Database for DOTA2

Abstract

Our project is creating a web application that allows exploration of game statistics from the popular multiplayer online battle arena game DOTA2. Using our application you will be able to explore various statistics as well as the raw game data based on a large collection of match data. We believe that this tool can be used by DOTA2 players to learn more about the current meta-game and make more data driven decision when it comes to improving their game play.

Keywords: Database, Web application, DOTA2.

Frank Egan
Chengwei Ye
Ruonan Feng
Xinting Yao

Overview

Our project is creating a web application that allows exploration of game statistics from the popular multiplayer online battle arena game DOTA 2. The way the game works as far as this project is concerned can be summed up as: Users have a choice of characters called Heroes which they can use to compete in Matches. Each match is played by two teams of five players. Within these matches these players can buy and use various kinds of item in order to defeat the other team. Using our application you will be able to explore various statistics as well as the raw game data based on a large collection of match data. Some examples include looking at the stats for individual Users, see the outcomes of specific matches, explore which Users have the highest win rate, and what kind of Heroes are most popular. The full list of queries is listed below. We believe that this tool can be used by DOTA 2 players to learn more about the current meta-game and make more data driven decision when it comes to improving their game play.

We obtained our data using an API called OpenDota. It supports various kinds of query such as for example retrieving all matches of a given user, or all players in a specific match. We wrote a python script to scrape the API, and retrieve the data set in a given type, list or dictionary. We then cleaned and converted the data into comma-separated values (CSV) files. This CSV data can be imported directly into our DBMS.

We believe that this application qualifies as a database project because it requires us to download in one form (CSV), transform it into another more suitable for a relational database, preserve the relationship between the entities we identify. We must also define a series of support queries and define any SQL views, and triggers we find necessary.

We support the following queries based on what believe will best enable exploration of the dataset and allow users to make interesting discoveries in regards to the dataset we've curated. Additionally, attempting to implement these queries will necessitate applying many of the topics we have discussed in class.

We develop a web application to show all the views and functions. The database and web applications are based on Heroku. Back end is developed with Node.js.

Background Material

Architecture: Heroku

Our application is deployed on a Heroku server. By choosing to host our backend on Heroku we are able to deploy updates to our project quickly and securely. Additionally, Heroku provides a web console for configuring our application and adding any of the built in database management systems. The Heroku Backend as a Service (BaaS) also allows us to a great number of choices in what programming language we implement the application that accesses our database such as Node.js, Python, or Java. Because of our team's background in programming we decided to develop the application using Node.js. The application interface is written in HTML with Javascript.

DBMS Tools: PostgreSQL

As for our choice of DBMS we have decided to use the open source PostgreSQL. The main reason for our decision was because of its ease of configuration on the Heroku platform. Additionally, we had multiple team members with at least some experience using this DBMS. Finally, based on our research it supports all of the features we have used on the Oracle DBMS from class. Our resource for provisioning the database was provided by Heroku.

Web Tools: Node.js

The backend code for our project is written in Node.js and deployed to the Heroku Backend as a Service (BaaS). The server code listens for HTTP requests and executes SQL queries on our PostgreSQL database which is also hosted on Heroku. After the queries resolves the server returns html rendered with the data from the query. This html then gets sent back to the browser to be displayed.

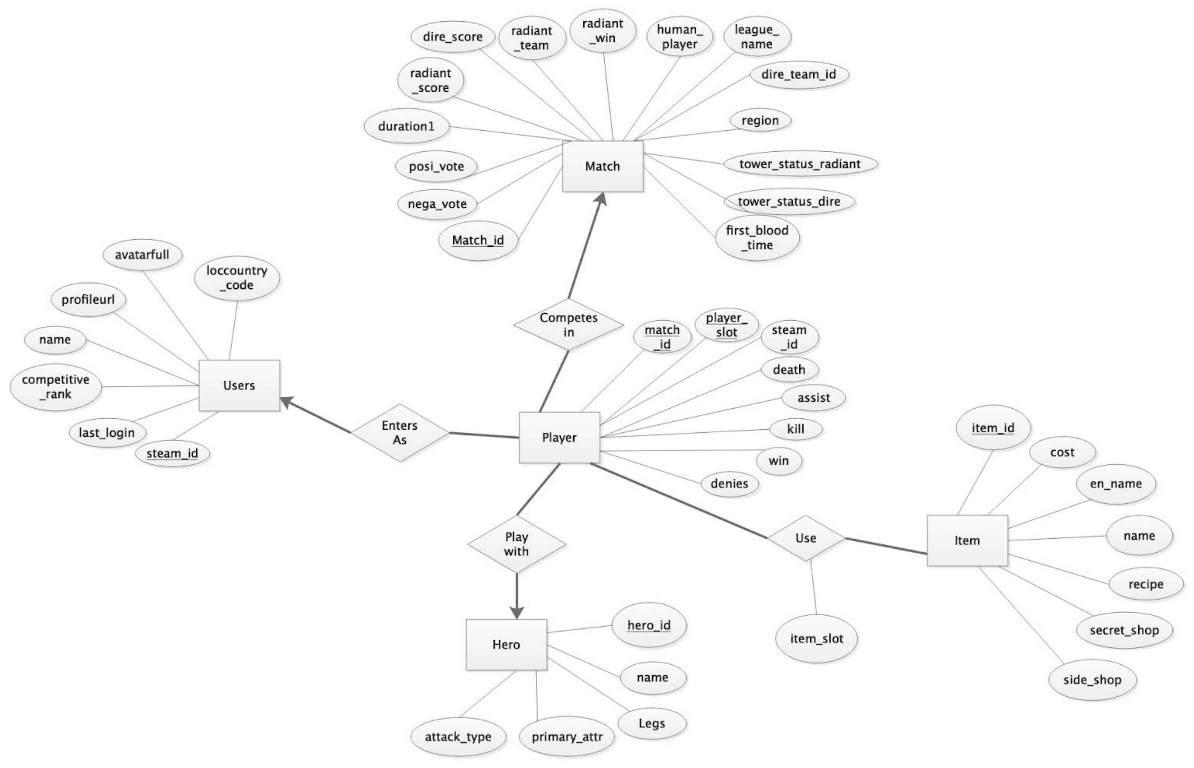
Implementation Process

General Approach

We developed several strategies for organizing the group work. During the design phase we worked together to discuss the various ways we could structure our data, the kinds of queries we could support, and the kind of triggers or constraints we would need to maintain the integrity of our data. During the implementation phase we divided our efforts. You can see in greater detail how we divided the work in the Member Contribution section. Essentially, we had one member doing data collection and cleaning, two members dedicated to writing queries and one final member handling the server side code.

