Ticket to Ride Database

Kylie Wasserman

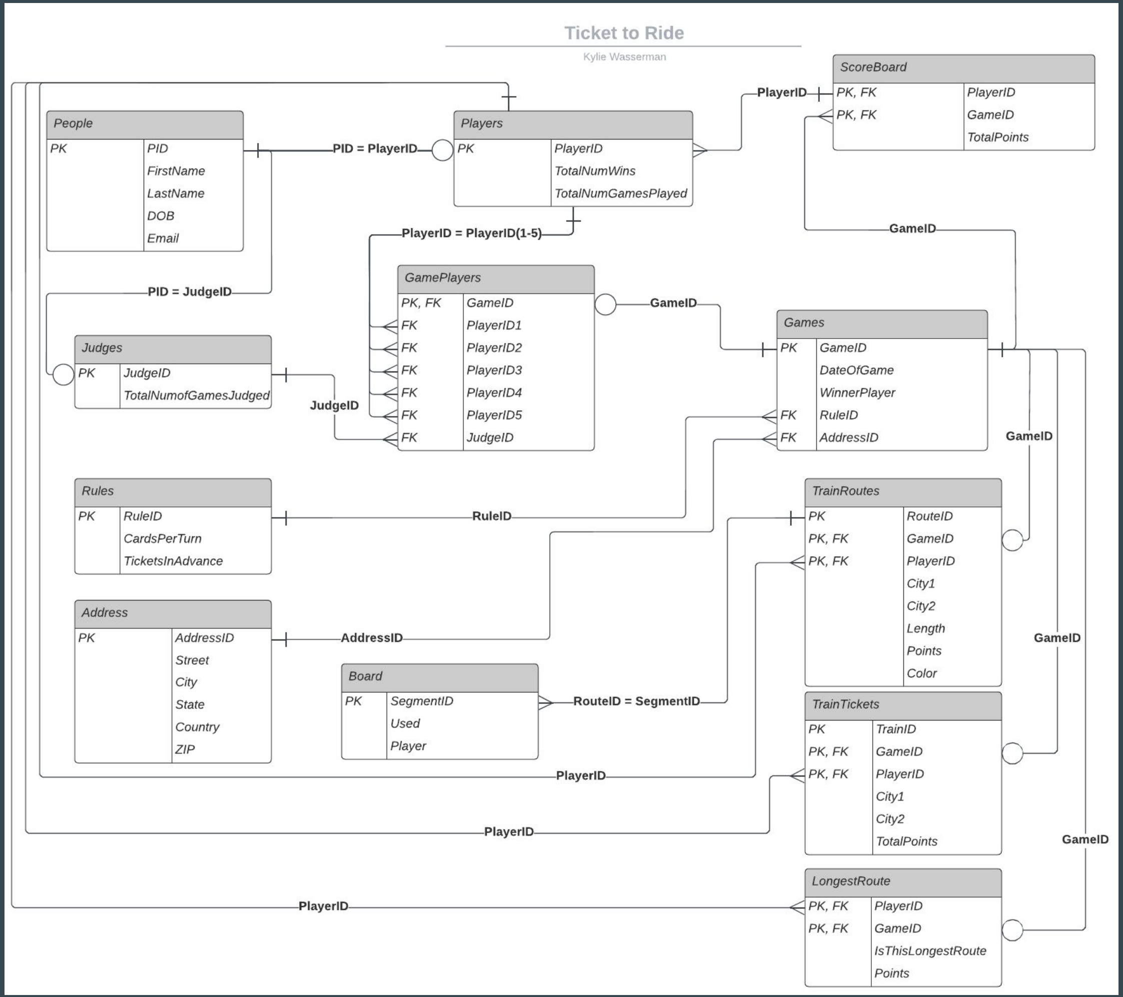
Table of Contents

Table of Contents	2
Executive Summary	3
Entity-Relationship Diagram	4
Tables	5
Stored Procedures	18
Views	22
Triggers	26
Reports	29
Security	35
Implementation Notes	37
Known Problems	37
Future Advancements	37

Executive Summary

This document outlines the design of a database to hold all of the information about the board game *Ticket to Ride*. The goal of *Ticket to Ride* is to earn points by completing train tickets that prompt you to build train routes from one city to another. Players collect cards of various colors that you then use to claim train routes in North America. The longer the route, the more points you earn. You can also earn an additional ten points if you have the longest continuous train in the game. The design of this database is to show the framework for the data that is involved in a game of *Ticket to Ride* as well as the ability to look back on previous games. With this database you are able to provide information about multiple games with specific players, judges, and point values. The information implemented into this database is fictional, with some exceptions. All people and their correlating information are fictional. The objective is to design a database that is fully functional and fully normalized in third normal form that can help game players store information about previous and current games of *Ticket to Ride* that they have played.

Entity-Relationship Diagram

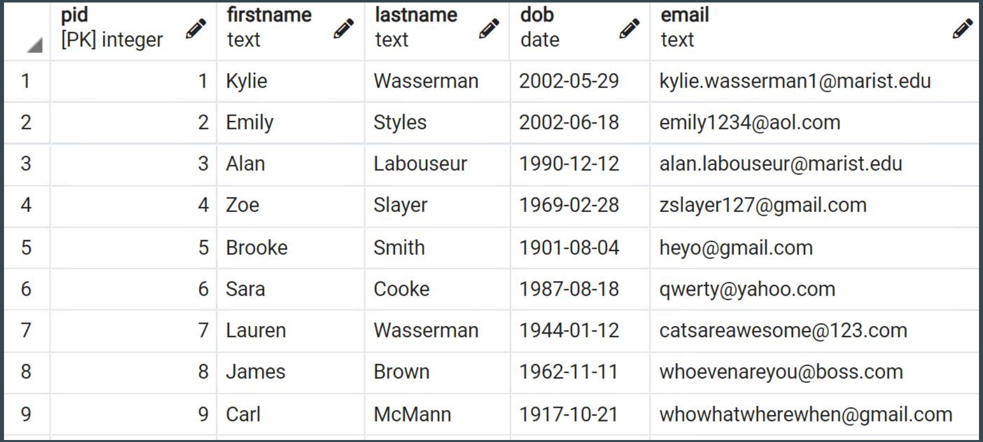


Tables

People Table

The **People** table contains all of the people and their common attributes. There are two subtypes for the People table: players and judges.

```
DROP TABLE IF  EXISTS People;
CREATE TABLE People (
   PID            int  not null unique,
   FirstName   text not null,
   LastName    text,
   DOB            date not null,
   Email          text,
 primary key(PID)
);
```



