CIS 550 FINAL PROJECT: STEAM-RECOMMENDATION

Ye Dong, Xin Gu, Shuliang Tian, Shu Yang

1

December 10, 2019

ABSTRACT

Recently gaming has become more popular as more people choose to become professional E-sport players and more people are able to pick up electronic devices, from phones/laptops to x-box and Nintendo switches. Yet with the uprising of the gaming industry, there are a lot options. As a result, the customers easily spend a lot of time trying out random games yet still not finding a game that they like. This is how the steam-database came into play. By allowing the users to searching up for specific games and filtering specific genres, we will provide the most helpful reviews and relevant game info like trailers, pictures, and tags to help the user find their next favorite game!

1. Introduction and Goals

The website is aimed to recommend the games and show reviews for the user. Additionally, users can sign up and log in to their account to search the games for details that they want to have. On the home page, we can link to the hottest, the latest and the funnest game detail page. Then on the search page, users can find all games contains the input words, and clicking the game can link to detail page of each game. On the filter page, users can filter by release year, genre, price, and language to get their ideal name, and clicking the game name can link to detail page. Detail page contains details and a piece of review of the game and a short introduction video. Additionally, we have login page for users to log in, or can link to sign up page if user doesn't have an account yet. In this case, the user can set a username and password which will be saved on MongoDB.

2. Data sources and Technologies used

In this project, we used AngularJS as our basic frame work and a variety of tools. We used AngularJS(including Promise), Bootstrap, HTML 5, CSS/BOOTSTRAP, JS, JSON, JQuery, and a few APIs for the UI.(including the API to retrieve image and video from STEAM) We used Express and NodeJS for the server side function and OracleDB(hosted on AWS), MongoDB for the database. We used npm as the building tool and angular controller to control the DOM.

3. Relational Schema

- **Description**(GAME_DESCRIPTION, PUBLISHER, DEVELOPER, TYPES, URL, APPID, NAME)
- **Detail**(DETAIL, NAME)
  FOREIGN KEY NAME REFERENCE (Description) NAME
- **Genre**(GENRE, NAME)
  FOREIGN KEY NAME REFERENCE (Description) NAME
- **Language**(LANGUAGE, NAME)
  FOREIGN KEY NAME REFERENCE (Description) NAME
- **Price**(ORIGINAL_PRICE, NAME)
  FOREIGN KEY NAME REFERENCE (Description) NAME
- **Release_date**(RELEASE_DATE, NAME)
  FOREIGN KEY NAME REFERENCE (Description) NAME
- **Tag**(TAG, NAME)
  FOREIGN KEY NAME REFERENCE (Description) NAME
- **Review_content**(TITLE,REVIEW, REVIEW_ID)
  FOREIGN KEY TITLE REFERENCE (Description) NAME
- **Review_criteria**(Recommendation, HELPFUL, FUNNY, HOUR_PLAYED, TITLE, DATE_POSTED, REVIEW_ID)
  FOREIGN KEY TITLE REFERENCE (Description) NAME
  FOREIGN KEY REVIEW_ID REFERENCE (Review_content) REVIEW_ID

4. Description of our datasets

The original datasets are two large datasets. One contains all the information of the games, and another contains all the information of the reviews.
The game name is the primary key of the game dataset, and there is no other dependency among the rest attributes. So we split and reorganized the game dataset depend on the function of our website. In the original game dataset, one game may have an array containing multiple words as GENRE, TAG, DETAIL and LANGUAGE. So we separate columns of these four attributes and form four new tables in which each attribute is 1 vs. 1 to game name(like unwind in mongoDB). Then we separate columns of RELEASE_DATE and PRICE, since we will filter games based on these two attributes, it seems to be most efficient to separate them into two smaller tables.
For review datasets, it is hard to find primary keys, or we can say that the primary key is consisted of all attributes in the dataset. And there is no dependency in the relation. So we assign each piece of review with a reviewid to be the primary key. And since the column of review content holds up large spaces, we separate it and primary key(reviewid) from the original dataset into a new table. And leave reviewid and all the rest attributes as reviewcriteria table used for review filtering.

We can say the each attribute is only determined by the primary key of each relation. In other words, for each dependency $X \rightarrow Y$ in every table, Y is a superkey of each relation. So all the relations in our database belong to BCNF.(a specific description of the dataset can be found in index D).

