_by = p.pid
    where gp.owned_by = playerID;
end;
$$
language plpgsql;
```



	getpieceowner record
1	(Iron,Ray)

Sample Reports

Here is a report on the players table and their location on the board in the last few turns

select p.name, p.money_mm, p.num_props, p.num_mortgage, p.jail_status, p.current_cid, l.turn, s.name as position

from players p

inner join locations l on l.pid=p.pid

inner join squares s on s.sid=l.sid

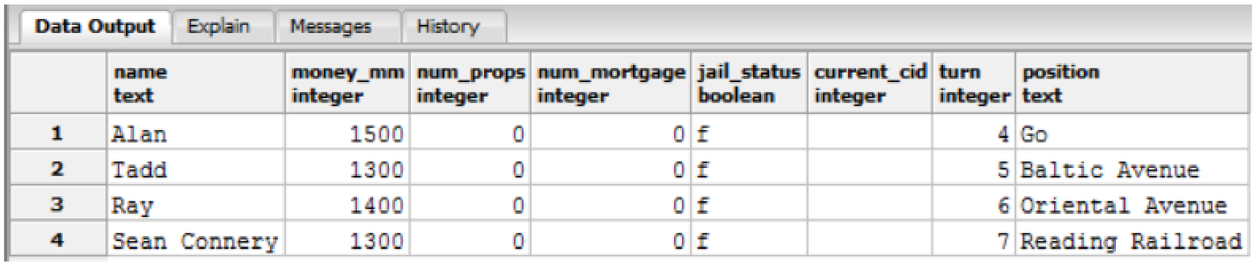
where l.turn = (select max(turn) from locations)

    or l.turn = (select max(turn) from locations)-1

        or l.turn = (select max(turn) from locations)-2

            or l.turn = (select max(turn) from locations)-3

order by l.turn asc;



Data Output | Explain | Messages | History

	name text	money_mm integer	num_props integer	num_mortgage integer	jail_status boolean	current_cid integer	turn integer	position text
1	Alan	1500	0	0	f		4	Go
2	Tadd	1300	0	0	f		5	Baltic Avenue
3	Ray	1400	0	0	f		6	Oriental Avenue
4	Sean Connery	1300	0	0	f		7	Reading Railroad

Here is a report on the properties that are left on the board

select name, price_mm, type, color

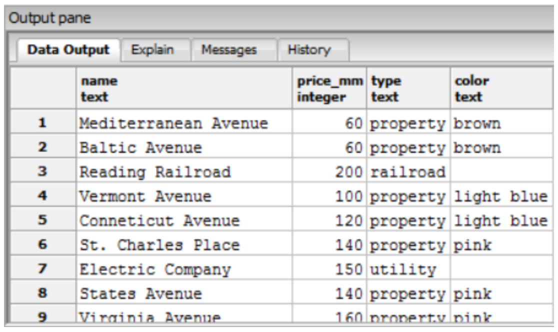
from squares

where owner = 5

    and type='property'

        or type='utility'

            or type = 'railroad';



Output pane

Data Output | Explain | Messages | History

	name text	price_mm integer	type text	color text
1	Mediterranean Avenue	60	property	brown
2	Baltic Avenue	60	property	brown
3	Reading Railroad	200	railroad	
4	Vermont Avenue	100	property	light blue
5	Conneticut Avenue	120	property	light blue
6	St. Charles Place	140	property	pink
7	Electric Company	150	utility	
8	States Avenue	140	property	pink
9	Virginia Avenue	160	property	pink

Roles

There are three roles in a game of monopoly

```
create role admin;
create role player;
create role banker;
```

Admins have full control over the database

```
grant select, insert, update, delete on squares to admin;
grant select, insert, update, delete on rent_info to admin;
grant select, insert, update, delete on squares_rent to admin;
grant select, insert, update, delete on cards to admin;
grant select, insert, update, delete on jail to admin;
grant select, insert, update, delete on game_pieces to admin;
grant select, insert, update, delete on players to admin;
grant select, insert, update, delete on bank to admin;
grant select, insert, update, delete on locations to admin;
```

Player can only select certain tables, can insert and update on players and locations, but cannot see the cards in the deck.

```
revoke all privileges on squares from player;
revoke all privileges on rent_info from player;
revoke all privileges on squares_rent from player;
revoke all privileges on cards from player;
revoke all privileges on jail from player;
revoke all privileges on game_pieces from player;
revoke all privileges on players from player;
revoke all privileges on bank from player;
revoke all privileges on locations from player;
```

```
grant select on squares to player;
grant select on rent_info to player;
grant select on squares_rent to player;
grant select on jail to player;
grant select on game_pieces to player;
grant select, insert, update on players to player;
grant select on bank to player;
grant select, insert on locations to player;
```

The banker can select and update on all tables, except for square_rent, and can insert on bank, players, and locations

```
revoke all privileges on squares from banker;
revoke all privileges on rent_info from banker;
revoke all privileges on squares_rent from banker;
revoke all privileges on cards from banker;
revoke all privileges on jail from banker;
revoke all privileges on game_pieces from banker;
revoke all privileges on players from banker;
revoke all privileges on bank from banker;
revoke all privileges on locations from banker;
```

```
grant select, update on squares to banker;
grant select, update on rent_info to banker;
grant select on squares_rent to banker;
grant select, update on cards to banker;
grant select, update on jail to banker;
grant select, update on game_pieces to banker;
grant select, insert, update on players to banker;
grant select, insert, update on bank to banker;
grant select, insert on locations to banker;
```

Implementation Notes

- There are only 4 players for simplicity.

- I only implemented 7 turns since monopoly can go for a very long time.

- Every property and every card is inserted into the database.

- In jail and just visiting are two different spaces in the database because they have two different actions

- Income tax pay $200 and Income tax pay $10 are also two different spaces in the database.

Known Issues

-You need to multiply the rent_mm by two whenever the squares table says that space has a monopoly on it.

-There is no function to check if you have enough money to buy something

-If there is an auction the banker must handle that separately.

-The player cannot access all the cards due to security so the banker must select a card for the player.

Future Enhancements

- The only way to track the roll of a dice is to do the math between the last known sid and the current sid in the locations table. A future stored procedure could calculate this.

- Add a function to check to see if you have enough money to buy a property

- Auctions are not included in the database

-There is no function to give the user the free parking money when they land on that space.