	pid [PK] integer	firstname text	lastname text	dob date	email text
1	1	Kylie	Wasserman	2002-05-29	kylie.wasserman1@marist.edu
2	2	Emily	Styles	2002-06-18	emily1234@aol.com
3	3	Alan	Labouseur	1990-12-12	alan.labouseur@marist.edu
4	4	Zoe	Slayer	1969-02-28	zslayer127@gmail.com
5	5	Brooke	Smith	1901-08-04	heyo@gmail.com
6	6	Sara	Cooke	1987-08-18	qwerty@yahoo.com
7	7	Lauren	Wasserman	1944-01-12	catsareawesome@123.com
8	8	James	Brown	1962-11-11	whoevenareyou@boss.com
9	9	Carl	McMann	1917-10-21	whowhatwherewhen@gmail.com

Functional Dependencies

PID → FirstName, LastName, DOB, Email

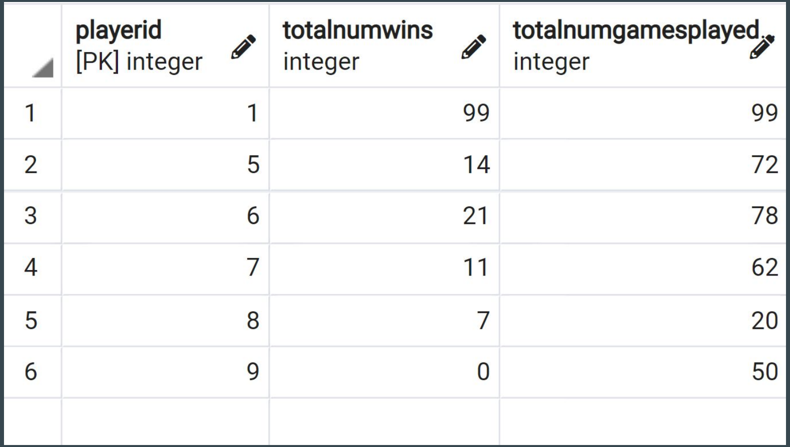
Players Table

The **Players** table contains all of the players and their common attributes. A players needs to already be a person.

```
DROP TABLE IF EXISTS Players;
CREATE TABLE Players (
  PlayerID                int not null references People(PID),
  TotalNumWins            int,
  TotalNumGamesPlayed     int,
 primary key(PlayerID)
);
```

Functional Dependencies

PlayerID → TotalNumWins, TotalNumGamesPlayed



	playerid [PK] integer	totalnumwins integer	totalnumgamesplayed integer
1	1	99	99
2	5	14	72
3	6	21	78
4	7	11	62
5	8	7	20
6	9	0	50

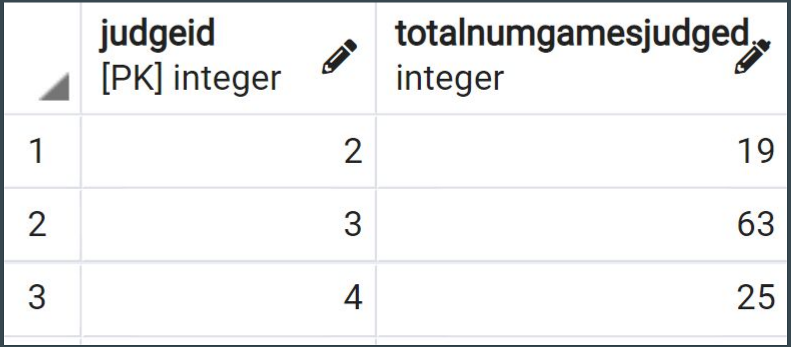
Judges Table

The **Judges** table contains all of the judges and their common attributes. A judge needs to already be a person.

```
DROP TABLE IF EXISTS Judges;
CREATE TABLE Judges (
   JudgeID                    int not null references People(pid),
   TotalNumGamesJudged  int,
 primary key(JudgeID)
);
```

Functional Dependencies

JudgeID → TotalNumGamesJudged



	judgeid [PK] integer	totalnumgamesjudged integer
1	2	19
2	3	63
3	4	25

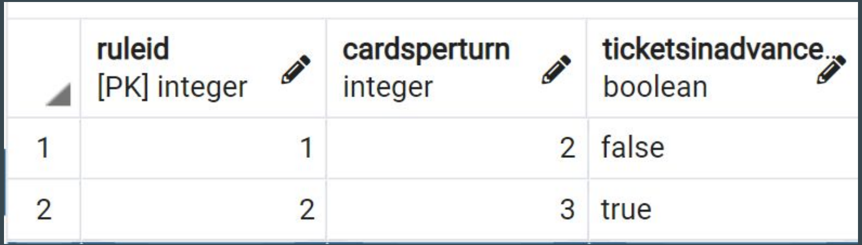
Rules Table

The **Rules** table contains all of the rules that can change depending on the way that you play them. RuleID 1 is the official rules, and RuleID 2 is the rules that my family plays by.

```
DROP TABLE IF EXISTS Rules;
CREATE TABLE Rules (
  RuleID              int    not null unique,
  CardsPerTurn        int    not null,
  TicketsInAdvance boolean not null,
CONSTRAINT CheckCards CHECK (CardsPerTurn = 2 or CardsPerTurn = 3),
 primary key(RuleID)
);
```

