ZombieFox:
App Based Human Vs Zombie Game

Created by
Michael Shershin

Table of Contents

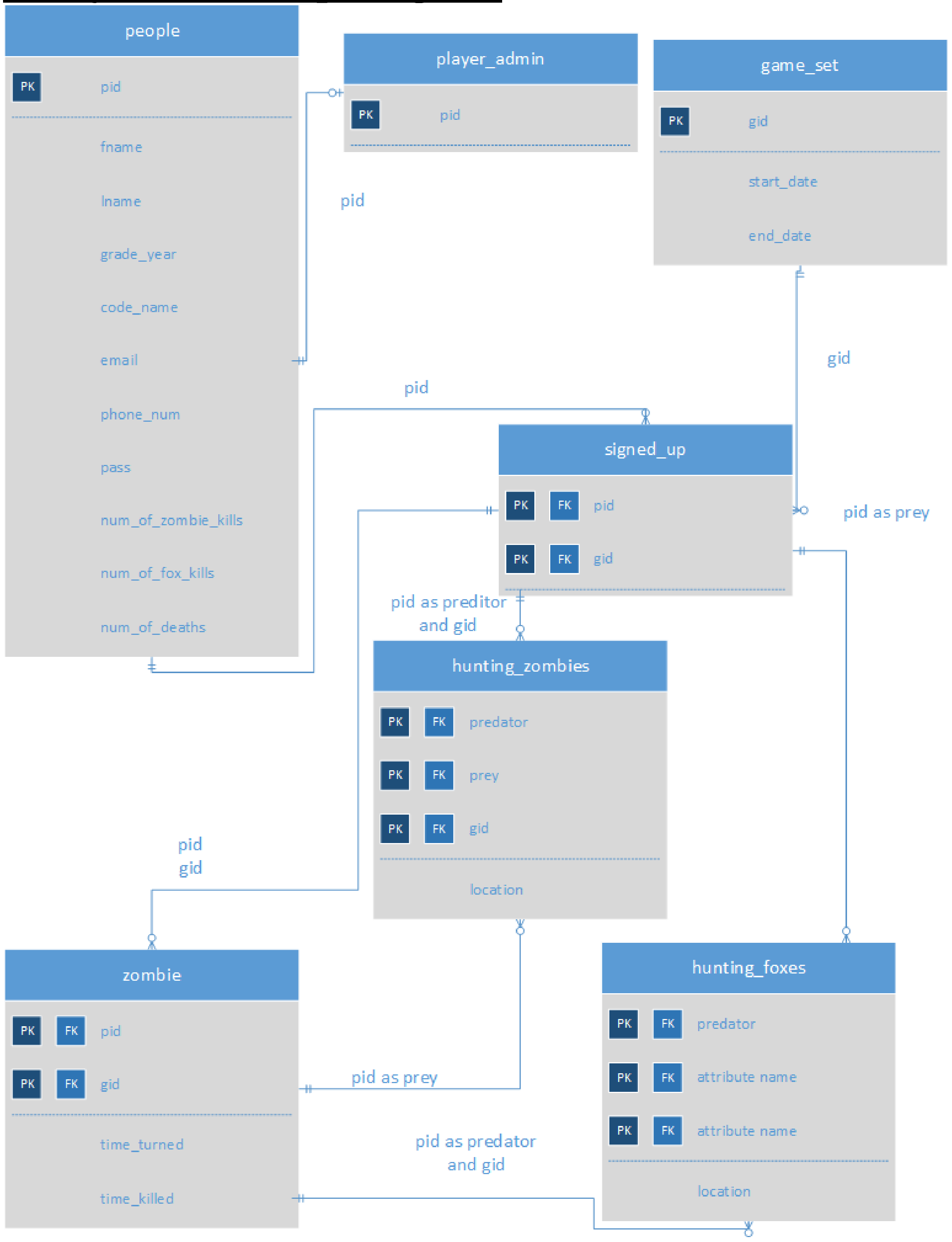
Executive Summary

This document is a strong representation of my design and implementation for my group's app, ZombieFox. This database is a way to see all the players in the app as well as their current role in the game. The users would be anyone who uses the app and play the game.

ZombieFox is an app based off of the popular college game Humans Vs Zombies. The game consists of two groups of people Humans and Zombies, and they both hunt each other for victory. This app allows the users to have access to who is playing the game when and where ever they are.

To start things off I have the Entity Relationship Diagram to show all the tables and their relations. Then comes the tables and their create statements and a bit of data that is in each table. Followed by enums, views and stored procedures. Next will be Security, followed by known problems and plans for the future.

