

# EECS 341: Project Final Report

Project Name: Vapor

Yue Yao, Kaiqi Yao

## Project Background

[Steam®](#) by Valve© is a game platform for vendors to sell game as well as a gamer's community. It's official website claim that *"We have thousands of games from Action to Indie and everything in-between."* and as a game distribution platform they let people *"Enjoy exclusive deals, automatic game updates and other great perks"*. As a game community, players will be able to *"meet new people, join game groups, form clans, chat in-game and more"* in this platform.

Each Steam user has his/her own game "library", which is a collection of games they owned. For each game the system keeps track of the player's playing statistics. These includes playing time, in-game achievement status and more. A player could "friend" other players, this allows them to cooperate with or play against each other in a commonly owned game. A lot of other interactions are also possible (e.g. watch your friend's live broadcast when your friend is in a game).

## Project Feature

Often one would wonder if they are missing out popular games. Am I not playing something very popular among my friends? Perhaps a bunch of your friends all have played one game but you haven't. Probably you don't want to be kept out of the conversation. If most of you and your friends are playing the same game and you would like to know if you are falling behind. Who has the most number of achievements, is a question you might raise.

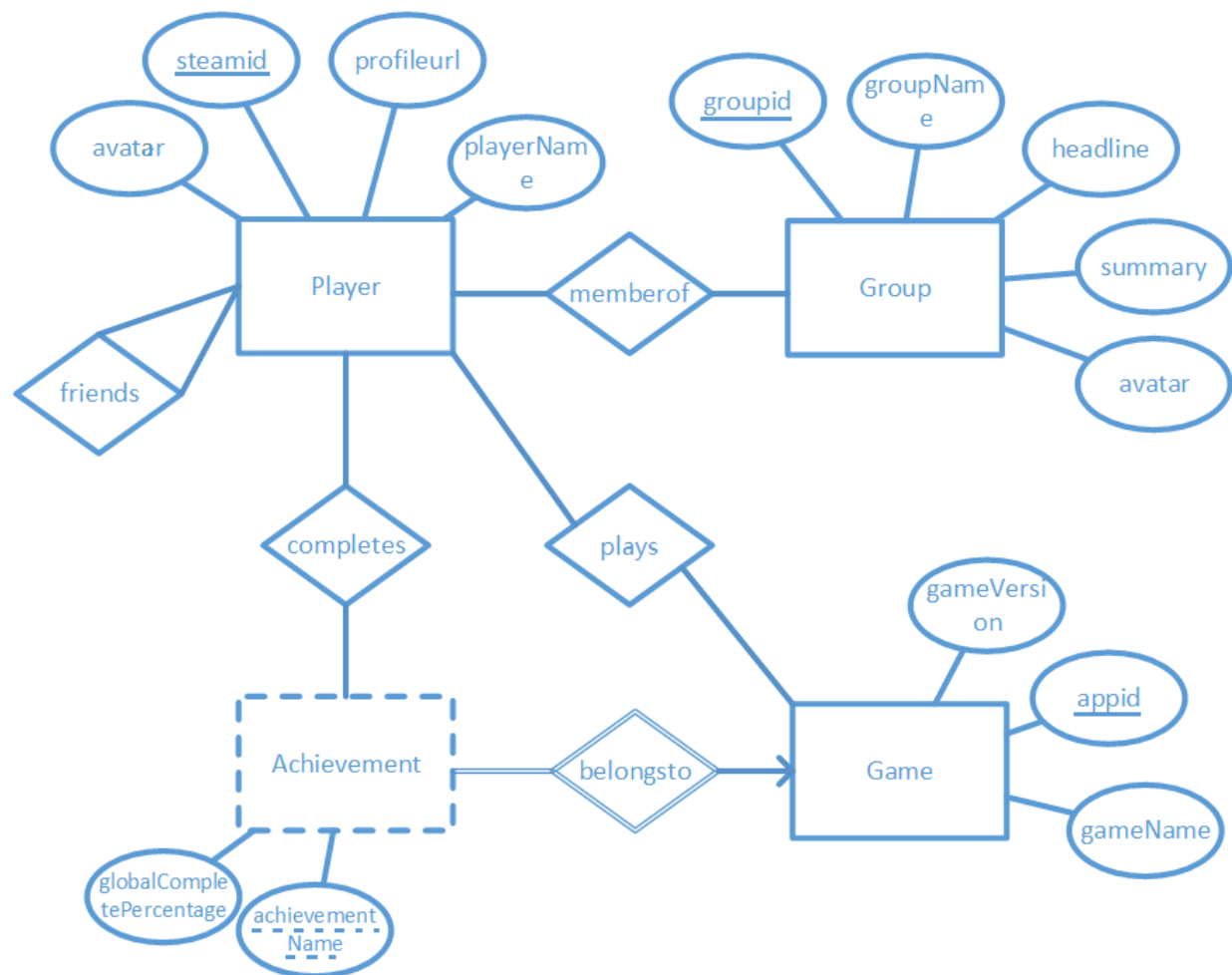
Fortunately Steam provides a set of [Web APIs](#) that allows you to acquire user statistics as well as game info. We could pull related data from the server and use database systems to store and analyze them, we would be able to answer all the questions raised above. Our project aims to help user understand what is popular around them and how well they fit into the picture.