ER Diagram

We have finalized the design of our ER Diagram as followed:



Implementation

Collect Data

We wrote Python script to get data from OpenDota API, then we did data processing to make it more clear and organised.

Design & Map ERD

The ERD we designed is shown above. After designing the ERD, we mapped it to seven relations using mapping strategies learned from class. For example, how can we map multi-value attribute and how can we map many to many relationship. The relations can be found in the Appendix.

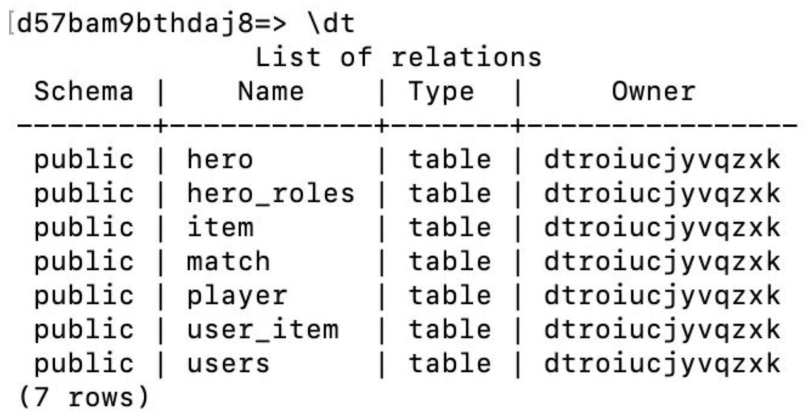
Design Constraints & Triggers

Before creating tables, we considered all Primary Key, Foreign Key, NOT NULL, Unique, Default constraints and triggers we may need, in order to guarantee the performance of our database. Some trigger samples are shown below and the SQL to create trigger can be found in the Appendix.

- Trigger 1 : This trigger prevents more than one user from choosing the same hero during a match.
- Trigger 2 : In table 'Player', kda is a derived attribute, so we write a trigger to compute kda.
- Trigger 3 : Some users don't reveal their information, thus the steam_id would be NULL. We write a trigger to convert all NULL steam_id to 0, representing anonymous users.

Create Tables & Triggers

After designing all Primary Key, Foreign Key, NOT NULL, Unique, Default constraints and needed triggers, we wrote DDL to create tables and the result is shown below. Also, the DDL can be found in the Appendix.



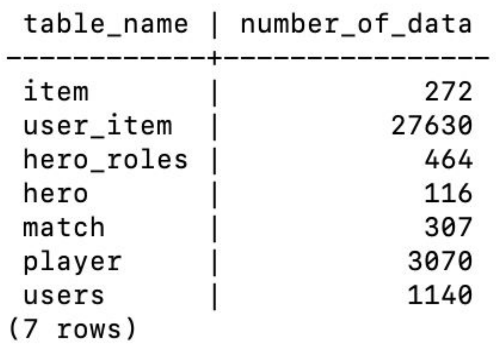
```
[d57bam9bthdaj8=> \dt
                 List of relations
 Schema |    Name     | Type  |     Owner
--------+-------------+-------+----------------
 public | hero        | table | dtroiucjyvqzxk
 public | hero_roles  | table | dtroiucjyvqzxk
 public | item        | table | dtroiucjyvqzxk
 public | match       | table | dtroiucjyvqzxk
 public | player      | table | dtroiucjyvqzxk
 public | user_item   | table | dtroiucjyvqzxk
 public | users       | table | dtroiucjyvqzxk
(7 rows)
```

Load Data

To import the csv data into our remote database we used the PostgreSQL COPY command. The command takes a file path as input and inserts data into the specified table. However, due to security constraints enforced by Heroku we cannot read a file from our local machines directly into a remote connection. Nor can we upload our own files onto Heroku. The solution was to use the unix pipe and cat commands to pipe data from stdout into the COPY command which can then be read our remote database. An example of the command is provided below:

```
cat /Users/frankegan/Developer/cs542-dota2/data/players.csv |
\psql `heroku config:get DATABASE_URL --app cs542-dota2` -c
"COPY tmp_table FROM STDIN DELIMITER ',' CSV HEADER;"
```

After loading data, we checked the number of data in our database and the result is provided below:



```
 table_name | number_of_data
------------+----------------
 item       |            272
 user_item  |          27630
 hero_roles |            464
 hero       |            116
 match      |            307
 player     |           3070
 users      |           1140
(7 rows)
```

Core Functions

1. For Each match:
   1.1> Find the winner of a specific match;
   1.2> Find 10 players and their individual performance (kill, death, assist, damage_to_hero, damage_to_tower, last_hits, denies, win/loss, ect.);
   1.3> Find team information of a specific match (dire_gold, radiant_gold, dire_score, radiant_score, etc.);
   1.4> Find information of a specific match (first_blood_time, duration, positive_votes, negative_votes,etc.).

2. For Each User:
   2.1> Find the favorite hero of a user by giving his steam_id;
   2.2> Find win rate of a user by giving his steam_id;
   2.3> Find the item used most frequently by a user;
   2.4> Find all matches of a user;
   2.5> Find teammates of a user by giving a specific match or a period of time;
   2.6> Find similar users for a user and recommend them to be friends. (Having the same favourite hero and item, having similar performance, etc.);
   2.7> Find all hero statistics of a user; (the average/best/worst performance of a user while picking a specific hero);

3. For Each Hero:
   3.1> Find fundamental information of a hero such as image, name, roles, legs;
   3.2> Find pick rate and win rate of a specific hero;
   3.3> Find heros playing most frequently in the same match and same team;
   3.4> Find average/best/worst performance (kill, death, assist, kda, last_hits, etc.) of a specific hero;

4. For Each Item:
   4.1> Find fundamental information of a item such as image, name, cost;
   4.2> Find using frequency of a item (for each hero and in total);
   4.3> Find win rate of a item (for each hero and in total);

Create Views
According to the functions we want, we wrote the Views(The SQL are in the Appendix). We tried to make the queries as efficient as possible. We also modified our tables when we met problems on queries.