5. Description of system architecture

The Steam database is composed of the following:

Index page

This is the welcome page which contains the navigation bar and three buttons linking to the detail page which is real time data. The three buttons are hottest, latest and funniest. These features may cater to people the most. So we put it on the front page. Then when you click on any of the letter on the navbar, it will link directly to each page.

Search page

The search page allows users to search for their beloved games. Game name along with a short description will be displayed, as well as a game cover fetched from another public api. Further, the users can click on the cover displayed to navigate to the detail page, where game price, review(if exists) and a trailer will be shown. Note that search page allows 'vague search' to some extent, but we couldn't implement a NLP system in this project.

Filter page

The navigation page recommend games to the user based on the games' cost, released year, language, and genre using a filter. Both the game title and a review will be displayed for each games that satisfies the criteria and the user can click on the game title to view the details of the game.(i.e. by jump to the details page). The user can choose to get result by filtering through any number of those criteria(i.e. from none of them to all of them). The results will be returned in the order of helpful reviews(since popular games are more likely to have helpful reviews) and only the top 1000 will be displayed. Also note not all games have ratings and reviews so those without ratings and reviews will be displayed at the end.

Detail page

The detail page can be linked from the filter, the front page and the search page. The page contains picture of the game and a trailer that auto player once the user opens the website. Note, both the trailer and auto player were taken from the website real time by the steam API so if there weren't a photo or trailer for the game on Steam, it won't show up on our detail page for corresponding games as well. In addition, this page shows the details of each game such as release data, price, genre, developer, type, tags and description. It includes a link to visit the official website for more detailed website.

Login page

this page is to send a query to the database for the check of the user information. We may develop the authentication and the user profile page in the future.

Detail page

this is a page to show the details of each game. The release data,price, genre, developer, type, tags and the description. It has the link to visit the official website for more detailed website. This is a page which is linked from the filter, the front page and the search page.

6. Performance evaluation

For filtering criteria and output satisfying results on the navigation page, we tried to order the selects, joins, and projections in the most optimal way as possible and used index. Our runtime went from 2000ms all the way down to 200ms after the optimization. Note since our sql query for filter depends on the number of criterias the user select, the runtime we reported here is an average. We used exactly the same optimization techniques for the searching operation in details page and the runtime went from 2000ms to 700ms.

For searching games, if the given game name is 'kind of' accurate, the average recorded timing is around 200ms. For example, we searched 'grand theft auto' series, 'total war' series and 'doom' series and recorded corresponding time shown in console, then took average as final timing.

7. Technical challenges

First we had a lot of trouble building and connecting to the oracleDB and mongoDB. After reading over the documents and searching for tutorials online, we realized that unlike the mysql database, those two databases required us to connect and then release to close the database.

For oracledb, we found out that the node version and the oracledb version should match and then hardcoded the connect and release the database. About mongodb, there are two tricky things. The first is to set all the IP address in the whitelist. And then when do a find function, we need to pass one variable as the total to the database and then it can get the data as the result.

The next major challenge facing us is passing the data via different urls. Since we have several pages jumping to detail page, this is a difficulty we have to work out. Firstly, we tried several ways described online, such as ui-router and service to transfer data. But they not work ideally. After further understanding the mechanism of web application, we successfully use urls to pass the data among different pages at last.

There is one problem when passing the data that it may contain the '%20'. So we use the function toJson and decode the JSON file to make sure that the data will not contain any special characters. Also, when there is a single quote in the game name, the query will import error. So we need to first split the name with single quotes and then join it.

To further complete our web application, we develop logIn and signUp function. But posting and fetching data from MongoDB is new to us. After searching online and trial, we figured out to set the post method instead to transfer data and contain everything in the body. Get MongoDB needs "call back" in the router.

