Final Report

1. **Briefly describe what the project accomplished.**

Our application “SteamIt” is a website that provides game recommendations to users according to different aspects such as age, tag, genres. It records the search history of each user to improve our accuracy of recommendation.

1. **Discuss the usefulness of our project.**

Our application provides an advanced way of recommending games. Popular game search engines would only return the result related to the user’s already bought game, while our application provides a new way of providing a list of games that people who at the same or nearby age have once searched or were interested in. We also provide another game list according to the search history which offers a different view.

1. **Discuss the data in our database.**

The datasets are cleaned. We customize and design some attributes based on users’ needs. The following relational schemas as created based on the datasets. We make assumptions on these relational schemas.

**Users**( Id:int [PK], name VARCHAR(225), age: INT, tags :VARCHAR(225), game: VARCHAR(225))

**machine**( steam\_appid:int[PK], pc\_requirements\_id: INT, mac\_requirements\_id: INT, pc\_requirements :TEXT, pc\_requirements :TEXT );

**pc**( pc\_requirements\_id: INT, steam\_appid:int[PK]);

**mac**( mac\_requirements\_id: INT, steam\_appid:int[PK]);

**Game**( appid: int [PK], name VARCHAR(225), release\_date: date, developer :VARCHAR(225), Platforms: VARCHAR(225), required\_age: INT, genres: VARCHAR(225), tags: VARCHAR(225), Price: FLOAT);

**Ratings**( appid: int [PK], achievements: INTEGER, positive\_ratings: INTEGER, negative\_ratings: INTEGER, average\_playtime: INTEGER, median\_playtime: INTEGER);

**Description**( index\_id :INTEGER, appid: int [PK], detailed\_description: TEXT, about\_the\_game: TEXT, short\_description: TEXT );

**Media**(index\_id: INTEGER, appid: int [PK], header\_image: TEXT, screenshots: TEXT, background: TEXT, movies: TEXT );

**describe**(appid:int[PK], description\_id:int[PK])

**Support\_link**( index\_id: INTEGER, appid:int[PK], website: TEXT, support\_url: TEXT, support\_email: TEXT );

Table Assumptions:

1. Users are uniquely identified by the user\_id. Each user also has information about their name age tags and games.

2. Machines are uniquely identified by the steam\_appid. Each machine also has information about its pc\_requirements and mac\_requirements.

3. Games are uniquely identified by the appid. Each game also has information about its platform, name, genres, released date, tags, required\_age, and price.

4. Descriptions are uniquely identified by the app\_id. Each description also has information about the game.

5. Media are uniquely identified by the app\_id. Each media also has information about the game.

6. Ratings are uniquely identified by the app\_id and also have information about achievements, positive ratings, negative ratings, average\_playtime, and median\_playtime.

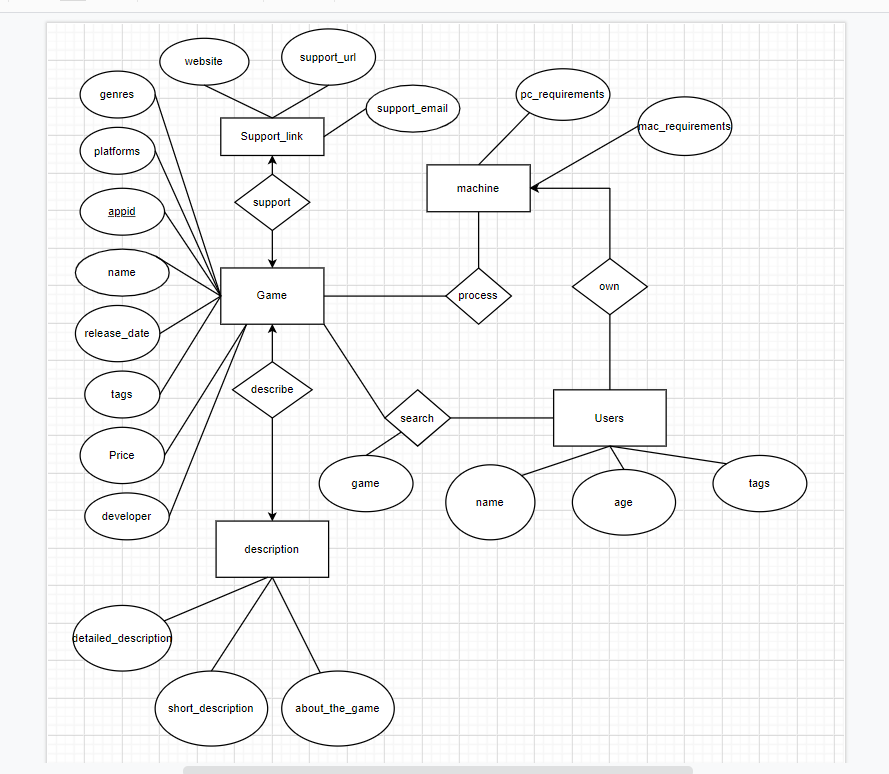
7. Support\_links are uniquely identified by appid. Each support link has information about the game.