Issues

Anonymous players
To protect users' privacy, Valve doesn't reveal data of those who choose to conceal their information. Each match has 10 players. If some of them do not display their data, their steam id would be null and we call them anonymous players. To identify anonymous players, we assign them a common steam id like '000000000'. However, this causes a new problem: steam id become no longer unique in players table and use_item table, so that steam id

cannot be a key. Therefore, we use the slot attribute to take the place of steam id as keys in these tables. Slot is unique for each match, so along with match id, we can identify the player in the match. Moreover, this assigned steam id remains in backend database and will never be displayed in the search result.

Boolean Value

For some attributes in our tables, such as “radiant_win” in the “Match” table and “is_contributor” in the “user” table, theirs values are “TRUE” or “FALSE”, we constraint them in different ways.
As for “is_contributor”, we changed the values to 0/1 to save more space and add constraint to make it to be restricted to 2 numbers.
As for “radiant_win”, it’s a important attribute that will be used very often, “TRUE” means in radiant won in that match while “FALSE” means dire won in that match. When we were writing queries, we found that it should be better if we preprocess it first. So we change “radiant_win” to attribute “win” which is a 0/1 attributes. “1” means win and “0” means lose. We also restricted its values by constraints. The good thing of this change is, when we do queries related to winner, we can easily realize by “WHERE win=1 ” instead of judging if radiant won in this match and finding who are radiant or dire.

Add Role to Hero

We find that one hero can play different roles. So we add one more table to record the hero-role pairs.

Improve Query

When writing queries (views), we found that there are many ways to realize the functions. So we judged which one can be faster. We tried to do more selection before joins to avoid large joins. We thought a lot about how can we improve the query efficiency when our database has a huge amount of data.

Database Endpoint

When writing queries, we found that many queries are strongly determined by what our users’ input. So we tried to deal with the endpoint of our database.

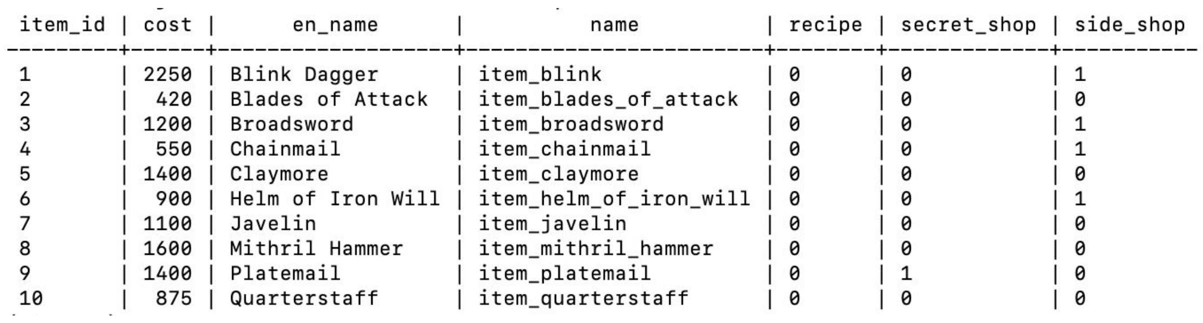
Improving Our Database

We found some problems when writing queries, so we thought about them thoroughly. Sometimes we need more information so we went back and collected more data and added more attributes to our table so that we can achieve the functions we want.

Validation

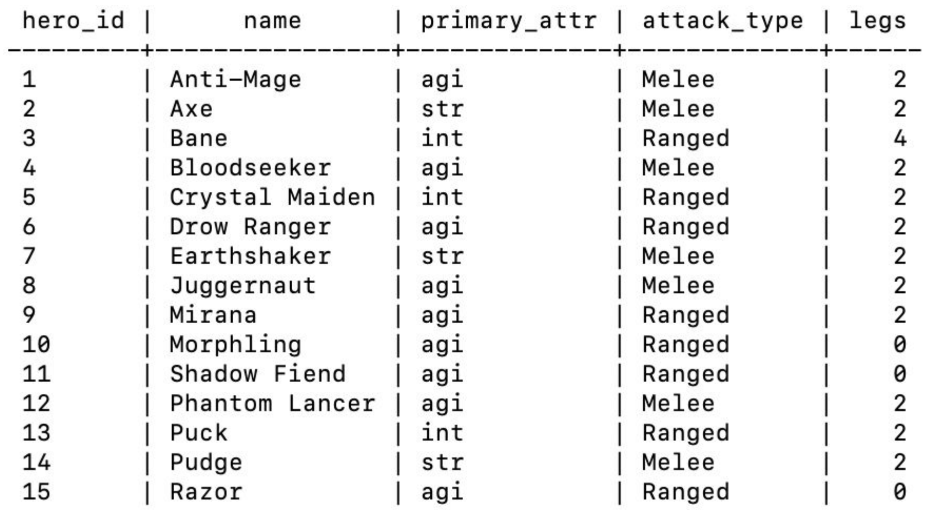
We’ve finished our database part of our system. We created tables, constraints and triggers, loaded data and create views. Our database contains seven tables. We run the views to do queries. It worked well.

Sample 1: Find information for the items.