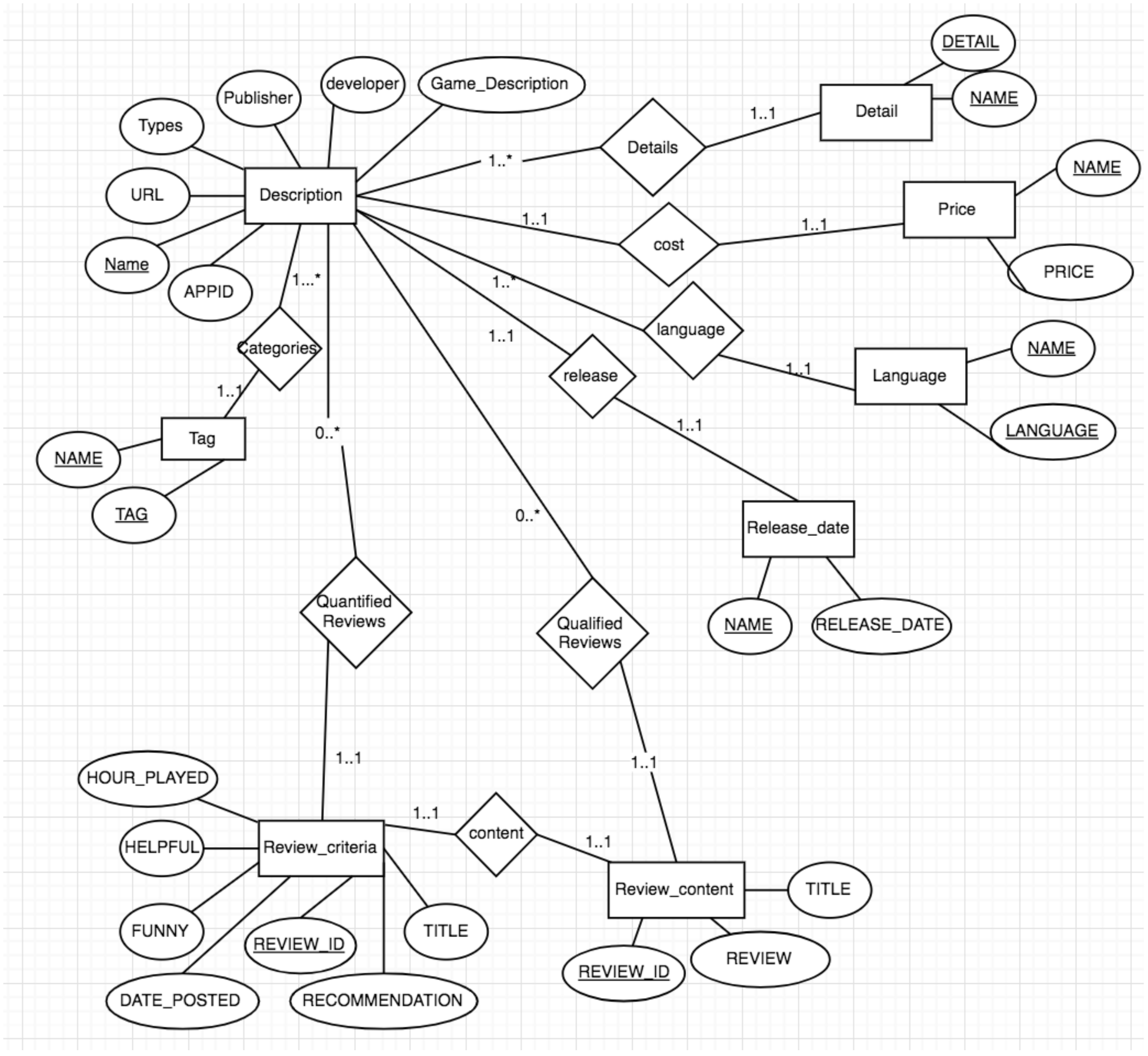
When design search page, we had trouble finding the cover of each returned game. In our original datasets, we don't have url linking to the images, hence we need to query an external api. After we found an api, we couldn't fetch data directly, due to block of CORS policy. We searched a lot and finally solved the problem with proxy. In addition, when fetching cover url for each game and then display the result, we also needed to wait for asynchronous requests. We used promise.all solving this problem.

Last but not least, we had a lot of trouble designing the code for the filter system. The first difficulty we encountered was determine the optimal algorithm so the user can decide which features to select and select as much as they want. And to make the implementation possible without using 10+ cases, we decided to first select all the game titles for games that satisfies our criteria and then rank them by ones that have received the highest reviews. Therefore I could essentially just write code to filter for each features and concatenate whichever ones that was selected. In addition, while we have 27k games overall, only 50 of them have reviews.(I was really confused that we only got so many results despite that we had such a large dataset) However, we want to pick **all** the ones that satisfies the criteria but just rank the ones without reviews at the end. So I picked out all the features using game titles from the genre set(since all games had a genre) and then kept right joining with other review content/review criteria on game names so I always have all the genres that satisfies my criteria(regardless of whether it has reviews or not).