Functional Dependencies

RuleID → CardsPerTurn, TicketsInAdvance



	ruleid [PK] integer	cardsperturn integer	ticketsinadvance boolean
1	1	2	false
2	2	3	true

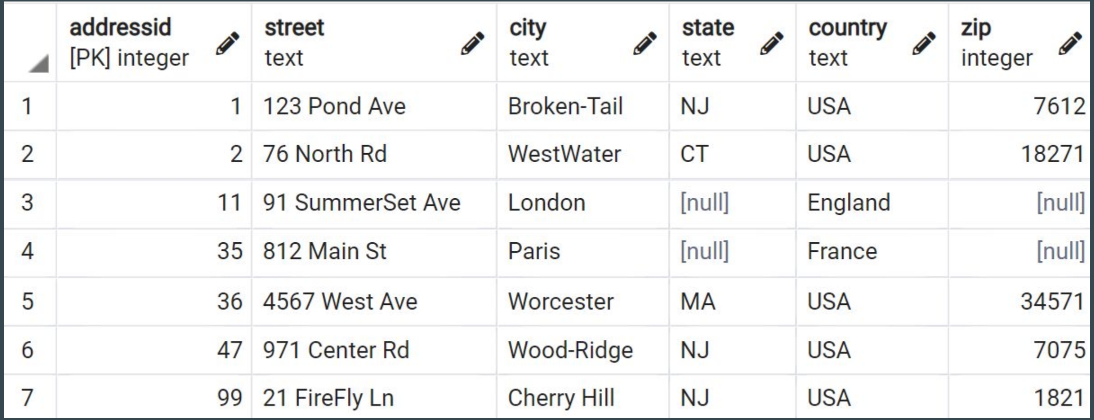
Constraints

CheckCards → Checks cards per turn only has an input of 2 or 3

Address Table

The **Address** table contains all of the addresses for where the individual games took place. Only the AddressID, Street, City, and Country need to be filled as a game could be held in another country that is not USA.

```
DROP TABLE IF EXISTS Address;
CREATE TABLE Address (
  AddressID  int  not null unique,
  Street        text not null,
  City            text not null,
  State          text,
  Country      text not null,
  ZIP             int,
 primary key(AddressID)
);
```



	addressid [PK] integer	street text	city text	state text	country text	zip integer
1	1	123 Pond Ave	Broken-Tail	NJ	USA	7612
2	2	76 North Rd	WestWater	CT	USA	18271
3	11	91 SummerSet Ave	London	[null]	England	[null]
4	35	812 Main St	Paris	[null]	France	[null]
5	36	4567 West Ave	Worcester	MA	USA	34571
6	47	971 Center Rd	Wood-Ridge	NJ	USA	7075
7	99	21 FireFly Ln	Cherry Hill	NJ	USA	1821

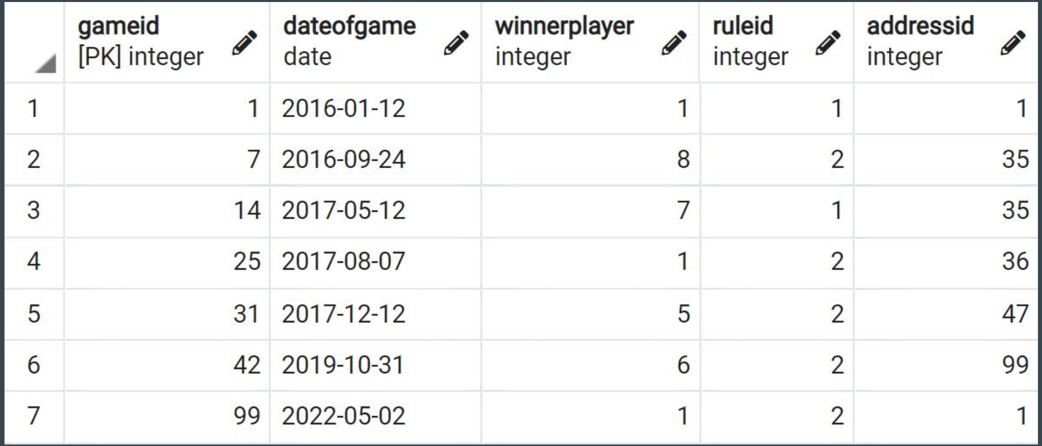
Functional Dependencies

AddressID → Street, City, State, Country, ZIP

Games Table

The **Games** table contains information about the game besides the people apart of an individual game.

```
DROP TABLE IF EXISTS Games;
CREATE TABLE Games (
  GameID          int not null unique,
  DateOfGame      date,
  WinnerPlayer    int,
  RuleID          int,
  AddressID       int,
 primary key(GameID)
);
```



	gameid [PK] integer	dateofgame date	winnerplayer integer	ruleid integer	addressid integer
1	1	2016-01-12	1	1	1
2	7	2016-09-24	8	2	35
3	14	2017-05-12	7	1	35
4	25	2017-08-07	1	2	36
5	31	2017-12-12	5	2	47
6	42	2019-10-31	6	2	99
7	99	2022-05-02	1	2	1

Functional Dependencies

GameID → DateOfGame, WinnerPlayer, RuleID, AddressID

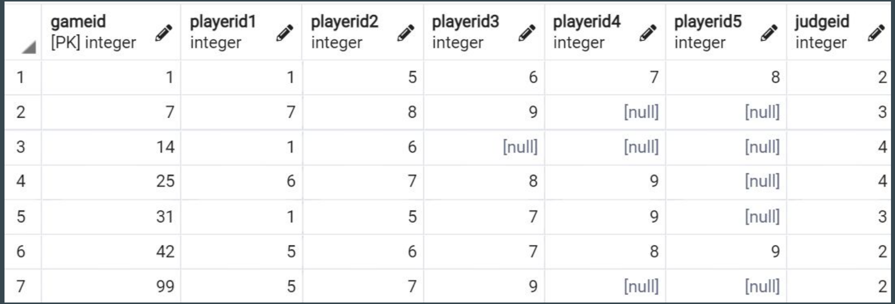
GamePlayers Table

The **GamePlayers** table contains all of the people apart of the game. Since the game is 2-5 players, only Players 1 and 2 have to be entered in order to make a game. There is only 1 judge per game.

```
DROP TABLE IF EXISTS GamePlayers;
CREATE TABLE GamePlayers (
  GameID     int    not null references Games(GameID),
  PlayerID1  int    not null references Players(PlayerID),
  PlayerID2  int    not null references Players(PlayerID),
  PlayerID3  int             references Players(PlayerID),
  PlayerID4  int             references Players(PlayerID),
  PlayerID5  int             references Players(PlayerID),
  JudgeID    int    not null references Judges(JudgeID),
 primary key(GameID)
);
```