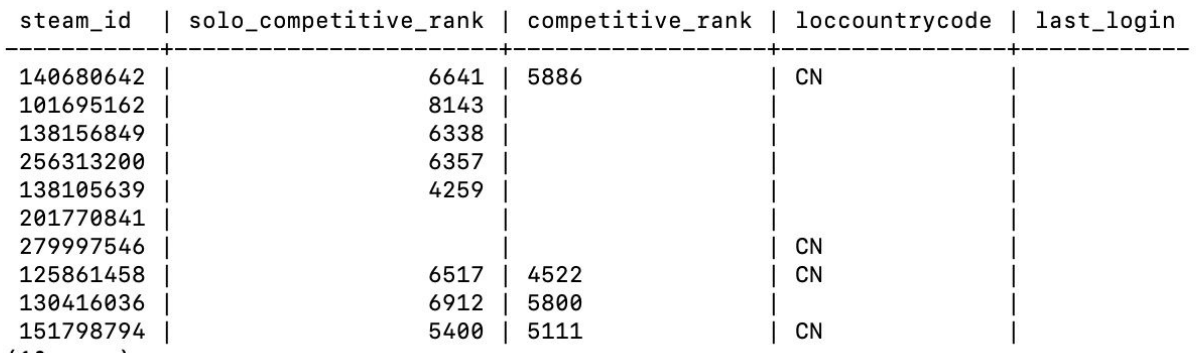
```
 item_id | cost |      en_name       |          name           | recipe | secret_shop | side_shop
---------+------+--------------------+-------------------------+--------+-------------+-----------
 1       | 2250 | Blink Dagger       | item_blink              | 0      | 0           | 1
 2       |  420 | Blades of Attack   | item_blades_of_attack   | 0      | 0           | 0
 3       | 1200 | Broadsword         | item_broadsword         | 0      | 0           | 1
 4       |  550 | Chainmail          | item_chainmail          | 0      | 0           | 1
 5       | 1400 | Claymore           | item_claymore           | 0      | 0           | 0
 6       |  900 | Helm of Iron Will  | item_helm_of_iron_will  | 0      | 0           | 1
 7       | 1100 | Javelin            | item_javelin            | 0      | 0           | 0
 8       | 1600 | Mithril Hammer     | item_mithril_hammer     | 0      | 0           | 0
 9       | 1400 | Platemail          | item_platemail          | 0      | 1           | 0
 10      |  875 | Quarterstaff       | item_quarterstaff       | 0      | 0           | 0
```

Sample 2: Find information for the heros:



```
 hero_id |      name       | primary_attr | attack_type | legs
---------+-----------------+--------------+-------------+------
 1       | Anti-Mage       | agi          | Melee       |    2
 2       | Axe             | str          | Melee       |    2
 3       | Bane            | int          | Ranged      |    4
 4       | Bloodseeker     | agi          | Melee       |    2
 5       | Crystal Maiden  | int          | Ranged      |    2
 6       | Drow Ranger     | agi          | Ranged      |    2
 7       | Earthshaker     | str          | Melee       |    2
 8       | Juggernaut      | agi          | Melee       |    2
 9       | Mirana          | agi          | Ranged      |    2
 10      | Morphling       | agi          | Ranged      |    0
 11      | Shadow Fiend    | agi          | Ranged      |    0
 12      | Phantom Lancer  | agi          | Melee       |    2
 13      | Puck            | int          | Ranged      |    2
 14      | Pudge           | str          | Melee       |    2
 15      | Razor           | agi          | Ranged      |    0
```

Sample 3: Find information for the users:

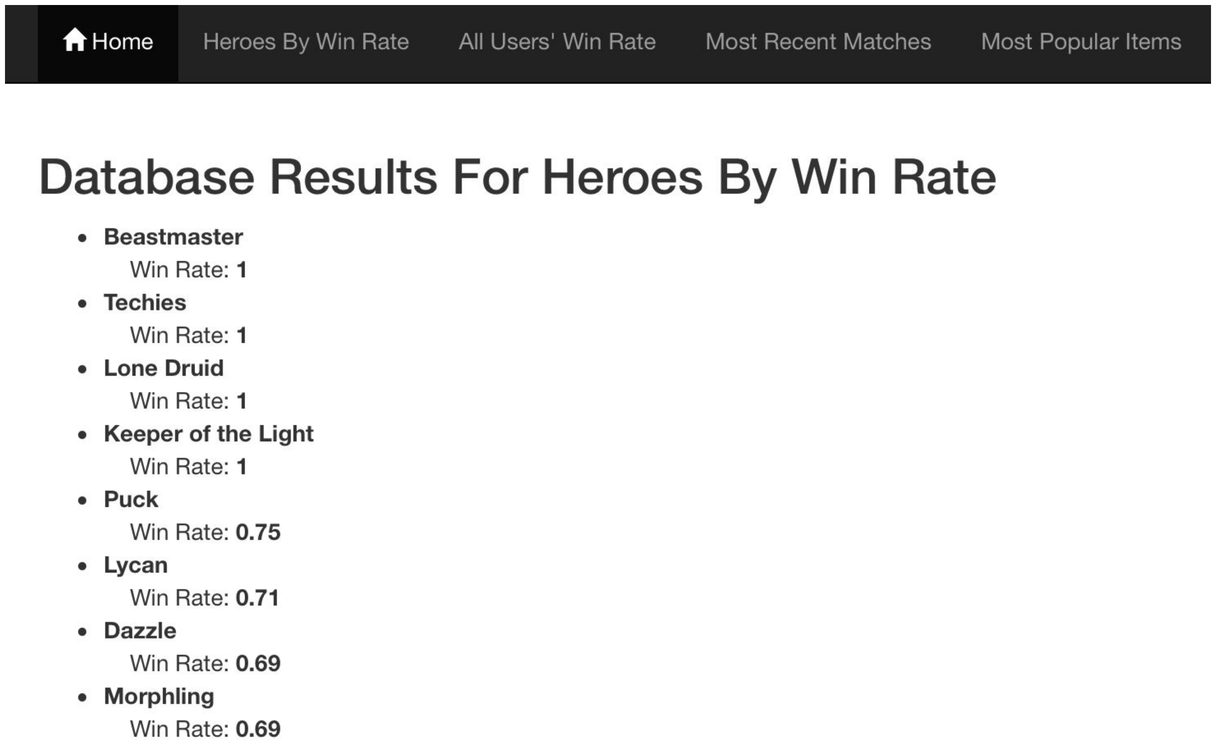


```
 steam_id  | solo_competitive_rank | competitive_rank | loccountrycode | last_login
-----------+-----------------------+------------------+----------------+------------
 140680642 |                  6641 | 5886             | CN             |
 101695162 |                  8143 |                  |                |
 138156849 |                  6338 |                  |                |
 256313200 |                  6357 |                  |                |
 138105639 |                  4259 |                  |                |
 201770841 |                       |                  |                |
 279997546 |                       |                  | CN             |
 125861458 |                  6517 | 4522             | CN             |
 130416036 |                  6912 | 5800             |                |
 151798794 |                  5400 | 5111             | CN             |
```

Next step we'll move on the interface part. We will design the web page and modify it to make it more user-friendly, learn how to make connections from web page to database and modify our system as much as possible.