## Data Description

We have 4 entities and 5 relationships in our database according to our needs. The entities are Player, Group, Game, Achievement. They are determined by steamid, groupid, appid, and achievementName, respectively. We consider the friend relationship between players, the relationship that a player is a member of a group, the relationship that a player plays a game

and complete an achievement. These are all many-to-many relationship. The foreign key of these relationships are just the primary keys of the related entities and the primary key of them are the combination of the foreign keys. There is also one many-to-one relationship. The achievement entity and belongsto relationship are special. Achievement is a weak entity. We can only know the achievement after knowing the game, so appid is also in the primary key of Achievement. For the same reason, we can combine Achievement and belongsto in one table and the primary key are appid and achievementName together this time.

## Database ER Diagram



# Database Schema

## Player:

```
CREATE TABLE vapor_player (  
    steamid VARCHAR(20) PRIMARY KEY,  
    playerName VARCHAR(50),  
    profileurl VARCHAR(200),  
    avatar VARCHAR(200)  
);
```

Functional Dependency:

$\text{steamid} \rightarrow (\text{playerName}, \text{profileurl}, \text{avatar})$

$\text{profileurl} \rightarrow (\text{steamid}, \text{playerName}, \text{avatar})$

Since steamid and profileurl are both superkeys of vapor\_player, it's in BCNF.

## Friends:

```
CREATE TABLE vapor_friends (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    player1id_id VARCHAR(20),  
    player2id_id VARCHAR(20),  
    FOREIGN KEY (player1id_id) REFERENCES vapor_player(steamid),  
    FOREIGN KEY (player2id_id) REFERENCES vapor_player(steamid)  
);
```

Functional Dependency:

$\text{id} \rightarrow (\text{player1id\_id}, \text{player2id\_id})$

$(\text{player1id\_id}, \text{player2id\_id}) \rightarrow \text{id}$

Both of the LHS are superkeys of vapor\_friends. It's in BCNF.

## Groups:

```
CREATE TABLE vapor_groups (  
    groupid VARCHAR(20) PRIMARY KEY,  
    groupName VARCHAR(50),  
    headline VARCHAR(200),  
    summary VARCHAR(500),  
    avatar VARCHAR(200)  
);
```

Functional Dependency:

$\text{groupid} \rightarrow (\text{groupName}, \text{headline}, \text{summary}, \text{avatar})$

Since groupid is a superkey of vapor\_group, it's in BCNF.

### MemberOf:

```
CREATE TABLE vapor_memberof (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    steamid_id VARCHAR(20),  
    groupid_id VARCHAR(20),  
    FOREIGN KEY (steamid_id) REFERENCES vapor_player(steamid),  
    FOREIGN KEY (groupid_id) REFERENCES vapor_groups(groupid)  
);
```

Functional Dependency:

$id \rightarrow (steamid\_id, groupid\_id)$

$(steamid\_id, groupid\_id) \rightarrow id$

Both of the LHS are superkeys of vapor\_memberof. It's in BCNF.