Entity Relationship Diagram



Tables

People table

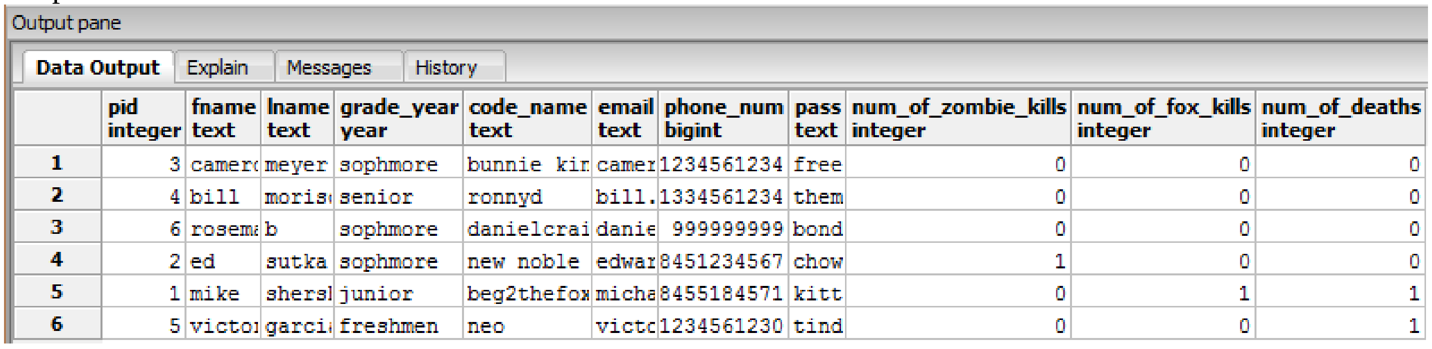
The people table is the starting point for all users of the app.

```
CREATE TABLE people(
	pid serial not null,
	fname text,
	lname text,
	grade_year year,
	code_name text UNIQUE,
	email text UNIQUE,
	phone_num bigint UNIQUE,
	pass text,
	num_of_zombie_kills int,
	num_of_fox_kills int,
	num_of_deaths int,
	primary key(pid));
```

Functional Dependencies

pid -> fname, lname, grade_year, code_name, email, phone_num, pass, num_of_zombie_kills, num_of_fox_kills, num_of_deaths.

Output:



Output pane

Data Output | Explain | Messages | History

	pid integer	fname text	lname text	grade_year year	code_name text	email text	phone_num bigint	pass text	num_of_zombie_kills integer	num_of_fox_kills integer	num_of_deaths integer
1	3	camer	meyer	sophmore	bunnie kir	camer	1234561234	free	0	0	0
2	4	bill	moris	senior	ronnyd	bill.	1334561234	them	0	0	0
3	6	rosema	b	sophmore	danielcrai	danie	999999999	bond	0	0	0
4	2	ed	sutka	sophmore	new noble	edwar	8451234567	chow	1	0	0
5	1	mike	shersl	junior	beg2thefox	micha	8455184571	kitt	0	1	1
6	5	victo	garci	freshmen	neo	victo	1234561230	tind	0	0	1

Game table

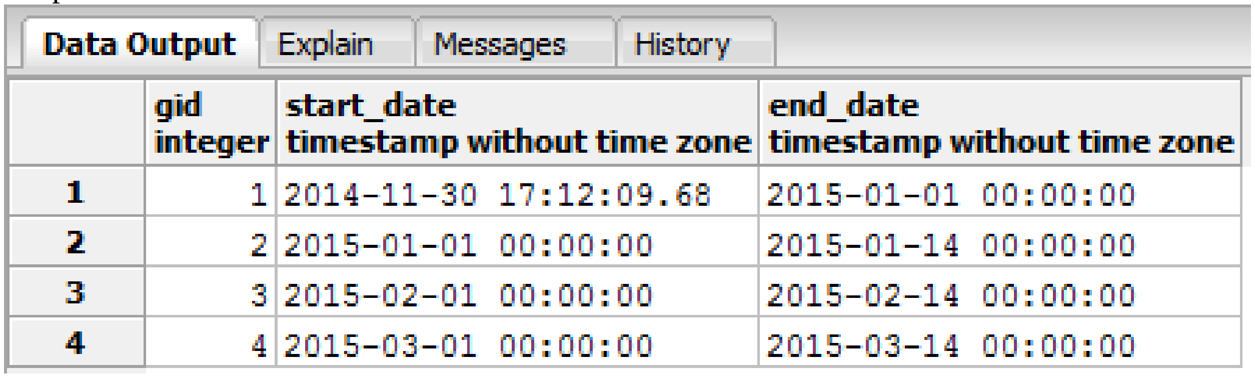
The game table is where each session of a game is started.

```
CREATE TABLE game(
	gid SERIAL not null,
	start_date TIMESTAMP,
	end_date TIMESTAMP,
	primary key(gid));
```

Functional Dependencies

	gid -> start_date, end_date

Output:



	gid integer	start_date timestamp without time zone	end_date timestamp without time zone
1	1	2014-11-30 17:12:09.68	2015-01-01 00:00:00
2	2	2015-01-01 00:00:00	2015-01-14 00:00:00
3	3	2015-02-01 00:00:00	2015-02-14 00:00:00
4	4	2015-03-01 00:00:00	2015-03-14 00:00:00