Functional Dependencies

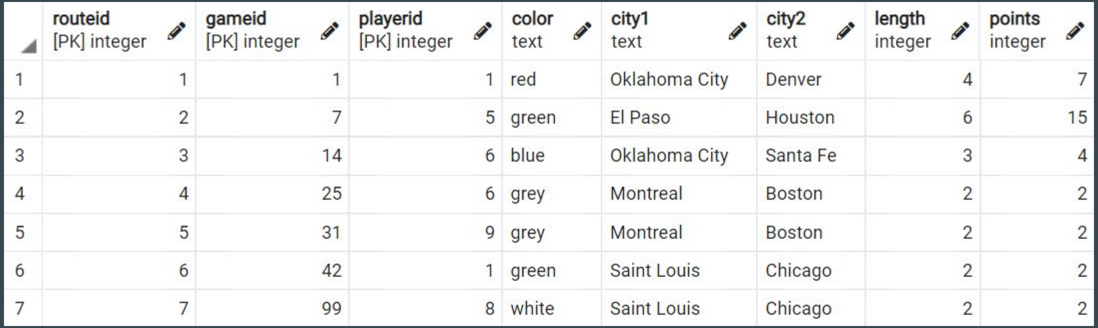
GameID → PlayerID1, PlayerID2, PlayerID3, PlayerID4, PlayerID5, JudgeID



	gameid [PK] integer	playerid1 integer	playerid2 integer	playerid3 integer	playerid4 integer	playerid5 integer	judgeid integer
1	1	1	5	6	7	8	2
2	7	7	8	9	[null]	[null]	3
3	14	1	6	[null]	[null]	[null]	4
4	25	6	7	8	9	[null]	4
5	31	1	5	7	9	[null]	3
6	42	5	6	7	8	9	2
7	99	5	7	9	[null]	[null]	2

TrainRoutes Table

The **TrainRoutes** table contains information about the individual train routes from city to city per game. The primary key is the RouteID, GameID, and the PlayerID as multiple games and different players can use the same train route.



	routeid [PK] integer	gameid [PK] integer	playerid [PK] integer	color text	city1 text	city2 text	length integer	points integer
1	1	1	1	red	Oklahoma City	Denver	4	7
2	2	7	5	green	El Paso	Houston	6	15
3	3	14	6	blue	Oklahoma City	Santa Fe	3	4
4	4	25	6	grey	Montreal	Boston	2	2
5	5	31	9	grey	Montreal	Boston	2	2
6	6	42	1	green	Saint Louis	Chicago	2	2
7	7	99	8	white	Saint Louis	Chicago	2	2

```
DROP TABLE IF EXISTS TrainRoutes;
CREATE TABLE TrainRoutes (
  RouteID     int    not null unique,
  GameID      int    not null references Games(GameID),
  PlayerID    int    not null references Players(PlayerID),
  Color       text   not null,
  City1       text   not null,
  City2       text   not null,
  Length      int    not null,
  Points      int    not null,
  CONSTRAINT CheckColor  Check (Color = 'red' OR Color = 'orange' OR Color = 'yellow' OR Color = 'green' OR Color = 'blue' OR
    Color = 'purple' OR Color = 'black' OR Color = 'white' OR Color = 'grey'),
  CONSTRAINT CheckLength Check (Length = 1 OR Length = 2 OR Length = 3 OR Length = 4 OR Length = 5 OR Length = 6),
  CONSTRAINT CheckPoints Check (Points = 1 OR Points = 2 OR Points = 4 OR Points = 7 OR Points = 10 OR Points = 15),
primary key(RouteID, GameID, PlayerID)
);
```

Constraints

CheckColor → Checks color is only one of the game options

CheckLength → Checks length is only one of the game options

CheckPoints → Checks points is only one of the game options

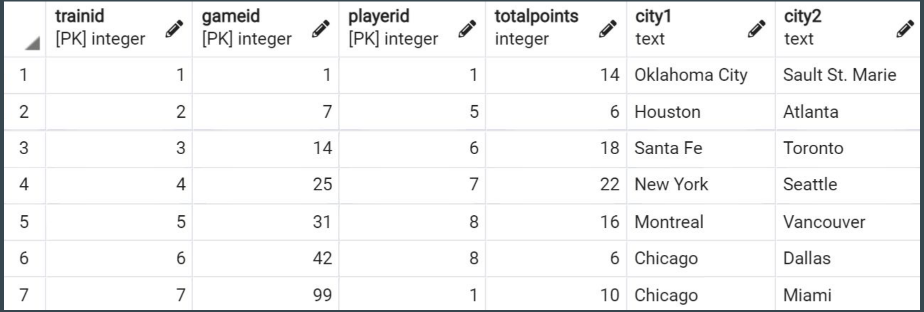
Functional Dependencies

RouteID, GameID, PlayerID → Color, City1, City2, Length, Points

TrainTickets Table

The **TrainTickets** table contains information about train tickets that players receive throughout the game. The ticket can be used by different players in each game, but only once. No two tickets have the same two cities, but many tickets share the same number of points.

```
DROP TABLE IF EXISTS TrainTickets;
CREATE TABLE TrainTickets (
  TrainID       int     not null unique,
  GameID        int     not null references Games(GameID),
  PlayerID      int     not null references Players(PlayerID),
  TotalPoints   int     not null,
  City1         text    not null,
  City2         text    not null,
primary key(TrainID, GameID, PlayerID)
);
```



	trainid [PK] integer	gameid [PK] integer	playerid [PK] integer	totalpoints integer	city1 text	city2 text
1	1	1	1	14	Oklahoma City	Sault St. Marie
2	2	7	5	6	Houston	Atlanta
3	3	14	6	18	Santa Fe	Toronto
4	4	25	7	22	New York	Seattle
5	5	31	8	16	Montreal	Vancouver
6	6	42	8	6	Chicago	Dallas
7	7	99	1	10	Chicago	Miami