### Game:

```
CREATE TABLE vapor_game (  
    appid INTEGER PRIMARY KEY,  
    gameName VARCHAR(50),  
    iconUrl VARCHAR(200),  
    logoUrl VARCHAR(200)  
);
```

Functional Dependency:

$appid \rightarrow (gameName, iconUrl, logoUrl)$

Since appid is a superkey of vapor\_game, it's in BCNF.

### Plays:

```
CREATE TABLE vapor_plays (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    steamid_id VARCHAR(20),  
    appid_id INTEGER,  
    playtime INTEGER,  
    FOREIGN KEY (steamid_id) REFERENCES vapor_player(steamid),  
    FOREIGN KEY (appid_id) REFERENCES vapor_game(appid)  
);
```

Functional Dependency:

$id \rightarrow (steamid\_id, appid\_id, playtime)$

$(steamid\_id, appid\_id) \rightarrow (id, playtime)$

Both of the LHS are superkeys of vapor\_plays. It's in BCNF.

## Achievement:

```
CREATE TABLE vapor_achievement (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  appid_id INTEGER,  
  achievementName VARCHAR(100),  
  globalCompletePercentage REAL,  
  FOREIGN KEY (appid_id) REFERENCES vapor_game(appid)  
);
```

Functional Dependency:

$id \rightarrow (appid\_id, achievementName, globalCompletePercentage)$

$(appid\_id, achievementName) \rightarrow (id, globalCompletePercentage)$

Both of the LHS are superkeys of vapor\_achievement. It's in BCNF.

## Completes:

```
CREATE TABLE vapor_completes (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  steamid VARCHAR(20),  
  achievementid_id INTEGER,  
  FOREIGN KEY (steamid) REFERENCES vapor_player(steamid),  
  FOREIGN KEY (achievementid_id) REFERENCES vapor_achievement(id)  
);
```

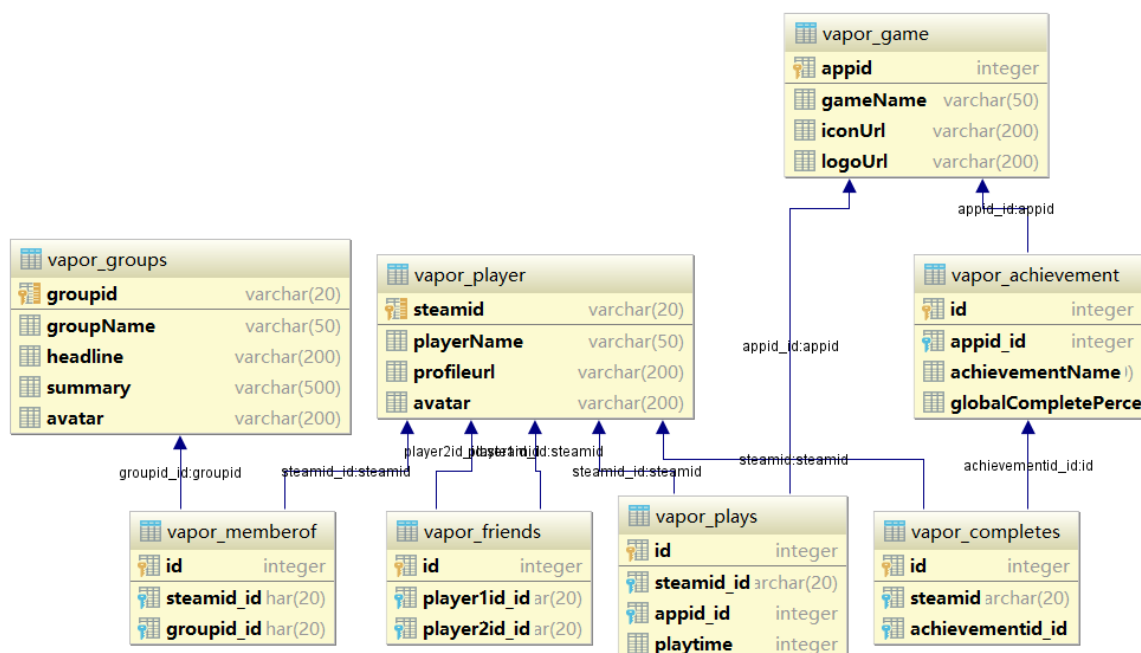
Functional Dependency:

$id \rightarrow (steamid, achievementid\_id)$

$(steamid, achievementid\_id) \rightarrow id$

Both of the LHS are superkeys of vapor\_achievement. It's in BCNF.

## Overall Structure of Tables:



# Sample Queries

## Query1:

Find the appid and gameName of the games played by at least 5 friends of player with steamid “76561198152495215”

### SQL:

```
SELECT P.appid_id, G.gameName
FROM vapor_plays P, vapor_game G
WHERE P.appid_id = G.appid AND
      P.steamid_id in (
        SELECT F.player2id_id
        FROM vapor_friends F
        WHERE F.player1id_id = '76561198152495215'
      )
GROUP BY P.appid_id
HAVING count(P.steamid_id) >= 5;
```

### Relational Algebra:

$A = \Pi_{\text{player2id\_id}} (\sigma_{F.\text{player1id\_id}='76561198152495215'} (\text{vapor\_friend } F))$   
 $B = \Pi_{\text{Plays.appid\_id}} (\text{Plays.appid\_id } G_{\text{Count(Plays.steamid\_id)} \geq 5} (\text{vapor\_plays } \bowtie_{\text{Plays.steamid\_id}=A.\text{player2id\_id}} A))$   
 $C = \Pi_{\text{Game.appid, Game.gameName}} (B \bowtie_{B.\text{appid\_id}=\text{Game.appid}} \text{vapor\_game})$   
C is the final query.

### Tuple Relational Calculus:

Aggregate functions included. No TRC.

## Query2:

Find the groupid, groupName, summary, and avatar of the groups having at least 3 friends of player “76561198152495215”

### SQL:

```
SELECT M0.groupid_id, G.groupName, G.summary, G.avatar
FROM vapor_groups G, vapor_memberof M0
WHERE g.groupid = M0.groupid_id AND
      M0.steamid_id in (
        SELECT F.player2id_id
        FROM vapor_friends F
        WHERE F.player1id_id = '76561198152495215'
      )
GROUP BY M0.groupid_id
HAVING count(M0.steamid_id) >= 3;
```

### Relational Algebra:

$A = \Pi_{\text{player2id\_id}} (\sigma_{\text{F.player1id\_id}='76561198152495215'} (\text{vapor\_friend } F))$

$B = \Pi_{\text{MemberOf.groupid\_id}} (\text{MemberOf.groupid\_id } G \text{Count}(\text{MemberOf.steamid\_id}) \geq 3 (\text{vapor\_memberof}$   
 $\bowtie_{\text{MemberOf.steamid\_id}=\text{A.player2id\_id}} A))$

$C = \Pi_{\text{Groups.groupid, Groups.groupName, Groups.summary, Groups.avatar}} (B \bowtie_{B.groupid\_id=\text{Groups.groupid}} \text{vapor\_groups}))$

C is the final query.

### Tuple Relational Calculus:

Aggregate functions included. No TRC.

### Query3:

List the top 20 achievements completed most by friends of player "76561198152495215"

### SQL:

```
SELECT A.appid_id, A.achievementName, count(C.steamid)
FROM vapor_completes C, vapor_achievement A
WHERE A.id = C.achievementid_id AND
      C.steamid in (
        SELECT F.player2id_id
        FROM vapor_friends F
        WHERE F.player1id_id = '76561198152495215'
      )
GROUP BY C.achievementid_id
ORDER BY count(C.steamid) DESC
LIMIT 20;
```

### Relational Algebra (Without Top 20):

$A = \Pi_{\text{player2id\_id}} (\sigma_{\text{F.player1id\_id}='76561198152495215'} (\text{vapor\_friends } F))$

$B = \text{vapor\_completes.achievementid\_id } G \text{Count}(\text{vapor\_completes.steamid}) (\text{vapor\_completes}$   
 $\bowtie_{\text{Completes.steamid\_id}=\text{A.player2id\_id}} A)$

$C = \Pi_{\text{Achi.appid\_id, Achi.achievementName, B.Count(steamid)}} ((\text{vapor\_achievement } Achi) \bowtie_{\text{Achi.id}=\text{B.achievementid\_id}} B)$

C is the final query.