Web Application

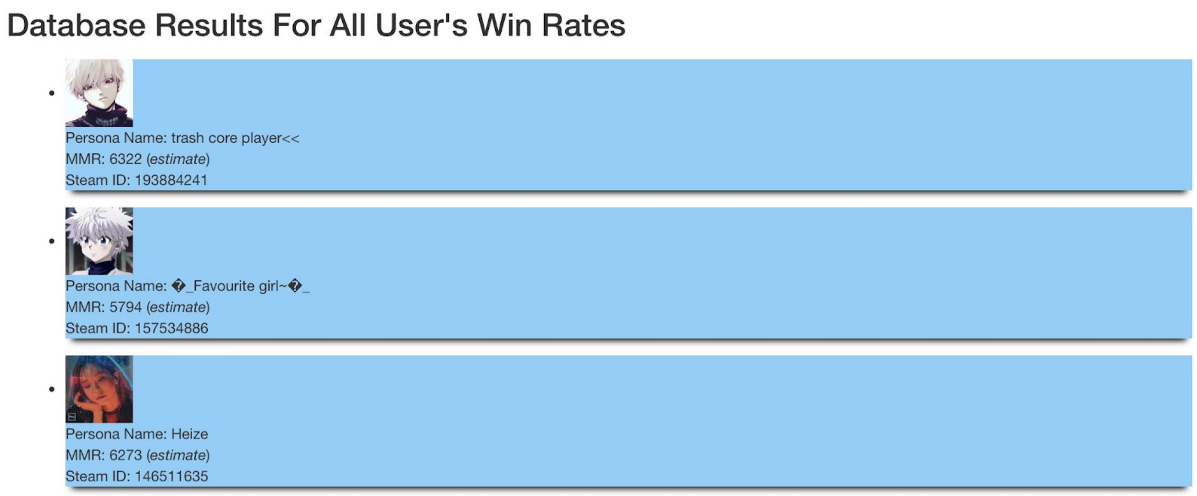
We developed web application with Node.js. It delivers all functions we planned to realize. Heroes ordered by win rate. The web applciation can be found at https://cs542-dota2.herokuapp.com/



Database Results For Heroes By Win Rate

- **Beastmaster**
  Win Rate: **1**
- **Techies**
  Win Rate: **1**
- **Lone Druid**
  Win Rate: **1**
- **Keeper of the Light**
  Win Rate: **1**
- **Puck**
  Win Rate: **0.75**
- **Lycan**
  Win Rate: **0.71**
- **Dazzle**
  Win Rate: **0.69**
- **Morphling**
  Win Rate: **0.69**

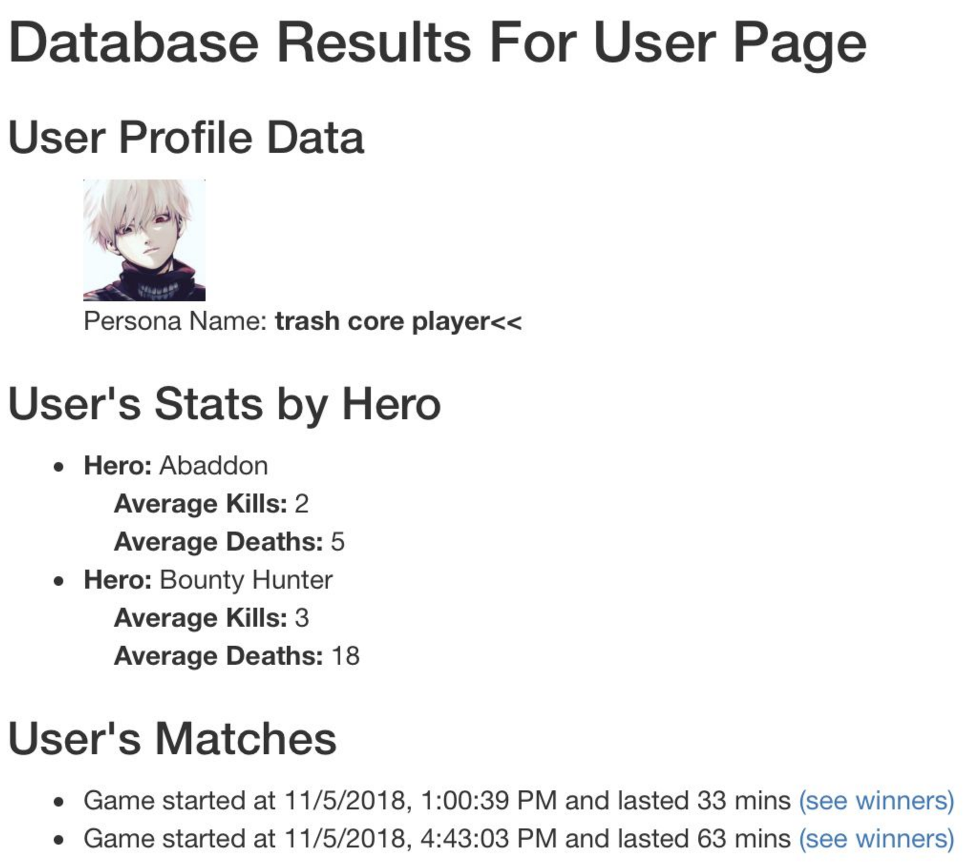
All users ordered by win rate.