Relation Assumptions:

1. One type of machine can support multiple games. Each game can run on multiple machines. (many to many)

2. Each game has exactly one description. Each description can only describe only one game. (one to one)

3. A game can be searched by multiple users by name or tags. Each user can search multiple games. (many to many)

1. **ER-Diagram**
2. **database schema (DDLs) and index design analysis**

CREATE TABLE Support\_link(

index\_id INTEGER,

appid INTEGER NOTNULL PRIMARY KEY,

website TEXT,

support\_url TEXT,

support\_email TEXT

);

CREATE TABLE Media(

index\_id INTEGER,

appid INTEGER NOT NULL PRIMARY KEY,

header\_image TEXT,

screenshots TEXT,

background TEXT,

movies TEXT

);

CREATE TABLE Description(

index\_id INTEGER,

appid INTEGER NOT NULL PRIMARY KEY,

detailed\_description TEXT,

about\_the\_game TEXT,

short\_description TEXT

);

CREATE TABLE Game(

appid INTEGER NOT NULL PRIMARY KEY,

name VARCHAR(225),

release\_date date,

developer VARCHAR(225),

platforms VARCHAR(225),

required\_age INT NOT NULL,

genres VARCHAR(225),

tags VARCHAR(225),

Price FLOAT,

);

CREATE TABLE Ratings(

appid INTEGER NOT NULL PRIMARY KEY,

achievements INTEGER NOT NULL,

positive\_ratings INTEGER NOT NULL,

negative\_ratings INTEGER NOT NULL,

average\_playtime INTEGER NOT NULL,

median\_playtime INTEGER NOT NULL

);

CREATE TABLE mac(

mac\_requirements\_id INT,

steam\_appid INTEGER NOT NULL PRIMARY KEY

);

CREATE TABLE pc(

pc\_requirements\_id INT,

steam\_appid INTEGER NOT NULL PRIMARY KEY

);

CREATE TABLE machine(

steam\_appid INTEGER NOT NULL PRIMARY KEY,

pc\_requirements\_id INT,

mac\_requirements\_id INT,

pc\_requirements TEXT,

pc\_requirements TEXT

);

CREATE TABLE Users(

Id INTEGER NOT NULL PRIMARY KEY,

name VARCHAR(225),

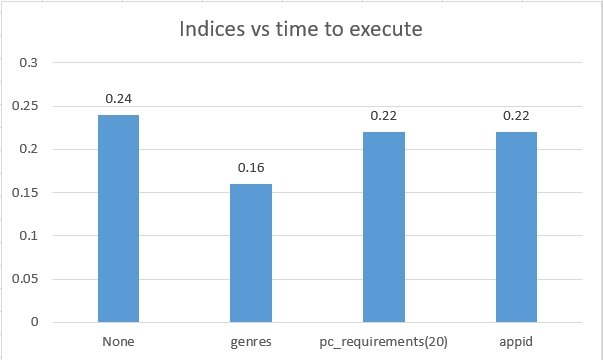
age INT,

tags VARCHAR(225),

game, VARCHAR(225)

);

Index design:



Advanced query: select genres, avg(Price) from Game where appid in (select steam\_appid from the machine where mac\_requirements is not null group by pc\_requirements) group by genres limit 15;

After a different combination of the index, choose the index on genres to optimize

1. **Briefly discuss from where we collected data and how we did it**

We collect the data in Kaggle: <https://www.kaggle.com/nikdavis/steam-store-games>. The data are in csv files. We download the data and pre-process it in Juypter Notebook using pandas DataFrame to manipulate columns and table index. Then we uploaded the processed datasets to GCP Storage bucket. Then we create corresponding empty tables then import the dataset by csv.

1. **Briefly discuss application design and the features involved.**

There are three main parts. The first part is game name search, users can enter the Game name to get the platform requirement, developer, and price, also the frequencies that each game has been searched will be recorded in our GCP. The second part is the user login/game recommend part, user can use their name, their age, and the tag they want to search as one record to insert into our database. We will see the persons have the same name and same age as the same user. So, if you insert two records with the same name and same age, the second insert will be denied and the first insert will be kept, if you want to change your tags, you can only update your previous record. After inserting the user information, our database will recommend several games under the tagged user choice based on our advanced query (depend on the positive ratings and average playtime). At the same time, all the records that insert into our database will be recorded, and it does not matter whether you delete or update your records in some time, all the insert information will be recorded. The Third part is the store procedure part. Our store procedure will return three history tables, which are gamehistory, taghistory, and agehistory, all these history tables are based on the information that users searched or inserted in our APP before.

1. We choose trigger plus store procedures. For the trigger, we use it to prevent the situation that the same user login in twice.