### Tuple Relational Calculus:

Aggregate functions included. No TRC.

#### Query4:

show the player information of the players who only get the achievement having global percentage  $\geq 90$

#### SQL:

```
SELECT DISTINCT P.steamid, P.playerName, P.avatar
FROM vapor_player P, vapor_completes C
WHERE P.steamid = C.steamid AND
      NOT EXISTS(
        SELECT *
        FROM vapor_achievement A
        WHERE A.id = C.achievementid_id AND
              A.globalCompletePercentage < 90
      );
```

#### Relational Algebra:

$A = \Pi_{Achi.id} (\sigma_{Achi.globalCompletePercentage < 90} (\text{vapor\_achievement } Achi))$

$B = \Pi_{Comp.steamid\_id} (A \bowtie_{A.id = Comp.steamid\_id} (\text{vapor\_completes } Comp))$

$C = \Pi_{steamid, playerName, avatar} ((\Pi_{\text{vapor\_player.steamid}} (\text{vapor\_player}) - B) \bowtie (\text{vapor\_player } P))$

C is the final query.

#### Tuple Relational Calculus:

$\{t^3 \mid (\exists p) (p \in \text{vapor\_player} \wedge t[steamid] = p[steamid] \wedge t[playerName] = p[playerName] \wedge$   
 $t[avatar] = p[avatar] \wedge (\forall c) (c \in \text{vapor\_completes} \wedge t[steamid] = c[steamid] \Rightarrow \neg (\exists a)$   
 $(a \in \text{vapor\_achievement} \wedge a[id] = c[achievement\_id] \wedge (a.globalCompletePercentage \geq 90))))))\}$

#### Query5:

show all the players who play no game in database

#### SQL:

```
SELECT DISTINCT Player.steamid, Player.playerName, Player.avatar
FROM vapor_player Player
WHERE NOT EXISTS(
  SELECT *
  FROM vapor_plays Plays
  WHERE Plays.steamid_id = Player.steamid
);
```

#### Relational Algebra:

$\Pi_{steamid, playerName, avatar} ((\Pi_{P1.steamid} (\text{vapor\_player } P1) - \Pi_{PS.steamid\_id} (\text{vapor\_plays } PS)) \bowtie (\text{vapor\_player } P2))$



### **Tuple Relational Calculus:**

$\{t^3 \mid (\exists p) (p \in \text{vapor\_player} \wedge t[\text{steamid}] = p[\text{steamid}] \wedge t[\text{playerName}] = p[\text{playerName}] \wedge t[\text{avatar}] = p[\text{avatar}] \wedge \neg (\exists ps) (ps \in \text{vapor\_plays} \wedge p[\text{steamid}] = ps[\text{steamid\_id}]))\}$

## **Data**

When you open our website, if you are a new user, you will be asked to register using a steamid. After you click the register icon, our program will automatically search for the data we need from your steamid as a root steamid using steam web api.

The database will immediately start populating itself using your data. However this must happen in an asynchronous way. We face the following 2 difficulties:

1. Amount of API queries to the steam WebAPI is enormous
2. Time for executing these queries are long and tedious, and the application must remain responsive in the process

We solve the problem by introducing a message queue to dispatch API calls to multiple threads, and use a database backend as a synchronization tool. The atomicity of database allows us to do this rather easily. In a larger scale we recommend to implement the message using NoSQLs such as redis.

We provide a sample steam ID to test our service 76561198152495215.

We have an administration site available at “/admin/” we provide an admin account with username “eecs341” and password “eecs341n” (all lower cases) to view our data. Note this account is not associated with any steamid.