Player Admin table

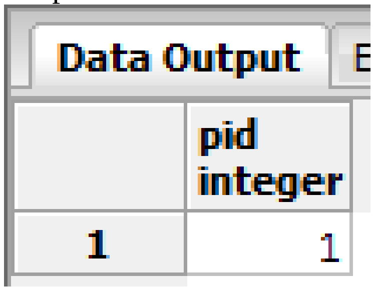
This table is to identify if a user of the app has admin power within the app itself and not so much in the database.

```
CREATE TABLE player_admin(
        pid SERIAL not null references people(pid),
        primary key(pid));
```

Functional Dependencies

pid

Output:



	pid integer
1	1

Hunting Zombies Table

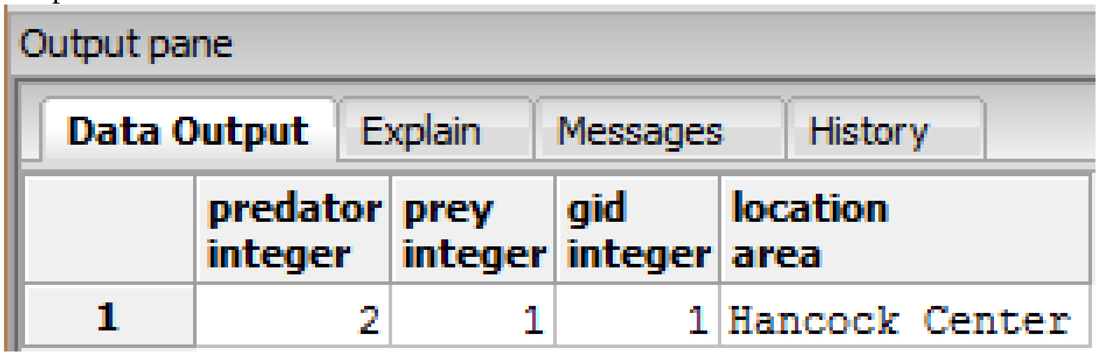
The hunting zombies table is to keep track of which fox player killed which zombie player in a given game.

```
CREATE TABLE hunting_zombies(
        predator SERIAL not null references people(pid),
        prey SERIAL not null references people(pid),
        gid SERIAL not null references game(gid),
        location area,
        primary key(predator, prey, gid));
```

Functional Dependencies

predator, prey, gid -> location

Output:



	predator integer	prey integer	gid integer	location area
1	2	1	1	Hancock Center

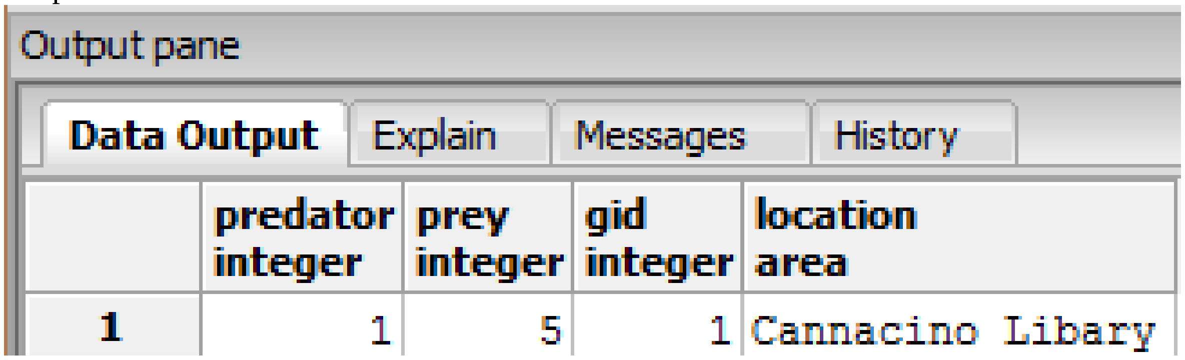
Hunting Foxes Table

The hunting foxes table is to keep track of which zombie player killed which gox player in a given game.

```
CREATE TABLE hunting_foxes(
	predator SERIAL not null references people(pid),
	prey SERIAL not null references people(pid),
	gid SERIAL not null references game(gid),
	location area,
	primary key(predator, prey, gid));
```

Functional Dependencies
	predator, prey, gid -> location

Output:



	predator integer	prey integer	gid integer	location area
1	1	5	1	Cannacino Libary

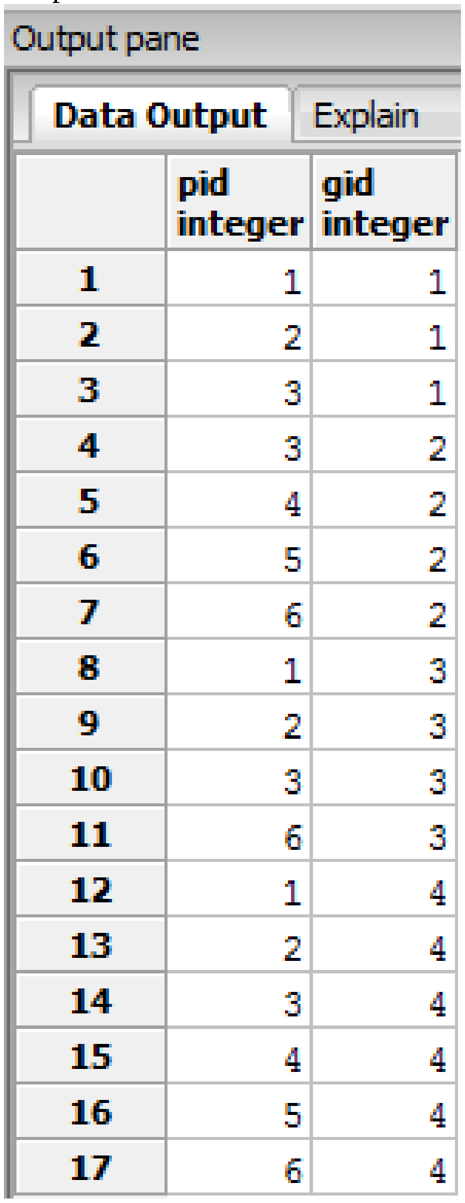
Signed Up Table

The signed up table is to show which players are signed up for a select game.

```
CREATE TABLE signed_up(
        pid SERIAL not null references people(pid),
        gid SERIAL not null references game(gid),
        primary key(pid, gid));
```

Functional Dependencies
    Pid, gid

Output:



	pid integer	gid integer
1	1	1
2	2	1
3	3	1
4	3	2
5	4	2
6	5	2
7	6	2
8	1	3
9	2	3
10	3	3
11	6	3
12	1	4
13	2	4
14	3	4
15	4	4
16	5	4
17	6	4

Zombie Table

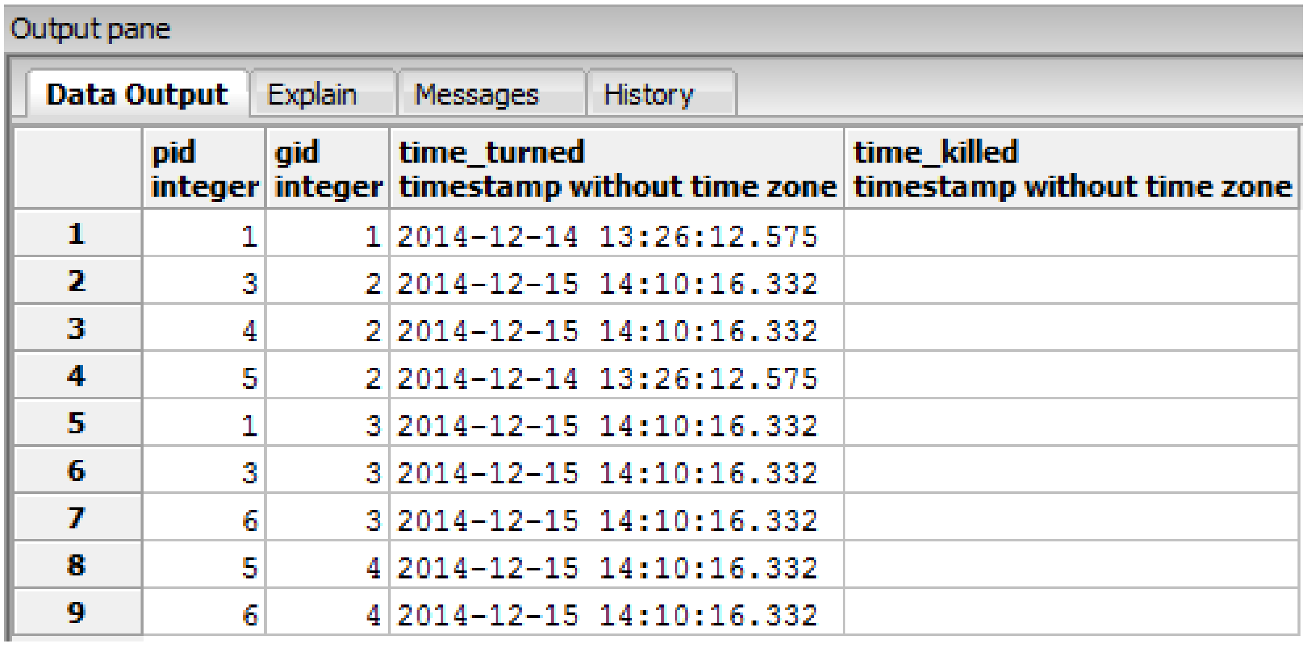
The zombie table is so we can see out of the players signed up for a game who are zombies.

```
CREATE TABLE zombie(
        pid serial not null references people(pid),
        gid SERIAL not null references game(gid),
        time_turned TIMESTAMP,
        time_killed TIMESTAMP,
        primary key(pid, gid));
```

Functional Dependencies

pid, gid -> time_turned, time_killed

Output:



	pid integer	gid integer	time_turned timestamp without time zone	time_killed timestamp without time zone
1	1	1	2014-12-14 13:26:12.575	
2	3	2	2014-12-15 14:10:16.332	
3	4	2	2014-12-15 14:10:16.332	
4	5	2	2014-12-14 13:26:12.575	
5	1	3	2014-12-15 14:10:16.332	
6	3	3	2014-12-15 14:10:16.332	
7	6	3	2014-12-15 14:10:16.332	
8	5	4	2014-12-15 14:10:16.332	
9	6	4	2014-12-15 14:10:16.332	