Database Results For All User's Win Rates

- Persona Name: trash core player<<
  MMR: 6322 (*estimate*)
  Steam ID: 193884241
- Persona Name: �_Favourite girl~�_
  MMR: 5794 (*estimate*)
  Steam ID: 157534886
- Persona Name: Heize
  MMR: 6273 (*estimate*)
  Steam ID: 146511635

Users' profile



Database Results For User Page

User Profile Data

Persona Name: **trash core player<<**

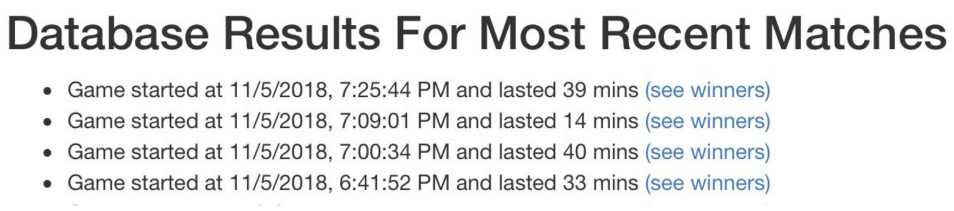
User's Stats by Hero

- **Hero:** Abaddon
  Average Kills: 2
  Average Deaths: 5
- **Hero:** Bounty Hunter
  Average Kills: 3
  Average Deaths: 18

User's Matches

- Game started at 11/5/2018, 1:00:39 PM and lasted 33 mins (see winners)
- Game started at 11/5/2018, 4:43:03 PM and lasted 63 mins (see winners)

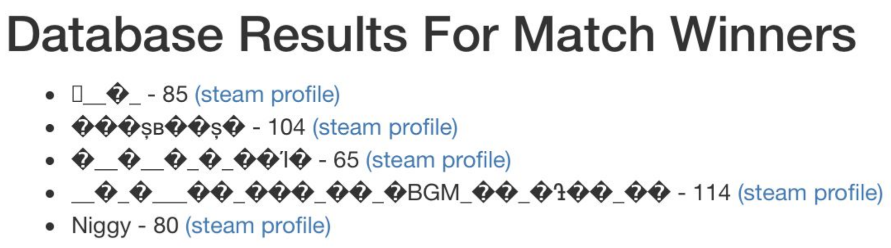
Most recent matches



Database Results For Most Recent Matches

- Game started at 11/5/2018, 7:25:44 PM and lasted 39 mins (see winners)
- Game started at 11/5/2018, 7:09:01 PM and lasted 14 mins (see winners)
- Game started at 11/5/2018, 7:00:34 PM and lasted 40 mins (see winners)
- Game started at 11/5/2018, 6:41:52 PM and lasted 33 mins (see winners)

Winners of a match



Database Results For Match Winners

- ▯__�_ - 85 (steam profile)
- ���ȘB��Ș� - 104 (steam profile)
- �__�__�_�_��Ί� - 65 (steam profile)
- __�_�___��_���_��_�BGM_��_�1��_�� - 114 (steam profile)
- Niggy - 80 (steam profile)

Lessons Learned

1. We learned to install PostgreSQL 10.5 on our laptops. We created accounts so that we can work on one database together.
2. There's some difference between SQL in Oracle and PostgreSQL, so we search more tutorials to learned and wrote.

3. The majority of our time has been spent designing the relations between the data. As we began collecting the data we discovered that some of fields we believed to be candidate keys were in fact nullable. This meant we then had to rely on more complex composite keys for these tables. In the case of the Player table we discovered that the steam_id attribute was null when competitors were competing anonymously. To resolve this issue we combined the player's match_id and player_slot to form a unique key.
4. Additionally, there was a fair amount of data cleaning that had to be done to format the data appropriately for a relational database. For example, replacing true and false boolean values with Zeros and Ones, or adding constraints to enum values such that they are restricted to certain string values.
5. Finally, despite our team members having experience with the Heroku backend service configuring our project to connect to our local development database as well as the remote production database was quite a challenge that required installing the heroku command line interface (CLI), the postgresql CLI, as well as the Node.js CLI. Through this process we discovered that certain steps could be skipped over. For example, you only needed to install the Heroku CLI if you were developing server code, but are not necessary for connecting to the remote backend database.

Conclusions

So far we have done our dota2 database. These tasks include, collecting the game match data, cleaning the collected data, finalizing our relation models, writing constraints, writing triggers, writing SQL views, importing our data into the PostgreSQL database on Heroku, and writing a Node.js program that will respond to HTTP requests with results from some of our queries. We realized the basic functions of our database.

In the future, our database are expected to have more functions for users like searching by steam ID in the input-boxes and clicking from drop-down boxes. It will have more fancy interface in the future.

Schedule

We plan to work individually throughout the week and hold group meeting every Friday.

Time	Tasks	Description	Team members
11.08-11.14	Design the web	Implement all of the api endpoints.	Xinting, Reina
		Use new backend API to populate data in the front end web app.	Frank, Chengwei

11.15-11.21	Build the web	Finish all of the things of our project.	All
11.22-11.28	Report		All
11.29-12.5	Presentation		All

Appendix

Links

- Web Application
- Source Code

Relations

Users

Attribute name	Type	Description
Steam_id (PK)	**varchar(9)**	**Steam_id**
personaname	varchar(50)	personaname
name	varchar(50)	name
leaderboard	varchar(50)	Ranking on leaderboard
solo_competitive_rank	int	solo_competitive_rank
mmr_estimate	int	mmr_estimate
competitive_rank	varchar(50)	competitive_rank
is_contributor	varchar(5)	If the user is a contributor to OpenDota
last_login	varchar(20)	Last login date
profileurl	varchar(255)	Url to profile
loccountrycode	varchar(10)	Country code
avatarfull	varchar(255)	Image of the user