8. Extra Credit features

- Using nosql to store user login information
- Query from the website the current pictures and video for the games.

Appendix A - Entity Relationship diagram



Appendix B - Queries

```sql
SELECT t1.name, t1.genre, t1.recommended_times
FROM
(
SELECT g.genre, g.name, count(*) as recommended_times
FROM genre g
JOIN review_criteria r
ON g.name = r.title
WHERE r.recommendation = 'Recommended'
GROUP BY genre, name
) t1
JOIN
(select genre, max(recommended_times) as maxrec from (
SELECT g.genre, g.name, count(*) as recommended_times
FROM genre g
JOIN review_criteria r
ON g.name = r.title
WHERE r.recommendation = 'Recommended'
GROUP BY genre, name
) group by genre) t2
ON t1.recommended_times = t2.maxrec and t1.genre = t2.genre
ORDER BY t1.genre
```

Listing 1. Hottest: It returns the name, genre and recommended times of the most recommended game of each genre.

```sql
SELECT * FROM
(select d.name, d.url, d.types, d.game_description, d.developer, d.publisher, p.original_price, r.release_date,
nvl(rt.review,'No reviews yet'),nvl(rc.helpful,0),nvl(rc.funny,0),genres,tags,languages,d.appid
FROM description d
JOIN price p ON d.name = p.name AND d.name = '${myGame}'
JOIN release_date r ON r.name = p.name
JOIN (select name , listagg(genre,',') within group (order by name) as genres from ( SELECT distinct name,genre FROM genre) GROUP BY name) g ON d.name = g.name
JOIN (select name , listagg(tag,',') within group (order by name) as tags from (select distinct name,tag from tag) GROUP BY name) t ON d.name = t.name
JOIN (select name , listagg(language,',') within group (order by name) as languages from (select distinct name,language from language) GROUP BY name) l ON d.name = l.name
LEFT JOIN review_criteria rc ON rc.title = d.name
LEFT JOIN review_content rt ON rc.review_id = rt.review_id
ORDER BY rc.helpful,rc.funny,rc.date_posted) WHERE ROWNUM<=5
```

Listing 2. Detail: It returns all the relative information of the certain input game from all tables.

```sql
SELECT * FROM(
  SELECT name AS title, MAX(r2.review) as review, MAX(r3.helpful) as helpful
FROM review_content r2
  RIGHT JOIN (
  SELECT r1.review_id, t1.name, r1.helpful FROM review_criteria r1
  RIGHT JOIN(
  SELECT name, max(helpful) as maxhelp FROM review_criteria
  RIGHT JOIN (SELECT name FROM GENRE  WHERE name IN
(SELECT name FROM price p1 WHERE p1.ORIGINAL_PRICE>500)
AND  name IN
(SELECT name FROM genre g1 WHERE g1.genre='Action')
AND  name IN (SELECT name FROM RELEASE_DATE r1
WHERE EXTRACT(year FROM r1.RELEASE_DATE) = 2018 )  AND  name IN (SELECT name FROM LANGUAGE l1 WHERE LANGUAGE = 'Danish' )   ) g ON title = name
  GROUP BY name) t1 ON r1.title = t1.name and r1.helpful = t1.maxhelp
  ) r3
  ON r2.review_id = r3.review_id
  GROUP BY name) final
  WHERE ROWNUM<10000
  ORDER BY helpful DESC NULLS LAST
```

Listing 3. Select all queries that matches the criteria(ex. specified genre, price range etc.) and output titles and reviews by most helpful reviews in decreasing order.

```sql
SELECT title
  FROM(
  SELECT title FROM
  (SELECT title, max(helpful) AS max
  FROM review_criteria
  GROUP BY title)
  ORDER BY max)
  WHERE rownum <= 1
```

Listing 4. Return name of game whose review is most helpful

```sql
SELECT name
  FROM
  (SELECT name,release_date
  FROM release_date
  WHERE release_date < '12-DEC-19'
  ORDER BY release_date DESC)
  WHERE ROWNUM <= 1
```

Listing 5. Return most recent released game before 12/12/19

```sql
SELECT title
  FROM
  (SELECT title
  FROM review_criteria
  ORDER BY funny DESC, title ASC)
  WHERE ROWNUM <=1
```