Enums

Year

I decided to create an enum for grade year. I decided to do this because in the app we would like to know which year they are in so we know which grade is using the app. Instead of just asking the user to fill out a little text box we thought it would be easier if the user just had a drop down menu this way we can cut down on spelling mistakes.

```
CREATE TYPE year AS ENUM ('freshmen', 'sophmore', 'junior', 'senior');
```

Area

The same thought process for year came into play again for area. But with area we would be able to let the user tell the other users where he just got a kill. This would give the game a bit of a first person shooter feel because the user who died in that area team mates would see where they died. This would give the other players the ability to stay far away from that area or go and avenge their comrade's death.

```
CREATE TYPE area AS ENUM ('Bryne House', 'Cannacino Libary', 'Champagnat Hall', 'Our
Lady Seat of Wisdom Chapel', 'Cornell Boathouse', 'Donnelly Hall', 'Dyson Center', 'Fern Tor',
'Fontaine Hall', 'Gartland Apartments', 'Greystone Hall', 'Hancock Center', 'Kieran Gatehouse',
'Kirk House', 'Leo Hall', 'Longview Park', 'Lowell Thomas Communications Center', 'Lower
Townhouses', 'Marian Hall', 'Marist Boathouse', 'McCann Recreation Center', 'Mid-rise Hall', 'St.
Anns Hermitage', 'St. Peter', 'Sheahan Hall', 'Steel Plant Studios and Gallery', 'Student Center',
'Foy Townhouses', 'Lower West Cedar Townhouses', 'Upper West Cedar Townhouses', 'Fulton
street Townhouses', 'Lower Fulton Townhouses');
```

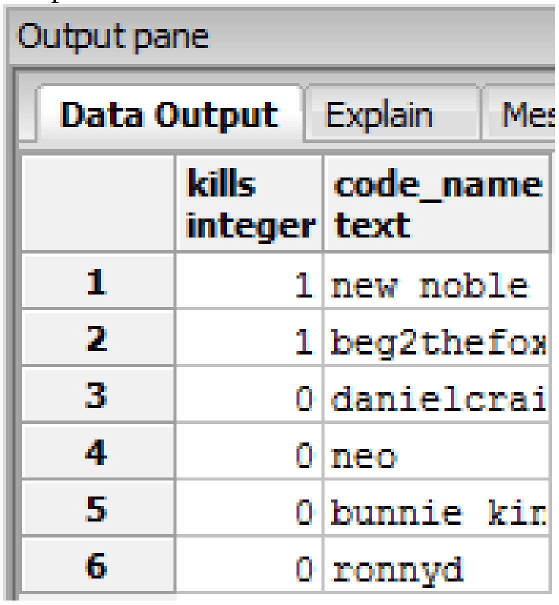
Queries

Total Kills

A basic query that returns the players code name with their current total kills.

```
select max(num_of_zombie_kills) + max(num_of_fox_kills) as Kills, code_name
from people
group by code_name
order by Kills DESC;
```

Output:



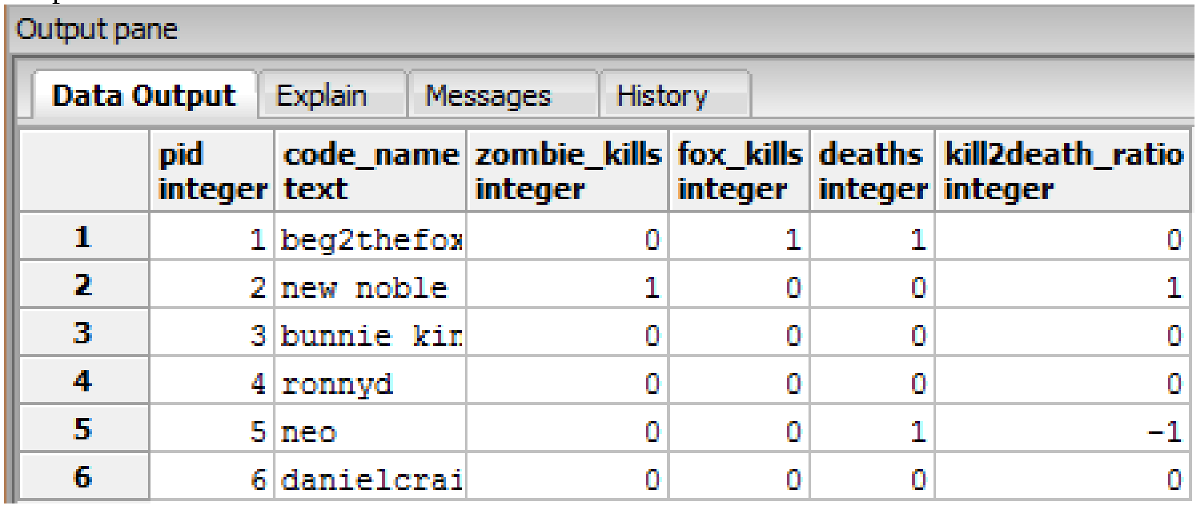
	kills integer	code_name text
1	1	new noble
2	1	beg2thefox
3	0	danielcrai
4	0	neo
5	0	bunnie kir
6	0	ronnyd