Note that in our submitted work we removed API KEY in settings due to security reasons.

The progress of searching are:

1. Your friends, second-level of friends, and their friend relationships.
2. New Groups from all new players and MemberOf relationships.
3. New Games from all new players and Plays relationships. (done parallel with step 2 using multi-thread)
4. The Achievements of all new Games and Complete relationships.

In our tests, we register with steamid 76561198152495215. The data searching will take about 10 minutes. You can also register using other steamids.

The amount of data we get (from steamid 76561198152495215):

601 players, 1475 friend relationships, 48 groups, 68 memberof relationships, 875 games, 1503 plays relationships, 28121 achievements, 4940 completes relationship

We save these data in 'sample.sqlite3'. The data scratched from internet will be stalled in 'db.sqlite3' automatically.

## Implementation

### Tech Stack for this project:

- Front End: Bootstrap 3
- Web framework: Django (website: <https://www.djangoproject.com/>)
- Programming Language: Python
  - ◆ DBMS: SQLite
  - ✓ Defend against SQL Injections
  - ✓ Defend against XSS attacks
  - ✓ ...

### Source code:

- |                               |  |
|-------------------------------|--|
| • vapor/models.py             | Create database  |
| • vapor/create_table.sql      | SQL corresponding to models.py                             |
| • vapor/view.py               | Implementation of all the website                          |
| • vapor/templates/base.html   | The base template of our website, used by other html files |
| • vapor/templates/home.html   | The home page template, used by view.py                    |
| • vapor/templates/login.html  | The login page template, used by view.py                   |
| • vapor/templates/query*.html | The template page of query demo                            |
| • vapor/tasks.py              | The model for multi-thread method                          |
| • vapor/data.py               | The program that automatically extract data for new users  |

### Requirements:

- Django 1.11 installed (website: <https://www.djangoproject.com/>)
- Python 3.5 installed
- Python Libraries:
  - ✧ steam library (pip install steam)
  - ✧ celery library (pip install celery)

## User Guide

1. Open website <http://eecs341-final.wmhi8zvdaa.us-west-1.elasticbeanstalk.com>
2. Type in the username, password, and steamid (e.g. 76561198152495215) to register

3. Wait for the program to collect data (you may refresh the page to see the steps)
4. After the data collecting progress is done, refresh the page and play with the website.

# User Interface

Homepage:

[Project Vapor](#) [Popular Games](#) [Interesting Groups](#) [Achievement](#) [Global Achievement](#) [I don't play](#)

You are 'yao', SteamID '76561198152495215' [Logout](#)

## Project Vapor

Let's feel the *Steam*



## Curious about what's popular?

Steam

- Steam by Valve is a game platform for vendors to sell game as well as a gamer's community.
- Players are able to "meet new people, join game groups, form clans, chat in-game and more" in this platform.

Help Understand what's going on around **YOU**

- Get more information about your friend circle
- The popular games among your friends.
- The interesting groups your friends are in.
- The achievements your friends get.

Developers of this project

This project is developed by Yue YAO [yxy686] and Kaiqi YAO [kxy184]

Query 1:

### Query Prameters

steamID

number of friends

### Query Results

#### Query 1: Popular Games

Find the appid and gameName of the games played by at least 5 friends of player **76561198152495215**

Query String:

```
SELECT P.appid_id, G.gameName
From vapor_plays P, vapor_game G
WHERE P.appid_id = G.appid AND
      P.steamid_id in (
        SELECT F.player2id_id
        FROM vapor_friends F
        WHERE F.player1id_id = ?
      )
GROUP BY P.appid_id
HAVING count(P.steamid_id) >= ?
```

Parameter:

( 76561198152495215, 5, )

Results:

400	Portal
550	Left 4 Dead 2
570	Dota 2
620	Portal 2
730	Counter-Strike: Global Offensive
8930	Sid Meier's Civilization V

Query 2:

Query Prameters

steamID

76561198152495215

number of friends

2

Submit

Query Results

Query 2: Interesting Groups

Find the groupid, groupName, and group summary of the groups having at least 2 friends of player 76561198152495215


Query String:

```
SELECT MO.groupid_id, G.groupName, G.summary, G.avatar
FROM vapor_groups G, vapor_memberof MO
WHERE g.groupid = MO.groupid_id AND
      MO.steamid_id in (
        SELECT F.player2id_id
        FROM vapor_friends F
        WHERE F.player1id_id = ?
      )
GROUP BY MO.groupid_id
HAVING count(MO.steamid_id) >=?
```

Parameter:

( 76561198152495215, 2, )

Results:

103582791430845171	SJTU	我替你们捉鸡啊真的，你们有一个好，玩起游戏来跑的比谁都快，但是玩来玩去都不知道去学习。你们要知道，我们某著名校友，不知道比你们强到哪里去了，一把年纪了都还在期刊上谈笑风生[1][2][3]。所以啊，你们还是需要提高一下自己的姿势水平，洗得不洗的噯。 我今天是作为一个学长，来和你们说一下人生的一些经验。中国有句古话叫“闷声学大习”，就是什么游戏都不玩，这是最好的。但是我看你们这么累啊，偶尔放松一下也是可以的，但是一切要按照	
--------------------	------	---	---

Query 3:

Query Prameters

steamID

76561198152495215

Submit

Query Results

Query 3: Achievements

List the top 20 achievements completed most by friends of player 76561198152495215

Query String:

```
SELECT A.appid_id, A.achievementName, count(C.steamid_id)
FROM vapor_completes C, vapor_achievement A
WHERE A.id = C.achievementid_id AND
      C.steamid_id in (
        SELECT F.player2id_id
        FROM vapor_friends F
        WHERE F.player1id_id = ?
      )
GROUP BY C.achievementid_id
ORDER BY count(C.steamid_id) DESC
LIMIT 20
```

Parameter:

( 76561198152495215, )

Results:

730	GIVE_DAMAGE_LOW	14
730	KILL_ENEMY_RELOADING	14
730	KILL_ENEMY_LOW	14
730	UNSTOPPABLE_FORCE	14
730	WIN_ROUNDS_LOW	14

Query 4:

Query Prameters

Percentage

90

Submit

Query Results

Query 4: High-completion-achievement person

show the player information of the players who only get the achievement having global percentage >= 90

Query String:

```
SELECT DISTINCT P.steamid, P.playerName, P.avatar
FROM vapor_player P, vapor_completes C
WHERE P.steamid = C.steamid_id AND
NOT EXISTS(
  SELECT *
  FROM vapor_achievement A
  WHERE A.id = C.achievementid_id AND
        A.globalCompletePercentage < ?
)
```

Parameter:

( 90, )

Results:

76561198142087512	回去干活！	
76561198167862910	passing_gunner	
76561198180264862	onethree_13	
76561198190069398	Redwood	
76561198194158398	faneater	

Query 5:

Static Query

Query Results

Query 5: I don't play

show all the players who play no game in database.

Query String:

```
SELECT DISTINCT Player.steamid, Player.playerName, Player.avatar
FROM vapor_player Player
WHERE NOT EXISTS(
  SELECT *
  FROM vapor_plays Plays
  WHERE Plays.steamid_id = Player.steamid
)
```

Parameter:

No parameter in this query.

Results:

76561198283652745	Anewer	
76561198273637871	Liamiao	
76561198259696379	西方记者卢西奥	
76561198207377336	丘皮	
76561198255085083	foreat	

login/register page:

## Please Sign In

**Username**

**Password**

Sign in

**SteamID**

Register

Administrative Interface

Administration page:

Django administration

WELCOME, ADMIN / VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

VAPOR

Achievements	+ Add	Change
Completeness	+ Add	Change
Friendss	+ Add	Change
Games	+ Add	Change
Groupss	+ Add	Change
Member ofs	+ Add	Change
Players	+ Add	Change
Playss	+ Add	Change
User profiles	+ Add	Change

Recent actions

My actions

+ admin, SteamID = 0000  
User profile