Match

Attribute name	Type	Description
Match_id(PK)	**varchar(20)**	**The ID number of the match assigned by Valve**
all_gg_count	int	The time that gg is called.
barracks_status_dire	int	Bitmask. An integer that represents a binary of which barracks are still standing. 63 would mean all barracks still stand at the end of the game.
barracks_status_radiant	int	Bitmask. An integer that represents a binary of which barracks are still standing. 63 would mean all barracks still stand at the end of the game.
cluster1	int	cluster
dire_logo	varchar(255)	Url of dire team logo
dire_score	int	Final score for Dire (number of kills on Radiant)
dire_team	varchar(20)	Dire team name
duration1	int	Duration of the game in seconds
engine	int	engine
first_blood_time	varchar(20)	Time in seconds at which first blood occurred
game_mode	int	Integer corresponding to game mode played. List of constants can be found here: https://github.com/odota/dotaconstants/blob/master/json/game_mode.json
human_players	int	Number of human players in the game
league_name	varchar(50)	League name
league_tier		League tier
leagueid	varchar(50)	League id
negative_votes	int	Number of negative votes the replay received in the in-game client
positive_votes	int	Number of positive votes the replay received in the in-game client
radiant_gold_adv	int	Advantage of radiant gold
radiant_score	int	Final score for Radiant (number of kills on Radiant)

radiant_logo	varchar(255)	Image of radiant team logo
radiant_team	varchar(20)	Radiant team name
radiant_win	int	Boolean indicating whether Radiant won the match
radiant_xp_adv	int	Advantage of radiant xp
region	int	Integer corresponding to the region the game
start_time	varchar(12)	The Unix timestamp at which the game started
tower_status_dire	int	Bitmask. An integer that represents a binary of which Dire towers are still standing.
tower_status_radiant	int	Bitmask. An integer that represents a binary of which Radiant towers are still standing.
version1	int	Parse version, used internally by OpenDota

Player

Attribute name	Type	Description
steam_id (FK references User(steam_id)) (default '000000000')'	varchar(9)	steam_id
match_id (PK; FK references Match(Match_id))	**varchar(20)**	**Match ID**
player_slot (PK)	**int**	**Which slot the player is in. 0-127 are Radiant, 128-255 are Dire**
assists	int	Number of assists of the player
deaths	int	Number of deaths of the player
denies	int	Number of deaths of the player
game_mode	char(2)	Integer corresponding to game mode played. List of constants can be found on Github.
gpm	int	Gold per minute
hero_damage	int	Damage to opponents

Hero_id (FK references Hero(hero_id))	varchar(3)	Id of the picked hero
kills	int	Number of kills of the player
last_hits	int	Number of last hits of the player
leaver_status	int	Whether the player left the match before it ended
llevel	int	Level of the hero when the match ends
total_gold	int	Total gold of the player when the match ends
tower_damage	int	Damage to enemies’ tower
win	int	Boolean indicating whether the player wins
xpm	int	Xp per minute

Hero

Attribute name	Type	Description
hero_id(PK)	**varchar(3)**	**hero_id**
name	varchar(30)	Hero name
primary_attr	varchar(30)	Hero primary shorthand attribute name
attack_type	varchar(30)	Hero attack type, either 'Melee' or 'Ranged'
legs	int	The number of legs of a hero

Hero_Role

Attribute name	Type	Description
hero_id(PK)	**varchar(3)**	**hero_id**
role(PK)	**varchar(24)**	Hero role in a match

Item

Attribute name	Type	Description
Item_id (PK)	varchar(20)	An unique ID of each item
cost	int	Price of the item

en_name	varchar(50)	English name of the item
name (Unique)	varchar(50)	name
recipe	varchar(5)	Boolean indicating
secret_shop	varchar(5)	Boolean indicating whether the item can be bought in secret shop (True/False)
side_shop	varchar(5)	Boolean indicating whether the item can be bought in side shop (True/False)

User_Item

Attribute name	Type	Description
item	varchar(20)	An unique ID of each item
item_slot(PK)	**varchar(50)**	**Item slot**
match_id(PK)	**varchar(50)**	**Match id**
player_slot(PK)	**int**	**Which slot the player is in. 0-127 are Radiant, 128-255 are Dire**

Create Table DDL

```
DROP TABLE user_item;
DROP TABLE hero_roles;
DROP TABLE Player;
DROP TABLE users;
DROP TABLE Match;
DROP TABLE hero;
DROP TABLE item;

CREATE TABLE item (
 item_id varchar(20) PRIMARY KEY,
 cost int,
 en_name varchar(50),
 name varchar(50) UNIQUE,
 recipe varchar(5),
 secret_shop varchar(5),
 side_shop varchar(5)
);

CREATE TABLE hero (
 hero_id varchar(3) PRIMARY KEY,
 name varchar(30),
 primary_attr varchar(30),
 attack_type varchar(30),
 legs int
);

CREATE TABLE Match(
 all_gg_counts int,
 barracks_status_dire int,
 barracks_status_radiant int,
 cluster1 int,
 dire_logo varchar(255),
 dire_score int,
 dire_team_id varchar(20),
 duration1 int,
 engine int,
 first_blood_time varchar(20),
 game_mode int,
 human_players int,
 league_name varchar(50),
 league_tier varchar(50),
 leagueid varchar(50),
 match_id varchar(20) primary key,
 negative_votes int,
 positive_votes int,
 radiant_gold_adv int,
 radiant_logo varchar(255),
 radiant_score int,
 radiant_team varchar(20),
 radiant_win int,
 radiant_xp_adv int,
 region int,
 replay_url varchar(255),
 start_time varchar(12),
 tower_status_dire int,
 tower_status_radiant int,
 version1 int,
 constraint bool_radiant_win check(radiant_win in(1,0))
);

CREATE TABLE users(
   avatarfull varchar(255),
   competitive_rank varchar(50),
   is_contributor varchar(5) NOT NULL,
   last_login varchar(20),
   leaderboard_rank varchar(50),
   loccountrycode varchar(10),
   mmr_estimate int,
   name varchar(50),
   personaname varchar(50),
   profileurl varchar(255),
   solo_competitive_rank int,
   steam_id varchar(9) primary key,
   constraint contributor check(is_contributor in ('True','False'))
);
```

```
CREATE TABLE Player(
kda int,
assists int NOT NULL,
death int NOT NULL,
denies int NOT NULL,
game_mode char(2),
gpm int NOT NULL,
hero_demage int NOT NULL,
hero_id varchar(3) NOT NULL,
kills int NOT NULL,
last_hits int NOT NULL,
leaver_status int NOT NULL,
llevel int NOT NULL,
match_id varchar(20),
player_slot int,
steam_id varchar(9) default '000000000',
total_gold int NOT NULL,
tower_damage int NOT NULL,
win int,
xpm int NOT NULL,
Constraint player_pk Primary Key (match_id, player_slot),
Constraint fk_steam_id Foreign Key (steam_id) References Users (steam_id),
Constraint fk_match_id Foreign Key (match_id) References Match (match_id),
Constraint fk_hero_id Foreign Key(hero_id) References hero (hero_id),
Constraint player_slot check (player_slot in (1,2,3,4,5,6,7,8,9,10)),
Constraint leaver_status check (leaver_status in (0,1)),
Constraint win check(win in (0, 1))
);

CREATE TABLE hero_roles (
 hero_id varchar(3) REFERENCES hero(hero_id),
 role varchar(24),
 Constraint pk Primary Key (hero_id, role)
);

CREATE TABLE user_item (
 item_id varchar(20) REFERENCES item(item_id),
 item_slot varchar(1),
 match_id varchar(20) REFERENCES match(match_id),
 player_slot varchar(3),
 Constraint user_item_pk Primary key (match_id, player_slot, item_slot)
);
```