Player Stats

A query that is to show the players current kill to death ratio

select pid, code_name, num_of_zombie_kills as zombie_kills, num_of_fox_kills as fox_kills, num_of_deaths as deaths, ((num_of_zombie_kills + num_of_fox_kills)- num_of_deaths) as Kill2Death_ratio
from people
order by pid;

Output:



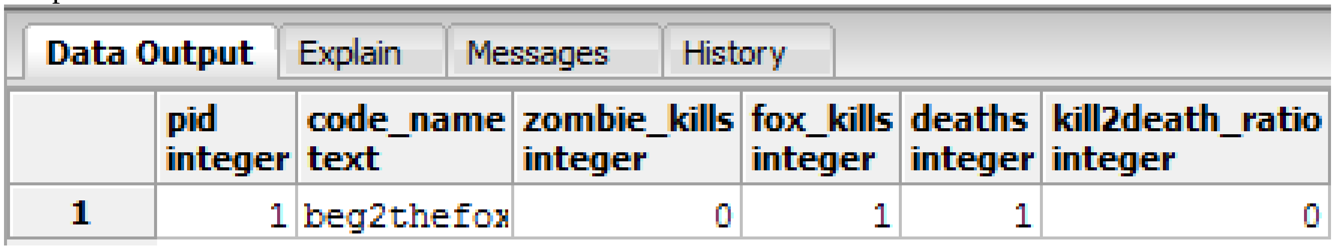
	pid integer	code_name text	zombie_kills integer	fox_kills integer	deaths integer	kill2death_ratio integer
1	1	beg2thefox	0	1	1	0
2	2	new noble	1	0	0	1
3	3	bunnie kir	0	0	0	0
4	4	ronnyd	0	0	0	0
5	5	neo	0	0	1	-1
6	6	danielcrai	0	0	0	0

Admin Stats

The same as the player stats but this one focuses just on the admins, so players can see how much better they are than the admins.

```
select pa.pid, p.code_name, p.num_of_zombie_kills as zombie_kills, p.num_of_fox_kills as
fox_kills, p.num_of_deaths as deaths, ((p.num_of_zombie_kills + p.num_of_fox_kills)-
p.num_of_deaths) as Kill2Death_ratio
from player_admin pa, people p
where p.pid = pa.pid;
```

Output:



	pid integer	code_name text	zombie_kills integer	fox_kills integer	deaths integer	kill2death_ratio integer
1	1	beg2thefox	0	1	1	0

Views

Zombie Wins

This view is to show how many times the zombies have beaten the foxes in their matches.

```
CREATE VIEW amount_of_zombie_wins as
select count(z.pid)
from zombie z, signed_up s
where z.pid > s.pid and z.gid = s.gid
group by z.gid;
```

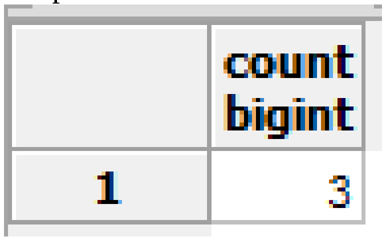
I currently do not have anything to show in this view because in my sample data the zombies still have yet to receive a victory over the foxes.

Fox Wins

This view is to show how many times foxes have survived the attacks of the zombie horde.

```
CREATE VIEW amount_of_fox_wins as
select count(s.pid)
from signed_up s, zombie z
where s.pid > z.pid and s.gid = z.gid
group by s.gid;
```

Output:



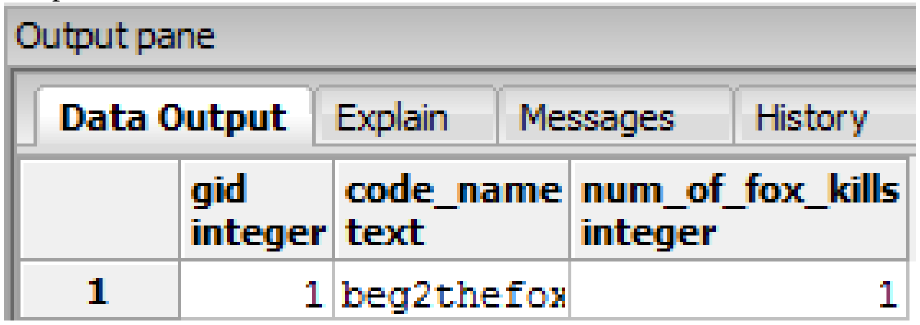
	count bigint
1	3

Zombie Predators

This view is to list all people who have made a kill as a zombie, and it also shows how many kills they have both as a zombie and as a fox.

```
CREATE VIEW predator as
select distinct g.gid, p.code_name, p.num_of_fox_kills
from people p, hunting_foxes hf, game g, zombie z, signed_up s
where p.pid = s.pid and
    g.gid = s.gid and
    s.pid = hf.predator and
    s.gid = hf.gid
order by p.num_of_fox_kills DESC;
```