Our trigger is shown below, which set the new insert’s name to ‘dup’ (means duplicate) if it has been in our user tables.

delimiter //

create trigger userTrigger

before insert on Users

for each row

begin

if new.name in (select name from Users where age = new.age)

then set new.name = 'dup';

end if;

end; //

delimiter;

For the store procedure, we want to give the users an overall view about the searched and inserted history, such as which game has been searched a lot.

delimiter //

Create procedure searchHistory()

Begin

Declare done int default 0;

Declare curgame varchar(255);

Declare curtag varchar(255);

Declare curage varchar(255);

Declare gamecur cursor for select distinct game from steamit.Users where game is not null and (name is null or name <> 'dup');

Declare tagcur cursor for select distinct tags from steamit.Users where tags is not null and (name is null or name <> 'dup');

Declare agecur cursor for select distinct age from steamit.Users where age is not null and (name is null or name <> 'dup');

Declare continue handler for not found set done = 1;

Drop table if exists GameHistory;

Drop table if exists TagHistory;

Drop table if exists AgeHistory;

Create table GameHistory (

Game varchar(255),

Num int,

Rating int,

Achievements int);

Create table TagHistory(

Tag varchar(255),

Search\_frequent int,

Game\_Num int

);

Create table AgeHistory(

Age int,

Freq int,

recommendNum int);

Open gamecur;

Repeat

Fetch gamecur into curgame;

Insert into GameHistory(Game, Num, Rating, Achievements)

(select distinct g.GameName, (select count(id) from steamit.Users u where u.game = curgame and (u.name is null or u.name <> 'dup')) as count, r.positive\_ratings, r.achievements from steamit.Game g join steamit.Ratings r on g.appid = r.appid where g.GameName = curgame);

Until done

End Repeat;

Close gamecur;

set done = 0;

Open tagcur;

Repeat

Fetch tagcur into curtag;

Insert into TagHistory(Tag, Search\_frequent, Game\_Num)

(select (select distinct tags from steamit.Users u where u.tags = curtag ) as tag, (select count(id) from steamit.Users u where u.tags = curtag and (u.name is null or u.name <> 'dup')) as count, count(g.appid) from steamit.Game g join steamit.Ratings r on g.appid = r.appid where r.positive\_ratings > 10 and locate(curtag,g.tags)>0);

Until done

End Repeat;

Close tagcur;

set done = 0;

Open agecur;

Repeat

Fetch agecur into curage;

Insert into AgeHistory(Age, Freq, recommendNum)

(select (select distinct age from steamit.Users u where u.age = curage) as age, (select count(id) from steamit.Users u where u.age = curage and (u.name is null or u.name <> 'dup')) as freq, count(g.appid) from steamit.Game g join steamit.Ratings r on g.appid = r.appid where r.positive\_ratings > 10 and g.required\_age<= curage);

Until done

End Repeat;

Close agecur;

End //

delimiter ;

1. **Dataflow**

After a user enter the data in the frontend text box, and click the corresponding button. The function binding on that button will request for a backend query which is used for executing insertion, deletion, search, or update. When the backend query is successfully executed, it sends a json, response, to frontend. Frontend map the json components of the response to each element on the information ‘card’.



The trigger is integrated into the search function. When the user is trying to insert a user twice, which means two users same name and the same age, we assume that these are the same person thus prevent duplicate insertion. We implement the store procedures with a backend query that calls search history and a frontend functional button that calls that query. When clicking the store\_procedure button, the backend will send the corresponding history: SearchHistory, TagHistory, and AgeHistory to the frontend and map to the elements in card div.

1. **Describe one technical challenge that the team encountered.**

We met a technical challenge when deploying our app onto GCP. We follow this [blog post](https://betterprogramming.pub/deploy-a-react-app-to-google-cloud-platform-using-google-app-engine-3f74fbd537ec) step by step but there are some problems that occur. First, we configure a new project in the GCP App Engine, but we did not download the Google Could SDK (you should download it). Only the host whose billing account was charge could download the SDK; others can only start a free trial using the default 300 free credits. It will take a long time to install. Make sure to turn off the VPN, otherwise, it may encounter some fetching problem in installing. When the installation is done, the following steps should execute on the Google Could SDK Shell but not GCP web console. At this point we need to create an app.yaml file in the root folder of the project, we created the app.yaml paralleled to the ‘backend’ and ‘frontend’ folder. Then we need to customize the configuration by your preference. After all, the tutorial said run the command ‘gcloud app deploy’. Unfortunately, it didn’t work and shows ‘ERROR: (gcloud.app.deploy) INVALID\_ARGUMENT: unable to resolve source’. To resolve this, we find that we need to add the path of the app.yaml you just created at the end of the command, that is, ‘gcloud app deploy [path of app.yaml on your computer]’. After a long time of deployment, it will finally give you a link start with https://.