Functional Dependencies

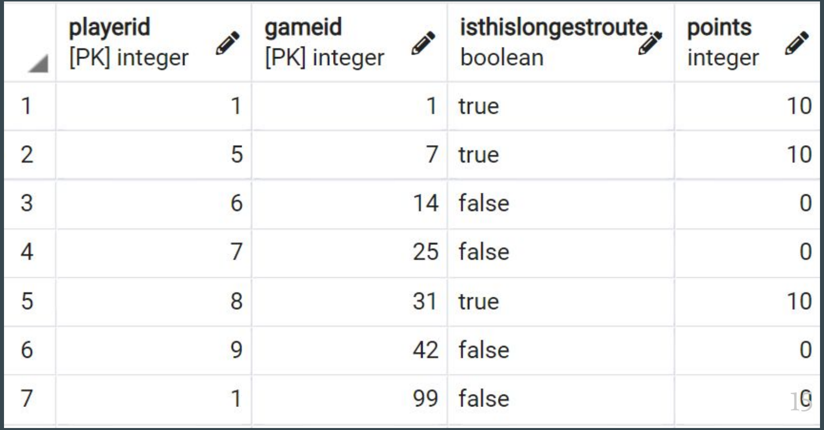
TrainID, GameID, PlayerID → TotalPoints, City1, City2

LongestRoute Table

The **LongestRoute** table contains whether or not a certain player in a game has the longest continuous train. If they do, then they are awarded an additional 10 points, if not they receive 0 points.

```
DROP TABLE IF EXISTS LongestRoute;
CREATE TABLE LongestRoute (
  PlayerID              int       not null references Players(PlayerID),
  GameID                int       not null references Games(GameID),
  IsThisLongestRoute  boolean not null,
  Points                int       not null,
primary key(PlayerID, GameID)
);
```

Functional Dependencies
PlayerID, GameID → IsThisLongestRoute, Points



	playerid [PK] integer	gameid [PK] integer	isthislongestroute boolean	points integer
1	1	1	true	10
2	5	7	true	10
3	6	14	false	0
4	7	25	false	0
5	8	31	true	10
6	9	42	false	0
7	1	99	false	0

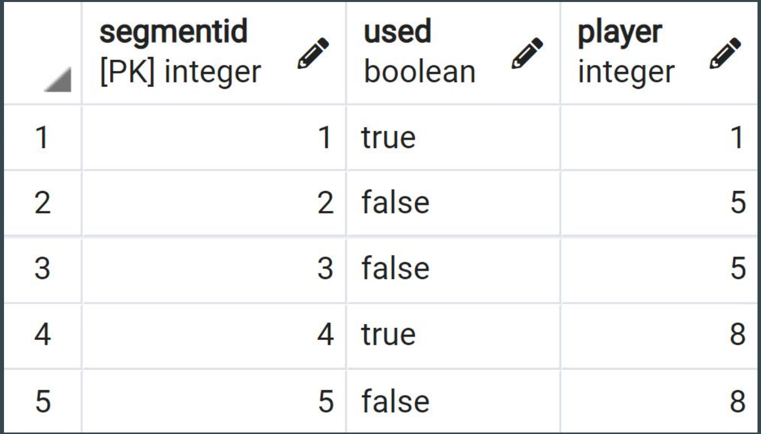
Board Table

The **Board** table informs you of which player is using certain train routes in a given game. If no player is using a train route, then the ID of the player is not needed.

```
DROP TABLE IF EXISTS Board;
CREATE TABLE Board (
  SegmentID    int        not null references TrainRoutes(RouteID),
  Used         boolean    not null,
  Player       int,
primary key(SegmentID)
);
```

Functional Dependencies

SegmentID → Used, Player

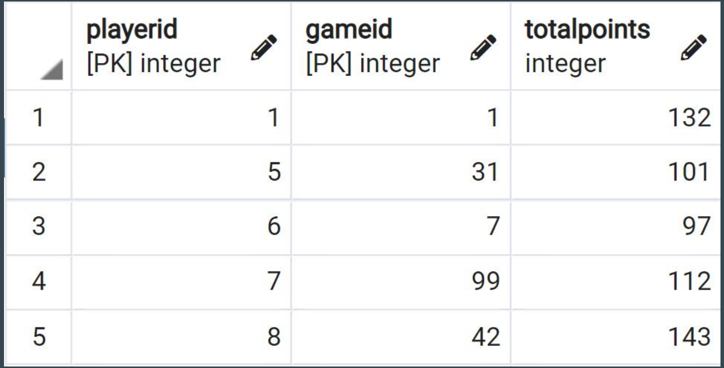


	segmentid [PK] integer	used boolean	player integer
1	1	true	1
2	2	false	5
3	3	false	5
4	4	true	8
5	5	false	8

ScoreBoard Table

The **ScoreBoard** table contains of the scores of all of the players in each game. A player can play in multiple games.

```
DROP TABLE IF EXISTS ScoreBoard;
CREATE TABLE ScoreBoard (
  PlayerID        int     not null references Players(PlayerID),
  GameID          int     not null references Games(GameID),
  TotalPoints     int     not null,
primary key(PlayerID, GameID)
);
```



	playerid [PK] integer	gameid [PK] integer	totalpoints integer
1	1	1	132
2	5	31	101
3	6	7	97
4	7	99	112
5	8	42	143