Output:



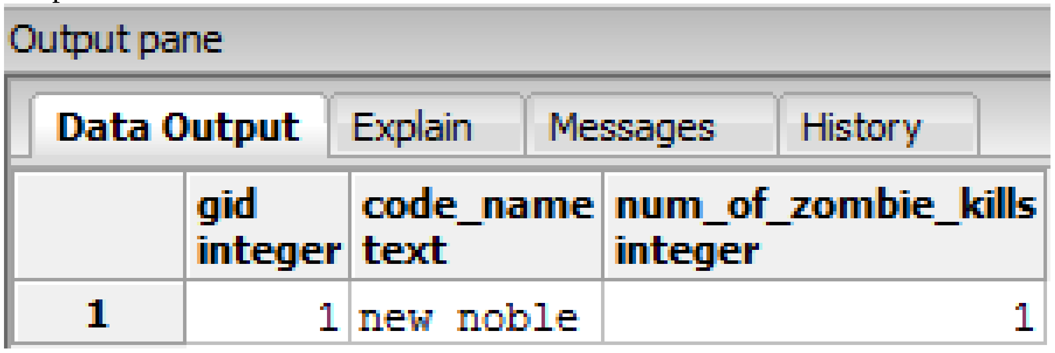
	gid integer	code_name text	num_of_fox_kills integer
1	1	beg2thefox	1

Fox Predators

This view is to show case all the people who have fallen victim to a predator as well as the players amount of deaths.

```
CREATE VIEW zombie_killer as
select distinct g.gid, p.code_name, p.num_of_zombie_kills
from people p, hunting_zombies hz, game g, zombie z, signed_up s
where p.pid = s.pid and
      g.gid = s.gid and
      s.pid = hz.predator and
      s.gid = hz.gid
order by p.num_of_zombie_kills DESC;
```

Output:



	gid integer	code_name text	num_of_zombie_kills integer
1	1	new noble	1

Functions

Career

List the complete career as in all their kills and how many times they died.

```
CREATE OR REPLACE FUNCTION show_career(pid integer, OUT code_name text,
                                       OUT zombie_kills integer,
                                       OUT fox_kills integer,
                                       OUT deaths integer) returns setof record as $$
BEGIN
return QUERY select code_name, num_of_zombie_kills, num_of_fox_kills, num_of_deaths
        from people
        where pid = $1;
END;
$$ LANGUAGE plpgsql;
```

Update Death

This function is used to update the players death once they are killed by a zombie or a fox.

```
CREATE OR REPLACE FUNCTION update_deaths(pid integer)returns void AS $$

BEGIN
update people p
set num_of_deaths = num_of_deaths + 1
where p.pid = $1;
END;
$$ LANGUAGE plpgsql;
```

Update Fox Kill

Just like the update death function this one is used to update a players fox kill after killing a fox.

```
CREATE OR REPLACE FUNCTION update_fox_kill(pid integer)returns void AS $$

BEGIN
update people p
set num_of_fox_kills = num_of_fox_kills + 1
where p.pid = $1;
END;
$$ LANGUAGE plpgsql;
```

Update Zombie Kill

Again another updating function to but this time it is used to update the zombie kill.

```
CREATE OR REPLACE FUNCTION update_zombie_kill(pid integer)returns void AS $$

BEGIN
update people p
set num_of_zombie_kills = num_of_zombie_kills + 1
where p.pid = $1;
END;
$$ LANGUAGE plpgsql;
```

Signed up to Battle

Time for a different function at last, this one is used to give a list of every user who is signed up for the give game.

```
CREATE OR REPLACE FUNCTION signed_up_to_battle(gid INTEGER, OUT fname text,
                                               OUT lname text,
                                               OUT code_name text,
                                               OUT email text) RETURNS SETOF RECORD AS
$$

BEGIN
RETURN QUERY select p.fname, p.lname, p.code_name, p.email
from people p, signed_up su, game g
where p.pid = su.pid and su.gid = g.gid and g.gid = $1
order by p.pid;
END;
$$ LANGUAGE plpgsql;
```

Zombies to Battle

Just like the last function this one does the same except it is used to show just the zombies that are signed up for that game.

```
CREATE OR REPLACE FUNCTION zombies_to_battle(gid INTEGER, OUT fname text,
                                             OUT lname text,
                                             OUT code_name text,
                                             OUT email text) RETURNS SETOF RECORD AS
$$

BEGIN
RETURN QUERY select p.fname, p.lname, p.code_name, p.email
from people p, signed_up su, zombie z, game g
where p.pid = su.pid and z.pid = su.pid and z.gid= su.gid and su.gid = g.gid and g.gid = $1
order by p.pid;
END;
$$ LANGUAGE plpgsql;
```