Triggers DDL

In table Player, kda is a derived attribute, so we write this trigger to compute kda.

```
CREATE OR REPLACE TRIGGER compute_kda
BEFORE INSERT OR UPDATE ON Player
BEGIN
   if (:new.death = 0) then :new.kda := :new.kills + :new.assists;
   elsif (:new.death <> 0) then :new.kda := (:new.kills + :new.assists)/(:new.death);
   end if;
```

```
END;
```

SQL Queries

View Win Rate for User

This query will allow you to find the percentage of games that a specific User has won.

```
CREATE OR REPLACE VIEW user_view1 AS
Select a.steam_id
         ,(a.win_match)/(b.total_match) as win_rate
From (select steam_id, count(match_id) as win_match
           from player
           where win=1
           group by steam_id) a left join
         (select steam_id, count(match_id) as total_match
          from player
         group by steam_id) b on a.steam_id = b.steam_id
```

View Users By Win Rate

This query will allow you to sort all the Users in our database by their win rate.

```
CREATE OR REPLACE VIEW user_view2 AS
Select a.steam_id
         ,(a.win_match)/(b.total_match) as win_rate
From (select steam_id, count(match_id) as win_match
           from player
           where win=1
           group by steam_id) a left join
         (select steam_id, count(match_id) as total_match
          from player
         group by steam_id) b on a.steam_id = b.steam_id
Order by win_rate;
```

View Hero By Win Rate

This query will allow you to sort the Heroes by their win rate.

```
CREATE OR REPLACE VIEW hero_view1 AS
SELECT t2.name as hero_name, t1.win_rate
FROM (
  SELECT
    a.hero_id, cast((a.win_match) AS decimal(7,2))/cast((b.total_match) AS decimal(7,2)) as
win_rate
  FROM (
    SELECT
      hero_id, count(match_id) as win_match
    FROM
      player
     WHERE win=1
     GROUP BY hero_id) a
  LEFT JOIN (
    SELECT
      hero_id, count(match_id) as total_match
    FROM
      player
    GROUP BY hero_id) b
    ON a.hero_id = b.hero_id) t1
    LEFT JOIN hero t2 on t1.hero_id = t2.hero_id
ORDER BY win_rate
```

View The Most Common Item for a Hero

This query will tell you what Item is most commonly used by a given Hero.

```
CREATE OR REPLACE VIEW  item_hero AS
select hero.name, item.name, it_times.times
from(
select hero_item.hero_id,hero_item.item_id, count(*)as times
from (
       select player.match_id, player.hero_id, use_item.item_id
       from use_item, player
       where player.steam_id=use_item.steam_id and player.match_id=use_item.match_id and
player.hero_id='?????')as hero_item
group by hero_item.item_id)as it_times, hero, item
where hero.hero_id=it_times.hero_id and item.item_id=it_times.item_id
order by it_times.times
```

View Winners of a Match

This query will return which Players won a specific Match.

```
CREATE OR REPLACE VIEW match_view1 AS
Select *
from Player, user
where player.match_id = ‘4193417965’ and Player.radiant_win=1 and
user.steam_id=player.steam_id
```

See Most Recent Matches

This query returns all the Matches sorted by how recently they happened.

```
SELECT *
FROM
  match
ORDER BY
  start_time;
```

Find Hero Stats By Player

```
CREATE OR REPLACE VIEW user_hero_view1
select Player.steam_id
     ,Hero.name as hero
     ,max(kills) as max_kill
     ,avg(kills) as avg_kill
     ,min(kills) as min_kill
     ,max(deaths) as max_death
     ,avg(deaths) as avg_death
     ,min(deaths) as min_death
     ,max(assists) as max_assist
     ,avg(assists) as avg_assist
     ,min(assists) as min_assist
from Player p, hero h
where p.hero_id = h.hero_id
group by P.steam_id, H.name;
```

Find User Stats

```
CREATE OR REPLACE VIEW user_view1
select steam_id
     ,max(kills) as max_kill
     ,avg(kills) as avg_kill
     ,min(kills) as min_kill
     ,max(deaths) as max_death
     ,avg(deaths) as avg_death
     ,min(deaths) as min_death
     ,max(assists) as max_assist
     ,avg(assists) as avg_assist
     ,min(assists) as min_assist
from Player
group by steam_id
```

Heroes By Win Rate

```
CREATE OR REPLACE VIEW hero_view1 AS
SELECT t2.name as hero_name, t1.win_rate
FROM (
  SELECT
    a.hero_id, cast((a.win_match) AS decimal(7,2))/cast((b.total_match) AS decimal(7,2)) as
win_rate
  FROM (
    SELECT
      hero_id, count(match_id) as win_match
    FROM
      player
     WHERE win=1
     GROUP BY hero_id) a
  LEFT JOIN (
    SELECT
      hero_id, count(match_id) as total_match
    FROM
      player
    GROUP BY hero_id) b
    ON a.hero_id = b.hero_id) t1
    LEFT JOIN hero t2 on t1.hero_id = t2.hero_id
ORDER BY win_rate;
```