Functional Dependencies

PlayerID, GameID → TotalPoints

Stored Procedures

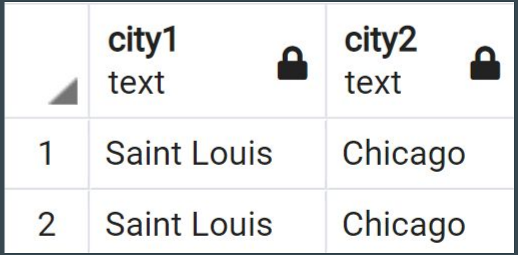
get_other_city

This will get all of the city route options given one city.

```
create or replace function get_other_city(text, REFCURSOR) returns refcursor as
$$
declare
   givencity   text      := $1;
   resultset   REFCURSOR := $2;
begin
   open resultset for
      select city1, city2
      from   TrainRoutes
      where  city1 = givencity or city2 = givencity;
   return resultset;
end;
$$
language plpgsql;
```

Sample Output:

```
select get_other_city('Chicago', 'results');
Fetch all from results;
```



	city1 text	city2 text
1	Saint Louis	Chicago
2	Saint Louis	Chicago

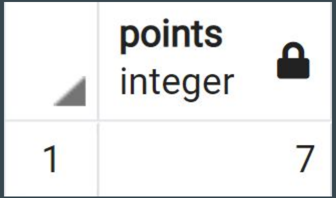
get_points

This will get the amount of points of a train route given the length of the train route.

```
create or replace function get_points(int, REFCURSOR) returns refcursor as
$$
declare
   givenlength int       := $1;
   resultset   REFCURSOR := $2;
begin
   open resultset for
      select points
      from   TrainRoutes
       where length = givenlength;
   return resultset;
end;
$$
language plpgsql;
```

Sample Output:

```
select get_points(4, 'results');
Fetch all from results;
```



	points integer
1	7

did_i_win

This will return if a player won a certain game given the id of each the player and the game. If something returns, then that means that the playerid matched the winnerid. If nothing is returned in the table, then the player that you entered, did not win the game that was entered.

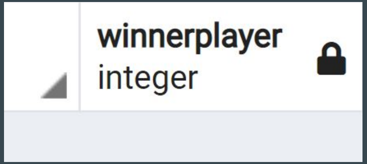
```
create or replace function did_i_win(int, int, REFCURSOR) returns refcursor as
$$
declare
  player     int       := $1;
  game       int       := $2;
  resultset  REFCURSOR := $3;
begin
  open resultset for
    select winnerplayer
    from games
    where winnerplayer = player and gameid = game;
  return resultset;

end;
$$
language plpgsql;
```

Sample Output:

Player didn't win the game entered:

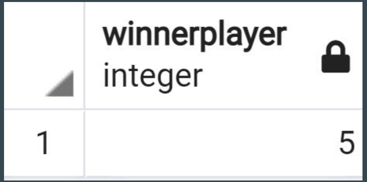
```
select did_i_win(5, 32, 'results');
Fetch all from results;
```



```
winnerplayer
integer
```

Player did win the game entered:

```
select did_i_win(5, 31, 'results');
Fetch all from results;
```



```
   winnerplayer
   integer
1             5
```

Views

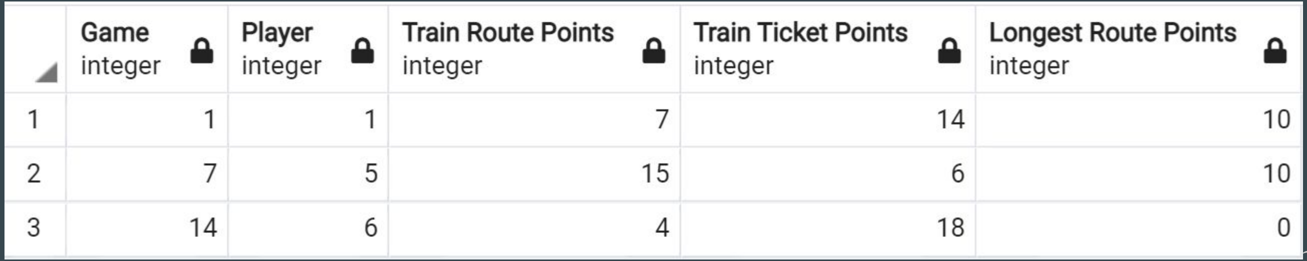
TotalGamePlayerPoints

This view contains all three areas of points of a given player and game.

```
CREATE OR REPLACE VIEW TotalGamePlayerPoints as (
    select tr.gameid as "Game", tr.playerid as "Player", tr.points as "Train Route Points",
              tt.totalpoints as "Train Ticket Points", lr.points as "Longest Route Points"
          from trainroutes tr inner join traintickets tt on tr.playerid = tt.playerid
                                    inner join longestroute lr on tt.playerid = lr.playerid
          where tt.gameid = lr.gameid
              and tt.gameid = tr.gameid
);
```

Sample Output:

```
select *
from TotalGamePlayerPoints;
```



	Game integer	Player integer	Train Route Points integer	Train Ticket Points integer	Longest Route Points integer
1	1	1	7	14	10
2	7	5	15	6	10
3	14	6	4	18	0

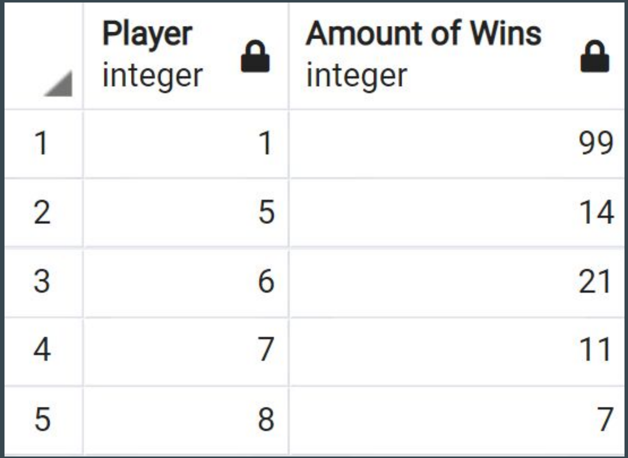
WinnerPlayers

This view contains all of the Players that have won a game and that amount.

```
CREATE OR REPLACE VIEW WinnerPlayers as (
        select p.playerid as "Player", p.totalnumwins as "Amount of Wins"
        from players p
        where p.totalnumwins != 0
);
```