Triggers

Admin Login

This trigger is used to make sure the admin on the app credentials check out, I am not too sure if I could have just done this in a regular function or not. This trigger would go off more in front end when the user tried to login.

```
CREATE FUNCTION admin_login_check() returns trigger as $admin_login_check$
BEGIN
        if
                exists(select p.code_name, ps.pass
                       from people p, passwords ps, player_admin pa
                       where p.pid = ps.pid and
                            p.pid = pa.pid and
                            p.code_name = NEW.code_name and
                            ps.pass = NEW.pass)
        then
        return NEW;
        END if;
        return exception 'Sorry you are not an admin please never try again';
END;
$admin_login_check$ LANGUAGE plpgsql;
```

Player Login

This function is used to just login a regular player to the app, it would go off when a player tried to logon.

```
CREATE FUNCTION login_check() returns trigger as $login_check$
BEGIN
        if
                exists(select p.code_name, ps.pass
                       from people p, passwords ps
                       where p.pid = ps.pid and
                            p.code_name = NEW.code_name and
                            ps.pass = NEW.pass)
        then
        return NEW;
        END if;
        return exception 'Sorry, you are not in the database. Please sign up for our amazing app';
END;
$login_check$ LANGUAGE plpgsql;
```

People Check

This function is used to make sure that a user doesn’t register another version of them self’s onto the database, this would trigger upon registration.

```
CREATE FUNCTION check_people() returns trigger as $check_people$
BEGIN
	if
		exists(select *
		      from people
		      where fname = NEW.fname and
		            lname = NEW.lname and
		            grade_year = NEW.grade_year and
		            code_name = NEW.code_name and
		            email = NEW.email and
		            phone_num = NEW.phone_num)
		then
			raise exception 'Sorry, but you already exists and we do not want another
one of you';
	END if;
	return NEW;
END;
$check_people$ LANGUAGE plpgsql;
CREATE trigger check_people before insert on people
	for each row execute procedure check_people();
```

Security

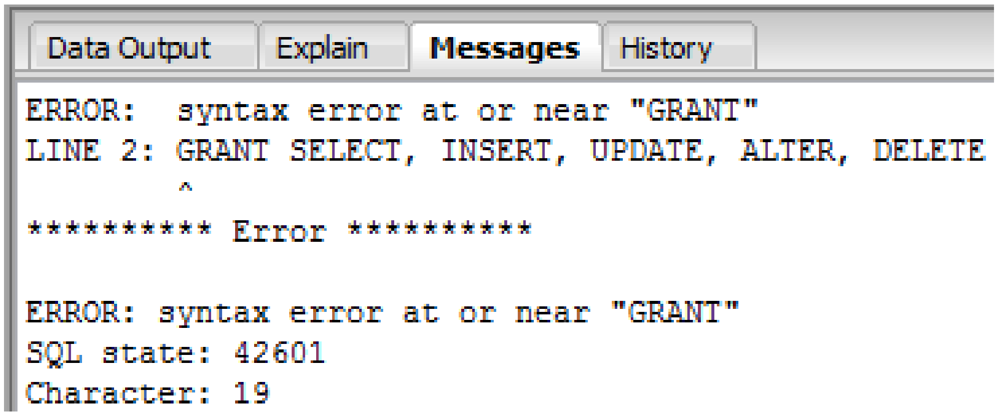
I tried to implement some added security features into my database but it just didn't like me. The features I wanted to add was an admin for the database to make changes if needed and to preform basic maintenance. I also wanted to add the app itself as a user but just give it only a few properties like insert, update, and select.

```
CREATE ROLE admin
GRANT SELECT, INSERT, UPDATE, ALTER, DELETE
ON ALL TABLES IN SCHEMA PUBLIC
TO admin

CREATE ROLE app
GRANT SELECT, INSERT, UPDATE
ON ALL TABLES IN SCHEMA PUBLIC
TO app
```

Whenever I attempted to implement these two I would always get an error.

Error:



```
ERROR:  syntax error at or near "GRANT"
LINE 2: GRANT SELECT, INSERT, UPDATE, ALTER, DELETE
        ^
********** Error **********

ERROR: syntax error at or near "GRANT"
SQL state: 42601
Character: 19
```

Problems

During the implementation of the database I ran into numerous problems. One being what I just described in security. My first major problem is that I built an entire database filled with data to just relies I couldn’t really do any form of triggers on it or functions. So I decided to go with a database I made for my hackathons group app. Just to realize I shouldn’t code when next to no sleep and running off of red bull. I found a lot of poor designed that I needed to fix from my caffeine raged weekend. Although these problems were annoying they weren’t as bad as the fact that I cannot spell one bit, though out my project you will probably notice many of my variables misspelt which caused many errors when I realized I spelt predator as predator. Lastly I had a few problems figuring out exactly how stored procedures and triggers work but I am confident that I have figured them out.

Plans for the Future

My plans for this data base are very simple. I plan to keep changing it as the production of the app continues to fit the need of the users. One plan I thought of while doing this project was to transform my enums into their own tables, so I would have a table for grade year and another one for locations.

I just wanted to add that if you want to use my project to show future students you are more than welcome to. Even if this is the worst project you have ever seen in your life. That way future students hopefully learn from my mistakes.