Listing 6. Return the first game name(ascending order) with funniest review

```sql
WITH recent_release AS
(SELECT name, release_date
FROM description
WHERE release_date between '1-JAN-19' AND '31-
    JAN-19'
ORDER BY release_date DESC)
SELECT name, genre
FROM
(SELECT genre.name AS name, genre
FROM recent_release JOIN genre ON
recent_release.name = genre.name)
WHERE ROWNUM <= 10
```

Listing 7. Show top 10 newest release within a certain period(say past 7 days) ranked by descending release_date for all or each genre

```sql
SELECT *
FROM
(SELECT B.name, C.best_rates, C.
    max_hours_played
FROM
(SELECT name
FROM language l
WHERE l.language IN
(SELECT language
FROM language
WHERE language.name = 'Beat Saber'))B
JOIN
(SELECT rc.title AS game_name, MAX(rc.helpful)
    AS best_rates, MAX(rc.hour_played) AS
    max_hours_played
FROM review_criteria rc
GROUP BY rc.title) C
ON B.name = C.game_name
ORDER BY C.max_hours_played DESC)D
ORDER BY D.best_rates DESC
```

Listing 8. Show name, best_rates(descending order) and max_hours_played(descending order) of game who shares at least one common language with game "beat saber"

Appendix C - Data Summeries

We used https://cors-anywhere.herokuapp.com/ as the proxy to access our https://store.steampowered.com/ for our web api database. This is where we got the real time image and videos in the detail page. Our data set was taken from Kaggle's Steam games complete dataset(https://www.kaggle.com/trolukovich/steam-games-complete-dataset). The data summaries are as follows:

Description

NUM_ROWS	20186
BLOCKS	4150
AVG_ROW_LEN	1306
SAMPLE_SIZE	20186

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
PUBLISHER	11454	0.0000873057447180024	31
DEVELOPER	13553	0.000073784401977422	16
GAME_DESCRIPTION	20152	0.0000496228662167527	1171
TYPES	1	1	4
URL	20184	0.0000495441934205311	62
NAME	20184	0.0000495392846527296	19
APPID	20086	0.0000497859205416708	5

Detail

NUM_ROWS	76459
BLOCKS	496
AVG_ROW_LEN	38
SAMPLE_SIZE	76459

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
DETAIL	32	0.03125	18
NAME	21502	0.0000465073016463585	20

Genre

NUM_ROWS	117982
BLOCKS	622
AVG_ROW_LEN	28
SAMPLE_SIZE	117982

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
GENRE	24	0.0416666666666667	8
NAME	21502	0.0000495392846527296	20

Language

NUM_ROWS	49755
BLOCKS	244
AVG_ROW_LEN	30
SAMPLE_SIZE	49755

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
LANGUAGE	31	0.0000103309957107584	10
NAME	20163	0.0000495392846527296	20

Price

NUM_ROWS	20186
BLOCKS	95
AVG_ROW_LEN	28
SAMPLE_SIZE	20186

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
PRIGINAL_PRICE	286	0.00154659073216489	4
NAME	20186	0.0000495392846527296	20

Release_date

NUM_ROWS	20148
BLOCKS	103
AVG_ROW_LEN	32
SAMPLE_SIZE	20148

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
RELEASE_DATE	3184	0.000314070351758794	8
NAME	20138	0.0000496573641871089	20

_Tag

NUM_ROWS	178026
BLOCKS	874
AVG_ROW_LEN	30
SAMPLE_SIZE	178026

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
TAG	376	0.00267379679144385	10
NAME	21484	0.0000465462669893875	20

Review_content

NUM_ROWS	357670
BLOCKS	14177
AVG_ROW_LEN	254
SAMPLE_SIZE	357670

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
TITLE	42	0.00000140766231239688	21
REVIEW	316128	0.00000316327563518575	229
REVIEW_ID	357670	0.00000279587329102245	5

Review_criteria

NUM_ROWS	433375
BLOCKS	3646
AVG_ROW_LEN	54
SAMPLE_SIZE	433375

COLUMN_NAME	NUM_DISTINCT	DENSITY	AVG_COL_LEN
TITLE	42	0.00000115583673911136	21
RECOMMENDATION	2	0.00000115625796241279	14
HOUR_PLAYED	4665	0.000214362272240086	4
HELPFUL	447	0.00148367952522255	3
FUNNY	245	0.00408163265306122	3
DATE_POSTED	1953	0.000512032770097286	8
REVIEW_ID	433375	0.00000230747043553504	5