Sample Output:

```
select *
from WinnerPlayers;
```



	Player integer	Amount of Wins integer
1	1	99
2	5	14
3	6	21
4	7	11
5	8	7

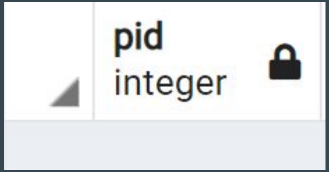
JudgePlayer

This view contains all of the people who are both players and judges.

```
CREATE OR REPLACE VIEW JudgePlayer as (
        select pe.PID
        from People pe inner join Players pl on pe.PID = pl.PlayerID
                                inner join Judges j on pl.PlayerID = j.JudgeID
);
```

Sample Output:

```
select *
from JudgePlayer;
```



```
pid
integer
```

Triggers

ValidatePeople

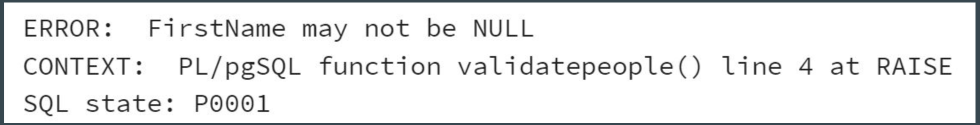
When a new person is entered into the People table, this trigger is called to make sure that a first name and DOB is inputted.

```
CREATE OR REPLACE FUNCTION ValidatePeople()
RETURNS TRIGGER AS
$$
BEGIN
  IF NEW.FirstName IS NULL THEN
  RAISE EXCEPTION 'FirstName may not be NULL';
  END IF;
  RETURN NEW;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER validPeople
BEFORE INSERT OR UPDATE ON People
FOR EACH ROW
EXECUTE PROCEDURE ValidatePeople();
```

Sample Output:

```
INSERT INTO People (PID, FirstName,
                    LastName, DOB, Email)
       VALUES(010, NULL, NULL, '2000-01-01',
              'sample@email.com');
```



```
ERROR:  FirstName may not be NULL
CONTEXT:  PL/pgSQL function validatepeople() line 4 at RAISE
SQL state: P0001
```

EnoughPlayers

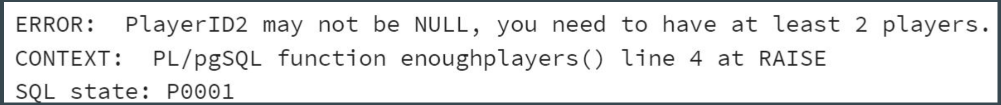
When a game is added to GamePlayers, this trigger is called to make sure that there is a player entered in both PlayerID1 and PlayerID2 as a game needs at least 2 players.

```
CREATE OR REPLACE FUNCTION EnoughPlayers()
RETURNS TRIGGER AS
$$
BEGIN
 IF NEW.PlayerID2 IS NULL THEN
 RAISE EXCEPTION 'PlayerID2 may not be NULL, you need to have at least 2 players.';
 END IF;
 RETURN NEW;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER EnoughPlay
BEFORE INSERT OR UPDATE ON GamePlayers
FOR EACH ROW
EXECUTE PROCEDURE EnoughPlayers();
```

Sample Output:

```
INSERT INTO GamePlayers(GameID, PlayerID1,
        PlayerID2, PlayerID3, PlayerID4, PlayerID5, JudgeID)
        VALUES(002, 001, null, null, null, null, 003);
```



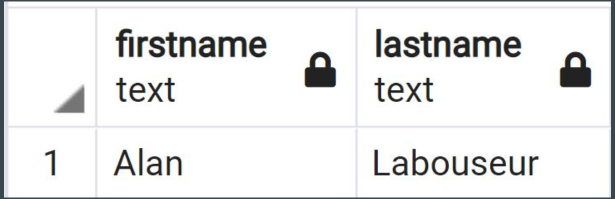
```
ERROR:  PlayerID2 may not be NULL, you need to have at least 2 players.
CONTEXT:  PL/pgSQL function enoughplayers() line 4 at RAISE
SQL state: P0001
```

Reports

Report 1

This report selects all judges that were born in December of any year.

```
select p.FirstName, p.LastName
from People p inner join Judges j on p.PID = j.JudgeID
where extract(month from p.DOB)='12';
```

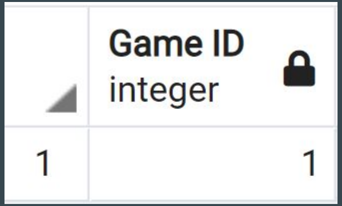


	firstname text	lastname text
1	Alan	Labouseur

Report 2

This report selects all of the games that Judge 002 judged in 2016 with Rule 1.

```
select g.GameID as "Game ID"
from Judges j inner join GamePlayers gp on j.JudgeID = gp.JudgeID
              inner join Games g        on gp.GameID = g.GameID
where j.JudgeID = 2
  and g.RuleID = 1
  and extract(year from g.DateOfGame)='2016';
```



	Game ID integer
1	1

Report 3

This report selects the first and last name of all of the players who have played a game in MA. These players are then sorted by last name in order A-Z.

1

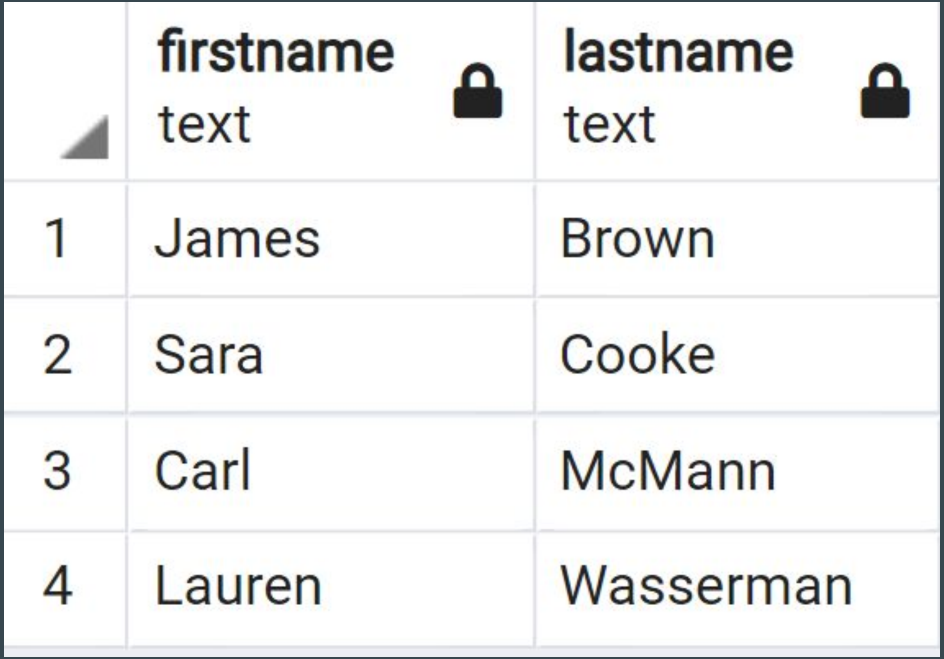
```
select pe.firstname, pe.lastname
from people pe
where pe.pid in (select pl.playerid
                 from players pl
                 where pl.playerid in (select gp.playerid1
                                       from gameplayers gp
                                       where gp.gameid in (select g.gameid
                                                           from games g
                                                           where g.addressid in (select a.addressid
                                                                                 from address a
                                                                                 where a.state = 'MA')))
                 or pl.playerid in (select gp.playerid2
                                    from gameplayers gp
                                    where gp.gameid in (select g.gameid
                                                        from games g
                                                        where g.addressid in (select a.addressid
                                                                              from address a
                                                                              where a.state = 'MA')))
                 or pl.playerid in (select gp.playerid3
                                    from gameplayers gp
                                    where gp.gameid in (select g.gameid
                                                        from games g
                                                        where g.addressid in (select a.addressid
                                                                              from address a
                                                                              where a.state = 'MA')))
                 or pl.playerid in (select gp.playerid4
                                    from gameplayers gp
                                    where gp.gameid in (select g.gameid
                                                        from games g
                                                        where g.addressid in (select a.addressid
                                                                              from address a
                                                                              where a.state = 'MA')))
```

2

```
                 or pl.playerid in (select gp.playerid5
                                    from gameplayers gp
                                    where gp.gameid in (select g.gameid
                                                        from games g
                                                        where g.addressid in (select a.addressid
                                                                              from address a
                                                                              where a.state = 'MA'))))
```

3

```
order by lastname ASC;
```



	firstname text 🔒	lastname text 🔒
1	James	Brown
2	Sara	Cooke
3	Carl	McMann
4	Lauren	Wasserman

Security

User Roles: Admin, GameMaster, GameInfoFinder

Admin: Database Administrator has full control of the DB
GameMaster: People in charge of entering information about the people involved in the individual games
GameInfoFinder: People in charge of altering rules and information about the game Ticket To Ride

```
CREATE ROLE ADMIN;
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO ADMIN;

CREATE ROLE GameInfoFinder;
REVOKE ALL ON ALL TABLES IN SCHEMA PUBLIC
FROM GameInfoFinder;
GRANT SELECT ON Rules, Games, TrainRoutes,
TrainTickets, LongestRoute
TO GameInfoFinder;
GRANT INSERT ON Rules, Games, TrainRoutes,
TrainTickets, LongestRoute
TO GameInfoFinder;
GRANT UPDATE ON Rules, Games, TrainRoutes,
TrainTickets, LongestRoute
TO GameInfoFinder;
```

```
CREATE ROLE GameMaster;
REVOKE ALL ON ALL TABLES IN SCHEMA PUBLIC
FROM GameMaster;
GRANT SELECT ON ALL TABLES IN SCHEMA
PUBLIC TO GameMaster;
GRANT INSERT ON People, Players, Judges, Address,
Games, GamePlayers, TrainRoutes, TrainTickets,
LongestRoute, Board, ScoreBoard
TO GameMaster;
GRANT UPDATE ON People, Players, Judges, Address,
Games, GamePlayers, TrainRoutes, TrainTickets,
LongestRoute, Board, ScoreBoard
TO GameMaster;
```

Implementation Notes/Known Problems/Future Advancements

- Implementation Notes
  - With a larger data sample (all games from 1-99, all train route options, and all train ticket options) you would be able to make many more interesting queries.
  - I have added GameID and PlayerID to tables TrainRoutes and TrainTickets for ease of searching as every game has a different player using a route from TrainRoutes and ticket from TrainTickets.
  - There are more rules then just cards per turn and tickets in advance, but these are the two rules that change for my family when we play the game.
- Known Problems
  - Since the total points for each train tickets, train routes, and longest route are not an FK or a PK, I am unable to add them to another table by referencing them, instead I have to do this through creating a view.
- Future Advancements
  - Implement checks on cities to make sure it is a valid city that is on the gameboard (ie Trenton would not be allowed, but Atlanta would be)
  - Implement a way to make sure that if a game is larger than two players, that a player is inputted for PlayerID3 before a player is inputted for PlayerID4
  - Implement a table and procedures for making moves in a game so that the database includes more details about current games, and not past games
  - Add a ranking of players so that the database caters to a tournament style play